

Measure and Probability Theory

September 11, 2023

Contents

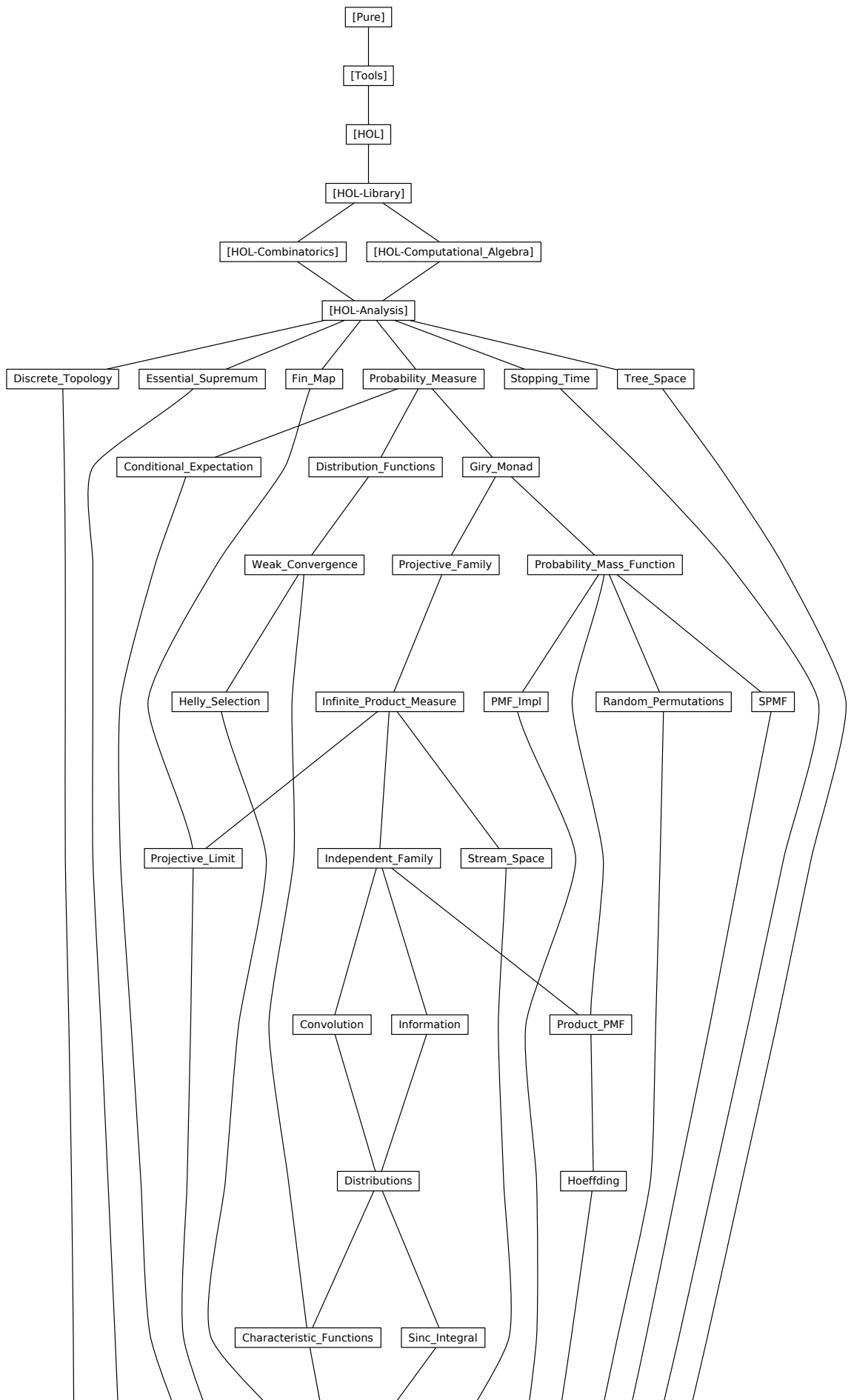
1	Probability measure	3
1.1	Introduce binder for probability	8
1.2	Distributions	14
2	Distribution Functions	32
2.1	Properties of cdf's	32
2.2	Uniqueness	35
3	Weak Convergence of Functions and Distributions	38
4	Weak Convergence of Functions	38
5	Weak Convergence of Distributions	38
6	Skorohod's theorem	39
7	The Giry monad	47
7.1	Sub-probability spaces	47
7.2	Properties of "return"	59
7.3	Join	64
7.4	Giry monad on probability spaces	82
8	Projective Family	87
9	Infinite Product Measure	103
9.1	Sequence space	109
10	Independent families of events, event sets, and random variables	113
11	Convolution Measure	144

12 Information theory	149
12.1 Information theory	150
12.2 Kullback–Leibler divergence	151
12.3 Finite Entropy	156
12.4 Mutual Information	158
12.5 Entropy	166
12.6 Conditional Mutual Information	170
12.7 Conditional Entropy	184
12.8 Equalities	188
13 Properties of Various Distributions	194
13.1 Erlang	196
13.2 Exponential distribution	202
13.3 Uniform distribution	208
13.4 Normal distribution	212
14 Characteristic Functions	226
14.1 Application of the FTC: integrating e^{ix}	227
14.2 The Characteristic Function of a Real Measure.	227
14.3 Independence	228
14.4 Approximations to e^{ix}	229
14.5 Calculation of the Characteristic Function of the Standard Distribution	237
15 Helly’s selection theorem	239
16 Integral of sinc	245
16.1 Various preparatory integrals	246
17 The sinc function, and the sine integral (Si)	248
17.1 The final theorems: boundedness and scalability	253
18 The Levy inversion theorem, and the Levy continuity theo- rem.	255
18.1 The Levy inversion theorem	255
18.2 The Levy continuity theorem	261
19 The Central Limit Theorem	267
20 Probability mass function	272
20.1 PMF as measure	273
20.2 Monad Interpretation	279
20.3 PMFs as function	288
20.4 Conditional Probabilities	297
20.5 Relator	299

20.6	Distributions	310
20.6.1	Bernoulli Distribution	310
20.6.2	Geometric Distribution	311
20.6.3	Uniform Multiset Distribution	313
20.6.4	Uniform Distribution	314
20.6.5	Poisson Distribution	317
20.6.6	Binomial Distribution	318
20.7	Negative Binomial distribution	320
20.8	PMFs from association lists	325
21	Code generation for PMFs	330
21.1	General code generation setup	330
21.2	Code abbreviations for integrals and probabilities	339
22	Finite Maps	342
22.1	Domain and Application	342
22.2	Constructor of Finite Maps	343
22.3	Product set of Finite Maps	343
22.3.1	Basic Properties of Pi'	344
22.4	Topological Space of Finite Maps	344
22.5	Metric Space of Finite Maps	347
22.6	Complete Space of Finite Maps	351
22.7	Second Countable Space of Finite Maps	353
22.8	Polish Space of Finite Maps	355
22.9	Product Measurable Space of Finite Maps	355
22.10	Isomorphism between Functions and Finite Maps	367
23	Projective Limit	371
23.1	Sequences of Finite Maps in Compact Sets	371
23.2	Daniell-Kolmogorov Theorem	373
24	Random Permutations	382
25	Discrete subprobability distribution	386
25.1	Auxiliary material	386
25.1.1	More about extended reals	387
25.1.2	More about <i>'a option</i>	387
25.1.3	A relator for sets that treats sets like predicates	390
25.1.4	Monotonicity rules	390
25.1.5	Bijections	391
25.2	Subprobability mass function	391
25.3	Support	394
25.4	Functorial structure	396
25.5	Monad operations	398

25.5.1	Return	398
25.5.2	Bind	398
25.6	Relator	401
25.7	From 'a pmf to 'a spmf	405
25.8	Weight of a subprobability	406
25.9	From density to spmfs	408
25.10	Ordering on spmfs	409
25.11	CCPO structure for the flat cppo <i>ord-option</i> (=)	415
25.11.1	Admissibility of <i>rel-spmf</i>	426
25.12	Restrictions on spmfs	429
25.13	Subprobability distributions of sets	431
25.14	Losslessness	435
25.15	Scaling	437
25.16	Conditional spmfs	442
25.17	Product spmf	443
25.18	Assertions	447
25.19	Try	448
25.20	Miscellaneous	450
26	Indexed products of PMFs	451
26.1	Preliminaries	451
26.2	Definition	452
26.3	Dependent product sets with a default	454
26.4	Common PMF operations on products	457
26.5	Merging and splitting PMF products	461
26.6	Additional properties	466
26.7	Applications	469
27	Hoeffding's Lemma and Hoeffding's Inequality	471
27.1	Hoeffding's Lemma	471
27.2	Hoeffding's Inequality	475
27.3	Hoeffding's inequality for i.i.d. bounded random variables	478
27.4	Hoeffding's Inequality for the Binomial distribution	481
27.5	Tail bounds for the negative binomial distribution	484
28	Conditional Expectation	514
28.1	Restricting a measure to a sub-sigma-algebra	514
28.2	Nonnegative conditional expectation	518
28.3	Real conditional expectation	525
29	The essential supremum	547
30	Stopping times	550
30.1	Stopping Time	550

31 Filtration	551
31.1 σ -algebra of a Stopping Time	551



1 Probability measure

```
theory Probability-Measure
  imports HOL-Analysis.Analysis
begin
```

```
locale prob-space = finite-measure +
  assumes emeasure-space-1: emeasure M (space M) = 1
```

```
lemma prob-spaceI[Pure.intro!]:
  assumes *: emeasure M (space M) = 1
  shows prob-space M
proof -
  interpret finite-measure M
  proof
    show emeasure M (space M)  $\neq$   $\infty$  using * by simp
  qed
  show prob-space M by standard fact
qed
```

```
lemma prob-space-imp-sigma-finite: prob-space M  $\implies$  sigma-finite-measure M
  unfolding prob-space-def finite-measure-def by simp
```

```
abbreviation (in prob-space) events  $\equiv$  sets M
abbreviation (in prob-space) prob  $\equiv$  measure M
abbreviation (in prob-space) random-variable  $M' X \equiv X \in$  measurable  $M M'$ 
abbreviation (in prob-space) expectation  $\equiv$  integralL M
abbreviation (in prob-space) variance  $X \equiv$  integralL M ( $\lambda x. (X x -$  expectation  $X)^2$ )
```

```
lemma (in prob-space) finite-measure [simp]: finite-measure M
  by unfold-locales
```

```
lemma (in prob-space) prob-space-distr:
  assumes f:  $f \in$  measurable  $M M'$  shows prob-space (distr M M' f)
proof (rule prob-spaceI)
  have f - ' space  $M' \cap$  space  $M =$  space  $M$  using f by (auto dest: measurable-space)
  with f show emeasure (distr M M' f) (space (distr M M' f)) = 1
  by (auto simp: emeasure-distr emeasure-space-1)
qed
```

```
lemma prob-space-distrD:
  assumes f:  $f \in$  measurable  $M N$  and M: prob-space (distr M N f) shows
  prob-space M
proof
  interpret M: prob-space distr M N f by fact
  have f - ' space  $N \cap$  space  $M =$  space  $M$ 
  using f[THEN measurable-space] by auto
```

then show $\text{emeasure } M \text{ (space } M) = 1$
using $M.\text{emeasure-space-1}$ **by** ($\text{simp add: emeasure-distr}[OF f]$)
qed

lemma (**in** prob-space) $\text{prob-space: prob (space } M) = 1$
using emeasure-space-1 **unfolding** measure-def **by** ($\text{simp add: one-ennreal.rep-eq}$)

lemma (**in** prob-space) $\text{prob-le-1}[\text{simp, intro}]: \text{prob } A \leq 1$
using $\text{bounded-measure}[of A]$ **by** ($\text{simp add: prob-space}$)

lemma (**in** prob-space) $\text{not-empty: space } M \neq \{\}$
using prob-space **by** auto

lemma (**in** prob-space) emeasure-eq-1-AE :
 $S \in \text{sets } M \implies \text{AE } x \text{ in } M. x \in S \implies \text{emeasure } M S = 1$
by ($\text{subst emeasure-eq-AE}[\text{where } B=\text{space } M]$) ($\text{auto simp: emeasure-space-1}$)

lemma (**in** prob-space) $\text{emeasure-le-1: emeasure } M S \leq 1$
unfolding $\text{ennreal-1}[\text{symmetric}]$ $\text{emeasure-eq-measure}$ **by** ($\text{subst ennreal-le-iff}$)
 auto

lemma (**in** prob-space) $\text{emeasure-ge-1-iff: emeasure } M A \geq 1 \longleftrightarrow \text{emeasure } M A = 1$
by ($\text{rule iffI, intro antisym emeasure-le-1}$) simp-all

lemma (**in** prob-space) $\text{AE-iff-emeasure-eq-1}$:
assumes $[\text{measurable}]: \text{Measurable.pred } M P$
shows $(\text{AE } x \text{ in } M. P x) \longleftrightarrow \text{emeasure } M \{x \in \text{space } M. P x\} = 1$

proof –

have $*$: $\{x \in \text{space } M. \neg P x\} = \text{space } M - \{x \in \text{space } M. P x\}$

by auto

show $?thesis$

by ($\text{auto simp add: ennreal-minus-eq-0 * emeasure-compl emeasure-space-1}$
 $\text{AE-iff-measurable}[OF - refl]$
 $\text{intro: antisym emeasure-le-1}$)

qed

lemma (**in** prob-space) $\text{measure-le-1: measure } M X \leq 1$
using $\text{emeasure-space}[of M X]$ **by** ($\text{simp add: emeasure-space-1}$)

lemma (**in** prob-space) $\text{measure-ge-1-iff: measure } M A \geq 1 \longleftrightarrow \text{measure } M A = 1$
by ($\text{auto intro!: antisym}$)

lemma (**in** prob-space) AE-I-eq-1 :
assumes $\text{emeasure } M \{x \in \text{space } M. P x\} = 1$ $\{x \in \text{space } M. P x\} \in \text{sets } M$
shows $\text{AE } x \text{ in } M. P x$

proof (rule AE-I)

show $\text{emeasure } M (\text{space } M - \{x \in \text{space } M. P x\}) = 0$

using *assms emeasure-space-1* **by** (*simp add: emeasure-compl*)
qed (*insert assms, auto*)

lemma *prob-space-restrict-space*:

$S \in \text{sets } M \implies \text{emeasure } M S = 1 \implies \text{prob-space } (\text{restrict-space } M S)$
by (*intro prob-spaceI*)
(simp add: emeasure-restrict-space space-restrict-space)

lemma (*in prob-space*) *prob-compl*:

assumes $A: A \in \text{events}$
shows $\text{prob } (\text{space } M - A) = 1 - \text{prob } A$
using *finite-measure-compl[OF A]* **by** (*simp add: prob-space*)

lemma (*in prob-space*) *AE-in-set-eq-1*:

assumes $A[\text{measurable}]: A \in \text{events}$ **shows** $(AE x \text{ in } M. x \in A) \longleftrightarrow \text{prob } A = 1$
proof –
have $*$: $\{x \in \text{space } M. x \in A\} = A$
using $A[\text{THEN sets.sets-into-space}]$ **by** *auto*
show *?thesis*
by (*subst AE-iff-emeasure-eq-1*) (*auto simp: emeasure-eq-measure **)
qed

lemma (*in prob-space*) *AE-False*: $(AE x \text{ in } M. \text{False}) \longleftrightarrow \text{False}$

proof

assume $AE x \text{ in } M. \text{False}$
then have $AE x \text{ in } M. x \in \{\}$ **by** *simp*
then show *False*
by (*subst (asm) AE-in-set-eq-1*) *auto*
qed *simp*

lemma (*in prob-space*) *AE-prob-1*:

assumes $\text{prob } A = 1$ **shows** $AE x \text{ in } M. x \in A$
proof –
from $\langle \text{prob } A = 1 \rangle$ **have** $A \in \text{events}$
by (*metis measure-notin-sets zero-neq-one*)
with *AE-in-set-eq-1 assms* **show** *?thesis* **by** *simp*
qed

lemma (*in prob-space*) *AE-const[simp]*: $(AE x \text{ in } M. P) \longleftrightarrow P$

by (*cases P*) (*auto simp: AE-False*)

lemma (*in prob-space*) *ae-filter-bot*: $ae\text{-filter } M \neq \text{bot}$

by (*simp add: trivial-limit-def*)

lemma (*in prob-space*) *AE-contr*:

assumes $ae: AE \omega \text{ in } M. P \omega$ $AE \omega \text{ in } M. \neg P \omega$
shows *False*
proof –

from *ae* **have** $AE \omega$ *in* M . *False* **by** *eventually-elim auto*
then show *False* **by** *auto*
qed

lemma (*in prob-space*) *integral-ge-const*:
fixes $c :: real$
shows *integrable* $M f \implies (AE x \text{ in } M. c \leq f x) \implies c \leq (\int x. f x \partial M)$
using *integral-mono-AE*[*of* $M \lambda x. c f$] *prob-space* **by** *simp*

lemma (*in prob-space*) *integral-le-const*:
fixes $c :: real$
shows *integrable* $M f \implies (AE x \text{ in } M. f x \leq c) \implies (\int x. f x \partial M) \leq c$
using *integral-mono-AE*[*of* $M f \lambda x. c$] *prob-space* **by** *simp*

lemma (*in prob-space*) *nn-integral-ge-const*:
 $(AE x \text{ in } M. c \leq f x) \implies c \leq (\int^+ x. f x \partial M)$
using *nn-integral-mono-AE*[*of* $\lambda x. c f M$] *emeasure-space-1*
by (*simp split: if-split-asm*)

lemma (*in prob-space*) *expectation-less*:
fixes $X :: - \Rightarrow real$
assumes [*simp*]: *integrable* $M X$
assumes *gt*: $AE x \text{ in } M. X x < b$
shows *expectation* $X < b$
proof –
have *expectation* $X < \text{expectation } (\lambda x. b)$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show *?thesis* **using** *prob-space* **by** *simp*
qed

lemma (*in prob-space*) *expectation-greater*:
fixes $X :: - \Rightarrow real$
assumes [*simp*]: *integrable* $M X$
assumes *gt*: $AE x \text{ in } M. a < X x$
shows $a < \text{expectation } X$
proof –
have *expectation* $(\lambda x. a) < \text{expectation } X$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show *?thesis* **using** *prob-space* **by** *simp*
qed

lemma (*in prob-space*) *jensens-inequality*:
fixes $q :: real \Rightarrow real$
assumes X : *integrable* $M X$ $AE x \text{ in } M. X x \in I$
assumes I : $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$
assumes q : *integrable* $M (\lambda x. q (X x))$ *convex-on* $I q$
shows $q (\text{expectation } X) \leq \text{expectation } (\lambda x. q (X x))$

proof –

let $?F = \lambda x. \text{Inf } ((\lambda t. (q\ x - q\ t) / (x - t)) \text{ ‘ } (\{x < ..\} \cap I))$
from $X(2)$ *AE-False* **have** $I \neq \{\}$ **by** *auto*

from I **have** *open* I **by** *auto*

note I

moreover

{ **assume** $I \subseteq \{a < ..\}$
with X **have** $a < \text{expectation } X$
by (*intro expectation-greater*) *auto* }

moreover

{ **assume** $I \subseteq \{.. < b\}$
with X **have** $\text{expectation } X < b$
by (*intro expectation-less*) *auto* }

ultimately have $\text{expectation } X \in I$

by (*elim disjE*) (*auto simp: subset-eq*)

moreover

{ **fix** y **assume** $y: y \in I$
with $q(2)$ *⟨open I⟩* **have** $\text{Sup } ((\lambda x. q\ x + ?F\ x * (y - x)) \text{ ‘ } I) = q\ y$
by (*auto intro!: cSup-eq-maximum convex-le-Inf-differential image-eqI [OF -*
 $y]$ *simp: interior-open*) }

ultimately have $q(\text{expectation } X) = \text{Sup } ((\lambda x. q\ x + ?F\ x * (\text{expectation } X - x)) \text{ ‘ } I)$

by *simp*

also have $\dots \leq \text{expectation } (\lambda w. q(X\ w))$

proof (*rule cSup-least*)

show $(\lambda x. q\ x + ?F\ x * (\text{expectation } X - x)) \text{ ‘ } I \neq \{\}$

using $\langle I \neq \{\} \rangle$ **by** *auto*

next

fix k **assume** $k \in (\lambda x. q\ x + ?F\ x * (\text{expectation } X - x)) \text{ ‘ } I$

then obtain x

where $x: k = q\ x + (\text{INF } t \in \{x < ..\} \cap I. (q\ x - q\ t) / (x - t)) * (\text{expectation } X - x)$ $x \in I$..

have $q\ x + ?F\ x * (\text{expectation } X - x) = \text{expectation } (\lambda w. q\ x + ?F\ x * (X\ w - x))$

using *prob-space* **by** (*simp add: X*)

also have $\dots \leq \text{expectation } (\lambda w. q(X\ w))$

using $\langle x \in I \rangle$ *⟨open I⟩* $X(2)$

apply (*intro integral-mono-AE Bochner-Integration.integrable-add Bochner-Integration.integrable-mult-right Bochner-Integration.integrable-diff*

integrable-const X q)

apply (*elim eventually-mono*)

apply (*intro convex-le-Inf-differential*)

apply (*auto simp: interior-open q*)

done

finally show $k \leq \text{expectation } (\lambda w. q(X\ w))$ **using** x **by** *auto*

qed

finally show $q(\text{expectation } X) \leq \text{expectation } (\lambda x. q(X\ x))$.

qed

1.1 Introduce binder for probability

syntax

-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* (⟨('P'((/- in ./ -)')⟩)⟩)

translations

$\mathcal{P}(x \text{ in } M. P) \Rightarrow \text{CONST measure } M \{x \in \text{CONST space } M. P\}$

print-translation ‹

let

fun to-pattern (*Const* (**const-syntax** ‹*Pair*›, -) \$ *l* \$ *r*) =
Syntax.const **const-syntax** ‹*Pair*› :: *to-pattern l* @ *to-pattern r*
 | *to-pattern* (*t as* (*Const* (**syntax-const** ‹*-bound*›, -)) \$ -) = [*t*]

fun mk-pattern ((*t*, *n*) :: *xs*) = *mk-patterns n xs* |>> *curry list-comb t*
 and *mk-patterns 0 xs* = ([], *xs*)
 | *mk-patterns n xs* =

let

val (*t*, *xs'*) = *mk-pattern xs*
val (*ts*, *xs''*) = *mk-patterns (n - 1) xs'*

in

(*t* :: *ts*, *xs''*)

end

fun unnest-tuples

(*Const* (**syntax-const** ‹*-pattern*›, -) \$
t1 \$
 (*t as* (*Const* (**syntax-const** ‹*-pattern*›, -) \$ - \$ -)))

= *let*

val (- \$ *t2* \$ *t3*) = *unnest-tuples t*

in

Syntax.const **syntax-const** ‹*-pattern*› \$
unnest-tuples t1 \$
 (*Syntax.const* **syntax-const** ‹*-patterns*› \$ *t2* \$ *t3*)

end

| *unnest-tuples pat* = *pat*

fun tr' [*sig-alg*, *Const* (**const-syntax** ‹*Collect*›, -) \$ *t*] =

let

val bound-dummyT = *Const* (**syntax-const** ‹*-bound*›, *dummyT*)

fun go pattern elem

(*Const* (**const-syntax** ‹*conj*›, -) \$

(*Const* (**const-syntax** ‹*Set.member*›, -) \$ *elem'* \$ (*Const* (**const-syntax** ‹*space*›, -) \$ *sig-alg'*)) \$

u)

= *let*

```

    val - = if sig-alg aconv sig-alg' andalso to-pattern elem' = rev elem then
() else raise Match;
    val (pat, rest) = mk-pattern (rev pattern);
    val - = case rest of [] => () | - => raise Match
    in
      Syntax.const syntax-const ⟨-prob⟩ $ unnest-tuples pat $ sig-alg $ u
    end
  | go pattern elem (Abs abs) =
    let
      val (x as (- $ tx), t) = Syntax-Trans.atomic-abs-tr' abs
    in
      go ((x, 0) :: pattern) (bound-dummyT $ tx :: elem) t
    end
  | go pattern elem (Const (const-syntax ⟨case-prod⟩, -) $ t) =
    go
      ((Syntax.const syntax-const ⟨-pattern⟩, 2) :: pattern)
      (Syntax.const const-syntax ⟨Pair⟩ :: elem)
      t
    in
      go [] [] t
    end
  in
    [(const-syntax ⟨Sigma-Algebra.measure⟩, K tr')]
  end
end

```

definition

$$\text{cond-prob } M P Q = \mathcal{P}(\omega \text{ in } M. P \omega \wedge Q \omega) / \mathcal{P}(\omega \text{ in } M. Q \omega)$$
syntax

$$\text{-conditional-prob} :: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \langle \langle 'P'(- \text{ in } -. - \text{ || } -') \rangle \rangle$$
translations

$$\mathcal{P}(x \text{ in } M. P \mid Q) \Rightarrow \text{CONST cond-prob } M (\lambda x. P) (\lambda x. Q)$$
lemma (in prob-space) AE-E-prob:

assumes *ae*: AE *x* in *M*. *P* *x*

obtains *S* **where** $S \subseteq \{x \in \text{space } M. P x\}$ *S* ∈ events *prob* *S* = 1

proof –

from *ae*[*THEN* AE-E] **obtain** *N*

where $\{x \in \text{space } M. \neg P x\} \subseteq N$ *emeasure* *M* *N* = 0 *N* ∈ events **by** *auto*

then show *thesis*

by (*intro that*[of space *M* – *N*])

(*auto simp: prob-compl prob-space emeasure-eq-measure measure-nonneg*)

qed

lemma (in prob-space) prob-neg: $\{x \in \text{space } M. P x\} \in \text{events} \implies \mathcal{P}(x \text{ in } M. \neg P x) = 1 - \mathcal{P}(x \text{ in } M. P x)$

by (*auto intro!: arg-cong*[**where** *f*=*prob*] *simp add: prob-compl*[*symmetric*])

lemma (in *prob-space*) *prob-eq-AE*:

$(AE\ x\ in\ M.\ P\ x \longleftrightarrow Q\ x) \implies \{x \in space\ M.\ P\ x\} \in events \implies \{x \in space\ M.\ Q\ x\} \in events \implies \mathcal{P}(x\ in\ M.\ P\ x) = \mathcal{P}(x\ in\ M.\ Q\ x)$

by (rule *finite-measure-eq-AE*) *auto*

lemma (in *prob-space*) *prob-eq-0-AE*:

assumes not: $AE\ x\ in\ M.\ \neg\ P\ x$ shows $\mathcal{P}(x\ in\ M.\ P\ x) = 0$

proof *cases*

assume $\{x \in space\ M.\ P\ x\} \in events$

with not have $\mathcal{P}(x\ in\ M.\ P\ x) = \mathcal{P}(x\ in\ M.\ False)$

by (*intro prob-eq-AE*) *auto*

then show *?thesis* by *simp*

qed (*simp add: measure-notin-sets*)

lemma (in *prob-space*) *prob-Collect-eq-0*:

$\{x \in space\ M.\ P\ x\} \in sets\ M \implies \mathcal{P}(x\ in\ M.\ P\ x) = 0 \longleftrightarrow (AE\ x\ in\ M.\ \neg\ P\ x)$

using *AE-iff-measurable[OF - refl, of M $\lambda x.\ \neg\ P\ x$]* by (*simp add: emeasure-eq-measure measure-nonneg*)

lemma (in *prob-space*) *prob-Collect-eq-1*:

$\{x \in space\ M.\ P\ x\} \in sets\ M \implies \mathcal{P}(x\ in\ M.\ P\ x) = 1 \longleftrightarrow (AE\ x\ in\ M.\ P\ x)$

using *AE-in-set-eq-1* [of $\{x \in space\ M.\ P\ x\}$] by *simp*

lemma (in *prob-space*) *prob-eq-0*:

$A \in sets\ M \implies prob\ A = 0 \longleftrightarrow (AE\ x\ in\ M.\ x \notin A)$

using *AE-iff-measurable[OF - refl, of M $\lambda x.\ x \notin A$]*

by (*auto simp add: emeasure-eq-measure Int-def[symmetric] measure-nonneg*)

lemma (in *prob-space*) *prob-eq-1*:

$A \in sets\ M \implies prob\ A = 1 \longleftrightarrow (AE\ x\ in\ M.\ x \in A)$

using *AE-in-set-eq-1* [of A] by *simp*

lemma (in *prob-space*) *prob-sums*:

assumes $P: \bigwedge n.\ \{x \in space\ M.\ P\ n\ x\} \in events$

assumes $Q: \{x \in space\ M.\ Q\ x\} \in events$

assumes $ae: AE\ x\ in\ M.\ (\forall n.\ P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists !n.\ P\ n\ x))$

shows $(\lambda n.\ \mathcal{P}(x\ in\ M.\ P\ n\ x))\ sums\ \mathcal{P}(x\ in\ M.\ Q\ x)$

proof –

from ae [THEN *AE-E-prob*] obtain S

where S :

$S \subseteq \{x \in space\ M.\ (\forall n.\ P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists !n.\ P\ n\ x))\}$

$S \in events$

$prob\ S = 1$

by *auto*

then have *disj*: *disjoint-family* $(\lambda n.\ \{x \in space\ M.\ P\ n\ x\} \cap S)$

by (*auto simp: disjoint-family-on-def*)

from S have ae - S :

$AE\ x\ in\ M.\ x \in \{x \in space\ M.\ Q\ x\} \longleftrightarrow x \in (\bigcup n.\ \{x \in space\ M.\ P\ n\ x\} \cap S)$

$\bigwedge n. AE\ x\ in\ M. x \in \{x \in space\ M. P\ n\ x\} \longleftrightarrow x \in \{x \in space\ M. P\ n\ x\} \cap S$
using *ae* **by** (*auto dest!*: *AE-prob-1*)
from *ae-S* **have** *:
 $\mathcal{P}(x\ in\ M. Q\ x) = prob\ (\bigcup n. \{x \in space\ M. P\ n\ x\} \cap S)$
using *P Q S* **by** (*intro finite-measure-eq-AE*) *auto*
from *ae-S* **have** **:
 $\bigwedge n. \mathcal{P}(x\ in\ M. P\ n\ x) = prob\ (\{x \in space\ M. P\ n\ x\} \cap S)$
using *P Q S* **by** (*intro finite-measure-eq-AE*) *auto*
show *?thesis*
unfolding * ** **using** *S P disj*
by (*intro finite-measure-UNION*) *auto*
qed

lemma (*in prob-space*) *prob-sum*:
assumes [*simp, intro*]: *finite I*
assumes *P*: $\bigwedge n. n \in I \implies \{x \in space\ M. P\ n\ x\} \in events$
assumes *Q*: $\{x \in space\ M. Q\ x\} \in events$
assumes *ae*: *AE x in M. $(\forall n \in I. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists ! n \in I. P\ n\ x))$*
shows $\mathcal{P}(x\ in\ M. Q\ x) = (\sum n \in I. \mathcal{P}(x\ in\ M. P\ n\ x))$
proof –
from *ae*[*THEN AE-E-prob*] **obtain** *S*
where *S*:
 $S \subseteq \{x \in space\ M. (\forall n \in I. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists ! n. n \in I \wedge P\ n\ x))\}$
 $S \in events$
 $prob\ S = 1$
by *auto*
then **have** *disj*: *disjoint-family-on* $(\lambda n. \{x \in space\ M. P\ n\ x\} \cap S)$ *I*
by (*auto simp: disjoint-family-on-def*)
from *S* **have** *ae-S*:
 $AE\ x\ in\ M. x \in \{x \in space\ M. Q\ x\} \longleftrightarrow x \in (\bigcup n \in I. \{x \in space\ M. P\ n\ x\} \cap S)$
 $\bigwedge n. n \in I \implies AE\ x\ in\ M. x \in \{x \in space\ M. P\ n\ x\} \longleftrightarrow x \in \{x \in space\ M. P\ n\ x\} \cap S$
using *ae* **by** (*auto dest!*: *AE-prob-1*)
from *ae-S* **have** *:
 $\mathcal{P}(x\ in\ M. Q\ x) = prob\ (\bigcup n \in I. \{x \in space\ M. P\ n\ x\} \cap S)$
using *P Q S* **by** (*intro finite-measure-eq-AE*) (*auto intro!*: *sets.Int*)
from *ae-S* **have** **:
 $\bigwedge n. n \in I \implies \mathcal{P}(x\ in\ M. P\ n\ x) = prob\ (\{x \in space\ M. P\ n\ x\} \cap S)$
using *P Q S* **by** (*intro finite-measure-eq-AE*) *auto*
show *?thesis*
using *S P disj*
by (*auto simp add: * ** simp del: UN-simps intro!*: *finite-measure-finite-Union*)
qed

lemma (*in prob-space*) *prob-EX-countable*:
assumes *sets*: $\bigwedge i. i \in I \implies \{x \in space\ M. P\ i\ x\} \in sets\ M$ **and** *I*: *countable I*
assumes *disj*: *AE x in M. $\forall i \in I. \forall j \in I. P\ i\ x \longrightarrow P\ j\ x \longrightarrow i = j$*
shows $\mathcal{P}(x\ in\ M. \exists i \in I. P\ i\ x) = (\int^{+i. \mathcal{P}(x\ in\ M. P\ i\ x)} \partial count-space\ I)$

proof –

let $?N = \lambda x. \exists ! i \in I. P i x$
have $\text{ennreal } (\mathcal{P}(x \text{ in } M. \exists i \in I. P i x)) = \mathcal{P}(x \text{ in } M. (\exists i \in I. P i x \wedge ?N x))$
unfolding $\text{ennreal-inj}[OF \text{ measure-nonneg measure-nonneg}]$
proof (rule prob-eq-AE)
show $AE x \text{ in } M. (\exists i \in I. P i x) = (\exists i \in I. P i x \wedge ?N x)$
using $\text{disj by eventually-elim blast}$
qed ($\text{auto intro! : sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1'}$
 $I \text{ sets}$) +
also have $\mathcal{P}(x \text{ in } M. (\exists i \in I. P i x \wedge ?N x)) = \text{emeasure } M (\bigcup_{i \in I. \{x \in \text{space } M. P i x \wedge ?N x\}})$
unfolding $\text{emeasure-eq-measure by (auto intro! : arg-cong[where f=prob] simp:}$
 $\text{measure-nonneg})$
also have $\dots = (\int^{+i. \text{emeasure } M \{x \in \text{space } M. P i x \wedge ?N x\} \partial \text{count-space } I)$
by ($\text{rule emeasure-UN-countable}$)
 $(\text{auto intro! : sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1'}$
 $I \text{ sets}$
 $\text{simp: disjoint-family-on-def})$
also have $\dots = (\int^{+i. \mathcal{P}(x \text{ in } M. P i x) \partial \text{count-space } I)$
unfolding $\text{emeasure-eq-measure using disj}$
by ($\text{intro nn-integral-cong ennreal-inj[THEN iffD2] prob-eq-AE}$)
 $(\text{auto intro! : sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1'}$
 $I \text{ sets measure-nonneg}) +$
finally show $?thesis .$
qed

lemma (in prob-space) cond-prob-eq-AE :

assumes $P: AE x \text{ in } M. Q x \longrightarrow P x \longleftrightarrow P' x \{x \in \text{space } M. P x\} \in \text{events}$
 $\{x \in \text{space } M. P' x\} \in \text{events}$
assumes $Q: AE x \text{ in } M. Q x \longleftrightarrow Q' x \{x \in \text{space } M. Q x\} \in \text{events}$
 $\{x \in \text{space } M. Q' x\} \in \text{events}$
shows $\text{cond-prob } M P Q = \text{cond-prob } M P' Q'$
using $P Q$
by ($\text{auto simp: cond-prob-def intro! : arg-cong2[where f=(/)] prob-eq-AE sets.sets-Collect-conj}$)

lemma (in prob-space) $\text{joint-distribution-Times-le-fst}$:

$\text{random-variable } MX X \Longrightarrow \text{random-variable } MY Y \Longrightarrow A \in \text{sets } MX \Longrightarrow B \in \text{sets } MY$
 $\Longrightarrow \text{emeasure } (\text{distr } M (MX \otimes_M MY) (\lambda x. (X x, Y x))) (A \times B) \leq \text{emeasure}$
 $(\text{distr } M MX X) A$
by ($\text{auto simp: emeasure-distr measurable-pair-iff comp-def intro! : emeasure-mono}$
 measurable-sets)

lemma (in prob-space) $\text{joint-distribution-Times-le-snd}$:

$\text{random-variable } MX X \Longrightarrow \text{random-variable } MY Y \Longrightarrow A \in \text{sets } MX \Longrightarrow B \in \text{sets } MY$
 $\Longrightarrow \text{emeasure } (\text{distr } M (MX \otimes_M MY) (\lambda x. (X x, Y x))) (A \times B) \leq \text{emeasure}$
 $(\text{distr } M MY Y) B$

by (*auto simp: emeasure-distr measurable-pair-iff comp-def intro!: emeasure-mono measurable-sets*)

lemma (*in prob-space*) *variance-eq*:

fixes $X :: 'a \Rightarrow \text{real}$

assumes [*simp*]: *integrable* M X

assumes [*simp*]: *integrable* M $(\lambda x. (X\ x)^2)$

shows *variance* $X = \text{expectation} (\lambda x. (X\ x)^2) - (\text{expectation } X)^2$

by (*simp add: field-simps prob-space power2-diff power2-eq-square[symmetric]*)

lemma (*in prob-space*) *variance-positive*: $0 \leq \text{variance } (X :: 'a \Rightarrow \text{real})$

by (*intro integral-nonneg-AE*) (*auto intro!: integral-nonneg-AE*)

lemma (*in prob-space*) *variance-mean-zero*:

expectation $X = 0 \implies \text{variance } X = \text{expectation} (\lambda x. (X\ x)^2)$

by *simp*

theorem (*in prob-space*) *Chebyshev-inequality*:

assumes [*measurable*]: *random-variable borel* f

assumes *integrable* M $(\lambda x. f\ x^2)$

defines $\mu \equiv \text{expectation } f$

assumes $a > 0$

shows *prob* $\{x \in \text{space } M. |f\ x - \mu| \geq a\} \leq \text{variance } f / a^2$

unfolding $\mu\text{-def}$

proof (*rule second-moment-method*)

have *integrable*: *integrable* M f

using *assms by* (*blast dest: square-integrable-imp-integrable*)

show *integrable* M $(\lambda x. (f\ x - \text{expectation } f)^2)$

using *assms integrable unfolding power2-eq-square ring-distrib*

by (*intro Bochner-Integration.integrable-diff*) *auto*

qed (*use assms in auto*)

locale *pair-prob-space* = *pair-sigma-finite* $M1\ M2 + M1$: *prob-space* $M1 + M2$:
prob-space $M2$ **for** $M1\ M2$

sublocale *pair-prob-space* $\subseteq P?$: *prob-space* $M1 \otimes_M M2$

proof

show *emeasure* $(M1 \otimes_M M2)$ (*space* $(M1 \otimes_M M2)$) = 1

by (*simp add: M2.emeasure-pair-measure-Times M1.emeasure-space-1 M2.emeasure-space-1 space-pair-measure*)

qed

locale *product-prob-space* = *product-sigma-finite* M **for** $M :: 'i \Rightarrow 'a \text{ measure} +$

fixes $I :: 'i \text{ set}$

assumes *prob-space*: $\bigwedge i. \text{prob-space } (M\ i)$

sublocale *product-prob-space* $\subseteq M?$: *prob-space* $M\ i$ **for** i

by (*rule prob-space*)

locale *finite-product-prob-space* = *finite-product-sigma-finite* $M I$ + *product-prob-space* $M I$ for $M I$

sublocale *finite-product-prob-space* \subseteq *prob-space* $\prod_{M i \in I} M i$

proof

show *emeasure* $(\prod_{M i \in I} M i)$ (*space* $(\prod_{M i \in I} M i)$) = 1

by (*simp add: measure-times M.emeasure-space-1 prod.neutral-const space-PiM*)

qed

lemma (in *finite-product-prob-space*) *prob-times*:

assumes $X: \bigwedge i. i \in I \implies X i \in \text{sets } (M i)$

shows *prob* $(\prod_{E i \in I} X i) = (\prod_{i \in I} M.\text{prob } i (X i))$

proof –

have *ennreal* (*measure* $(\prod_{M i \in I} M i)$ $(\prod_{E i \in I} X i)$) = *emeasure* $(\prod_{M i \in I} M i)$ $(\prod_{E i \in I} X i)$

using X **by** (*simp add: emeasure-eq-measure*)

also have $\dots = (\prod_{i \in I} \text{emeasure } (M i) (X i))$

using *measure-times X* **by** *simp*

also have $\dots = \text{ennreal } (\prod_{i \in I} \text{measure } (M i) (X i))$

using X **by** (*simp add: M.emeasure-eq-measure prod-ennreal measure-nonneg*)

finally show *?thesis* **by** (*simp add: measure-nonneg prod-nonneg*)

qed

lemma *product-prob-spaceI*:

assumes $\bigwedge i. \text{prob-space } (M i)$

shows *product-prob-space* M

unfolding *product-prob-space-def product-prob-space-axioms-def product-sigma-finite-def*

proof *safe*

fix i

interpret *prob-space* $M i$

by (*rule assms*)

show *sigma-finite-measure* $(M i)$ *prob-space* $(M i)$

by *unfold-locales*

qed

1.2 Distributions

definition *distributed* :: 'a *measure* \implies 'b *measure* \implies ('a \implies 'b) \implies ('b \implies *ennreal*) \implies *bool*

where

distributed $M N X f \longleftrightarrow$

distr $M N X = \text{density } N f \wedge f \in \text{borel-measurable } N \wedge X \in \text{measurable } M N$

lemma

assumes *distributed* $M N X f$

shows *distributed-distr-eq-density*: *distr* $M N X = \text{density } N f$

and *distributed-measurable*: $X \in \text{measurable } M N$

and *distributed-borel-measurable*: $f \in \text{borel-measurable } N$

using *assms* **by** (*simp-all add: distributed-def*)

lemma

assumes D : *distributed* $M N X f$

shows *distributed-measurable'*[*measurable-dest*]:

$g \in \text{measurable } L M \implies (\lambda x. X (g x)) \in \text{measurable } L N$

and *distributed-borel-measurable'*[*measurable-dest*]:

$h \in \text{measurable } L N \implies (\lambda x. f (h x)) \in \text{borel-measurable } L$

using *distributed-measurable*[*OF D*] *distributed-borel-measurable*[*OF D*]

by *simp-all*

lemma *distributed-real-measurable*:

$(\bigwedge x. x \in \text{space } N \implies 0 \leq f x) \implies \text{distributed } M N X (\lambda x. \text{ennreal } (f x)) \implies f \in \text{borel-measurable } N$

by (*simp-all add: distributed-def*)

lemma *distributed-real-measurable'*:

$(\bigwedge x. x \in \text{space } N \implies 0 \leq f x) \implies \text{distributed } M N X (\lambda x. \text{ennreal } (f x)) \implies$

$h \in \text{measurable } L N \implies (\lambda x. f (h x)) \in \text{borel-measurable } L$

using *distributed-real-measurable*[*measurable*] **by** *simp*

lemma *joint-distributed-measurable1*:

distributed $M (S \otimes_M T) (\lambda x. (X x, Y x)) f \implies h1 \in \text{measurable } N M \implies (\lambda x. X (h1 x)) \in \text{measurable } N S$

by *simp*

lemma *joint-distributed-measurable2*:

distributed $M (S \otimes_M T) (\lambda x. (X x, Y x)) f \implies h2 \in \text{measurable } N M \implies (\lambda x. Y (h2 x)) \in \text{measurable } N T$

by *simp*

lemma *distributed-count-space*:

assumes X : *distributed* $M (\text{count-space } A) X P$ **and** a : $a \in A$ **and** A : *finite* A

shows $P a = \text{emeasure } M (X - \{a\} \cap \text{space } M)$

proof –

have $\text{emeasure } M (X - \{a\} \cap \text{space } M) = \text{emeasure } (\text{distr } M (\text{count-space } A) X) \{a\}$

using $X a A$ **by** (*simp add: emeasure-distr*)

also have $\dots = \text{emeasure } (\text{density } (\text{count-space } A) P) \{a\}$

using X **by** (*simp add: distributed-distr-eq-density*)

also have $\dots = (\int^+ x. P a * \text{indicator } \{a\} x \partial \text{count-space } A)$

using $X a$ **by** (*auto simp add: emeasure-density distributed-def indicator-def intro!: nn-integral-cong*)

also have $\dots = P a$

using $X a$ **by** (*subst nn-integral-cmult-indicator*) (*auto simp: distributed-def one-ennreal-def[symmetric] AE-count-space*)

finally show *?thesis ..*

qed

lemma *distributed-cong-density*:

$(AE\ x\ in\ N.\ f\ x = g\ x) \implies g \in \text{borel-measurable } N \implies f \in \text{borel-measurable } N$
 \implies
 $\text{distributed } M\ N\ X\ f \longleftrightarrow \text{distributed } M\ N\ X\ g$
by (*auto simp: distributed-def intro!: density-cong*)

lemma (*in prob-space*) *distributed-imp-emeasure-nonzero*:

assumes X : *distributed* $M\ MX\ X\ Px$

shows *emeasure* $MX\ \{x \in \text{space } MX.\ Px\ x \neq 0\} \neq 0$

proof

note $Px = \text{distributed-borel-measurable}[OF\ X]$

interpret X : *prob-space distr* $M\ MX\ X$

using *distributed-measurable*[$OF\ X$] **by** (*rule prob-space-distr*)

assume *emeasure* $MX\ \{x \in \text{space } MX.\ Px\ x \neq 0\} = 0$

with Px **have** $AE\ x\ in\ MX.\ Px\ x = 0$

by (*intro AE-I*[$OF\ \text{subset-refl}$]) (*auto simp: borel-measurable-ennreal-iff*)

moreover

from $X.\text{emeasure-space-1}$ **have** $(\int^+ x.\ Px\ x\ \partial MX) = 1$

unfolding *distributed-distr-eq-density*[$OF\ X$] **using** Px

by (*subst (asm) emeasure-density*)

(*auto simp: borel-measurable-ennreal-iff intro!: integral-cong cong: nn-integral-cong*)

ultimately show *False*

by (*simp add: nn-integral-cong-AE*)

qed

lemma *subdensity*:

assumes T : $T \in \text{measurable } P\ Q$

assumes f : *distributed* $M\ P\ X\ f$

assumes g : *distributed* $M\ Q\ Y\ g$

assumes Y : $Y = T \circ X$

shows $AE\ x\ in\ P.\ g\ (T\ x) = 0 \longrightarrow f\ x = 0$

proof –

have $\{x \in \text{space } Q.\ g\ x = 0\} \in \text{null-sets } (\text{distr } M\ Q\ (T \circ X))$

using $g\ Y$ **by** (*auto simp: null-sets-density-iff distributed-def*)

also have $\text{distr } M\ Q\ (T \circ X) = \text{distr } (\text{distr } M\ P\ X)\ Q\ T$

using $T\ f$ [*THEN distributed-measurable*] **by** (*rule distr-distr*[*symmetric*])

finally have $T - \{x \in \text{space } Q.\ g\ x = 0\} \cap \text{space } P \in \text{null-sets } (\text{distr } M\ P\ X)$

using T **by** (*subst (asm) null-sets-distr-iff*) *auto*

also have $T - \{x \in \text{space } Q.\ g\ x = 0\} \cap \text{space } P = \{x \in \text{space } P.\ g\ (T\ x) = 0\}$

using T **by** (*auto dest: measurable-space*)

finally show *?thesis*

using $f\ g$ **by** (*auto simp add: null-sets-density-iff distributed-def*)

qed

lemma *subdensity-real*:

fixes $g :: 'a \Rightarrow \text{real}$ **and** $f :: 'b \Rightarrow \text{real}$

assumes T : $T \in \text{measurable } P\ Q$

assumes f : *distributed* $M\ P\ X\ f$

assumes g : *distributed* $M\ Q\ Y\ g$

assumes $Y: Y = T \circ X$
shows $(AE\ x\ in\ P.\ 0 \leq g\ (T\ x)) \implies (AE\ x\ in\ P.\ 0 \leq f\ x) \implies AE\ x\ in\ P.\ g\ (T\ x) = 0 \implies f\ x = 0$
using *subdensity*[*OF* T , of $M\ X\ \lambda x.\ ennreal\ (f\ x)\ Y\ \lambda x.\ ennreal\ (g\ x)$] *assms*
by *auto*

lemma *distributed-emeasure*:

$distributed\ M\ N\ X\ f \implies A \in sets\ N \implies emeasure\ M\ (X\ -' A \cap space\ M) = (\int^+ x.\ f\ x * indicator\ A\ x\ \partial N)$
by (*auto simp: distributed-distr-eq-density*[*symmetric*] *emeasure-density*[*symmetric*] *emeasure-distr*)

lemma *distributed-nn-integral*:

$distributed\ M\ N\ X\ f \implies g \in borel-measurable\ N \implies (\int^+ x.\ f\ x * g\ x\ \partial N) = (\int^+ x.\ g\ (X\ x)\ \partial M)$
by (*auto simp: distributed-distr-eq-density*[*symmetric*] *nn-integral-density*[*symmetric*] *nn-integral-distr*)

lemma *distributed-integral*:

$distributed\ M\ N\ X\ f \implies g \in borel-measurable\ N \implies (\bigwedge x.\ x \in space\ N \implies 0 \leq f\ x) \implies (\int x.\ f\ x * g\ x\ \partial N) = (\int x.\ g\ (X\ x)\ \partial M)$
supply *distributed-real-measurable*[*measurable*]
by (*auto simp: distributed-distr-eq-density*[*symmetric*] *integral-real-density*[*symmetric*] *integral-distr*)

lemma *distributed-transform-integral*:

assumes $Px: distributed\ M\ N\ X\ Px \bigwedge x.\ x \in space\ N \implies 0 \leq Px\ x$
assumes $distributed\ M\ P\ Y\ Py \bigwedge x.\ x \in space\ P \implies 0 \leq Py\ x$
assumes $Y: Y = T \circ X$ **and** $T: T \in measurable\ N\ P$ **and** $f: f \in borel-measurable\ P$
shows $(\int x.\ Py\ x * f\ x\ \partial P) = (\int x.\ Px\ x * f\ (T\ x)\ \partial N)$
proof –
have $(\int x.\ Py\ x * f\ x\ \partial P) = (\int x.\ f\ (Y\ x)\ \partial M)$
by (*rule distributed-integral*) *fact+*
also have $\dots = (\int x.\ f\ (T\ (X\ x))\ \partial M)$
using Y **by** *simp*
also have $\dots = (\int x.\ Px\ x * f\ (T\ x)\ \partial N)$
using *measurable-comp*[*OF* $T\ f$] Px **by** (*intro distributed-integral*[*symmetric*])
(auto simp: comp-def)
finally show *?thesis* .
qed

lemma (*in prob-space*) *distributed-unique*:

assumes $Px: distributed\ M\ S\ X\ Px$
assumes $Py: distributed\ M\ S\ X\ Py$
shows $AE\ x\ in\ S.\ Px\ x = Py\ x$
proof –
interpret $X: prob-space\ distr\ M\ S\ X$

```

    using Px by (intro prob-space-distr) simp
    have sigma-finite-measure (distr M S X) ..
    with sigma-finite-density-unique[of Px S Py ] Px Py
    show ?thesis
    by (auto simp: distributed-def)
qed

```

lemma (in prob-space) distributed-jointI:

```

    assumes sigma-finite-measure S sigma-finite-measure T
    assumes X[measurable]: X ∈ measurable M S and Y[measurable]: Y ∈ measurable M T
    assumes [measurable]: f ∈ borel-measurable (S ⊗M T) and f: AE x in S ⊗M T. 0 ≤ f x
    assumes eq: ⋀A B. A ∈ sets S ⇒ B ∈ sets T ⇒
      emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B} = (∫+x. (∫+y. f (x, y) *
indicator B y ∂T) * indicator A x ∂S)
    shows distributed M (S ⊗M T) (λx. (X x, Y x)) f
    unfolding distributed-def
proof safe
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..

```

from ST.sigma-finite-up-in-pair-measure-generator

```

obtain F :: nat ⇒ ('b × 'c) set
  where F: range F ⊆ {A × B | A B. A ∈ sets S ∧ B ∈ sets T} ∧ incseq F ∧
    ⋃ (range F) = space S × space T ∧ (∀ i. emeasure (S ⊗M T) (F i) ≠ ∞) ..
let ?E = {a × b | a b. a ∈ sets S ∧ b ∈ sets T}
let ?P = S ⊗M T
show distr M ?P (λx. (X x, Y x)) = density ?P f (is ?L = ?R)
proof (rule measure-eqI-generator-eq[OF Int-stable-pair-measure-generator[of S
T]])
  show ?E ⊆ Pow (space ?P)
  using sets.space-closed[of S] sets.space-closed[of T] by (auto simp: space-pair-measure)
  show sets ?L = sigma-sets (space ?P) ?E
    by (simp add: sets-pair-measure space-pair-measure)
  then show sets ?R = sigma-sets (space ?P) ?E
    by simp
next
interpret L: prob-space ?L
  by (rule prob-space-distr) (auto intro!: measurable-Pair)
show range F ⊆ ?E (⋃ i. F i) = space ?P ∧ i. emeasure ?L (F i) ≠ ∞
  using F by (auto simp: space-pair-measure)
next
fix E assume E ∈ ?E
then obtain A B where E[simp]: E = A × B
  and A[measurable]: A ∈ sets S and B[measurable]: B ∈ sets T by auto
have emeasure ?L E = emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B}
  by (auto intro!: arg-cong[where f=emeasure M] simp add: emeasure-distr

```

measurable-Pair)

also have ... = $(\int^+ x. (\int^+ y. (f(x, y) * \text{indicator } B y) * \text{indicator } A x) \partial T)$
 $\partial S)$

using *f* **by** (*auto simp add: eq nn-integral-multc intro!: nn-integral-cong*)

also have ... = *emeasure ?R E*

by (*auto simp add: emeasure-density T.nn-integral-fst[symmetric]*
intro!: nn-integral-cong split: split-indicator)

finally show *emeasure ?L E = emeasure ?R E* .

qed

qed (*auto simp: f*)

lemma (*in prob-space*) *distributed-swap*:

assumes *sigma-finite-measure S sigma-finite-measure T*

assumes *Pxy: distributed M (S \otimes_M T) ($\lambda x. (X x, Y x)$) Pxy*

shows *distributed M (T \otimes_M S) ($\lambda x. (Y x, X x)$) ($\lambda(x, y). Pxy(y, x)$)*

proof –

interpret *S: sigma-finite-measure S* **by** *fact*

interpret *T: sigma-finite-measure T* **by** *fact*

interpret *ST: pair-sigma-finite S T* ..

interpret *TS: pair-sigma-finite T S* ..

note *Pxy[measurable]*

show *?thesis*

apply (*subst TS.distr-pair-swap*)

unfolding *distributed-def*

proof *safe*

let *?D = distr (S \otimes_M T) (T \otimes_M S) ($\lambda(x, y). (y, x)$)*

show *1: ($\lambda(x, y). Pxy(y, x)) \in \text{borel-measurable } ?D$*

by *auto*

show *2: random-variable (distr (S \otimes_M T) (T \otimes_M S) ($\lambda(x, y). (y, x))) (\lambda x. (Y x, X x))$*

using *Pxy* **by** *auto*

{ **fix** *A* **assume** *A: A \in sets (T \otimes_M S)*

let *?B = ($\lambda(x, y). (y, x) - ' A \cap \text{space } (S \otimes_M T)$)*

from *sets.sets-into-space[OF A]*

have *emeasure M (($\lambda x. (Y x, X x)$) - ' A \cap \text{space } M) =*

emeasure M (($\lambda x. (X x, Y x)$) - ' ?B \cap \text{space } M)

by (*auto intro!: arg-cong2[where f=emeasure] simp: space-pair-measure*)

also have ... = $(\int^+ x. Pxy x * \text{indicator } ?B x \partial (S \otimes_M T))$

using *Pxy A* **by** (*intro distributed-emeasure*) *auto*

finally have *emeasure M (($\lambda x. (Y x, X x)$) - ' A \cap \text{space } M) =*

$(\int^+ x. Pxy x * \text{indicator } A (\text{snd } x, \text{fst } x) \partial (S \otimes_M T))$

by (*auto intro!: nn-integral-cong split: split-indicator*) }

note ** = this*

show *distr M ?D ($\lambda x. (Y x, X x)$) = density ?D ($\lambda(x, y). Pxy(y, x)$)*

apply (*intro measure-eqI*)

apply (*simp-all add: emeasure-distr[OF 2] emeasure-density[OF 1]*)

apply (*subst nn-integral-distr*)

apply (*auto intro!: * simp: comp-def split-beta*)

done
qed
qed

lemma (in prob-space) distr-marginal1:

assumes sigma-finite-measure S sigma-finite-measure T
assumes Pxy: distributed M (S \otimes_M T) ($\lambda x. (X x, Y x)$) Pxy
defines Px $\equiv \lambda x. (\int^+ z. Pxy (x, z) \partial T)$
shows distributed M S X Px
unfolding distributed-def

proof safe

interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..

note Pxy[measurable]
show X: X \in measurable M S by simp

show borel: Px \in borel-measurable S
by (auto intro!: T.nn-integral-fst simp: Px-def)

interpret Pxy: prob-space distr M (S \otimes_M T) ($\lambda x. (X x, Y x)$)
by (intro prob-space-distr) simp

show distr M S X = density S Px

proof (rule measure-eqI)

fix A assume A: A \in sets (distr M S X)

with X measurable-space[of Y M T]

have emeasure (distr M S X) A = emeasure (distr M (S \otimes_M T) ($\lambda x. (X x, Y x)$)) (A \times space T)

by (auto simp add: emeasure-distr intro!: arg-cong[where f=emeasure M])

also have ... = emeasure (density (S \otimes_M T) Pxy) (A \times space T)

using Pxy by (simp add: distributed-def)

also have ... = $\int^+ x. \int^+ y. Pxy (x, y) * \text{indicator } (A \times \text{space } T) (x, y) \partial T$
 ∂S

using A borel Pxy

by (simp add: emeasure-density T.nn-integral-fst[symmetric])

also have ... = $\int^+ x. Px x * \text{indicator } A x \partial S$

proof (rule nn-integral-cong)

fix x assume x \in space S

moreover have eq: $\bigwedge y. y \in \text{space } T \implies \text{indicator } (A \times \text{space } T) (x, y) = \text{indicator } A x$

by (auto simp: indicator-def)

ultimately have $(\int^+ y. Pxy (x, y) * \text{indicator } (A \times \text{space } T) (x, y) \partial T) = (\int^+ y. Pxy (x, y) \partial T) * \text{indicator } A x$

by (simp add: eq nn-integral-multc cong: nn-integral-cong)

also have $(\int^+ y. Pxy (x, y) \partial T) = Px x$

by (simp add: Px-def)

finally show $(\int^+ y. Pxy (x, y) * \text{indicator } (A \times \text{space } T) (x, y) \partial T) = Px$

$x * \text{indicator } A \ x$.

qed

finally show $\text{emeasure } (\text{distr } M \ S \ X) \ A = \text{emeasure } (\text{density } S \ Px) \ A$

using $A \ \text{borel } Pxy$ **by** $(\text{simp add: emeasure-density})$

qed simp

qed

lemma $(\text{in prob-space}) \ \text{distr-marginal2}$:

assumes S : $\text{sigma-finite-measure } S$ **and** T : $\text{sigma-finite-measure } T$

assumes Pxy : $\text{distributed } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \ Pxy$

shows $\text{distributed } M \ T \ Y \ (\lambda y. (\int^+ x. Pxy \ (x, y) \ \partial S))$

using $\text{distr-marginal1}[OF \ T \ S \ \text{distributed-swap}[OF \ S \ T]] \ Pxy$ **by** simp

lemma $(\text{in prob-space}) \ \text{distributed-marginal-eq-joint1}$:

assumes T : $\text{sigma-finite-measure } T$

assumes S : $\text{sigma-finite-measure } S$

assumes Px : $\text{distributed } M \ S \ X \ Px$

assumes Pxy : $\text{distributed } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \ Pxy$

shows $AE \ x \ \text{in } S. \ Px \ x = (\int^+ y. Pxy \ (x, y) \ \partial T)$

using $Px \ \text{distr-marginal1}[OF \ S \ T \ Pxy]$ **by** $(\text{rule distributed-unique})$

lemma $(\text{in prob-space}) \ \text{distributed-marginal-eq-joint2}$:

assumes T : $\text{sigma-finite-measure } T$

assumes S : $\text{sigma-finite-measure } S$

assumes Py : $\text{distributed } M \ T \ Y \ Py$

assumes Pxy : $\text{distributed } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \ Pxy$

shows $AE \ y \ \text{in } T. \ Py \ y = (\int^+ x. Pxy \ (x, y) \ \partial S)$

using $Py \ \text{distr-marginal2}[OF \ S \ T \ Pxy]$ **by** $(\text{rule distributed-unique})$

lemma $(\text{in prob-space}) \ \text{distributed-joint-indep}'$:

assumes S : $\text{sigma-finite-measure } S$ **and** T : $\text{sigma-finite-measure } T$

assumes $X[\text{measurable}]$: $\text{distributed } M \ S \ X \ Px$ **and** $Y[\text{measurable}]$: $\text{distributed } M \ T \ Y \ Py$

assumes indep : $\text{distr } M \ S \ X \ \otimes_M \ \text{distr } M \ T \ Y = \text{distr } M \ (S \ \otimes_M \ T) \ (\lambda x. (X \ x, Y \ x))$

shows $\text{distributed } M \ (S \ \otimes_M \ T) \ (\lambda x. (X \ x, Y \ x)) \ (\lambda(x, y). Px \ x * Py \ y)$

unfolding distributed-def

proof safe

interpret S : $\text{sigma-finite-measure } S$ **by** fact

interpret T : $\text{sigma-finite-measure } T$ **by** fact

interpret ST : $\text{pair-sigma-finite } S \ T \ ..$

interpret X : $\text{prob-space density } S \ Px$

unfolding $\text{distributed-distr-eq-density}[OF \ X, \ \text{symmetric}]$

by $(\text{rule prob-space-distr}) \ \text{simp}$

have sf-X : $\text{sigma-finite-measure } (\text{density } S \ Px) \ ..$

interpret Y : $\text{prob-space density } T \ Py$

unfolding $\text{distributed-distr-eq-density}[OF \ Y, \ \text{symmetric}]$

by (rule prob-space-distr) simp
have sf-Y: sigma-finite-measure (density T Py) ..

show distr M (S \otimes_M T) ($\lambda x. (X x, Y x)$) = density (S \otimes_M T) ($\lambda(x, y). Px$
 $x * Py y$)
unfolding indep[symmetric] distributed-distr-eq-density[OF X] distributed-distr-eq-density[OF
Y]
using distributed-borel-measurable[OF X]
using distributed-borel-measurable[OF Y]
by (rule pair-measure-density[OF - - T sf-Y])

show random-variable (S \otimes_M T) ($\lambda x. (X x, Y x)$) **by** auto

show Pxy: ($\lambda(x, y). Px x * Py y$) \in borel-measurable (S \otimes_M T) **by** auto
qed

lemma distributed-integrable:
distributed M N X f $\implies g \in$ borel-measurable N $\implies (\bigwedge x. x \in$ space N $\implies 0 \leq$
f x) \implies
integrable N ($\lambda x. f x * g x$) \longleftrightarrow integrable M ($\lambda x. g (X x)$)
supply distributed-real-measurable[measurable]
by (auto simp: distributed-distr-eq-density[symmetric] integrable-real-density[symmetric]
integrable-distr-eq)

lemma distributed-transform-integrable:
assumes Px: distributed M N X Px $\bigwedge x. x \in$ space N $\implies 0 \leq Px x$
assumes distributed M P Y Py $\bigwedge x. x \in$ space P $\implies 0 \leq Py x$
assumes Y: Y = ($\lambda x. T (X x)$) **and** T: T \in measurable N P **and** f: f \in
borel-measurable P
shows integrable P ($\lambda x. Py x * f x$) \longleftrightarrow integrable N ($\lambda x. Px x * f (T x)$)
proof –
have integrable P ($\lambda x. Py x * f x$) \longleftrightarrow integrable M ($\lambda x. f (Y x)$)
by (rule distributed-integrable) fact+
also have ... \longleftrightarrow integrable M ($\lambda x. f (T (X x))$)
using Y **by** simp
also have ... \longleftrightarrow integrable N ($\lambda x. Px x * f (T x)$)
using measurable-comp[OF T f] Px **by** (intro distributed-integrable[symmetric])
(auto simp: comp-def)
finally show ?thesis .
qed

lemma distributed-integrable-var:
fixes X :: 'a \Rightarrow real
shows distributed M lborel X ($\lambda x. ennreal (f x)$) $\implies (\bigwedge x. 0 \leq f x) \implies$
integrable lborel ($\lambda x. f x * x$) \implies integrable M X
using distributed-integrable[of M lborel X f $\lambda x. x$] **by** simp

lemma (in prob-space) distributed-variance:
fixes f::real \Rightarrow real

assumes D : distributed M lborel $X f$ **and** $[simp]$: $\bigwedge x. 0 \leq f x$
shows variance $X = (\int x. x^2 * f (x + expectation X) \partial lborel)$
proof (subst distributed-integral[OF D , symmetric])
show $(\int x. f x * (x - expectation X)^2 \partial lborel) = (\int x. x^2 * f (x + expectation X) \partial lborel)$
by (subst lborel-integral-real-affine[where $c=1$ **and** $t=expectation X$]) (auto simp: ac-simps)
qed simp-all

lemma (in prob-space) variance-affine:
fixes $f::real \Rightarrow real$
assumes $[arith]$: $b \neq 0$
assumes $D[intro]$: distributed M lborel $X f$
assumes $[simp]$: prob-space (density lborel f)
assumes $I[simp]$: integrable $M X$
assumes $I2[simp]$: integrable $M (\lambda x. (X x)^2)$
shows variance $(\lambda x. a + b * X x) = b^2 * variance X$
by (subst variance-eq)
(auto simp: power2-sum power-mult-distrib prob-space variance-eq right-diff-distrib)

definition

simple-distributed $M X f \longleftrightarrow$
 $(\forall x. 0 \leq f x) \wedge$
distributed M (count-space $(X'space M)$) $X (\lambda x. ennreal (f x)) \wedge$
finite $(X'space M)$

lemma simple-distributed-nonneg[dest]: simple-distributed $M X f \Longrightarrow 0 \leq f x$
by (auto simp: simple-distributed-def)

lemma simple-distributed:
simple-distributed $M X P x \Longrightarrow$ distributed M (count-space $(X'space M)$) $X P x$
unfolding simple-distributed-def **by** auto

lemma simple-distributed-finite[dest]: simple-distributed $M X P \Longrightarrow$ finite $(X'space M)$
by (simp add: simple-distributed-def)

lemma (in prob-space) distributed-simple-function-superset:

assumes X : simple-function $M X \bigwedge x. x \in X'space M \Longrightarrow P x = measure M (X - \{x\} \cap space M)$

assumes A : $X'space M \subseteq A$ finite A

defines $S \equiv count-space A$ **and** $P' \equiv (\lambda x. if x \in X'space M then P x else 0)$

shows distributed $M S X P'$

unfolding distributed-def

proof safe

show $(\lambda x. ennreal (P' x)) \in$ borel-measurable S **unfolding** S -def **by** simp

show distr $M S X = density S P'$

proof (rule measure-eqI-finite)

show sets (distr $M S X) = Pow A$ sets (density $S P') = Pow A$

using A **unfolding** S -def **by** *auto*
show *finite* A **by** *fact*
fix a **assume** $a: a \in A$
then **have** $a \notin X \text{'space } M \implies X - \{a\} \cap \text{space } M = \{\}$ **by** *auto*
with A a X **have** *emeasure* (*distr* M S X) $\{a\} = P' a$
by (*subst emeasure-distr*)
(auto simp add: S-def P'-def simple-functionD emeasure-eq-measure measurable-count-space-eq2
intro!: arg-cong[where f=prob])
also **have** $\dots = (\int^+ x. \text{ennreal } (P' a) * \text{indicator } \{a\} x \partial S)$
using A X a
by (*subst nn-integral-cmult-indicator*)
(auto simp: S-def P'-def simple-distributed-def simple-functionD measurable-nonneg)
also **have** $\dots = (\int^+ x. \text{ennreal } (P' x) * \text{indicator } \{a\} x \partial S)$
by (*auto simp: indicator-def intro!: nn-integral-cong*)
also **have** $\dots = \text{emeasure } (\text{density } S P') \{a\}$
using a A **by** (*intro emeasure-density[symmetric]*) (*auto simp: S-def*)
finally **show** *emeasure* (*distr* M S X) $\{a\} = \text{emeasure } (\text{density } S P') \{a\}$.
qed
show *random-variable* S X
using $X(1)$ A **by** (*auto simp: measurable-def simple-functionD S-def*)
qed

lemma (*in prob-space*) *simple-distributedI*:

assumes X : *simple-function* M X
 $\bigwedge x. 0 \leq P x$
 $\bigwedge x. x \in X \text{'space } M \implies P x = \text{measure } M (X - \{x\} \cap \text{space } M)$
shows *simple-distributed* M X P
unfolding *simple-distributed-def*
proof (*safe intro!*: X)
have *distributed* M (*count-space* ($X \text{'space } M$)) X ($\lambda x. \text{ennreal } (\text{if } x \in X \text{'space } M \text{ then } P x \text{ else } 0)$)
(is ?A)
using *simple-functionD[OF X(1)]* **by** (*intro distributed-simple-function-superset[OF X(1,3)]*) *auto*
also **have** $?A \iff \text{distributed } M$ (*count-space* ($X \text{'space } M$)) X ($\lambda x. \text{ennreal } (P x)$)
by (*rule distributed-cong-density*) *auto*
finally **show** \dots .
qed (*rule simple-functionD[OF X(1)]*)

lemma *simple-distributed-joint-finite*:

assumes X : *simple-distributed* M ($\lambda x. (X x, Y x)$) Px
shows *finite* ($X \text{'space } M$) *finite* ($Y \text{'space } M$)
proof –
have *finite* ($(\lambda x. (X x, Y x)) \text{'space } M$)
using X **by** (*auto simp: simple-distributed-def simple-functionD*)
then **have** *finite* (*fst* $\text{' } (\lambda x. (X x, Y x)) \text{'space } M$) *finite* (*snd* $\text{' } (\lambda x. (X x, Y x))$)

```

‘ space M)
  by auto
  then show fin: finite (X ‘ space M) finite (Y ‘ space M)
    by (auto simp: image-image)
qed

```

```

lemma simple-distributed-joint2-finite:
  assumes X: simple-distributed M (λx. (X x, Y x, Z x)) Px
  shows finite (X ‘ space M) finite (Y ‘ space M) finite (Z ‘ space M)
proof –
  have finite ((λx. (X x, Y x, Z x)) ‘ space M)
    using X by (auto simp: simple-distributed-def simple-functionD)
  then have finite (fst ‘ (λx. (X x, Y x, Z x)) ‘ space M)
    finite ((fst ∘ snd) ‘ (λx. (X x, Y x, Z x)) ‘ space M)
    finite ((snd ∘ snd) ‘ (λx. (X x, Y x, Z x)) ‘ space M)
    by auto
  then show fin: finite (X ‘ space M) finite (Y ‘ space M) finite (Z ‘ space M)
    by (auto simp: image-image)
qed

```

```

lemma simple-distributed-simple-function:
  simple-distributed M X Px ⇒ simple-function M X
  unfolding simple-distributed-def distributed-def
  by (auto simp: simple-function-def measurable-count-space-eq2)

```

```

lemma simple-distributed-measure:
  simple-distributed M X P ⇒ a ∈ X‘space M ⇒ P a = measure M (X -‘ {a}
  ∩ space M)
  using distributed-count-space[of M X‘space M X P a, symmetric]
  by (auto simp: simple-distributed-def measure-def)

```

```

lemma (in prob-space) simple-distributed-joint:
  assumes X: simple-distributed M (λx. (X x, Y x)) Px
  defines S ≡ count-space (X‘space M) ⊗M count-space (Y‘space M)
  defines P ≡ (λx. if x ∈ (λx. (X x, Y x))‘space M then Px x else 0)
  shows distributed M S (λx. (X x, Y x)) P
proof –
  from simple-distributed-joint-finite[OF X, simp]
  have S-eq: S = count-space (X‘space M × Y‘space M)
    by (simp add: S-def pair-measure-count-space)
  show ?thesis
    unfolding S-eq P-def
  proof (rule distributed-simple-function-superset)
    show simple-function M (λx. (X x, Y x))
      using X by (rule simple-distributed-simple-function)
    fix x assume x ∈ (λx. (X x, Y x)) ‘ space M
    from simple-distributed-measure[OF X this]
    show Px x = prob ((λx. (X x, Y x)) -‘ {x} ∩ space M) .
  qed auto

```

qed

lemma (in *prob-space*) *simple-distributed-joint2*:

assumes X : *simple-distributed* M $(\lambda x. (X\ x, Y\ x, Z\ x))\ Px$

defines $S \equiv$ *count-space* $(X\text{'space}\ M) \otimes_M$ *count-space* $(Y\text{'space}\ M) \otimes_M$ *count-space* $(Z\text{'space}\ M)$

defines $P \equiv (\lambda x. \text{if } x \in (\lambda x. (X\ x, Y\ x, Z\ x))\text{'space}\ M \text{ then } Px\ x \text{ else } 0)$

shows *distributed* $M\ S$ $(\lambda x. (X\ x, Y\ x, Z\ x))\ P$

proof –

from *simple-distributed-joint2-finite*[*OF* X , *simp*]

have $S\text{-eq}$: $S =$ *count-space* $(X\text{'space}\ M \times Y\text{'space}\ M \times Z\text{'space}\ M)$

by (*simp add*: $S\text{-def pair-measure-count-space}$)

show *?thesis*

unfolding $S\text{-eq}\ P\text{-def}$

proof (*rule distributed-simple-function-superset*)

show *simple-function* M $(\lambda x. (X\ x, Y\ x, Z\ x))$

using X **by** (*rule simple-distributed-simple-function*)

fix x **assume** $x \in (\lambda x. (X\ x, Y\ x, Z\ x))\ \text{'space}\ M$

from *simple-distributed-measure*[*OF* X *this*]

show $Px\ x = \text{prob } ((\lambda x. (X\ x, Y\ x, Z\ x)) - \{x\} \cap \text{space}\ M)$.

qed *auto*

qed

lemma (in *prob-space*) *simple-distributed-sum-space*:

assumes X : *simple-distributed* $M\ X\ f$

shows $\text{sum } f\ (X\text{'space}\ M) = 1$

proof –

from X **have** $\text{sum } f\ (X\text{'space}\ M) = \text{prob } (\bigcup i \in X\text{'space}\ M. X - \{i\} \cap \text{space}\ M)$

by (*subst finite-measure-finite-Union*)

(*auto simp add*: *disjoint-family-on-def simple-distributed-measure simple-distributed-simple-function simple-functionD*)

intro!: sum.cong arg-cong [**where** $f = \text{prob}$]

also have $\dots = \text{prob } (\text{space}\ M)$

by (*auto intro!*: arg-cong [**where** $f = \text{prob}$])

finally show *?thesis*

using *emeasure-space-1* **by** (*simp add*: *emeasure-eq-measure*)

qed

lemma (in *prob-space*) *distributed-marginal-eq-joint-simple*:

assumes Px : *simple-function* $M\ X$

assumes Py : *simple-distributed* $M\ Y\ Py$

assumes Pxy : *simple-distributed* M $(\lambda x. (X\ x, Y\ x))\ Pxy$

assumes y : $y \in Y\text{'space}\ M$

shows $Pxy\ y = (\sum x \in X\text{'space}\ M. \text{if } (x, y) \in (\lambda x. (X\ x, Y\ x))\ \text{'space}\ M \text{ then } Pxy\ (x, y) \text{ else } 0)$

proof –

note $Px =$ *simple-distributedI*[*OF* Px *measure-nonneg refl*]

have $AE\ y$ *in count-space* $(Y\ \text{'space}\ M)$. $\text{ennreal } (Py\ y) =$

$\int^+ x. \text{ennreal } (\text{if } (x, y) \in (\lambda x. (X\ x, Y\ x))\ \text{'space}\ M \text{ then } Pxy\ (x, y) \text{ else } 0)$

```

0)  $\partial$ count-space (X ‘ space M)
  using sigma-finite-measure-count-space-finite sigma-finite-measure-count-space-finite
    simple-distributed[OF Py] simple-distributed-joint[OF Pxy]
  by (rule distributed-marginal-eq-joint2)
    (auto intro: Py Px simple-distributed-finite)
  then have ennreal (Py y) =
    ( $\sum x \in X$  ‘space M. ennreal (if (x, y)  $\in$  ( $\lambda x$ . (X x, Y x)) ‘ space M then Pxy (x,
y) else 0))
  using y Px [THEN simple-distributed-finite]
  by (auto simp: AE-count-space nn-integral-count-space-finite)
  also have ... = ( $\sum x \in X$  ‘space M. if (x, y)  $\in$  ( $\lambda x$ . (X x, Y x)) ‘ space M then
Pxy (x, y) else 0)
  using Pxy by (intro sum-ennreal) auto
  finally show ?thesis
  using simple-distributed-nonneg[OF Py] simple-distributed-nonneg[OF Pxy]
  by (subst (asm) ennreal-inj) (auto intro!: sum-nonneg)
qed

```

lemma distributedI-real:

```

fixes f :: 'a  $\Rightarrow$  real
assumes gen: sets M1 = sigma-sets (space M1) E and Int-stable E
  and A: range A  $\subseteq$  E ( $\bigcup i :: \text{nat}$ . A i) = space M1  $\wedge$  i. emeasure (distr M M1 X)
(A i)  $\neq$   $\infty$ 
  and X: X  $\in$  measurable M M1
  and f: f  $\in$  borel-measurable M1 AE x in M1. 0  $\leq$  f x
  and eq:  $\bigwedge A$ . A  $\in$  E  $\implies$  emeasure M (X -‘ A  $\cap$  space M) = ( $\int$   $^+$  x. f x *
indicator A x  $\partial$ M1)
shows distributed M M1 X f
unfolding distributed-def
proof (intro conjI)
  show distr M M1 X = density M1 f
  proof (rule measure-eqI-generator-eq[where A=A])
    { fix A assume A: A  $\in$  E
      then have A  $\in$  sigma-sets (space M1) E by auto
      then have A  $\in$  sets M1
        using gen by simp
      with f A eq[of A] X show emeasure (distr M M1 X) A = emeasure (density
M1 f) A
        by (auto simp add: emeasure-distr emeasure-density ennreal-indicator
intro!: nn-integral-cong split: split-indicator) }
  note eq-E = this
  show Int-stable E by fact
  { fix e assume e  $\in$  E
    then have e  $\in$  sigma-sets (space M1) E by auto
    then have e  $\in$  sets M1 unfolding gen .
    then have e  $\subseteq$  space M1 by (rule sets.sets-into-space) }
  then show E  $\subseteq$  Pow (space M1) by auto
  show sets (distr M M1 X) = sigma-sets (space M1) E
    sets (density M1 ( $\lambda x$ . ennreal (f x))) = sigma-sets (space M1) E

```

```

    unfolding gen[symmetric] by auto
  qed fact+
qed (insert X f, auto)

lemma distributedI-borel-atMost:
  fixes f :: real  $\Rightarrow$  real
  assumes [measurable]: X  $\in$  borel-measurable M
  and [measurable]: f  $\in$  borel-measurable borel and f[simp]:  $\forall x$  in lborel.  $0 \leq f x$ 
  and g-eq:  $\bigwedge a. (\int^+ x. f x * \text{indicator } \{..a\} x \partial \text{lborel}) = \text{ennreal } (g a)$ 
  and M-eq:  $\bigwedge a. \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal } (g a)$ 
  shows distributed M lborel X f
proof (rule distributedI-real)
  show sets (lborel::real measure) = sigma-sets (space lborel) (range atMost)
  by (simp add: borel-eq-atMost)
  show Int-stable (range atMost :: real set set)
  by (auto simp: Int-stable-def)
  have vimage-eq:  $\bigwedge a. (X -' \{..a\} \cap \text{space } M) = \{x \in \text{space } M. X x \leq a\}$  by auto
  define A where A i =  $\{.. \text{real } i\}$  for i :: nat
  then show range A  $\subseteq$  range atMost  $(\bigcup i. A i) = \text{space } \text{lborel}$ 
   $\bigwedge i. \text{emeasure } (\text{distr } M \text{ lborel } X) (A i) \neq \infty$ 
  by (auto simp: real-arch-simple emeasure-distr vimage-eq M-eq)

  fix A :: real set assume A  $\in$  range atMost
  then obtain a where A: A =  $\{..a\}$  by auto
  show emeasure M (X -' A  $\cap$  space M) =  $(\int^+ x. f x * \text{indicator } A x \partial \text{lborel})$ 
  unfolding vimage-eq A M-eq g-eq ..
qed auto

lemma (in prob-space) uniform-distributed-params:
  assumes X: distributed M MX X ( $\lambda x. \text{indicator } A x / \text{measure } MX A$ )
  shows A  $\in$  sets MX measure MX A  $\neq 0$ 
proof -
  interpret X: prob-space distr M MX X
  using distributed-measurable[OF X] by (rule prob-space-distr)

  show measure MX A  $\neq 0$ 
proof
  assume measure MX A = 0
  with X.emeasure-space-1 X.prob-space distributed-distr-eq-density[OF X]
  show False
  by (simp add: emeasure-density zero-ennreal-def[symmetric])
qed
with measure-notin-sets[of A MX] show A  $\in$  sets MX
  by blast
qed

```

lemma prob-space-uniform-measure:

assumes A: emeasure M A $\neq 0$ emeasure M A $\neq \infty$

shows *prob-space* (*uniform-measure* M A)
proof
show *emeasure* (*uniform-measure* M A) (*space* (*uniform-measure* M A)) = 1
using *emeasure-uniform-measure*[*OF* *emeasure-neq-0-sets*[*OF* $A(1)$], *of space* M]
using *sets.sets-into-space*[*OF* *emeasure-neq-0-sets*[*OF* $A(1)$]] A
by (*simp add: Int-absorb2 less-top*)
qed

lemma *prob-space-uniform-count-measure*: *finite* $A \implies A \neq \{\}$ \implies *prob-space* (*uniform-count-measure* A)
by *standard* (*auto simp: emeasure-uniform-count-measure space-uniform-count-measure one-enreal-def*)

lemma (**in** *prob-space*) *measure-uniform-measure-eq-cond-prob*:
assumes [*measurable*]: *Measurable.pred* M P *Measurable.pred* M Q
shows $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q\ x\}. P\ x) = \mathcal{P}(x \text{ in } M. P\ x \mid Q\ x)$
proof *cases*
assume Q : *measure* $M \{x \in \text{space } M. Q\ x\} = 0$
then have $*$: *AE* $x \text{ in } M. \neg Q\ x$
by (*simp add: prob-eq-0*)
then have *density* $M (\lambda x. \text{indicator } \{x \in \text{space } M. Q\ x\} x / \text{emeasure } M \{x \in \text{space } M. Q\ x\}) = \text{density } M (\lambda x. 0)$
by (*intro density-cong*) *auto*
with $*$ **show** ?*thesis*
unfolding *uniform-measure-def*
by (*simp add: emeasure-density measure-def cond-prob-def emeasure-eq-0-AE*)
next
assume Q : *measure* $M \{x \in \text{space } M. Q\ x\} \neq 0$
then show $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q\ x\}. P\ x) = \text{cond-prob } M\ P\ Q$
by (*subst measure-uniform-measure*)
(auto simp: emeasure-eq-measure cond-prob-def measure-nonneg intro!: arg-cong[where f=prob])
qed

lemma *prob-space-point-measure*:
finite $S \implies (\bigwedge s \in S \implies 0 \leq p\ s) \implies (\sum s \in S. p\ s) = 1 \implies$ *prob-space* (*point-measure* S p)
by (*rule prob-spaceI*) (*simp add: space-point-measure emeasure-point-measure-finite*)

lemma (**in** *prob-space*) *distr-pair-fst*: *distr* ($N \otimes_M M$) N *fst* = N
proof (*intro measure-eqI*)
fix A **assume** A : $A \in \text{sets } (\text{distr } (N \otimes_M M) N\ \text{fst})$
from A **have** *emeasure* (*distr* ($N \otimes_M M$) N *fst*) A = *emeasure* ($N \otimes_M M$) ($A \times \text{space } M$)
by (*auto simp add: emeasure-distr space-pair-measure dest: sets.sets-into-space intro!: arg-cong2[where f=emeasure]*)

with A **show** $\text{emeasure } (\text{distr } (N \otimes_M M) N \text{ fst}) A = \text{emeasure } N A$
by ($\text{simp add: emeasure-pair-measure-Times emeasure-space-1}$)
qed simp

lemma (**in** $\text{product-prob-space}$) distr-reorder :

assumes $\text{inj-on } t J t \in J \rightarrow K \text{ finite } K$
shows $\text{distr } (PiM K M) (PiM J (\lambda x. M (t x))) (\lambda \omega. \lambda n \in J. \omega (t n)) = PiM J$
 $(\lambda x. M (t x))$
proof ($\text{rule product-sigma-finite.PiM-eqI}$)
show $\text{product-sigma-finite } (\lambda x. M (t x)) ..$
have $t'J \subseteq K$ **using** assms by auto
then show [simp]: $\text{finite } J$
by ($\text{rule finite-imageD[OF finite-subset]}$) fact+
fix A **assume** $A: \bigwedge i. i \in J \implies A i \in \text{sets } (M (t i))$
moreover have $((\lambda \omega. \lambda n \in J. \omega (t n)) - ' PiE J A \cap \text{space } (PiM K M)) =$
 $(\Pi_E i \in K. \text{if } i \in t'J \text{ then } A (\text{the-inv-into } J t i) \text{ else } \text{space } (M i))$
using $A A[\text{THEN sets.sets-into-space}] \langle t \in J \rightarrow K \rangle \langle \text{inj-on } t J \rangle$
by ($\text{subst prod-emb-Pi[symmetric]}$) ($\text{auto simp: space-PiM PiE-iff the-inv-into-f-f}$
 prod-emb-def)
ultimately show $\text{distr } (PiM K M) (PiM J (\lambda x. M (t x))) (\lambda \omega. \lambda n \in J. \omega (t n))$
 $(PiE J A) = (\prod i \in J. M (t i) (A i))$
using assms
apply ($\text{subst emeasure-distr}$)
apply ($\text{auto intro!: sets-PiM-I-finite simp: Pi-iff}$)
apply ($\text{subst emeasure-PiM}$)
apply ($\text{auto simp: the-inv-into-f-f } \langle \text{inj-on } t J \rangle \text{ prod.reindex[OF } \langle \text{inj-on } t J \rangle$
 $\text{if-distrib[where } f = \text{emeasure } (M -) \text{] prod.If-cases emeasure-space-1 Int-absorb1}$
 $\langle t'J \subseteq K \rangle$)
done
qed simp

lemma (**in** $\text{product-prob-space}$) distr-restrict :

$J \subseteq K \implies \text{finite } K \implies (\Pi_M i \in J. M i) = \text{distr } (\Pi_M i \in K. M i) (\Pi_M i \in J. M$
 $i) (\lambda f. \text{restrict } f J)$
using $\text{distr-reorder[of } \lambda x. x J K]$ **by** ($\text{simp add: Pi-iff subset-eq}$)

lemma (**in** $\text{product-prob-space}$) $\text{emeasure-prod-emb[simp]}$:

assumes $L: J \subseteq L \text{ finite } L$ **and** $X: X \in \text{sets } (PiM J M)$
shows $\text{emeasure } (PiM L M) (\text{prod-emb } L M J X) = \text{emeasure } (PiM J M) X$
by ($\text{subst distr-restrict[OF L]}$)
 $(\text{simp add: prod-emb-def space-PiM emeasure-distr measurable-restrict-subset } L$
 $X)$

lemma $\text{emeasure-distr-restrict}$:

assumes $I \subseteq K$ **and** $Q[\text{measurable-cong}]$: $\text{sets } Q = \text{sets } (PiM K M)$ **and**
 $A[\text{measurable}]$: $A \in \text{sets } (PiM I M)$
shows $\text{emeasure } (\text{distr } Q (PiM I M) (\lambda \omega. \text{restrict } \omega I)) A = \text{emeasure } Q$
 $(\text{prod-emb } K M I A)$
using $\langle I \subseteq K \rangle \text{sets-eq-imp-space-eq[OF } Q]$

by (*subst emeasure-distr*)
 (*auto simp: measurable-cong-sets[OF Q] prod-emb-def space-PiM[symmetric]*)
intro!: measurable-restrict)

lemma (*in prob-space*) *prob-space-completion: prob-space (completion M)*
by (*rule prob-spaceI*) (*simp add: emeasure-space-1*)

lemma *distr-PiM-finite-prob-space:*

assumes *fin: finite I*
assumes *product-prob-space M*
assumes *product-prob-space M'*
assumes [*measurable*]: $\bigwedge i. i \in I \implies f \in \text{measurable } (M \ i) \ (M' \ i)$
shows $\text{distr } (PiM \ I \ M) \ (PiM \ I \ M') \ (\text{compose } I \ f) = PiM \ I \ (\lambda i. \text{distr } (M \ i) \ (M' \ i) \ f)$

proof –

interpret *M: product-prob-space M* **by fact**

interpret *M': product-prob-space M'* **by fact**

define *N* **where** $N = (\lambda i. \text{if } i \in I \text{ then } \text{distr } (M \ i) \ (M' \ i) \ f \text{ else } M' \ i)$

have [*intro*]: *prob-space (N i)* **for** *i*

by (*auto simp: N-def intro!: M.M.prob-space-distr M'.prob-space*)

interpret *N: product-prob-space N*

by (*intro product-prob-spaceI*) (*auto simp: N-def M'.prob-space intro: M.M.prob-space-distr*)

have $\text{distr } (PiM \ I \ M) \ (PiM \ I \ M') \ (\text{compose } I \ f) = PiM \ I \ N$

proof (*rule N.PiM-eqI*)

have *N-events-eq: sets (Pi_M I N) = sets (Pi_M I M')*

unfolding *N-def* **by** (*intro sets-PiM-cong*) *auto*

also have $\dots = \text{sets } (\text{distr } (PiM \ I \ M) \ (PiM \ I \ M') \ (\text{compose } I \ f))$

by *simp*

finally show $\text{sets } (\text{distr } (PiM \ I \ M) \ (PiM \ I \ M') \ (\text{compose } I \ f)) = \text{sets } (PiM \ I \ N)$..

fix *A* **assume** $A: \bigwedge i. i \in I \implies A \ i \in N.M.\text{events } i$

have $\text{emeasure } (\text{distr } (PiM \ I \ M) \ (PiM \ I \ M') \ (\text{compose } I \ f)) \ (PiE \ I \ A) =$
 $\text{emeasure } (PiM \ I \ M) \ (\text{compose } I \ f \ -' \ PiE \ I \ A \cap \text{space } (PiM \ I \ M))$

proof (*intro emeasure-distr*)

show $\text{compose } I \ f \in PiM \ I \ M \rightarrow_M \ PiM \ I \ M'$

unfolding *compose-def* **by** *measurable*

show $PiE \ I \ A \in \text{sets } (PiM \ I \ M')$

unfolding *N-events-eq [symmetric]* **by** (*intro sets-PiM-I-finite fin A*)

qed

also have $\text{compose } I \ f \ -' \ PiE \ I \ A \cap \text{space } (PiM \ I \ M) = PiE \ I \ (\lambda i. f \ -' \ A \ i \cap \text{space } (M \ i))$

using *A* **by** (*auto simp: space-PiM PiE-def Pi-def extensional-def N-def compose-def*)

also have $\text{emeasure } (PiM \ I \ M) \ (PiE \ I \ (\lambda i. f \ -' \ A \ i \cap \text{space } (M \ i))) =$
 $(\prod_{i \in I}. \text{emeasure } (M \ i) \ (f \ -' \ A \ i \cap \text{space } (M \ i)))$

using *A* **by** (*intro M.emeasure-PiM fin*) (*auto simp: N-def*)

```

also have ... = ( $\prod_{i \in I}. \text{emeasure } (\text{distr } (M \ i) \ (M' \ i) \ f) \ (A \ i))$ 
  using A by (intro prod.cong emeasure-distr [symmetric]) (auto simp: N-def)
also have ... = ( $\prod_{i \in I}. \text{emeasure } (N \ i) \ (A \ i))$ 
  unfolding N-def by (intro prod.cong) (auto simp: N-def)
finally show  $\text{emeasure } (\text{distr } (Pi_M \ I \ M) \ (Pi_M \ I \ M') \ (\text{compose } I \ f)) \ (Pi_E \ I$ 
A) = ... .
qed fact+
also have  $Pi_M \ I \ N = Pi_M \ I \ (\lambda i. \text{distr } (M \ i) \ (M' \ i) \ f)$ 
  by (intro PiM-cong) (auto simp: N-def)
finally show ?thesis .
qed

end

```

2 Distribution Functions

Shows that the cumulative distribution function (cdf) of a distribution (a measure on the reals) is nondecreasing and right continuous, which tends to 0 and 1 in either direction.

Conversely, every such function is the cdf of a unique distribution. This direction defines the measure in the obvious way on half-open intervals, and then applies the Caratheodory extension theorem.

```

theory Distribution-Functions
  imports Probability-Measure
begin

```

```

lemma UN-Ioc-eq-UNIV: ( $\bigcup n. \{ -real \ n <.. \ real \ n \}$ ) = UNIV
  by auto
  (metis le-less-trans minus-minus neg-less-iff-less not-le real-arch-simple
of-nat-0-le-iff reals-Archimedean2)

```

2.1 Properties of cdf's

```

definition
  cdf :: real measure  $\Rightarrow$  real  $\Rightarrow$  real
where
  cdf M  $\equiv$   $\lambda x. \text{measure } M \ \{..x\}$ 

```

```

lemma cdf-def2: cdf M x = measure M  $\{..x\}$ 
  by (simp add: cdf-def)

```

```

locale finite-borel-measure = finite-measure M for M :: real measure +
  assumes M-is-borel: sets M = sets borel
begin

```

```

lemma sets-M[intro]:  $a \in \text{sets borel} \implies a \in \text{sets } M$ 
  using M-is-borel by auto

```

lemma *cdf-diff-eq*:

assumes $x < y$

shows $\text{cdf } M \ y - \text{cdf } M \ x = \text{measure } M \ \{x < ..y\}$

proof –

from *assms* **have** $*$: $\{..x\} \cup \{x < ..y\} = \{..y\}$ **by** *auto*

have $\text{measure } M \ \{..y\} = \text{measure } M \ \{..x\} + \text{measure } M \ \{x < ..y\}$

by (*subst finite-measure-Union [symmetric], auto simp add: **)

thus *?thesis*

unfolding *cdf-def* **by** *auto*

qed

lemma *cdf-nondecreasing*: $x \leq y \implies \text{cdf } M \ x \leq \text{cdf } M \ y$

unfolding *cdf-def* **by** (*auto intro!: finite-measure-mono*)

lemma *borel-UNIV*: $\text{space } M = \text{UNIV}$

by (*metis in-mono sets.sets-into-space space-in-borel top-le M-is-borel*)

lemma *cdf-nonneg*: $\text{cdf } M \ x \geq 0$

unfolding *cdf-def* **by** (*rule measure-nonneg*)

lemma *cdf-bounded*: $\text{cdf } M \ x \leq \text{measure } M \ (\text{space } M)$

unfolding *cdf-def* **by** (*intro bounded-measure*)

lemma *cdf-lim-inf*:

$((\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow \text{measure } M \ (\text{space } M))$

proof –

have $(\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow \text{measure } M \ (\bigcup i::\text{nat}. \{.. \text{real } i\})$

unfolding *cdf-def* **by** (*rule finite-Lim-measure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i::\text{nat}. \{.. \text{real } i\}) = \text{space } M$

by (*auto simp: borel-UNIV intro: real-arch-simple*)

finally show *?thesis* .

qed

lemma *cdf-lim-at-top*: $(\text{cdf } M \longrightarrow \text{measure } M \ (\text{space } M)) \text{ at-top}$

by (*rule tendsto-at-topI-sequentially-real*)

(*simp-all add: mono-def cdf-nondecreasing cdf-lim-inf*)

lemma *cdf-lim-neg-inf*: $((\lambda i. \text{cdf } M \ (- \text{real } i)) \longrightarrow 0)$

proof –

have $(\lambda i. \text{cdf } M \ (- \text{real } i)) \longrightarrow \text{measure } M \ (\bigcap i::\text{nat}. \{.. - \text{real } i\})$

unfolding *cdf-def* **by** (*rule finite-Lim-measure-decseq*) (*auto simp: decseq-def*)

also have $(\bigcap i::\text{nat}. \{.. - \text{real } i\}) = \{\}$

by *auto* (*metis leD le-minus-iff reals-Archimedean2*)

finally show *?thesis*

by *simp*

qed

lemma *cdf-lim-at-bot*: $(\text{cdf } M \longrightarrow 0) \text{ at-bot}$

proof –

have *: $((\lambda x :: \text{real}. - \text{cdf } M (- x)) \longrightarrow 0)$ *at-top*
by (*intro tendsto-at-topI-sequentially-real monoI*)
(auto simp: cdf-nondecreasing cdf-lim-neg-infty tendsto-minus-cancel-left[symmetric])
from *filterlim-compose [OF *, OF filterlim-uminus-at-top-at-bot]*
show *?thesis*
unfolding *tendsto-minus-cancel-left[symmetric]* **by** *simp*
qed

lemma *cdf-is-right-cont: continuous (at-right a) (cdf M)*

unfolding *continuous-within*

proof (*rule tendsto-at-right-sequentially[where b=a + 1]*)

fix $f :: \text{nat} \Rightarrow \text{real}$ **and** x **assume** $f: \text{decseq } f f \longrightarrow a$

then have $(\lambda n. \text{cdf } M (f n)) \longrightarrow \text{measure } M (\bigcap i. \{.. f i\})$

using $\langle \text{decseq } f \rangle$ **unfolding** *cdf-def*

by (*intro finite-Lim-measure-decseq*) (*auto simp: decseq-def*)

also have $(\bigcap i. \{.. f i\}) = \{.. a\}$

using *decseq-ge[OF f]* **by** (*auto intro: order-trans LIMSEQ-le-const[OF f(2)]*)

finally show $(\lambda n. \text{cdf } M (f n)) \longrightarrow \text{cdf } M a$

by (*simp add: cdf-def*)

qed *simp*

lemma *cdf-at-left: (cdf M \longrightarrow measure M $\{..<a\}$) (at-left a)*

proof (*rule tendsto-at-left-sequentially[of a - 1]*)

fix $f :: \text{nat} \Rightarrow \text{real}$ **and** x **assume** $f: \text{incseq } f f \longrightarrow a \wedge x. f x < a \wedge x. a - 1 < f x$

then have $(\lambda n. \text{cdf } M (f n)) \longrightarrow \text{measure } M (\bigcup i. \{.. f i\})$

using $\langle \text{incseq } f \rangle$ **unfolding** *cdf-def*

by (*intro finite-Lim-measure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i. \{.. f i\}) = \{..<a\}$

by (*auto dest!: order-tendstoD(1)[OF f(2)] eventually-happens'[OF sequentially-bot]*)

intro: less-imp-le le-less-trans f(3))

finally show $(\lambda n. \text{cdf } M (f n)) \longrightarrow \text{measure } M \{..<a\}$

by (*simp add: cdf-def*)

qed *auto*

lemma *isCont-cdf: isCont (cdf M) x \longleftrightarrow measure M $\{x\} = 0$*

proof –

have *isCont (cdf M) x \longleftrightarrow cdf M x = measure M $\{..<x\}$*

by (*auto simp: continuous-at-split cdf-is-right-cont continuous-within[where s= $\{..<- \}$]*)

cdf-at-left tendsto-unique[OF - cdf-at-left])

also have *cdf M x = measure M $\{..<x\} \longleftrightarrow$ measure M $\{x\} = 0$*

unfolding *cdf-def ivl-disj-un(2)[symmetric]*

by (*subst finite-measure-Union*) *auto*

finally show *?thesis .*

qed

lemma *countable-atoms*: *countable* $\{x. \text{measure } M \{x\} > 0\}$
using *countable-support* **unfolding** *zero-less-measure-iff* .

end

locale *real-distribution* = *prob-space* *M* **for** *M* :: *real measure* +
assumes *events-eq-borel* [*simp*, *measurable-cong*]: *sets* *M* = *sets borel*
begin

lemma *finite-borel-measure-M*: *finite-borel-measure* *M*
by *standard auto*

sublocale *finite-borel-measure* *M*
by (*rule finite-borel-measure-M*)

lemma *space-eq-univ* [*simp*]: *space* *M* = *UNIV*
using *events-eq-borel*[*THEN sets-eq-imp-space-eq*] **by** *simp*

lemma *cdf-bounded-prob*: $\bigwedge x. \text{cdf } M \ x \leq 1$
by (*subst prob-space* [*symmetric*], *rule cdf-bounded*)

lemma *cdf-lim-inf-prob*: $(\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow 1$
by (*subst prob-space* [*symmetric*], *rule cdf-lim-inf*)

lemma *cdf-lim-at-top-prob*: $(\text{cdf } M \longrightarrow 1) \text{ at-top}$
by (*subst prob-space* [*symmetric*], *rule cdf-lim-at-top*)

lemma *measurable-finite-borel* [*simp*]:
 $f \in \text{borel-measurable borel} \implies f \in \text{borel-measurable } M$
by (*rule borel-measurable-subalgebra*[**where** $N = \text{borel}$]) *auto*

end

lemma (**in** *prob-space*) *real-distribution-distr* [*intro*, *simp*]:
random-variable borel *X* $\implies \text{real-distribution} (\text{distr } M \ \text{borel } X)$
unfolding *real-distribution-def* *real-distribution-axioms-def* **by** (*auto intro!*: *prob-space-distr*)

2.2 Uniqueness

lemma (**in** *finite-borel-measure*) *emeasure-Ioc*:
assumes $a \leq b$ **shows** $\text{emeasure } M \ \{a <.. b\} = \text{cdf } M \ b - \text{cdf } M \ a$
proof –
have $\{a <.. b\} = \{..b\} - \{..a\}$
by *auto*
moreover **have** $\{..x\} \in \text{sets } M$ **for** *x*
using *atMost-borel*[*of x*] *M-is-borel* **by** *auto*
moreover **note** $\langle a \leq b \rangle$
ultimately **show** *?thesis*
by (*simp add: emeasure-eq-measure finite-measure-Diff* *cdf-def*)

qed

lemma *cdf-unique'*:

fixes $M1\ M2$

assumes *finite-borel-measure* $M1$ **and** *finite-borel-measure* $M2$

assumes $\text{cdf } M1 = \text{cdf } M2$

shows $M1 = M2$

proof (*rule measure-eqI-generator-eq*[**where** $\Omega = UNIV$])

fix X **assume** $X \in \text{range } (\lambda(a, b). \{a <..b\}::\text{real})$

then obtain $a\ b$ **where** $X \in \{a <..b\}$ **by** *auto*

then show $\text{emeasure } M1\ X = \text{emeasure } M2\ X$

by (*cases* $a \leq b$)

(*simp-all add: assms(1,2)*[*THEN finite-borel-measure.emeasure-Ioc*] *assms(3)*)

next

show $(\bigcup i. \{- \text{real } (i::\text{nat}) <.. \text{real } i\}) = UNIV$

by (*rule UN-Ioc-eq-UNIV*)

qed (*auto simp: finite-borel-measure.emeasure-Ioc*[*OF assms(1)*])

assms(1,2)[*THEN finite-borel-measure.M-is-borel*] *borel-sigma-sets-Ioc*

Int-stable-def)

lemma *cdf-unique*:

real-distribution $M1 \implies \text{real-distribution } M2 \implies \text{cdf } M1 = \text{cdf } M2 \implies M1 = M2$

using *cdf-unique'*[*of* $M1\ M2$] **by** (*simp add: real-distribution.finite-borel-measure-M*)

lemma

fixes $F :: \text{real} \Rightarrow \text{real}$

assumes *nondecF* : $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$

and *right-cont-F* : $\bigwedge a. \text{continuous } (\text{at-right } a)\ F$

and *lim-F-at-bot* : $(F \longrightarrow 0)\ \text{at-bot}$

and *lim-F-at-top* : $(F \longrightarrow m)\ \text{at-top}$

and $m: 0 \leq m$

shows *interval-measure-UNIV*: $\text{emeasure } (\text{interval-measure } F)\ UNIV = m$

and *finite-borel-measure-interval-measure*: *finite-borel-measure* (*interval-measure* F)

proof –

let $?F = \text{interval-measure } F$

{ **have** $\text{ennreal } (m - 0) = (\text{SUP } i. \text{ennreal } (F\ (\text{real } i) - F\ (- \text{real } i)))$

by (*intro LIMSEQ-unique*[*OF - LIMSEQ-SUP*] *tendsto-ennrealI tendsto-intros*

lim-F-at-bot[*THEN filterlim-compose*] *lim-F-at-top*[*THEN filter-*

lim-compose]

lim-F-at-bot[*THEN filterlim-compose*] *filterlim-real-sequentially*

filterlim-uminus-at-top[*THEN iffD1*])

(*auto simp: incseq-def nondecF intro!: diff-mono*)

also have $\dots = (\text{SUP } i. \text{emeasure } ?F\ \{- \text{real } i <.. \text{real } i\})$

by (*subst emeasure-interval-measure-Ioc*) (*simp-all add: nondecF right-cont-F*)

also have $\dots = \text{emeasure } ?F\ (\bigcup i::\text{nat}. \{- \text{real } i <.. \text{real } i\})$

by (*rule SUP-emeasure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i. \{- \text{real } (i::\text{nat}) <.. \text{real } i\}) = \text{space } ?F$


```

    by (simp add: UN-Ioc-eq-UNIV)
  finally have emeasure ?F (space ?F) = m
    by simp }
note * = this
then show emeasure (interval-measure F) UNIV = m
  by simp

```

```

interpret finite-measure ?F
proof
  show emeasure ?F (space ?F)  $\neq \infty$ 
    using * by simp
qed
show finite-borel-measure (interval-measure F)
  proof qed simp-all
qed

```

```

lemma real-distribution-interval-measure:
  fixes F :: real  $\Rightarrow$  real
  assumes nondecF :  $\bigwedge x y. x \leq y \implies F x \leq F y$  and
    right-cont-F :  $\bigwedge a. \text{continuous (at-right a) } F$  and
    lim-F-at-bot :  $(F \longrightarrow 0)$  at-bot and
    lim-F-at-top :  $(F \longrightarrow 1)$  at-top
  shows real-distribution (interval-measure F)
proof -
  let ?F = interval-measure F
  interpret prob-space ?F
  proof qed (use interval-measure-UNIV[OF assms] in simp)
  show ?thesis
  proof qed simp-all
qed

```

```

lemma
  fixes F :: real  $\Rightarrow$  real
  assumes nondecF :  $\bigwedge x y. x \leq y \implies F x \leq F y$  and
    right-cont-F :  $\bigwedge a. \text{continuous (at-right a) } F$  and
    lim-F-at-bot :  $(F \longrightarrow 0)$  at-bot
  shows emeasure-interval-measure-Iic: emeasure (interval-measure F)  $\{.. x\} = F x$ 
  and measure-interval-measure-Iic: measure (interval-measure F)  $\{.. x\} = F x$ 
  unfolding cdf-def
proof -
  have F-nonneg[simp]:  $0 \leq F y$  for y
  using lim-F-at-bot by (rule tendsto-upperbound) (auto simp: eventually-at-bot-linorder
  nondecF intro!: exI[of - y])

  have emeasure (interval-measure F)  $(\bigcup i::nat. \{-real\ i <.. x\}) = F x - ennreal 0$ 
proof (intro LIMSEQ-unique[OF Lim-emeasure-incseq])
  have  $(\lambda i. F x - F (-real\ i)) \longrightarrow F x - 0$ 

```

```

  by (intro tendsto-intros lim-F-at-bot[THEN filterlim-compose] filterlim-real-sequentially
      filterlim-uminus-at-top[THEN iffD1])
  from tendsto-ennreal[OF this]
  show (λi. emeasure (interval-measure F) {- real i <.. x}) → F x - ennreal
0
  apply (rule filterlim-cong[THEN iffD1, rotated 3])
  apply simp
  apply simp
  apply (rule eventually-sequentiallyI[where c=nat (ceiling (- x))])
  apply (simp add: emeasure-interval-measure-Ioc right-cont-F nondecF)
  done
qed (auto simp: incseq-def)
also have (⋃ i::nat. {- real i <.. x}) = {..x}
  by auto (metis minus-minus neg-less-iff-less reals-Archimedean2)
finally show emeasure (interval-measure F) {..x} = F x
  by simp
then show measure (interval-measure F) {..x} = F x
  by (simp add: measure-def)
qed

```

lemma *cdf-interval-measure*:

```

(⋂ x y. x ≤ y ⇒ F x ≤ F y) ⇒ (⋂ a. continuous (at-right a) F) ⇒ (F →
0) at-bot ⇒ cdf (interval-measure F) = F
  by (simp add: cdf-def fun-eq-iff measure-interval-measure-Iic)

```

end

3 Weak Convergence of Functions and Distributions

Properties of weak convergence of functions and measures, including the portmanteau theorem.

```

theory Weak-Convergence
  imports Distribution-Functions
begin

```

4 Weak Convergence of Functions

definition

```

weak-conv :: (nat ⇒ (real ⇒ real)) ⇒ (real ⇒ real) ⇒ bool
where
  weak-conv F-seq F ≡ ∀ x. isCont F x → (λn. F-seq n x) → F x

```

5 Weak Convergence of Distributions

definition

weak-conv-m :: (nat \Rightarrow real measure) \Rightarrow real measure \Rightarrow bool
where
weak-conv-m *M-seq* *M* \equiv *weak-conv* ($\lambda n.$ *cdf* (*M-seq* *n*)) (*cdf* *M*)

6 Skorohod’s theorem

locale *right-continuous-mono* =
fixes *f* :: real \Rightarrow real **and** *a* *b* :: real
assumes *cont*: $\bigwedge x.$ *continuous* (*at-right* *x*) *f*
assumes *mono*: *mono* *f*
assumes *bot*: (*f* \longrightarrow *a*) *at-bot*
assumes *top*: (*f* \longrightarrow *b*) *at-top*
begin

abbreviation *I* :: real \Rightarrow real **where**
I $\omega \equiv$ *Inf* {*x*. $\omega \leq f$ *x*}

lemma *pseudoinverse*: **assumes** *a* < ω ω < *b* **shows** $\omega \leq f$ *x* \longleftrightarrow *I* $\omega \leq x$
proof

let *?F* = {*x*. $\omega \leq f$ *x*}
obtain *y* **where** *f* *y* < ω
by (*metis* *eventually-happens'* *trivial-limit-at-bot-linorder* *order-tendstoD*(2) *bot* $\langle a < \omega \rangle$)
with *mono* **have** *bdd*: *bdd-below* *?F*
by (*auto* *intro!*: *bdd-belowI*[*of* - *y*] *elim*: *mono-invE*[*OF* - *less-le-trans*])
have *ne*: *?F* \neq {}
using *order-tendstoD*(1)[*OF* *top* $\langle \omega < b \rangle$]
by (*auto* *dest!*: *eventually-happens'*[*OF* *trivial-limit-at-top-linorder*] *intro*: *less-imp-le*)
show $\omega \leq f$ *x* \implies *I* $\omega \leq x$
by (*auto* *intro!*: *cInf-lower* *bdd*)

{ **assume** *: *I* $\omega \leq x$
have $\omega \leq$ (*INF* *s* \in {*x*. $\omega \leq f$ *x*}. *f* *s*)
by (*rule* *cINF-greatest*[*OF* *ne*]) *auto*
also **have** ... = *f* (*I* ω)
using *continuous-at-Inf-mono*[*OF* *mono* *cont* *ne* *bdd*] ..
also **have** ... \leq *f* *x*
using * **by** (*rule* *monoD*[*OF* \langle *mono* *f* \rangle])
finally **show** $\omega \leq f$ *x* . }
qed

lemma *pseudoinverse'*: $\forall \omega \in \{a < .. < b\}.$ $\forall x.$ $\omega \leq f$ *x* \longleftrightarrow *I* $\omega \leq x$
by (*intro* *ballI* *allI* *impI* *pseudoinverse*) *auto*

lemma *mono-I*: *mono-on* {*a* < .. < *b*} *I*
unfolding *mono-on-def* **by** (*metis* *order.trans* *order.refl* *pseudoinverse'*)

end

locale *cdf-distribution = real-distribution*
begin

abbreviation $C \equiv \text{cdf } M$

sublocale *right-continuous-mono* C 0 1

by *standard*
(auto intro: cdf-nondecreasing cdf-is-right-cont cdf-lim-at-top-prob cdf-lim-at-bot monoI)

lemma *measurable-C[measurable]*: $C \in \text{borel-measurable borel}$
by *(intro borel-measurable-mono mono)*

lemma *measurable-CI[measurable]*: $I \in \text{borel-measurable (restrict-space borel } \{0 < .. < 1\})$
by *(intro borel-measurable-mono-on-fnc mono-I)*

lemma *emeasure-distr-I*: *emeasure (distr (restrict-space lborel } $\{0 < .. < 1::\text{real}\}$) borel I) UNIV = 1*
by *(simp add: emeasure-distr space-restrict-space emeasure-restrict-space)*

lemma *distr-I-eq-M*: *distr (restrict-space lborel } $\{0 < .. < 1::\text{real}\}$) borel I = M (is ?I = -)*

proof *(intro cdf-unique ext)*
let $?\Omega = \text{restrict-space lborel } \{0 < .. < 1\}::\text{real measure}$
interpret Ω : *prob-space ? Ω*
by *(auto simp add: emeasure-restrict-space space-restrict-space intro!: prob-spaceI)*
show *real-distribution ?I*
by *auto*

fix x

have *cdf ?I x = measure lborel } $\{\omega \in \{0 < .. < 1\}. \omega \leq C x\}$*
by *(subst cdf-def)*

(auto simp: pseudoinverse[symmetric] measure-distr space-restrict-space measure-restrict-space

intro!: arg-cong2[where f=measure])

also have $\dots = \text{measure lborel } \{0 < .. < C x\}$

using *cdf-bounded-prob[of x] AE-lborel-singleton[of C x]*

by *(auto intro!: arg-cong[where f=enn2real] emeasure-eq-AE simp: measure-def)*

also have $\dots = C x$

by *(simp add: cdf-nonneg)*

finally show *cdf (distr ? Ω borel I) x = C x .*

qed *standard*

end

context

fixes $\mu :: \text{nat} \Rightarrow \text{real measure}$

```

and  $M :: \text{real measure}$ 
assumes  $\mu: \bigwedge n. \text{real-distribution } (\mu \ n)$ 
assumes  $M: \text{real-distribution } M$ 
assumes  $\mu\text{-to-}M: \text{weak-conv-m } \mu \ M$ 
begin

```

theorem *Skorohod*:

```

 $\exists (\Omega :: \text{real measure}) (\text{Y-seq} :: \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}) (\text{Y} :: \text{real} \Rightarrow \text{real}).$ 
   $\text{prob-space } \Omega \wedge$ 
   $(\forall n. \text{Y-seq } n \in \text{measurable } \Omega \ \text{borel}) \wedge$ 
   $(\forall n. \text{distr } \Omega \ \text{borel } (\text{Y-seq } n) = \mu \ n) \wedge$ 
   $\text{Y} \in \text{measurable } \Omega \ \text{lborel} \wedge$ 
   $\text{distr } \Omega \ \text{borel } \text{Y} = M \wedge$ 
   $(\forall x \in \text{space } \Omega. (\lambda n. \text{Y-seq } n \ x) \longrightarrow \text{Y } x)$ 

```

proof –

```

interpret  $\mu: \text{cdf-distribution } \mu \ n \ \text{for } n$ 
unfolding  $\text{cdf-distribution-def}$  by (rule  $\mu$ )
interpret  $M: \text{cdf-distribution } M$ 
unfolding  $\text{cdf-distribution-def}$  by (rule  $M$ )

```

```

have  $\text{conv}: \text{measure } M \ \{x\} = 0 \implies (\lambda n. \mu.C \ n \ x) \longrightarrow M.C \ x \ \text{for } x$ 
using  $\mu\text{-to-}M \ M.\text{isCont-cdf}$  by (auto simp:  $\text{weak-conv-m-def}$   $\text{weak-conv-def}$ )

```

```

let  $?\Omega = \text{restrict-space lborel } \{0 < .. < 1\} :: \text{real measure}$ 

```

```

have  $\text{prob-space } ?\Omega$ 

```

```

by (auto simp:  $\text{space-restrict-space emeasure-restrict-space intro!}$ :  $\text{prob-spaceI}$ )

```

```

interpret  $\Omega: \text{prob-space } ?\Omega$ 

```

```

by fact

```

```

have  $\text{Y-distr}: \text{distr } ?\Omega \ \text{borel } M.I = M$ 

```

```

by (rule  $M.\text{distr-I-eq-M}$ )

```

```

have  $\text{Y-cts-cnvt}: (\lambda n. \mu.I \ n \ \omega) \longrightarrow M.I \ \omega$ 

```

```

if  $\omega: \omega \in \{0 < .. < 1\}$  isCont  $M.I \ \omega$  for  $\omega :: \text{real}$ 

```

```

proof (intro  $\text{limsup-le-liminf-real}$ )

```

```

show  $\text{liminf } (\lambda n. \mu.I \ n \ \omega) \geq M.I \ \omega$ 

```

```

unfolding  $\text{le-Liminf-iff}$ 

```

```

proof safe

```

```

fix  $B :: \text{ereal}$  assume  $B: B < M.I \ \omega$ 

```

```

then show  $\forall_F n$  in sequentially.  $B < \mu.I \ n \ \omega$ 

```

```

proof (cases  $B$ )

```

```

case (real  $r$ )

```

```

with  $B$  have  $r: r < M.I \ \omega$ 

```

```

by simp

```

```

then obtain  $x$  where  $x: r < x < x < M.I \ \omega$  measure  $M \ \{x\} = 0$ 

```

```

using  $\text{open-minus-countable[OF } M.\text{countable-support, of } \{r < .. < M.I \ \omega\}]$ 

```

```

by auto

```

```

then have  $\text{Fx-less}: M.C \ x < \omega$ 

```

```

using  $M.pseudoinverse'$   $\omega$  not-less by blast

have  $\forall_F n$  in sequentially.  $\mu.C n x < \omega$ 
  using order-tendstoD(2)[OF conv[OF x(3)] Fx-less] .
then have  $\forall_F n$  in sequentially.  $x < \mu.I n \omega$ 
  by eventually-elim (insert  $\omega$   $\mu.pseudoinverse[symmetric]$ , simp add: not-le[symmetric])
  then show ?thesis
    by eventually-elim (insert x(1), simp add: real)
qed auto
qed

have  $*$ :  $\limsup (\lambda n. \mu.I n \omega) \leq M.I \omega'$ 
if  $\omega'$ :  $0 < \omega' \omega' < 1 \omega < \omega'$  for  $\omega' :: real$ 
proof (rule dense-ge-bounded)
  fix  $B'$  assume ereal  $(M.I \omega') < B' B' < \text{ereal } (M.I \omega' + 1)$ 
then obtain  $B$  where  $M.I \omega' < B$  and [simp]:  $B' = \text{ereal } B$ 
  by (cases B') auto
then obtain  $y$  where  $y: M.I \omega' < y y < B$  measure M {y} = 0
  using open-minus-countable[OF M.countable-support, of {M.I  $\omega' < .. < B$ }]
by auto
  then have  $\omega' \leq M.C (M.I \omega')$ 
  using  $M.pseudoinverse'$   $\omega'$  by (metis greaterThanLessThan-iff order-refl)
  also have  $\dots \leq M.C y$ 
  using  $M.mono y$  unfolding mono-def by auto
finally have  $Fy\text{-gt}: \omega < M.C y$ 
  using  $\omega'(3)$  by simp

have  $\forall_F n$  in sequentially.  $\omega \leq \mu.C n y$ 
  using order-tendstoD(1)[OF conv[OF y(3)] Fy-gt] by eventually-elim (rule less-imp-le)
  then have  $2: \forall_F n$  in sequentially.  $\mu.I n \omega \leq \text{ereal } y$ 
  by simp (subst  $\mu.pseudoinverse'$ [rule-format, OF  $\omega(1)$ , symmetric])
then show  $\limsup (\lambda n. \mu.I n \omega) \leq B'$ 
  using  $\langle y < B \rangle$ 
  by (intro Limsup-bounded[rotated]) (auto intro: le-less-trans elim: eventually-mono)
qed simp

have  $**$ :  $(M.I \longrightarrow \text{ereal } (M.I \omega))$  (at-right  $\omega$ )
  using  $\omega(2)$  by (auto intro: tendsto-within-subset simp: continuous-at)
show  $\limsup (\lambda n. \mu.I n \omega) \leq M.I \omega$ 
  using  $\omega$ 
  by (intro tendsto-lowerbound[OF **])
  (auto intro!: exI[of - 1] * simp: eventually-at-right[of - 1])
qed

let  $?D = \{\omega \in \{0 < .. < 1\}. \neg \text{isCont } M.I \omega\}$ 
have  $D\text{-countable}$ : countable ?D

```

using *mono-on-ctble-discont*[*OF M.mono-I*] **by** (*simp add: at-within-open*[*of -*
 $\{0 <..< I\}$] *cong: conj-cong*)
hence *D: emeasure ? Ω ?D = 0*
using *emeasure-lborel-countable*[*OF D-countable*]
by (*subst emeasure-restrict-space*) *auto*

define *Y'* **where** $Y' \omega = (if \omega \in ?D then 0 else M.I \omega)$ **for** ω
have *Y'-AE: AE ω in ? Ω . $Y' \omega = M.I \omega$*
by (*rule AE-I* [*OF - D*]) (*auto simp: space-restrict-space sets-restrict-space-iff*
Y'-def)

define *Y-seq'* **where** $Y-seq' n \omega = (if \omega \in ?D then 0 else \mu.I n \omega)$ **for** $n \omega$
have *Y-seq'-AE: $\bigwedge n. AE \omega$ in ? Ω . $Y-seq' n \omega = \mu.I n \omega$*
by (*rule AE-I* [*OF - D*]) (*auto simp: space-restrict-space sets-restrict-space-iff*
Y-seq'-def)

have *Y'-cnv: $\forall \omega \in \{0 <..< I\}. (\lambda n. Y-seq' n \omega) \longrightarrow Y' \omega$*
by (*auto simp: Y'-def Y-seq'-def Y-cts-cnv*)

have [*simp*]: *Y-seq' n \in borel-measurable ? Ω for n*
by (*rule measurable-discrete-difference*[*of $\mu.I n - - ?D$*])
(insert μ .measurable-CI[*of n*] *D-countable, auto simp: sets-restrict-space*
Y-seq'-def)

moreover have *distr ? Ω borel (Y-seq' n) = μn for n*

using *μ .distr-I-eq-M* [*of n*] *Y-seq'-AE* [*of n*]

by (*subst distr-cong-AE*[**where** $f = Y-seq' n$ **and** $g = \mu.I n$], *auto*)

moreover have [*simp*]: *Y' \in borel-measurable ? Ω*

by (*rule measurable-discrete-difference*[*of M.I - - ?D*])

(insert M.measurable-CI D-countable, auto simp: sets-restrict-space Y'-def)

moreover have *distr ? Ω borel Y' = M*

using *M.distr-I-eq-M Y'-AE*

by (*subst distr-cong-AE*[**where** $f = Y'$ **and** $g = M.I$], *auto*)

ultimately have *prob-space ? Ω \wedge ($\forall n. Y-seq' n \in$ borel-measurable ? Ω) \wedge*

*($\forall n. distr ?\Omega$ borel (Y-seq' n) = μn) \wedge Y' \in measurable ? Ω lborel \wedge distr ? Ω
borel Y' = M \wedge*

($\forall x \in space ?\Omega. (\lambda n. Y-seq' n x) \longrightarrow Y' x$)

using *Y'-cnv* *prob-space ? Ω* **by** (*auto simp: space-restrict-space*)

thus *?thesis by metis*

qed

The Portmanteau theorem, that is, the equivalence of various definitions of weak convergence.

theorem *weak-conv-imp-bdd-ae-continuous-conv:*

fixes

f :: real \Rightarrow 'a::{banach, second-countable-topology}

assumes

*discont-null: M ($\{x. \neg isCont f x\}) = 0$ **and***

*f-bdd: $\bigwedge x. norm (f x) \leq B$ **and***

[measurable]: f \in borel-measurable borel

shows
 $(\lambda n. \text{integral}^L (\mu n) f) \longrightarrow \text{integral}^L M f$
proof –
 have $0 \leq B$
 using *norm-ge-zero f-bdd by (rule order-trans)*
 note *Skorohod*
 then obtain *Omega Y-seq Y where*
ps-Omega [simp]: prob-space Omega and
Y-seq-measurable [measurable]: $\bigwedge n. Y\text{-seq } n \in \text{borel-measurable } \text{Omega}$ and
distr-Y-seq: $\bigwedge n. \text{distr } \text{Omega borel } (Y\text{-seq } n) = \mu n$ and
Y-measurable [measurable]: $Y \in \text{borel-measurable } \text{Omega}$ and
distr-Y: $\text{distr } \text{Omega borel } Y = M$ and
YnY: $\bigwedge x :: \text{real}. x \in \text{space } \text{Omega} \implies (\lambda n. Y\text{-seq } n x) \longrightarrow Y x$ by force
 interpret *prob-space Omega by fact*
 have *: *emeasure Omega (Y - ‘ {x. $\neg \text{isCont } f x$ } $\cap \text{space } \text{Omega}$) = 0*
 by (*subst emeasure-distr [symmetric, where N=borel]*) (*auto simp: distr-Y*
discont-null)
 have *: *AE x in Omega. $(\lambda n. f (Y\text{-seq } n x)) \longrightarrow f (Y x)$*
 by (*rule AE-I [OF - *]*) (*auto intro: isCont-tendsto-compose YnY*)
 show ?thesis
 by (*auto intro!: integral-dominated-convergence[where w= $\lambda x. B$]*
*simp: f-bdd * integral-distr distr-Y-seq [symmetric] distr-Y [symmetric]*)
 qed

theorem *weak-conv-imp-integral-bdd-continuous-conv:*
 fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$
 assumes
 $\bigwedge x. \text{isCont } f x$ and
 $\bigwedge x. \text{norm } (f x) \leq B$
 shows
 $(\lambda n. \text{integral}^L (\mu n) f) \longrightarrow \text{integral}^L M f$
 using *assms*
 by (*intro weak-conv-imp-bdd-ae-continuous-conv*)
 (*auto intro!: borel-measurable-continuous-onI continuous-at-imp-continuous-on*)

theorem *weak-conv-imp-continuity-set-conv:*
 fixes $f :: \text{real} \Rightarrow \text{real}$
 assumes *[measurable]: $A \in \text{sets borel}$ and $M (\text{frontier } A) = 0$*
 shows $(\lambda n. \text{measure } (\mu n) A) \longrightarrow \text{measure } M A$
proof –
 interpret *M: real-distribution M by fact*
 interpret μ : *real-distribution μn for n by fact*

 have $(\lambda n. (\int x. \text{indicator } A x \partial \mu n) :: \text{real}) \longrightarrow (\int x. \text{indicator } A x \partial M)$
 by (*intro weak-conv-imp-bdd-ae-continuous-conv[where B=1]*)
 (*auto intro: assms simp: isCont-indicator*)
 then show ?thesis
 by *simp*
 qed

end

definition

cts-step :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow *real*

where

cts-step *a b x* \equiv if $x \leq a$ then 1 else if $x \geq b$ then 0 else $(b - x) / (b - a)$

lemma *cts-step-uniformly-continuous*:

assumes [*arith*]: $a < b$

shows *uniformly-continuous-on UNIV* (*cts-step* *a b*)

unfolding *uniformly-continuous-on-def*

proof *clarsimp*

fix *e* :: *real* **assume** [*arith*]: $0 < e$

let $?d = \min (e * (b - a)) (b - a)$

have $?d > 0$

by (*auto simp add: field-simps*)

moreover have $\text{dist } x' x < ?d \implies \text{dist } (\text{cts-step } a b x') (\text{cts-step } a b x) < e$ **for**
x x'

by (*auto simp: dist-real-def divide-simps cts-step-def*)

ultimately show $\exists d > 0. \forall x x'. \text{dist } x' x < d \implies \text{dist } (\text{cts-step } a b x') (\text{cts-step}$
a b x) $< e$

by *blast*

qed

lemma (**in** *real-distribution*) *integrable-cts-step*: $a < b \implies \text{integrable } M$ (*cts-step*
a b)

by (*rule integrable-const-bound [of - 1]*) (*auto simp: cts-step-def[abs-def]*)

lemma (**in** *real-distribution*) *cdf-cts-step*:

assumes [*arith*]: $x < y$

shows $\text{cdf } M x \leq \text{integral}^L M (\text{cts-step } x y)$ **and** $\text{integral}^L M (\text{cts-step } x y) \leq$
cdf *M y*

proof –

have $\text{cdf } M x = \text{integral}^L M (\text{indicator } \{..x\})$

by (*simp add: cdf-def*)

also have $\dots \leq \text{expectation } (\text{cts-step } x y)$

by (*intro integral-mono integrable-cts-step*)

(*auto simp: cts-step-def less-top[symmetric] split: split-indicator*)

finally show $\text{cdf } M x \leq \text{expectation } (\text{cts-step } x y)$.

next

have $\text{expectation } (\text{cts-step } x y) \leq \text{integral}^L M (\text{indicator } \{..y\})$

by (*intro integral-mono integrable-cts-step*)

(*auto simp: cts-step-def less-top[symmetric] split: split-indicator*)

also have $\dots = \text{cdf } M y$

by (*simp add: cdf-def*)

finally show $\text{expectation } (\text{cts-step } x y) \leq \text{cdf } M y$.

qed

context

fixes $M\text{-seq} :: \text{nat} \Rightarrow \text{real measure}$
and $M :: \text{real measure}$
assumes $\text{distr-}M\text{-seq}$ [*simp*]: $\bigwedge n. \text{real-distribution } (M\text{-seq } n)$
assumes $\text{distr-}M$ [*simp*]: $\text{real-distribution } M$

begin

theorem *continuity-set-conv-imp-weak-conv*:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $*$: $\bigwedge A. A \in \text{sets borel} \implies M (\text{frontier } A) = 0 \implies (\lambda n. (\text{measure } (M\text{-seq } n) A)) \longrightarrow \text{measure } M A$
shows *weak-conv-m* $M\text{-seq } M$

proof –

interpret *real-distribution* M **by** *simp*

show *?thesis*

by (*auto intro!*: $*$ *simp*: *frontier-real-atMost isCont-cdf emeasure-eq-measure weak-conv-m-def weak-conv-def cdf-def2*)

qed

theorem *integral-cts-step-conv-imp-weak-conv*:

assumes *integral-conv*: $\bigwedge x y. x < y \implies (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x y)) \longrightarrow \text{integral}^L M (\text{cts-step } x y)$

shows *weak-conv-m* $M\text{-seq } M$

unfolding *weak-conv-m-def weak-conv-def*

proof (*clarsimp*)

interpret *real-distribution* M **by** (*rule distr-M*)

fix x **assume** *isCont* (*cdf* M) x

hence *left-cont*: *continuous-at-left* x (*cdf* M)

unfolding *continuous-at-split ..*

{ **fix** $y :: \text{real}$ **assume** [*arith*]: $x < y$

have $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) x) \leq \text{limsup } (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x y))$

by (*auto intro!*: *Limsup-mono always-eventually real-distribution.cdf-cts-step*)

also have $\dots = \text{integral}^L M (\text{cts-step } x y)$

by (*intro lim-imp-Limsup*) (*auto intro: integral-conv*)

also have $\dots \leq \text{cdf } M y$

by (*simp add: cdf-cts-step*)

finally have $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) x) \leq \text{cdf } M y$.

} **note** $*$ = *this*

{ **fix** $y :: \text{real}$ **assume** [*arith*]: $x > y$

have $\text{cdf } M y \leq \text{ereal } (\text{integral}^L M (\text{cts-step } y x))$

by (*simp add: cdf-cts-step*)

also have $\dots = \text{liminf } (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } y x))$

by (*intro lim-imp-Liminf[symmetric]*) (*auto intro: integral-conv*)

also have $\dots \leq \text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x)$

by (*auto intro!*: *Liminf-mono always-eventually real-distribution.cdf-cts-step*)

finally have $\text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x) \geq \text{cdf } M y$.

} **note** $**$ = *this*

```

have limsup ( $\lambda n. \text{cdf } (M\text{-seq } n) x \leq \text{cdf } M x$ )
proof (rule tendsto-lowerbound)
  show  $\forall_F i \text{ in } \text{at-right } x. \text{limsup } (\lambda xa. \text{ereal } (\text{cdf } (M\text{-seq } xa) x)) \leq \text{ereal } (\text{cdf } M$ 
i)
    by (subst eventually-at-right[of - x + 1]) (auto simp: * intro: exI [of - x+1])
  qed (insert cdf-is-right-cont, auto simp: continuous-within)
moreover have  $\text{cdf } M x \leq \text{liminf } (\lambda n. \text{cdf } (M\text{-seq } n) x)$ 
proof (rule tendsto-upperbound)
  show  $\forall_F i \text{ in } \text{at-left } x. \text{ereal } (\text{cdf } M i) \leq \text{liminf } (\lambda xa. \text{ereal } (\text{cdf } (M\text{-seq } xa) x))$ 
    by (subst eventually-at-left[of x - 1]) (auto simp: ** intro: exI [of - x-1])
  qed (insert left-cont, auto simp: continuous-within)
ultimately show  $(\lambda n. \text{cdf } (M\text{-seq } n) x) \longrightarrow \text{cdf } M x$ 
  by (elim limsup-le-liminf-real)
qed

```

theorem *integral-bdd-continuous-conv-imp-weak-conv:*

```

assumes
   $\bigwedge f. (\bigwedge x. \text{isCont } f x) \implies (\bigwedge x. \text{abs } (f x) \leq 1) \implies (\lambda n. \text{integral}^L (M\text{-seq } n)$ 
f::real)  $\longrightarrow \text{integral}^L M f$ 
shows
  weak-conv-m M-seq M
apply (rule integral-cts-step-conv-imp-weak-conv [OF assms])
apply (rule continuous-on-interior)
apply (rule uniformly-continuous-imp-continuous)
apply (rule cts-step-uniformly-continuous)
apply (auto simp: cts-step-def)
done

```

end

end

7 The Giry monad

theory *Giry-Monad*

```

imports Probability-Measure HOL-Library.Monad-Syntax
begin

```

7.1 Sub-probability spaces

```

locale subprob-space = finite-measure +
  assumes emeasure-space-le-1: emeasure M (space M) ≤ 1
  assumes subprob-not-empty: space M ≠ {}

```

lemma *subprob-spaceI[Pure.intro]:*

```

assumes *: emeasure M (space M) ≤ 1
assumes space M ≠ {}
shows subprob-space M
proof –

```

```

interpret finite-measure M
proof
  show emeasure M (space M)  $\neq \infty$  using * by (auto simp: top-unique)
qed
show subprob-space M by standard fact+
qed

lemma (in subprob-space) emeasure-subprob-space-less-top: emeasure M A  $\neq$  top
  using emeasure-finite[of A] .

lemma prob-space-imp-subprob-space:
  prob-space M  $\implies$  subprob-space M
  by (rule subprob-spaceI) (simp-all add: prob-space.emeasure-space-1 prob-space.not-empty)

lemma subprob-space-imp-sigma-finite: subprob-space M  $\implies$  sigma-finite-measure
M
  unfolding subprob-space-def finite-measure-def by simp

sublocale prob-space  $\subseteq$  subprob-space
  by (rule subprob-spaceI) (simp-all add: emeasure-space-1 not-empty)

lemma subprob-space-sigma [simp]:  $\Omega \neq \{\}$   $\implies$  subprob-space (sigma  $\Omega$  X)
  by (rule subprob-spaceI) (simp-all add: emeasure-sigma space-measure-of-conv)

lemma subprob-space-null-measure: space M  $\neq \{\}$   $\implies$  subprob-space (null-measure
M)
  by (simp add: null-measure-def)

lemma (in subprob-space) subprob-space-distr:
  assumes f:  $f \in$  measurable M M' and space M'  $\neq \{\}$  shows subprob-space (distr
M M' f)
proof (rule subprob-spaceI)
  have f - ' space M'  $\cap$  space M = space M using f by (auto dest: measur-
able-space)
  with f show emeasure (distr M M' f) (space (distr M M' f))  $\leq 1$ 
  by (auto simp: emeasure-distr emeasure-space-le-1)
  show space (distr M M' f)  $\neq \{\}$  by (simp add: assms)
qed

lemma (in subprob-space) subprob-emeasure-le-1: emeasure M X  $\leq 1$ 
  by (rule order.trans[OF emeasure-space emeasure-space-le-1])

lemma (in subprob-space) subprob-measure-le-1: measure M X  $\leq 1$ 
  using subprob-emeasure-le-1[of X] by (simp add: emeasure-eq-measure)

lemma (in subprob-space) nn-integral-le-const:
  assumes  $0 \leq c$  AE x in M.  $f x \leq c$ 
  shows  $(\int^+ x. f x \partial M) \leq c$ 
proof -

```

have $(\int^+ x. f x \partial M) \leq (\int^+ x. c \partial M)$
by(rule nn-integral-mono-AE) fact
also have $\dots \leq c * \text{emeasure } M$ (space M)
using $\langle 0 \leq c \rangle$ **by** simp
also have $\dots \leq c * 1$ **using** emeasure-space-le-1 $\langle 0 \leq c \rangle$ **by**(rule mult-left-mono)
finally show ?thesis **by** simp
qed

lemma emeasure-density-distr-interval:

fixes $h :: \text{real} \Rightarrow \text{real}$ **and** $g :: \text{real} \Rightarrow \text{real}$ **and** $g' :: \text{real} \Rightarrow \text{real}$
assumes [simp]: $a \leq b$
assumes $Mf[\text{measurable}]$: $f \in \text{borel-measurable borel}$
assumes $Mg[\text{measurable}]$: $g \in \text{borel-measurable borel}$
assumes $Mg'[\text{measurable}]$: $g' \in \text{borel-measurable borel}$
assumes $Mh[\text{measurable}]$: $h \in \text{borel-measurable borel}$
assumes prob: subprob-space (density lborel f)
assumes nonnegf: $\bigwedge x. f x \geq 0$
assumes derivg: $\bigwedge x. x \in \{a..b\} \implies (g \text{ has-real-derivative } g' x)$ (at x)
assumes contg': continuous-on $\{a..b\}$ g'
assumes mono: strict-mono-on $\{a..b\}$ g **and** inv: $\bigwedge x. h x \in \{a..b\} \implies g (h x)$
 $= x$
assumes range: $\{a..b\} \subseteq \text{range } h$
shows emeasure (distr (density lborel f) lborel h) $\{a..b\} =$
emeasure (density lborel $(\lambda x. f (g x) * g' x)$) $\{a..b\}$

proof (cases $a < b$)

assume $a < b$
from mono **have** inj: inj-on $g \{a..b\}$ **by** (rule strict-mono-on-imp-inj-on)
from mono **have** mono': mono-on $\{a..b\}$ g **by** (rule strict-mono-on-imp-mono-on)
from mono' **derivg** **have** $\bigwedge x. x \in \{a < .. < b\} \implies g' x \geq 0$
by (rule mono-on-imp-deriv-nonneg) auto
from contg' **this** **have** derivg-nonneg: $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$
by (rule continuous-ge-on-Ioo) (simp-all add: $\langle a < b \rangle$)

from derivg **have** contg: continuous-on $\{a..b\}$ g **by** (rule has-real-derivative-imp-continuous-on)

have A: $h - \{a..b\} = \{g a..g b\}$

proof (intro equalityI subsetI)

fix x **assume** $x \in h - \{a..b\}$

hence $g (h x) \in \{g a..g b\}$ **by** (auto intro: mono-onD[OF mono'])

with inv **and** x **show** $x \in \{g a..g b\}$ **by** simp

next

fix y **assume** $y \in \{g a..g b\}$

with IVT'[OF - - - contg, of y] **obtain** x **where** $x \in \{a..b\}$ $y = g x$ **by** auto

with range **and** inv **show** $y \in h - \{a..b\}$ **by** auto

qed

have prob': subprob-space (distr (density lborel f) lborel h)

by (rule subprob-space.subprob-space-distr[OF prob]) (simp-all add: Mh)

have B: emeasure (distr (density lborel f) lborel h) $\{a..b\} =$

$$\int^+ x. f x * \text{indicator } (h - \{a..b\}) x \partial \text{lborel}$$

by (*subst emeasure-distr*) (*simp-all add: emeasure-density Mf Mh measurable-sets-borel[OF Mh]*)
also note A
also have $\text{emeasure } (\text{distr } (\text{density lborel } f) \text{ lborel } h) \{a..b\} \leq 1$
by (*rule subprob-space.subprob-emeasure-le-1*) (*rule prob'*)
hence $\text{emeasure } (\text{distr } (\text{density lborel } f) \text{ lborel } h) \{a..b\} \neq \infty$ **by** (*auto simp: top-unique*)
with *assms* **have** $(\int^+ x. f x * \text{indicator } \{g a..g b\} x \partial \text{lborel}) =$
 $(\int^+ x. f (g x) * g' x * \text{indicator } \{a..b\} x \partial \text{lborel})$
by (*intro nn-integral-substitution-aux*)
(auto simp: derivg-nonneg A B emeasure-density mult.commute <a < b>)
also have $\dots = \text{emeasure } (\text{density lborel } (\lambda x. f (g x) * g' x)) \{a..b\}$
by (*simp add: emeasure-density*)
finally show *?thesis* .
next
assume $\neg a < b$
with $\langle a \leq b \rangle$ **have** [*simp*]: $b = a$ **by** (*simp add: not-less del: <a < b>*)
from *inv* **and** *range* **have** $h - \{a\} = \{g a\}$ **by** *auto*
thus *?thesis* **by** (*simp-all add: emeasure-distr emeasure-density measurable-sets-borel[OF Mh]*)
qed

locale *pair-subprob-space* =
pair-sigma-finite M1 M2 + M1: subprob-space M1 + M2: subprob-space M2 for M1 M2

sublocale *pair-subprob-space* $\subseteq P?$: *subprob-space M1* \otimes_M *M2*

proof

from *mult-le-one[OF M1.emeasure-space-le-1 - M2.emeasure-space-le-1]*
show $\text{emeasure } (M1 \otimes_M M2) (\text{space } (M1 \otimes_M M2)) \leq 1$
by (*simp add: M2.emeasure-pair-measure-Times space-pair-measure*)
from *M1.subprob-not-empty* **and** *M2.subprob-not-empty* **show** $\text{space } (M1 \otimes_M M2) \neq \{\}$
by (*simp add: space-pair-measure*)
qed

lemma *subprob-space-null-measure-iff*:

$\text{subprob-space } (\text{null-measure } M) \longleftrightarrow \text{space } M \neq \{\}$
by (*auto intro!: subprob-spaceI dest: subprob-space.subprob-not-empty*)

lemma *subprob-space-restrict-space*:

assumes M : *subprob-space M*
and A : $A \cap \text{space } M \in \text{sets } M$ $A \cap \text{space } M \neq \{\}$
shows *subprob-space (restrict-space M A)*
proof(*rule subprob-spaceI*)
have $\text{emeasure } (\text{restrict-space } M A) (\text{space } (\text{restrict-space } M A)) = \text{emeasure } M$
 $(A \cap \text{space } M)$
using A **by**(*simp add: emeasure-restrict-space space-restrict-space*)
also have $\dots \leq 1$ **by**(*rule subprob-space.subprob-emeasure-le-1*)(*rule M*)

```

finally show emeasure (restrict-space M A) (space (restrict-space M A)) ≤ 1 .
next
  show space (restrict-space M A) ≠ {}
  using A by(simp add: space-restrict-space)
qed

```

```

definition subprob-algebra :: 'a measure ⇒ 'a measure measure where
  subprob-algebra K =
    (SUP A ∈ sets K. vimage-algebra {M. subprob-space M ∧ sets M = sets K}
    (λM. emeasure M A) borel)

```

```

lemma space-subprob-algebra: space (subprob-algebra A) = {M. subprob-space M
  ∧ sets M = sets A}
  by (auto simp add: subprob-algebra-def space-Sup-eq-UN)

```

```

lemma subprob-algebra-cong: sets M = sets N ⇒ subprob-algebra M = sub-
  prob-algebra N
  by (simp add: subprob-algebra-def)

```

```

lemma measurable-emeasure-subprob-algebra[measurable]:
  a ∈ sets A ⇒ (λM. emeasure M a) ∈ borel-measurable (subprob-algebra A)
  by (auto intro!: measurable-Sup1 measurable-vimage-algebra1 simp: subprob-algebra-def)

```

```

lemma measurable-measure-subprob-algebra[measurable]:
  a ∈ sets A ⇒ (λM. measure M a) ∈ borel-measurable (subprob-algebra A)
  unfolding measure-def by measurable

```

```

lemma subprob-measurableD:
  assumes N: N ∈ measurable M (subprob-algebra S) and x: x ∈ space M
  shows space (N x) = space S
  and sets (N x) = sets S
  and measurable (N x) K = measurable S K
  and measurable K (N x) = measurable K S
  using measurable-space[OF N x]
  by (auto simp: space-subprob-algebra intro!: measurable-cong-sets dest: sets-eq-imp-space-eq)

```

ML ‹

```

fun subprob-cong thm ctxt = (
  let
    val thm' = Thm.transfer' ctxt thm
    val free = thm' |> Thm.concl-of |> HOLogic.dest-Trueprop |> dest-comb |> fst
  |>
    dest-comb |> snd |> strip-abs-body |> head-of |> is-Free
  in
    if free then ([], Measurable.add-local-cong (thm' RS @{thm subprob-measurableD(2)}))
  ctxt
    else ([], ctxt)
  end

```

```

    handle THM - => ([], ctxt) | TERM - => ([], ctxt)

  ›
setup ‹
  Context.theory-map (Measurable.add-preprocessor subprob-cong subprob-cong)
  ›

context
  fixes K M N assumes K: K ∈ measurable M (subprob-algebra N)
begin

lemma subprob-space-kernel: a ∈ space M ⇒ subprob-space (K a)
  using measurable-space[OF K] by (simp add: space-subprob-algebra)

lemma sets-kernel: a ∈ space M ⇒ sets (K a) = sets N
  using measurable-space[OF K] by (simp add: space-subprob-algebra)

lemma measurable-emeasure-kernel[measurable]:
  A ∈ sets N ⇒ (λa. emeasure (K a) A) ∈ borel-measurable M
  using measurable-compose[OF K measurable-emeasure-subprob-algebra] .

end

lemma measurable-subprob-algebra:
  (∧a. a ∈ space M ⇒ subprob-space (K a)) ⇒
  (∧a. a ∈ space M ⇒ sets (K a) = sets N) ⇒
  (∧A. A ∈ sets N ⇒ (λa. emeasure (K a) A) ∈ borel-measurable M) ⇒
  K ∈ measurable M (subprob-algebra N)
  by (auto intro!: measurable-Sup2 measurable-vimage-algebra2 simp: subprob-algebra-def)

lemma measurable-submarkov:
  K ∈ measurable M (subprob-algebra M) ‹↔›
  (∀x∈space M. subprob-space (K x) ∧ sets (K x) = sets M) ∧
  (∀A∈sets M. (λx. emeasure (K x) A) ∈ measurable M borel)
proof
  assume (∀x∈space M. subprob-space (K x) ∧ sets (K x) = sets M) ∧
  (∀A∈sets M. (λx. emeasure (K x) A) ∈ borel-measurable M)
  then show K ∈ measurable M (subprob-algebra M)
  by (intro measurable-subprob-algebra) auto
next
  assume K ∈ measurable M (subprob-algebra M)
  then show (∀x∈space M. subprob-space (K x) ∧ sets (K x) = sets M) ∧
  (∀A∈sets M. (λx. emeasure (K x) A) ∈ borel-measurable M)
  by (auto dest: subprob-space-kernel sets-kernel)
qed

lemma measurable-subprob-algebra-generated:
  assumes eq: sets N = sigma-sets Ω G and Int-stable G G ⊆ Pow Ω

```



```

assumes subsp:  $\bigwedge a. a \in \text{space } M \implies \text{subprob-space } (K a)$ 
assumes sets:  $\bigwedge a. a \in \text{space } M \implies \text{sets } (K a) = \text{sets } N$ 
assumes  $\bigwedge A. A \in G \implies (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M$ 
assumes  $\Omega: (\lambda a. \text{emeasure } (K a) \Omega) \in \text{borel-measurable } M$ 
shows  $K \in \text{measurable } M \text{ (subprob-algebra } N)$ 
proof (rule measurable-subprob-algebra)
  fix a assume  $a \in \text{space } M$  then show  $\text{subprob-space } (K a) \text{ sets } (K a) = \text{sets } N$ 
by fact+
next
  interpret G: sigma-algebra  $\Omega$  sigma-sets  $\Omega$  G
    using  $\langle G \subseteq \text{Pow } \Omega \rangle$  by (rule sigma-algebra-sigma-sets)
    fix A assume  $A \in \text{sets } N$  with assms(2,3) show  $(\lambda a. \text{emeasure } (K a) A) \in$ 
borel-measurable  $M$ 
    unfolding  $\langle \text{sets } N = \text{sigma-sets } \Omega G \rangle$ 
    proof (induction rule: sigma-sets-induct-disjoint)
      case (basic A) then show ?case by fact
    next
      case empty then show ?case by simp
    next
      case (compl A)
        have  $(\lambda a. \text{emeasure } (K a) (\Omega - A)) \in \text{borel-measurable } M \longleftrightarrow$ 
           $(\lambda a. \text{emeasure } (K a) \Omega - \text{emeasure } (K a) A) \in \text{borel-measurable } M$ 
        using G.top G.sets-into-space sets eq compl subprob-space.emeasure-subprob-space-less-top[OF
subsp]
        by (intro measurable-cong emeasure-Diff) auto
        with compl  $\Omega$  show ?case
        by simp
    next
      case (union F)
        moreover have  $(\lambda a. \text{emeasure } (K a) (\bigcup i. F i)) \in \text{borel-measurable } M \longleftrightarrow$ 
           $(\lambda a. \sum i. \text{emeasure } (K a) (F i)) \in \text{borel-measurable } M$ 
        using sets union eq
        by (intro measurable-cong suminf-emeasure[symmetric]) auto
        ultimately show ?case
        by auto
    qed
qed

lemma space-subprob-algebra-empty-iff:
   $\text{space } (\text{subprob-algebra } N) = \{\}$   $\longleftrightarrow \text{space } N = \{\}$ 
proof
  have  $\bigwedge x. x \in \text{space } N \implies \text{density } N (\lambda-. 0) \in \text{space } (\text{subprob-algebra } N)$ 
    by (auto simp: space-subprob-algebra emeasure-density intro!: subprob-spaceI)
  then show  $\text{space } (\text{subprob-algebra } N) = \{\} \implies \text{space } N = \{\}$ 
    by auto
next
  assume  $\text{space } N = \{\}$ 
  hence  $\text{sets } N = \{\{\}\}$  by (simp add: space-empty-iff)
  moreover have  $\bigwedge M. \text{subprob-space } M \implies \text{sets } M \neq \{\{\}\}$ 

```

by (*simp add: subprob-space.subprob-not-empty space-empty-iff[symmetric]*)
ultimately show $\text{space } (\text{subprob-algebra } N) = \{\}$ **by** (*auto simp: space-subprob-algebra*)
qed

lemma *nn-integral-measurable-subprob-algebra[measurable]:*

assumes $f: f \in \text{borel-measurable } N$

shows $(\lambda M. \text{integral}^N M f) \in \text{borel-measurable } (\text{subprob-algebra } N)$ (**is** $- \in ?B$)

using f

proof *induct*

case (*cong f g*)

moreover have $(\lambda M'. \int^+ M''. f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \int^+ M''. g M'' \partial M') \in ?B$

by (*intro measurable-cong nn-integral-cong cong*)

(*auto simp: space-subprob-algebra dest!: sets-eq-imp-space-eq*)

ultimately show *?case* **by** *simp*

next

case (*set B*)

then have $(\lambda M'. \int^+ M''. \text{indicator } B M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \text{emeasure } M' B) \in ?B$

by (*intro measurable-cong nn-integral-indicator*) (*simp add: space-subprob-algebra*)

with *set show ?case*

by (*simp add: measurable-emeasure-subprob-algebra*)

next

case (*mult f c*)

then have $(\lambda M'. \int^+ M''. c * f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. c * \int^+ M''. f M'' \partial M') \in ?B$

by (*intro measurable-cong nn-integral-cmult*) (*auto simp add: space-subprob-algebra*)

with *mult show ?case*

by *simp*

next

case (*add f g*)

then have $(\lambda M'. \int^+ M''. f M'' + g M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. (\int^+ M''. f M'' \partial M') + (\int^+ M''. g M'' \partial M')) \in ?B$

by (*intro measurable-cong nn-integral-add*) (*auto simp add: space-subprob-algebra*)

with *add show ?case*

by (*simp add: ac-simps*)

next

case (*seq F*)

then have $(\lambda M'. \int^+ M''. (\text{SUP } i. F i) M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \text{SUP } i. \int^+ M''. F i M'' \partial M') \in ?B$

unfolding *SUP-apply*

by (*intro measurable-cong nn-integral-monotone-convergence-SUP*) (*auto simp add: space-subprob-algebra*)

with *seq show ?case*

by (*simp add: ac-simps*)

qed

lemma *measurable-distr:*

assumes [*measurable*]: $f \in \text{measurable } M N$

shows $(\lambda M'. \text{distr } M' N f) \in \text{measurable (subprob-algebra } M) \text{ (subprob-algebra } N)$
proof (*cases space* $N = \{\}$)
assume *not-empty: space* $N \neq \{\}$
show *?thesis*
proof (*rule measurable-subprob-algebra*)
fix A **assume** $A: A \in \text{sets } N$
then have $(\lambda M'. \text{emeasure (distr } M' N f) A) \in \text{borel-measurable (subprob-algebra } M)$ \longleftrightarrow
 $(\lambda M'. \text{emeasure } M' (f \text{ - ' } A \cap \text{space } M)) \in \text{borel-measurable (subprob-algebra } M)$
by (*intro measurable-cong*)
(auto simp: emeasure-distr space-subprob-algebra
intro!: arg-cong2[where f=emeasure] sets-eq-imp-space-eq arg-cong2[where
f=(\cap)])
also have ...
using A **by** (*intro measurable-emeasure-subprob-algebra simp*)
finally show $(\lambda M'. \text{emeasure (distr } M' N f) A) \in \text{borel-measurable (subprob-algebra } M)$.
qed (*auto intro!: subprob-space.subprob-space-distr simp: space-subprob-algebra*
not-empty cong: measurable-cong-sets)
qed (*insert assms, auto simp: measurable-empty-iff space-subprob-algebra-empty-iff*)

lemma *emeasure-space-subprob-algebra[measurable]:*

$(\lambda a. \text{emeasure } a \text{ (space } a)) \in \text{borel-measurable (subprob-algebra } N)$

proof –

have $(\lambda a. \text{emeasure } a \text{ (space } N)) \in \text{borel-measurable (subprob-algebra } N)$ (**is** $?f \in ?M$)

by (*rule measurable-emeasure-subprob-algebra simp*)

also have $?f \in ?M \longleftrightarrow (\lambda a. \text{emeasure } a \text{ (space } a)) \in ?M$

by (*rule measurable-cong (auto simp: space-subprob-algebra dest: sets-eq-imp-space-eq)*)

finally show *?thesis* .

qed

lemma *integrable-measurable-subprob-algebra[measurable]:*

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$

assumes [*measurable*]: $f \in \text{borel-measurable } N$

shows *Measurable.pred (subprob-algebra } N) (\lambda M. \text{integrable } M f)*

proof (*rule measurable-cong[THEN iffD2]*)

show $M \in \text{space (subprob-algebra } N) \implies \text{integrable } M f \longleftrightarrow (\int^{+x}. \text{norm } (f x) \partial M) < \infty$ **for** M

by (*auto simp: space-subprob-algebra integrable-iff-bounded*)

qed *measurable*

lemma *integral-measurable-subprob-algebra[measurable]:*

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$

assumes f [*measurable*]: $f \in \text{borel-measurable } N$

shows $(\lambda M. \text{integral}^L M f) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$

proof –

from *borel-measurable-implies-sequence-metric*[*OF f, of 0*]
obtain F **where** $F: \bigwedge i. \text{simple-function } N (F i)$
 $\bigwedge x. x \in \text{space } N \implies (\lambda i. F i x) \longrightarrow f x$
 $\bigwedge i x. x \in \text{space } N \implies \text{norm } (F i x) \leq 2 * \text{norm } (f x)$
unfolding *norm-conv-dist* **by** *blast*

have [*measurable*]: $F i \in N \rightarrow_M \text{count-space UNIV}$ **for** i
using $F(1)$ **by** (*rule measurable-simple-function*)

define F' **where** [*abs-def*]:
 $F' M i = (\text{if integrable } M f \text{ then } \text{integral}^L M (F i) \text{ else } 0)$ **for** $M i$

have $(\lambda M. F' M i) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$ **for** i
proof (*rule measurable-cong*[*THEN iffD2*])
fix M **assume** $M \in \text{space } (\text{subprob-algebra } N)$
then have [*simp*]: *sets* $M = \text{sets } N$ *space* $M = \text{space } N$ *subprob-space* M
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
interpret *subprob-space* M **by fact**
have $F' M i = (\text{if integrable } M f \text{ then } \text{Bochner-Integration.simple-bochner-integral}$
 $M (F i) \text{ else } 0)$
using $F(1)$
by (*subst simple-bochner-integrable-eq-integral*)
(*auto simp: simple-bochner-integrable.simps simple-function-def F'-def*)
then show $F' M i = (\text{if integrable } M f \text{ then } \sum_{y \in F i} \text{space } N. \text{measure } M$
 $\{x \in \text{space } N. F i x = y\} *_{\mathbb{R}} y \text{ else } 0)$
unfolding *simple-bochner-integral-def* **by** *simp*
qed *measurable*

moreover
have $F' M \longrightarrow \text{integral}^L M f$ **if** $M: M \in \text{space } (\text{subprob-algebra } N)$ **for** M
proof *cases*
from M **have** [*simp*]: *sets* $M = \text{sets } N$ *space* $M = \text{space } N$
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
assume *integrable* $M f$ **then show** *?thesis*
unfolding *F'-def* **using** $F(1)$ [*THEN borel-measurable-simple-function*] F
by (*auto intro!: integral-dominated-convergence*[**where** $w = \lambda x. 2 * \text{norm } (f x)$]
cong: measurable-cong-sets)
qed (*auto simp: F'-def not-integrable-integral-eq*)
ultimately show *?thesis*
by (*rule borel-measurable-LIMSEQ-metric*)
qed

lemma *measurable-pair-measure*:

assumes $f: f \in \text{measurable } M (\text{subprob-algebra } N)$
assumes $g: g \in \text{measurable } M (\text{subprob-algebra } L)$
shows $(\lambda x. f x \otimes_M g x) \in \text{measurable } M (\text{subprob-algebra } (N \otimes_M L))$
proof (*rule measurable-subprob-algebra*)
{ fix x **assume** $x \in \text{space } M$
with *measurable-space*[*OF f*] *measurable-space*[*OF g*]

```

have  $fx: f x \in \text{space } (\text{subprob-algebra } N)$  and  $gx: g x \in \text{space } (\text{subprob-algebra } L)$ 
  by auto
interpret  $F: \text{subprob-space } f x$ 
  using  $fx$  by (simp add: space-subprob-algebra)
interpret  $G: \text{subprob-space } g x$ 
  using  $gx$  by (simp add: space-subprob-algebra)

interpret pair-subprob-space  $f x g x ..$ 
show subprob-space  $(f x \otimes_M g x)$  by unfold-locales
show sets-eq:  $\text{sets } (f x \otimes_M g x) = \text{sets } (N \otimes_M L)$ 
  using  $fx gx$  by (simp add: space-subprob-algebra)

have  $1: \bigwedge A B. A \in \text{sets } N \implies B \in \text{sets } L \implies \text{emeasure } (f x \otimes_M g x) (A \times B) = \text{emeasure } (f x) A * \text{emeasure } (g x) B$ 
  using  $fx gx$  by (intro G.emeasure-pair-measure-Times) (auto simp: space-subprob-algebra)
have  $\text{emeasure } (f x \otimes_M g x) (\text{space } (f x \otimes_M g x)) = \text{emeasure } (f x) (\text{space } (f x)) * \text{emeasure } (g x) (\text{space } (g x))$ 
  by (subst G.emeasure-pair-measure-Times[symmetric]) (simp-all add: space-pair-measure)
hence  $2: \bigwedge A. A \in \text{sets } (N \otimes_M L) \implies \text{emeasure } (f x \otimes_M g x) (\text{space } N \times \text{space } L - A) = \dots - \text{emeasure } (f x \otimes_M g x) A$ 
  using emeasure-compl[simplified, OF - P.emeasure-finite]
  unfolding sets-eq
  unfolding sets-eq-imp-space-eq[OF sets-eq]
  by (simp add: space-pair-measure G.emeasure-pair-measure-Times)
note  $1\ 2\ \text{sets-eq}$  }
note  $\text{Times} = \text{this}(1)$  and  $\text{Compl} = \text{this}(2)$  and  $\text{sets-eq} = \text{this}(3)$ 

fix  $A$  assume  $A: A \in \text{sets } (N \otimes_M L)$ 
show  $(\lambda a. \text{emeasure } (f a \otimes_M g a) A) \in \text{borel-measurable } M$ 
  using Int-stable-pair-measure-generator pair-measure-closed A
  unfolding sets-pair-measure
proof (induct A rule: sigma-sets-induct-disjoint)
  case (basic A) then show ?case
    by (auto intro!: borel-measurable-times-ennreal simp: Times cong: measurable-cong)
    (auto intro!: measurable-emeasure-kernel f g)
  next
  case (compl A)
  then have  $A: A \in \text{sets } (N \otimes_M L)$ 
    by (auto simp: sets-pair-measure)
  have  $(\lambda x. \text{emeasure } (f x) (\text{space } (f x)) * \text{emeasure } (g x) (\text{space } (g x)) - \text{emeasure } (f x \otimes_M g x) A) \in \text{borel-measurable } M$  (is ?f ∈ ?M)
    using compl(2) f g by measurable
  thus ?case by (simp add: Compl A cong: measurable-cong)
next
case (union A)
then have  $\text{range } A \subseteq \text{sets } (N \otimes_M L)$  disjoint-family A

```

```

    by (auto simp: sets-pair-measure)
  then have  $(\lambda a. \text{emeasure } (f a \otimes_M g a) (\bigcup i. A i)) \in \text{borel-measurable } M \longleftrightarrow$ 
     $(\lambda a. \sum i. \text{emeasure } (f a \otimes_M g a) (A i)) \in \text{borel-measurable } M$ 
    by (intro measurable-cong suminf-emeasure[symmetric])
      (auto simp: sets-eq)
  also have ...
    using union by auto
  finally show ?case .
qed simp
qed

lemma restrict-space-measurable:
  assumes X:  $X \neq \{\}$   $X \in \text{sets } K$ 
  assumes N:  $N \in \text{measurable } M$  (subprob-algebra K)
  shows  $(\lambda x. \text{restrict-space } (N x) X) \in \text{measurable } M$  (subprob-algebra (restrict-space K X))
proof (rule measurable-subprob-algebra)
  fix a assume a:  $a \in \text{space } M$ 
  from N[THEN measurable-space, OF this]
  have subprob-space (N a) and [simp]:  $\text{sets } (N a) = \text{sets } K$   $\text{space } (N a) = \text{space } K$ 
  by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
  then interpret subprob-space N a
    by simp
  show subprob-space (restrict-space (N a) X)
  proof
    show  $\text{space } (\text{restrict-space } (N a) X) \neq \{\}$ 
      using X by (auto simp add: space-restrict-space)
    show  $\text{emeasure } (\text{restrict-space } (N a) X) (\text{space } (\text{restrict-space } (N a) X)) \leq 1$ 
      using X by (simp add: emeasure-restrict-space space-restrict-space subprob-emeasure-le-1)
  qed
  show  $\text{sets } (\text{restrict-space } (N a) X) = \text{sets } (\text{restrict-space } K X)$ 
    by (intro sets-restrict-space-cong) fact
next
fix A assume A:  $A \in \text{sets } (\text{restrict-space } K X)$ 
show  $(\lambda a. \text{emeasure } (\text{restrict-space } (N a) X) A) \in \text{borel-measurable } M$ 
proof (subst measurable-cong)
  fix a assume a  $\in \text{space } M$ 
  from N[THEN measurable-space, OF this]
  have [simp]:  $\text{sets } (N a) = \text{sets } K$   $\text{space } (N a) = \text{space } K$ 
    by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
  show  $\text{emeasure } (\text{restrict-space } (N a) X) A = \text{emeasure } (N a) (A \cap X)$ 
    using X A by (subst emeasure-restrict-space) (auto simp add: sets-restrict-space ac-simps)
next
show  $(\lambda w. \text{emeasure } (N w) (A \cap X)) \in \text{borel-measurable } M$ 
  using A X
  by (intro measurable-compose[OF N measurable-emeasure-subprob-algebra])

```

(*auto simp: sets-restrict-space*)

qed
qed

7.2 Properties of “return”

definition *return* :: 'a measure \Rightarrow 'a \Rightarrow 'a measure **where**
return R $x = \text{measure-of (space } R) (\text{sets } R) (\lambda A. \text{indicator } A x)$

lemma *space-return[simp]*: $\text{space (return } M x) = \text{space } M$
by (*simp add: return-def*)

lemma *sets-return[simp]*: $\text{sets (return } M x) = \text{sets } M$
by (*simp add: return-def*)

lemma *measurable-return1[simp]*: $\text{measurable (return } N x) L = \text{measurable } N L$
by (*simp cong: measurable-cong-sets*)

lemma *measurable-return2[simp]*: $\text{measurable } L (\text{return } N x) = \text{measurable } L N$
by (*simp cong: measurable-cong-sets*)

lemma *return-sets-cong*: $\text{sets } M = \text{sets } N \Longrightarrow \text{return } M = \text{return } N$
by (*auto dest: sets-eq-imp-space-eq simp: fun-eq-iff return-def*)

lemma *return-cong*: $\text{sets } A = \text{sets } B \Longrightarrow \text{return } A x = \text{return } B x$
by (*auto simp add: return-def dest: sets-eq-imp-space-eq*)

lemma *emeasure-return[simp]*:

assumes $A \in \text{sets } M$

shows $\text{emeasure (return } M x) A = \text{indicator } A x$

proof (*rule emeasure-measure-of[OF return-def]*)

show $\text{sets } M \subseteq \text{Pow (space } M)$ **by** (*rule sets.space-closed*)

show *positive* ($\text{sets (return } M x)$) ($\lambda A. \text{indicator } A x$) **by** (*simp add: positive-def*)

from *assms* **show** $A \in \text{sets (return } M x)$ **unfolding** *return-def* **by** *simp*

show *countably-additive* ($\text{sets (return } M x)$) ($\lambda A. \text{indicator } A x$)

by (*auto intro!: countably-additiveI suminf-indicator*)

qed

lemma *prob-space-return*: $x \in \text{space } M \Longrightarrow \text{prob-space (return } M x)$
by *rule simp*

lemma *subprob-space-return*: $x \in \text{space } M \Longrightarrow \text{subprob-space (return } M x)$
by (*intro prob-space-return prob-space-imp-subprob-space*)

lemma *subprob-space-return-ne*:

assumes $\text{space } M \neq \{\}$ **shows** $\text{subprob-space (return } M x)$

proof

show $\text{emeasure (return } M x) (\text{space (return } M x)) \leq 1$

by (*subst emeasure-return (auto split: split-indicator)*)

qed (*simp, fact*)

lemma *measure-return*: **assumes** $X: X \in \text{sets } M$ **shows** $\text{measure } (\text{return } M x) X = \text{indicator } X x$
unfolding *measure-def emeasure-return*[*OF X, of x*] **by** (*simp split: split-indicator*)

lemma *AE-return*:

assumes [*simp*]: $x \in \text{space } M$ **and** [*measurable*]: $\text{Measurable.pred } M P$
shows $(AE y \text{ in } \text{return } M x. P y) \longleftrightarrow P x$

proof –

have $(AE y \text{ in } \text{return } M x. y \notin \{x \in \text{space } M. \neg P x\}) \longleftrightarrow P x$

by (*subst AE-iff-null-sets[symmetric]*) (*simp-all add: null-sets-def split: split-indicator*)

also have $(AE y \text{ in } \text{return } M x. y \notin \{x \in \text{space } M. \neg P x\}) \longleftrightarrow (AE y \text{ in } \text{return } M x. P y)$

by (*rule AE-cong*) *auto*

finally show *?thesis* .

qed

lemma *nn-integral-return*:

assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$

shows $(\int^+ a. g a \partial \text{return } M x) = g x$

proof –

interpret *prob-space return M x* **by** (*rule prob-space-return*[*OF ⟨x ∈ space M⟩*])

have $(\int^+ a. g a \partial \text{return } M x) = (\int^+ a. g x \partial \text{return } M x)$ **using** *assms*

by (*intro nn-integral-cong-AE*) (*auto simp: AE-return*)

also have $\dots = g x$

using *nn-integral-const*[*of return M x*] *emeasure-space-1* **by** *simp*

finally show *?thesis* .

qed

lemma *integral-return*:

fixes $g :: - \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$

shows $(\int a. g a \partial \text{return } M x) = g x$

proof –

interpret *prob-space return M x* **by** (*rule prob-space-return*[*OF ⟨x ∈ space M⟩*])

have $(\int a. g a \partial \text{return } M x) = (\int a. g x \partial \text{return } M x)$ **using** *assms*

by (*intro integral-cong-AE*) (*auto simp: AE-return*)

then show *?thesis*

using *prob-space* **by** *simp*

qed

lemma *return-measurable*[*measurable*]: $\text{return } N \in \text{measurable } N$ (*subprob-algebra N*)

by (*rule measurable-subprob-algebra*) (*auto simp: subprob-space-return*)

lemma *distr-return*:

assumes $f \in \text{measurable } M N$ **and** $x \in \text{space } M$

shows $\text{distr } (\text{return } M x) N f = \text{return } N (f x)$

using *assms* **by** (*intro measure-eqI*) (*simp-all add: indicator-def emeasure-distr*)

lemma *return-restrict-space*:

$\Omega \in \text{sets } M \implies \text{return (restrict-space } M \ \Omega) \ x = \text{restrict-space (return } M \ x) \ \Omega$
by (*auto intro!: measure-eqI simp: sets-restrict-space emeasure-restrict-space*)

lemma *measurable-distr2*:

assumes *f[measurable]*: *case-prod* $f \in \text{measurable } (L \otimes_M M) \ N$
assumes *g[measurable]*: $g \in \text{measurable } L \ (\text{subprob-algebra } M)$
shows $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L \ (\text{subprob-algebra } N)$

proof –

have $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L \ (\text{subprob-algebra } N)$
 $\longleftrightarrow (\lambda x. \text{distr } (\text{return } L \ x \otimes_M g \ x) \ N \ (\text{case-prod } f)) \in \text{measurable } L \ (\text{subprob-algebra } N)$

proof (*rule measurable-cong*)

fix *x* **assume** $x \in \text{space } L$

have $g \ x \in \text{space } (\text{subprob-algebra } M)$

using *measurable-space[OF g x]* .

then have [*simp*]: $\text{sets } (g \ x) = \text{sets } M$

by (*simp add: space-subprob-algebra*)

then have [*simp*]: $\text{space } (g \ x) = \text{space } M$

by (*rule sets-eq-imp-space-eq*)

let $?R = \text{return } L \ x$

from *measurable-compose-Pair1[OF x f]* **have** $f \text{-}M'$: $f \ x \in \text{measurable } M \ N$

by *simp*

interpret *subprob-space g x*

using *gx* **by** (*simp add: space-subprob-algebra*)

have *space-pair-M'[simp]*: $\bigwedge X. \text{space } (X \otimes_M g \ x) = \text{space } (X \otimes_M M)$

by (*simp add: space-pair-measure*)

show $\text{distr } (g \ x) \ N \ (f \ x) = \text{distr } (?R \otimes_M g \ x) \ N \ (\text{case-prod } f)$ (**is** $?l = ?r$)

proof (*rule measure-eqI*)

show $\text{sets } ?l = \text{sets } ?r$

by *simp*

next

fix *A* **assume** $A \in \text{sets } ?l$

then have *A[measurable]*: $A \in \text{sets } N$

by *simp*

then have $\text{emeasure } ?r \ A = \text{emeasure } (?R \otimes_M g \ x) \ ((\lambda(x, y). f \ x \ y) \text{-}' A \cap \text{space } (?R \otimes_M g \ x))$

by (*auto simp add: emeasure-distr f-M' cong: measurable-cong-sets*)

also have $\dots = (\int^+ M''. \text{emeasure } (g \ x) \ (f \ M'' \text{-}' A \cap \text{space } M) \ \partial ?R)$

apply (*subst emeasure-pair-measure-alt*)

apply (*rule measurable-sets[OF - A]*)

apply (*auto simp add: f-M' cong: measurable-cong-sets*)

apply (*intro nn-integral-cong arg-cong[where f=emeasure (g x)]*)

apply (*auto simp: space-subprob-algebra space-pair-measure*)

done

also have $\dots = \text{emeasure } (g \ x) \ (f \ x \text{-}' A \cap \text{space } M)$

by (*subst nn-integral-return*)

```

      (auto simp: x intro!: measurable-emeasure)
    also have ... = emeasure ?l A
      by (simp add: emeasure-distr f-M' cong: measurable-cong-sets)
    finally show emeasure ?l A = emeasure ?r A ..
  qed
qed
also have ...
  apply (intro measurable-compose[OF measurable-pair-measure measurable-distr])
  apply (rule return-measurable)
  apply measurable
  done
  finally show ?thesis .
qed

```

lemma *nn-integral-measurable-subprob-algebra2*:

```

  assumes f[measurable]:  $(\lambda(x, y). f x y) \in \text{borel-measurable } (M \otimes_M N)$ 
  assumes N[measurable]:  $L \in \text{measurable } M \text{ (subprob-algebra } N)$ 
  shows  $(\lambda x. \text{integral}^N (L x) (f x)) \in \text{borel-measurable } M$ 
proof -
  note nn-integral-measurable-subprob-algebra[measurable]
  note measurable-distr2[measurable]
  have  $(\lambda x. \text{integral}^N (\text{distr } (L x) (M \otimes_M N) (\lambda y. (x, y))) (\lambda(x, y). f x y)) \in$ 
  borel-measurable M
    by measurable
  then show  $(\lambda x. \text{integral}^N (L x) (f x)) \in \text{borel-measurable } M$ 
    by (rule measurable-cong[THEN iffD1, rotated])
      (simp add: nn-integral-distr)
qed

```

lemma *emeasure-measurable-subprob-algebra2*:

```

  assumes A[measurable]:  $(\text{SIGMA } x:\text{space } M. A x) \in \text{sets } (M \otimes_M N)$ 
  assumes L[measurable]:  $L \in \text{measurable } M \text{ (subprob-algebra } N)$ 
  shows  $(\lambda x. \text{emeasure } (L x) (A x)) \in \text{borel-measurable } M$ 
proof -
  { fix x assume x ∈ space M
    then have Pair x - ‘Sigma (space M) A = A x
      by auto
    with sets-Pair1[OF A, of x] have A x ∈ sets N
      by auto }
  note ** = this

  have *:  $\bigwedge x. \text{fst } x \in \text{space } M \implies \text{snd } x \in A \iff x \in (\text{SIGMA } x:\text{space } M. A x)$ 
  by (auto simp: fun-eq-iff)
  have  $(\lambda(x, y). \text{indicator } (A x) y::\text{ennreal}) \in \text{borel-measurable } (M \otimes_M N)$ 
  apply measurable
  apply (subst measurable-cong)
  apply (rule *)
  apply (auto simp: space-pair-measure)

```

```

done
then have ( $\lambda x. \text{integral}^N (L x) (\text{indicator } (A x)) \in \text{borel-measurable } M$ )
  by (intro nn-integral-measurable-subprob-algebra2[where N=N] L)
then show ( $\lambda x. \text{emeasure } (L x) (A x) \in \text{borel-measurable } M$ )
  apply (rule measurable-cong[THEN iffD1, rotated])
  apply (rule nn-integral-indicator)
  apply (simp add: subprob-measurableD[OF L] **)
done
qed

lemma measure-measurable-subprob-algebra2:
  assumes A[measurable]: ( $\text{SIGMA } x:\text{space } M. A x \in \text{sets } (M \otimes_M N)$ )
  assumes L[measurable]:  $L \in \text{measurable } M (\text{subprob-algebra } N)$ 
  shows ( $\lambda x. \text{measure } (L x) (A x) \in \text{borel-measurable } M$ )
  unfolding measure-def
  by (intro borel-measurable-enn2real emeasure-measurable-subprob-algebra2[OF assms])

definition select-sets  $M = (\text{SOME } N. \text{sets } M = \text{sets } (\text{subprob-algebra } N))$ 

lemma select-sets1:
   $\text{sets } M = \text{sets } (\text{subprob-algebra } N) \implies \text{sets } M = \text{sets } (\text{subprob-algebra } (\text{select-sets } M))$ 
  unfolding select-sets-def by (rule someI)

lemma sets-select-sets[simp]:
  assumes sets:  $\text{sets } M = \text{sets } (\text{subprob-algebra } N)$ 
  shows  $\text{sets } (\text{select-sets } M) = \text{sets } N$ 
  unfolding select-sets-def
proof (rule someI2)
  show  $\text{sets } M = \text{sets } (\text{subprob-algebra } N)$ 
  by fact
next
  fix L assume  $\text{sets } M = \text{sets } (\text{subprob-algebra } L)$ 
  with sets have eq:  $\text{space } (\text{subprob-algebra } N) = \text{space } (\text{subprob-algebra } L)$ 
  by (intro sets-eq-imp-space-eq) simp
  show  $\text{sets } L = \text{sets } N$ 
  proof cases
    assume  $\text{space } (\text{subprob-algebra } N) = \{\}$ 
    with  $\text{space-subprob-algebra-empty-iff}[of N] \text{space-subprob-algebra-empty-iff}[of L]$ 
    show ?thesis
    by (simp add: eq space-empty-iff)
  next
    assume  $\text{space } (\text{subprob-algebra } N) \neq \{\}$ 
    with eq show ?thesis
    by (fastforce simp add: space-subprob-algebra)
  qed
qed
qed

```

lemma *space-select-sets[simp]*:

sets $M = \text{sets (subprob-algebra } N) \implies \text{space (select-sets } M) = \text{space } N$
by (*intro sets-eq-imp-space-eq sets-select-sets*)

7.3 Join

definition *join* :: 'a measure measure \Rightarrow 'a measure **where**

join $M = \text{measure-of (space (select-sets } M)) (\text{sets (select-sets } M)) (\lambda B. \int^+ M'.$
emeasure $M' B \partial M)$

lemma

shows *space-join[simp]*: $\text{space (join } M) = \text{space (select-sets } M)$
and *sets-join[simp]*: $\text{sets (join } M) = \text{sets (select-sets } M)$
by (*simp-all add: join-def*)

lemma *emeasure-join*:

assumes *M[simp, measurable-cong]*: $\text{sets } M = \text{sets (subprob-algebra } N)$ **and** $A:$
 $A \in \text{sets } N$

shows *emeasure (join M) A* = $(\int^+ M'. \text{emeasure } M' A \partial M)$

proof (*rule emeasure-measure-of[OF join-def]*)

show *countably-additive (sets (join M))* $(\lambda B. \int^+ M'. \text{emeasure } M' B \partial M)$

proof (*rule countably-additiveI*)

fix $A :: \text{nat} \Rightarrow$ 'a set **assume** $A: \text{range } A \subseteq \text{sets (join } M)$ *disjoint-family* A

have $(\sum i. \int^+ M'. \text{emeasure } M' (A i) \partial M) = (\int^+ M'. (\sum i. \text{emeasure } M' (A$
 $i)) \partial M)$

using A **by** (*subst nn-integral-suminf*) (*auto simp: measurable-emeasure-subprob-algebra*)

also have $\dots = (\int^+ M'. \text{emeasure } M' (\bigcup i. A i) \partial M)$

proof (*rule nn-integral-cong*)

fix M' **assume** $M' \in \text{space } M$

then show $(\sum i. \text{emeasure } M' (A i)) = \text{emeasure } M' (\bigcup i. A i)$

using A *sets-eq-imp-space-eq[OF M]* **by** (*simp add: suminf-emeasure*

space-subprob-algebra)

qed

finally show $(\sum i. \int^+ M'. \text{emeasure } M' (A i) \partial M) = (\int^+ M'. \text{emeasure } M'$
 $(\bigcup i. A i) \partial M)$.

qed

qed (*auto simp: A sets.space-closed positive-def*)

lemma *measurable-join*:

join $\in \text{measurable (subprob-algebra (subprob-algebra } N)) (\text{subprob-algebra } N)$

proof (*cases space N \neq {}, rule measurable-subprob-algebra*)

fix A **assume** $A \in \text{sets } N$

let $?B = \text{borel-measurable (subprob-algebra (subprob-algebra } N))$

have $(\lambda M'. \text{emeasure (join } M') A) \in ?B \longleftrightarrow (\lambda M'. (\int^+ M''. \text{emeasure } M'' A$
 $\partial M')) \in ?B$

proof (*rule measurable-cong*)

fix M' **assume** $M' \in \text{space (subprob-algebra (subprob-algebra } N))$

then show $\text{emeasure (join } M') A = (\int^+ M''. \text{emeasure } M'' A \partial M')$

by (*intro emeasure-join*) (*auto simp: space-subprob-algebra $\langle A \in \text{sets } N \rangle$*)

```

qed
also have  $(\lambda M'. \int^+ M''. \text{emeasure } M'' A \partial M') \in ?B$ 
  using measurable-emeasure-subprob-algebra[OF  $\langle A \in \text{sets } N \rangle$ ]
  by (rule nn-integral-measurable-subprob-algebra)
finally show  $(\lambda M'. \text{emeasure } (\text{join } M') A) \in \text{borel-measurable } (\text{subprob-algebra } N)$ 
  (subprob-algebra N) .
next
assume [simp]: space N  $\neq \{\}$ 
fix M assume M: M  $\in \text{space } (\text{subprob-algebra } (\text{subprob-algebra } N))$ 
then have  $(\int^+ M'. \text{emeasure } M' (\text{space } N) \partial M) \leq (\int^+ M'. 1 \partial M)$ 
  apply (intro nn-integral-mono)
  apply (auto simp: space-subprob-algebra
    dest!: sets-eq-imp-space-eq subprob-space.emeasure-space-le-1)
done
with M show subprob-space (join M)
  by (intro subprob-spaceI)
  (auto simp: emeasure-join space-subprob-algebra M dest: subprob-space.emeasure-space-le-1)
next
assume  $\neg(\text{space } N \neq \{\})$ 
thus ?thesis by (simp add: measurable-empty-iff space-subprob-algebra-empty-iff)
qed (auto simp: space-subprob-algebra)

```

lemma *nn-integral-join*:

```

assumes f: f  $\in \text{borel-measurable } N$ 
  and M[measurable-cong]: sets M = sets (subprob-algebra N)
shows  $(\int^+ x. f x \partial \text{join } M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$ 
  using f
proof induct
  case (cong f g)
  moreover have  $\text{integral}^N (\text{join } M) f = \text{integral}^N (\text{join } M) g$ 
    by (intro nn-integral-cong cong) (simp add: M)
  moreover from M have  $(\int^+ M'. \text{integral}^N M' f \partial M) = (\int^+ M'. \text{integral}^N$ 
    M' g  $\partial M)$ 
    by (intro nn-integral-cong cong)
    (auto simp add: space-subprob-algebra dest!: sets-eq-imp-space-eq)
  ultimately show ?case
    by simp
next
  case (set A)
  with M have  $(\int^+ M'. \text{integral}^N M' (\text{indicator } A) \partial M) = (\int^+ M'. \text{emeasure}$ 
    M' A  $\partial M)$ 
    by (intro nn-integral-cong nn-integral-indicator)
    (auto simp: space-subprob-algebra dest!: sets-eq-imp-space-eq)
  with set show ?case
    using M by (simp add: emeasure-join)
next
  case (mult f c)
  have  $(\int^+ M'. \int^+ x. c * f x \partial M' \partial M) = (\int^+ M'. c * \int^+ x. f x \partial M' \partial M)$ 
    using mult M M [THEN sets-eq-imp-space-eq]

```

by (*intro nn-integral-cong nn-integral-cmult*) (*auto simp add: space-subprob-algebra*)
also have $\dots = c * (\int^+ M'. \int^+ x. f x \partial M' \partial M)$
using *nn-integral-measurable-subprob-algebra*[*OF mult(2)*]
by (*intro nn-integral-cmult mult*) (*simp add: M*)
also have $\dots = c * (\text{integral}^N (\text{join } M) f)$
by (*simp add: mult*)
also have $\dots = (\int^+ x. c * f x \partial \text{join } M)$
using *mult(2,3)* **by** (*intro nn-integral-cmult[symmetric] mult*) (*simp add: M cong: measurable-cong-sets*)
finally show ?*case* **by** *simp*
next
case (*add f g*)
have $(\int^+ M'. \int^+ x. f x + g x \partial M' \partial M) = (\int^+ M'. (\int^+ x. f x \partial M') + (\int^+ x. g x \partial M') \partial M)$
using *add M M*[*THEN sets-eq-imp-space-eq*]
by (*intro nn-integral-cong nn-integral-add*) (*auto simp add: space-subprob-algebra*)
also have $\dots = (\int^+ M'. \int^+ x. f x \partial M' \partial M) + (\int^+ M'. \int^+ x. g x \partial M' \partial M)$
using *nn-integral-measurable-subprob-algebra*[*OF add(1)*]
using *nn-integral-measurable-subprob-algebra*[*OF add(4)*]
by (*intro nn-integral-add add*) (*simp-all add: M*)
also have $\dots = (\text{integral}^N (\text{join } M) f) + (\text{integral}^N (\text{join } M) g)$
by (*simp add: add*)
also have $\dots = (\int^+ x. f x + g x \partial \text{join } M)$
using *add* **by** (*intro nn-integral-add[symmetric] add*) (*simp-all add: M cong: measurable-cong-sets*)
finally show ?*case* **by** (*simp add: ac-simps*)
next
case (*seq F*)
have $(\int^+ M'. \int^+ x. (\text{SUP } i. F i) x \partial M' \partial M) = (\int^+ M'. (\text{SUP } i. \int^+ x. F i x \partial M') \partial M)$
using *seq M M*[*THEN sets-eq-imp-space-eq*] **unfolding** *SUP-apply*
by (*intro nn-integral-cong nn-integral-monotone-convergence-SUP*)
(*auto simp add: space-subprob-algebra*)
also have $\dots = (\text{SUP } i. \int^+ M'. \int^+ x. F i x \partial M' \partial M)$
using *nn-integral-measurable-subprob-algebra*[*OF seq(1)*] *seq*
by (*intro nn-integral-monotone-convergence-SUP*)
(*simp-all add: M incseq-nn-integral incseq-def le-fun-def nn-integral-mono*)
also have $\dots = (\text{SUP } i. \text{integral}^N (\text{join } M) (F i))$
by (*simp add: seq*)
also have $\dots = (\int^+ x. (\text{SUP } i. F i x) \partial \text{join } M)$
using *seq* **by** (*intro nn-integral-monotone-convergence-SUP[symmetric] seq*)
(*simp-all add: M cong: measurable-cong-sets*)
finally show ?*case* **by** (*simp add: ac-simps image-comp*)
qed

lemma *measurable-join1*:

$\llbracket f \in \text{measurable } N K; \text{ sets } M = \text{sets } (\text{subprob-algebra } N) \rrbracket$

$\implies f \in \text{measurable } (\text{join } M) K$

by(*simp add: measurable-def*)

lemma

fixes $f :: - \Rightarrow \text{real}$
assumes $f\text{-measurable}$ [*measurable*]: $f \in \text{borel-measurable } N$
and $f\text{-bounded}$: $\bigwedge x. x \in \text{space } N \implies |f x| \leq B$
and M [*measurable-cong*]: $\text{sets } M = \text{sets (subprob-algebra } N)$
and fin : *finite-measure* M
and $M\text{-bounded}$: $\text{AE } M' \text{ in } M. \text{emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
shows integrable-join : $\text{integrable (join } M) f$ (**is** *?integrable*)
and integral-join : $\text{integral}^L (\text{join } M) f = \int M'. \text{integral}^L M' f \partial M$ (**is** *?integral*)
proof(*case-tac* [!]) $\text{space } N = \{\}$
assume *: $\text{space } N = \{\}$
show *?integrable*
using $M * \text{by}$ (*simp add: real-integrable-def measurable-def nn-integral-empty*)
have $(\int M'. \text{integral}^L M' f \partial M) = (\int M'. 0 \partial M)$
proof(*rule Bochner-Integration.integral-cong*)
fix M'
assume $M' \in \text{space } M$
with $\text{sets-eq-imp-space-eq}[OF M]$ **have** $\text{space } M' = \text{space } N$
by(*auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq*)
with * **show** $(\int x. f x \partial M') = 0$ **by**(*simp add: Bochner-Integration.integral-empty*)
qed *simp*
then show *?integral*
using $M * \text{by}$ (*simp add: Bochner-Integration.integral-empty*)
next
assume *: $\text{space } N \neq \{\}$

from * **have** B [*simp*]: $0 \leq B$ **by**(*auto dest: f-bounded*)

have [*measurable*]: $f \in \text{borel-measurable (join } M)$ **using** $f\text{-measurable } M$
by(*rule measurable-join1*)

{ fix $f M'$
assume [*measurable*]: $f \in \text{borel-measurable } N$
and $f\text{-bounded}$: $\bigwedge x. x \in \text{space } N \implies f x \leq B$
and $M' \in \text{space } M$ $\text{emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
have $\text{AE } x \text{ in } M'. \text{ennreal } (f x) \leq \text{ennreal } B$
proof(*rule AE-I2*)
fix x
assume $x \in \text{space } M'$
with $\langle M' \in \text{space } M \rangle \text{sets-eq-imp-space-eq}[OF M]$
have $x \in \text{space } N$ **by**(*auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq*)
from $f\text{-bounded}[OF \text{ this}]$ **show** $\text{ennreal } (f x) \leq \text{ennreal } B$ **by** *simp*
qed
then have $(\int^+ x. \text{ennreal } (f x) \partial M') \leq (\int^+ x. \text{ennreal } B \partial M')$
by(*rule nn-integral-mono-AE*)
also have $\dots = \text{ennreal } B * \text{emeasure } M' (\text{space } M')$ **by**(*simp*)
also have $\dots \leq \text{ennreal } B * \text{ennreal } B'$ **by**(*rule mult-left-mono*)(*fact, simp*)
also have $\dots \leq \text{ennreal } B * \text{ennreal } |B'|$ **by**(*rule mult-left-mono*)(*simp-all*)

finally have $(\int^+ x. \text{ennreal } (f x) \partial M') \leq \text{ennreal } (B * |B'|)$ **by** (*simp add: ennreal-mult*) }

note *bounded1 = this*

have *bounded:*

$\bigwedge f. \llbracket f \in \text{borel-measurable } N; \bigwedge x. x \in \text{space } N \implies f x \leq B \rrbracket$
 $\implies (\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \text{top}$

proof –

fix *f*

assume [*measurable*]: $f \in \text{borel-measurable } N$

and *f-bounded*: $\bigwedge x. x \in \text{space } N \implies f x \leq B$

have $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) = (\int^+ M'. \int^+ x. \text{ennreal } (f x) \partial M' \partial M)$

by(*rule nn-integral-join[OF - M]*) *simp*

also have $\dots \leq \int^+ M'. B * |B'| \partial M$

using *bounded1*[*OF* $\langle f \in \text{borel-measurable } N \rangle$ *f-bounded*]

by(*rule nn-integral-mono-AE*[*OF* *AE-mp*[*OF* *M-bounded* *AE-I2*], *rule-format*])

also have $\dots = B * |B'| * \text{emeasure } M$ (*space M*) **by** *simp*

also have $\dots < \infty$

using *finite-measure.finite-emeasure-space*[*OF* *fin*]

by(*simp add: ennreal-mult-less-top less-top*)

finally show *?thesis f* **by** *simp*

qed

have *f-pos*: $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \infty$

and *f-neg*: $(\int^+ x. \text{ennreal } (- f x) \partial \text{join } M) \neq \infty$

using *f-bounded* **by**(*auto del: notI intro!*: *bounded simp add: abs-le-iff*)

show *?integrable* **using** *f-pos f-neg* **by**(*simp add: real-integrable-def*)

note [*measurable*] = *nn-integral-measurable-subprob-algebra*

have *int-f*: $(\int^+ x. f x \partial \text{join } M) = \int^+ M'. \int^+ x. f x \partial M' \partial M$

by(*simp add: nn-integral-join*[*OF* - *M*])

have *int-mf*: $(\int^+ x. - f x \partial \text{join } M) = (\int^+ M'. \int^+ x. - f x \partial M' \partial M)$

by(*simp add: nn-integral-join*[*OF* - *M*])

have *pos-finite*: $\text{AE } M' \text{ in } M. (\int^+ x. f x \partial M') \neq \infty$

using *AE-space M-bounded*

proof *eventually-elim*

fix *M'* **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq \text{ennreal } B'$

then have $(\int^+ x. \text{ennreal } (f x) \partial M') \leq \text{ennreal } (B * |B'|)$

using *f-measurable* **by**(*auto intro!*: *bounded1 dest: f-bounded*)

then show $(\int^+ x. \text{ennreal } (f x) \partial M') \neq \infty$

by (*auto simp: top-unique*)

qed

hence [*simp*]: $(\int^+ M'. \text{ennreal } (\text{enn2real } (\int^+ x. f x \partial M')) \partial M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$

by (*rule nn-integral-cong-AE*[*OF* *AE-mp*]) (*simp add: less-top*)

from *f-pos* **have** [*simp*]: *integrable* *M* $(\lambda M'. \text{enn2real } (\int^+ x. f x \partial M'))$

by(*simp add: int-f real-integrable-def nn-integral-0-iff-AE*[*THEN iffD2*] *en-*

nreal-neg enn2real-nonneg)

have *neg-finite*: $AE\ M'$ in M . $(\int^+ x. - f\ x\ \partial M') \neq \infty$
using *AE-space M-bounded*
proof *eventually-elim*
fix M' **assume** $M' \in \text{space } M\ \text{emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
then have $(\int^+ x. \text{ennreal } (- f\ x)\ \partial M') \leq \text{ennreal } (B * |B'|)$
using *f-measurable by(auto intro!: bounded1 dest: f-bounded)*
then show $(\int^+ x. \text{ennreal } (- f\ x)\ \partial M') \neq \infty$
by *(auto simp: top-unique)*
qed
hence [*simp*]: $(\int^+ M'. \text{ennreal } (\text{enn2real } (\int^+ x. - f\ x\ \partial M'))\ \partial M) = (\int^+ M'. \int^+ x. - f\ x\ \partial M'\ \partial M)$
by *(rule nn-integral-cong-AE[OF AE-mp]) (simp add: less-top)*
from *f-neg* **have** [*simp*]: $\text{integrable } M\ (\lambda M'. \text{enn2real } (\int^+ x. - f\ x\ \partial M'))$
by *(simp add: int-mf real-integrable-def nn-integral-0-iff-AE[THEN iffD2] ennreal-neg enn2real-nonneg)*

have $(\int x. f\ x\ \partial \text{join } M) = \text{enn2real } (\int^+ N. \int^+ x. f\ x\ \partial N\ \partial M) - \text{enn2real } (\int^+ N. \int^+ x. - f\ x\ \partial N\ \partial M)$
unfolding *real-lebesgue-integral-def[OF <?integrable>]* **by** *(simp add: nn-integral-join[OF - M])*
also have $\dots = (\int N. \text{enn2real } (\int^+ x. f\ x\ \partial N)\ \partial M) - (\int N. \text{enn2real } (\int^+ x. - f\ x\ \partial N)\ \partial M)$
using *pos-finite neg-finite by (subst (1 2) integral-eq-nn-integral) (auto simp: enn2real-nonneg)*
also have $\dots = (\int N. \text{enn2real } (\int^+ x. f\ x\ \partial N) - \text{enn2real } (\int^+ x. - f\ x\ \partial N))\ \partial M$
by *simp*
also have $\dots = \int M'. \int x. f\ x\ \partial M'\ \partial M$
proof *(rule integral-cong-AE)*
show $AE\ x$ in M .
 $\text{enn2real } (\int^+ x. \text{ennreal } (f\ x)\ \partial x) - \text{enn2real } (\int^+ x. \text{ennreal } (- f\ x)\ \partial x) = \text{integral}^L\ x\ f$
using *AE-space M-bounded*
proof *eventually-elim*
fix M' **assume** $M' \in \text{space } M\ \text{emeasure } M' (\text{space } M') \leq B'$
then interpret *subprob-space M'*
by *(auto simp: M[THEN sets-eq-imp-space-eq] space-subprob-algebra)*

from $\langle M' \in \text{space } M \rangle\ \text{sets-eq-imp-space-eq}[OF\ M]$
have [*measurable-cong*]: $\text{sets } M' = \text{sets } N$ **by** *(simp add: space-subprob-algebra)*
hence [*simp*]: $\text{space } M' = \text{space } N$ **by** *(rule sets-eq-imp-space-eq)*
have *integrable M' f*
by *(rule integrable-const-bound[where B=B])(auto simp add: f-bounded)*
then show $\text{enn2real } (\int^+ x. f\ x\ \partial M') - \text{enn2real } (\int^+ x. - f\ x\ \partial M') = \int x. f\ x\ \partial M'$
by *(simp add: real-lebesgue-integral-def)*
qed

qed *simp-all*
finally show *?integral by simp*
qed

lemma *join-assoc*:

assumes $M[\text{measurable-cong}]$: *sets M = sets (subprob-algebra (subprob-algebra N))*
shows $\text{join (distr M (subprob-algebra N) join) = join (join M)}$
proof (*rule measure-eqI*)
fix A **assume** $A \in \text{sets (join (distr M (subprob-algebra N) join))}$
then have A : $A \in \text{sets N}$ **by** *simp*
show $\text{emeasure (join (distr M (subprob-algebra N) join)) A = emeasure (join (join M)) A}$
using *measurable-join[of N]*
by (*auto simp: M A nn-integral-distr emeasure-join measurable-emeasure-subprob-algebra sets-eq-imp-space-eq[OF M] space-subprob-algebra nn-integral-join[OF - M]*)
intro!: nn-integral-cong emeasure-join
qed (*simp add: M*)

lemma *join-return*:

assumes sets M = sets N **and** *subprob-space M*
shows $\text{join (return (subprob-algebra N) M) = M}$
by (*rule measure-eqI*)
(simp-all add: emeasure-join space-subprob-algebra measurable-emeasure-subprob-algebra nn-integral-return assms)

lemma *join-return'*:

assumes sets N = sets M
shows $\text{join (distr M (subprob-algebra N) (return N)) = M}$
apply (*rule measure-eqI*)
apply (*simp add: assms*)
apply (*subgoal-tac return N \in measurable M (subprob-algebra N)*)
apply (*simp add: emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra assms*)
apply (*subst measurable-cong-sets, rule assms[symmetric], rule refl, rule return-measurable*)
done

lemma *join-distr-distr*:

fixes $f :: 'a \Rightarrow 'b$ **and** $M :: 'a \text{ measure measure}$ **and** $N :: 'b \text{ measure}$
assumes $\text{sets M = sets (subprob-algebra R)}$ **and** $f \in \text{measurable R N}$
shows $\text{join (distr M (subprob-algebra N) (\lambda M. distr M N f)) = distr (join M) N f}$ (*is ?r = ?l*)
proof (*rule measure-eqI*)
fix A **assume** $A \in \text{sets ?r}$
hence $A\text{-in-N}$: $A \in \text{sets N}$ **by** *simp*

from *assms* **have** $f \in \text{measurable (join M) N}$
by (*simp cong: measurable-cong-sets*)

moreover from *assms* **and** *A-in-N* **have** $f - 'A \cap \text{space } R \in \text{sets } R$
by (*intro measurable-sets*) *simp-all*
ultimately have $\text{emeasure } (\text{distr } (\text{join } M) N f) A = \int^+ M'. \text{emeasure } M' (f - 'A \cap \text{space } R) \partial M$
by (*simp-all add: A-in-N emeasure-distr emeasure-join assms*)

also have $\dots = \int^+ x. \text{emeasure } (\text{distr } x N f) A \partial M$ **using** *A-in-N*
proof (*intro nn-integral-cong, subst emeasure-distr*)
fix M' **assume** $M' \in \text{space } M$
from *assms* **have** $\text{space } M = \text{space } (\text{subprob-algebra } R)$
using *sets-eq-imp-space-eq* **by** *blast*
with $\langle M' \in \text{space } M \rangle$ **have** [*simp*]: $\text{sets } M' = \text{sets } R$ **using** *space-subprob-algebra*
by *blast*
show $f \in \text{measurable } M' N$ **by** (*simp cong: measurable-cong-sets add: assms*)
have $\text{space } M' = \text{space } R$ **by** (*rule sets-eq-imp-space-eq*) *simp*
thus $\text{emeasure } M' (f - 'A \cap \text{space } R) = \text{emeasure } M' (f - 'A \cap \text{space } M')$ **by**
simp
qed

also have $(\lambda M. \text{distr } M N f) \in \text{measurable } M (\text{subprob-algebra } N)$
by (*simp cong: measurable-cong-sets add: assms measurable-distr*)
hence $(\int^+ x. \text{emeasure } (\text{distr } x N f) A \partial M) =$
 $\text{emeasure } (\text{join } (\text{distr } M (\text{subprob-algebra } N) (\lambda M. \text{distr } M N f))) A$
by (*simp-all add: emeasure-join assms A-in-N nn-integral-distr measurable-emeasure-subprob-algebra*)
finally show $\text{emeasure } ?r A = \text{emeasure } ?l A ..$
qed *simp*

definition *bind* :: $'a \text{ measure} \Rightarrow ('a \Rightarrow 'b \text{ measure}) \Rightarrow 'b \text{ measure}$ **where**
 $\text{bind } M f = (\text{if } \text{space } M = \{\} \text{ then } \text{count-space } \{\} \text{ else}$
 $\text{join } (\text{distr } M (\text{subprob-algebra } (f (\text{SOME } x. x \in \text{space } M)))) f)$

adhoc-overloading *Monad-Syntax.bind* *bind*

lemma *bind-empty*:
 $\text{space } M = \{\} \implies \text{bind } M f = \text{count-space } \{\}$
by (*simp add: bind-def*)

lemma *bind-nonempty*:
 $\text{space } M \neq \{\} \implies \text{bind } M f = \text{join } (\text{distr } M (\text{subprob-algebra } (f (\text{SOME } x. x \in \text{space } M)))) f)$
by (*simp add: bind-def*)

lemma *sets-bind-empty*: $\text{sets } M = \{\} \implies \text{sets } (\text{bind } M f) = \{\{\}\}$
by (*auto simp: bind-def*)

lemma *space-bind-empty*: $\text{space } M = \{\} \implies \text{space } (\text{bind } M f) = \{\}$
by (*simp add: bind-def*)

lemma *sets-bind*[*simp*, *measurable-cong*]:
assumes $f: \bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } N$ **and** $M: \text{space } M \neq \{\}$
shows $\text{sets } (\text{bind } M f) = \text{sets } N$
using f [of *SOME* $x. x \in \text{space } M$] **by** (*simp add: bind-nonempty M some-in-eq*)

lemma *space-bind*[*simp*]:
assumes $\bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } N$ **and** $\text{space } M \neq \{\}$
shows $\text{space } (\text{bind } M f) = \text{space } N$
using *assms* **by** (*intro sets-eq-imp-space-eq sets-bind*)

lemma *bind-cong-All*:
assumes $\forall x \in \text{space } M. f x = g x$
shows $\text{bind } M f = \text{bind } M g$
proof (*cases space M = \{\}*)
assume $\text{space } M \neq \{\}$
hence (*SOME* $x. x \in \text{space } M$) $\in \text{space } M$ **by** (*rule-tac someI-ex*) *blast*
with *assms* **have** f (*SOME* $x. x \in \text{space } M$) $= g$ (*SOME* $x. x \in \text{space } M$) **by**
blast
with $\langle \text{space } M \neq \{\} \rangle$ **and** *assms* **show** *?thesis* **by** (*simp add: bind-nonempty*
cong: distr-cong)
qed (*simp add: bind-empty*)

lemma *bind-cong*:
 $M = N \implies (\bigwedge x. x \in \text{space } M \implies f x = g x) \implies \text{bind } M f = \text{bind } N g$
using *bind-cong-All*[of $M f g$] **by** *auto*

lemma *bind-nonempty'*:
assumes $f \in \text{measurable } M$ (*subprob-algebra* N) $x \in \text{space } M$
shows $\text{bind } M f = \text{join } (\text{distr } M$ (*subprob-algebra* N) f)
using *assms*
apply (*subst bind-nonempty, blast*)
apply (*subst subprob-algebra-cong[OF sets-kernel[OF assms(1) someI-ex]]*, *blast*)
apply (*simp add: subprob-algebra-cong[OF sets-kernel[OF assms]]*)
done

lemma *bind-nonempty''*:
assumes $f \in \text{measurable } M$ (*subprob-algebra* N) $\text{space } M \neq \{\}$
shows $\text{bind } M f = \text{join } (\text{distr } M$ (*subprob-algebra* N) f)
using *assms* **by** (*auto intro: bind-nonempty'*)

lemma *emeasure-bind*:
 $\llbracket \text{space } M \neq \{\}; f \in \text{measurable } M$ (*subprob-algebra* N); $X \in \text{sets } N \rrbracket$
 $\implies \text{emeasure } (M \ggg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$
by (*simp add: bind-nonempty'' emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra*)

lemma *nn-integral-bind*:
assumes $f: f \in \text{borel-measurable } B$
assumes $N: N \in \text{measurable } M$ (*subprob-algebra* B)
shows $(\int^+ x. f x \partial(M \ggg N)) = (\int^+ x. \int^+ y. f y \partial N x \partial M)$

proof cases

assume M : *space* $M \neq \{\}$ **show** *?thesis*
unfolding *bind-nonempty'*[$OF\ N\ M$] *nn-integral-join*[$OF\ f\ sets-distr$]
by (*rule nn-integral-distr*[$OF\ N$])
(*simp add: f nn-integral-measurable-subprob-algebra*)
qed (*simp add: bind-empty space-empty[of M] nn-integral-count-space*)

lemma *AE-bind*:

assumes N [*measurable*]: $N \in measurable\ M$ (*subprob-algebra* B)
assumes P [*measurable*]: *Measurable.pred* $B\ P$
shows ($AE\ x\ in\ M \ggg N. P\ x$) \longleftrightarrow ($AE\ x\ in\ M. AE\ y\ in\ N\ x. P\ y$)

proof cases

assume M : *space* $M = \{\}$ **show** *?thesis*
unfolding *bind-empty*[$OF\ M$] **unfolding** *space-empty*[$OF\ M$] **by** (*simp add: AE-count-space*)

next

assume M : *space* $M \neq \{\}$
note *sets-kernel*[$OF\ N, simp$]
have $*$: $(\int^+ x. indicator\ \{x. \neg P\ x\}\ x\ \partial(M \ggg N)) = (\int^+ x. indicator\ \{x \in space\ B. \neg P\ x\}\ x\ \partial(M \ggg N))$
by (*intro nn-integral-cong*) (*simp add: space-bind[OF - M] split: split-indicator*)

have ($AE\ x\ in\ M \ggg N. P\ x$) \longleftrightarrow $(\int^+ x. integral^N\ (N\ x)\ (indicator\ \{x \in space\ B. \neg P\ x\})\ \partial M) = 0$

by (*simp add: AE-iff- nn -integral sets-bind[OF - M] space-bind[OF - M] * nn-integral-bind[where B=B]*)

del: nn-integral-indicator)

also have $\dots = (AE\ x\ in\ M. AE\ y\ in\ N\ x. P\ y)$

apply (*subst nn-integral-0-iff-AE*)

apply (*rule measurable-compose[OF N nn-integral-measurable-subprob-algebra]*)

apply *measurable*

apply (*intro eventually-subst AE-I2*)

apply (*auto simp add: subprob-measurableD(1)[OF N]*)

intro!: AE-iff-measurable[symmetric])

done

finally show *?thesis* .

qed

lemma *measurable-bind'*:

assumes $M1$: $f \in measurable\ M$ (*subprob-algebra* N) **and**

$M2$: *case-prod* $g \in measurable\ (M \otimes_M N)$ (*subprob-algebra* R)

shows $(\lambda x. bind\ (f\ x)\ (g\ x)) \in measurable\ M$ (*subprob-algebra* R)

proof (*subst measurable-cong*)

fix x **assume** x -*in-M*: $x \in space\ M$

with *assms have space* $(f\ x) \neq \{\}$

by (*blast dest: subprob-space-kernel subprob-space.subprob-not-empty*)

moreover from $M2$ x -*in-M* **have** $g\ x \in measurable\ (f\ x)$ (*subprob-algebra* R)

by (*subst measurable-cong-sets[OF sets-kernel[OF M1 x-in-M] refl]*)

(*auto dest: measurable-Pair2*)

ultimately show $\text{bind } (f \ x) \ (g \ x) = \text{join } (\text{distr } (f \ x) \ (\text{subprob-algebra } R) \ (g \ x))$
by (*simp-all add: bind-nonempty'*)
next
show $(\lambda w. \text{join } (\text{distr } (f \ w) \ (\text{subprob-algebra } R) \ (g \ w))) \in \text{measurable } M \ (\text{subprob-algebra } R)$
apply (*rule measurable-compose[OF - measurable-join]*)
apply (*rule measurable-distr2[OF M2 M1]*)
done
qed

lemma *measurable-bind*[*measurable (raw)*]:
assumes $M1: f \in \text{measurable } M \ (\text{subprob-algebra } N)$ **and**
 $M2: (\lambda x. g \ (\text{fst } x) \ (\text{snd } x)) \in \text{measurable } (M \otimes_M N) \ (\text{subprob-algebra } R)$
shows $(\lambda x. \text{bind } (f \ x) \ (g \ x)) \in \text{measurable } M \ (\text{subprob-algebra } R)$
using *assms* **by** (*auto intro: measurable-bind' simp: measurable-split-conv*)

lemma *measurable-bind2*:
assumes $f \in \text{measurable } M \ (\text{subprob-algebra } N)$ **and** $g \in \text{measurable } N \ (\text{subprob-algebra } R)$
shows $(\lambda x. \text{bind } (f \ x) \ g) \in \text{measurable } M \ (\text{subprob-algebra } R)$
using *assms* **by** (*intro measurable-bind' measurable-const auto*)

lemma *subprob-space-bind*:
assumes *subprob-space* $M \ f \in \text{measurable } M \ (\text{subprob-algebra } N)$
shows *subprob-space* $(M \ggg f)$
proof (*rule subprob-space-kernel*[*of* $\lambda x. x \ggg f$])
show $(\lambda x. x \ggg f) \in \text{measurable } (\text{subprob-algebra } M) \ (\text{subprob-algebra } N)$
by (*rule measurable-bind, rule measurable-ident-sets, rule refl,*
rule measurable-compose[*OF measurable-snd assms(2)*])
from *assms(1)* **show** $M \in \text{space } (\text{subprob-algebra } M)$
by (*simp add: space-subprob-algebra*)
qed

lemma
fixes $f :: - \Rightarrow \text{real}$
assumes *f-measurable* [*measurable*]: $f \in \text{borel-measurable } K$
and *f-bounded*: $\bigwedge x. x \in \text{space } K \implies |f \ x| \leq B$
and N [*measurable*]: $N \in \text{measurable } M \ (\text{subprob-algebra } K)$
and *fin*: *finite-measure* M
and *M-bounded*: $\forall E \ x \ \text{in } M. \text{emeasure } (N \ x) \ (\text{space } (N \ x)) \leq \text{ennreal } B'$
shows *integrable-bind*: *integrable* $(\text{bind } M \ N) \ f$ (**is** *?integrable*)
and *integral-bind*: $\text{integral}^L \ (\text{bind } M \ N) \ f = \int x. \text{integral}^L \ (N \ x) \ f \ \partial M$ (**is** *?integral*)
proof(*case-tac* [!] *space* $M = \{\}$)
assume [*simp*]: *space* $M \neq \{\}$
interpret *finite-measure* M **by**(*rule fin*)

have *integrable* $(\text{join } (\text{distr } M \ (\text{subprob-algebra } K) \ N)) \ f$
using *f-measurable f-bounded*

by(rule *integrable-join*[**where** $B'=B'$])(*simp-all add: finite-measure-distr AE-distr-iff*
M-bounded)
then show *?integrable* **by**(*simp add: bind-nonempty''*[**where** $N=K$])

have $\text{integral}^L (\text{join} (\text{distr } M (\text{subprob-algebra } K) N)) f = \int M'. \text{integral}^L M'$
 $f \partial \text{distr } M (\text{subprob-algebra } K) N$
using *f-measurable f-bounded*
by(rule *integral-join*[**where** $B'=B'$])(*simp-all add: finite-measure-distr AE-distr-iff*
M-bounded)
also have $\dots = \int x. \text{integral}^L (N x) f \partial M$
by(rule *integral-distr*)(*simp-all add: integral-measurable-subprob-algebra*[*OF -*])
finally show *?integral* **by**(*simp add: bind-nonempty''*[**where** $N=K$])
qed(*simp-all add: bind-def integrable-count-space lebesgue-integral-count-space-finite*
Bochner-Integration.integral-empty)

lemma (*in prob-space*) *prob-space-bind*:
assumes *ae: AE x in M. prob-space (N x)*
and $N[\text{measurable}]: N \in \text{measurable } M (\text{subprob-algebra } S)$
shows *prob-space (M \ggg N)*
proof
have $\text{emeasure } (M \ggg N) (\text{space } (M \ggg N)) = (\int^+ x. \text{emeasure } (N x) (\text{space}$
 $(N x)) \partial M)$
by (*subst emeasure-bind*[**where** $N=S$])
(auto simp: not-empty space-bind[*OF sets-kernel*] *subprob-measurableD*[*OF*
 N] *intro!: nn-integral-cong*)
also have $\dots = (\int^+ x. 1 \partial M)$
using *ae* **by** (*intro nn-integral-cong-AE, eventually-elim*) (*rule prob-space.emeasure-space-1*)
finally show $\text{emeasure } (M \ggg N) (\text{space } (M \ggg N)) = 1$
by (*simp add: emeasure-space-1*)
qed

lemma (*in subprob-space*) *bind-in-space*:
 $A \in \text{measurable } M (\text{subprob-algebra } N) \implies (M \ggg A) \in \text{space } (\text{subprob-algebra}$
 $N)$
by (*auto simp add: space-subprob-algebra subprob-not-empty sets-kernel intro!:*
subprob-space-bind)
unfold-locales

lemma (*in subprob-space*) *measure-bind*:
assumes $f: f \in \text{measurable } M (\text{subprob-algebra } N)$ **and** $X: X \in \text{sets } N$
shows $\text{measure } (M \ggg f) X = \int x. \text{measure } (f x) X \partial M$
proof –
interpret $Mf: \text{subprob-space } M \ggg f$
by (*rule subprob-space-bind*[*OF - f*]) *unfold-locales*

{ fix x **assume** $x \in \text{space } M$
from f [*THEN measurable-space, OF this*] **interpret** *subprob-space* $f x$
by (*simp add: space-subprob-algebra*)
have $\text{emeasure } (f x) X = \text{ennreal } (\text{measure } (f x) X)$ $\text{measure } (f x) X \leq 1$

```

    by (auto simp: emeasure-eq-measure subprob-measure-le-1) }
  note this[simp]

  have emeasure (M ≫ f) X = ∫+x. emeasure (f x) X ∂M
    using subprob-not-empty f X by (rule emeasure-bind)
  also have ... = ∫+x. ennreal (measure (f x) X) ∂M
    by (intro nn-integral-cong) simp
  also have ... = ∫ x. measure (f x) X ∂M
    by (intro nn-integral-eq-integral integrable-const-bound[where B=1]
        measure-measurable-subprob-algebra2[OF f] pair-measureI X)
        (auto simp: measure-nonneg)
  finally show ?thesis
    by (simp add: Mf.emeasure-eq-measure measure-nonneg integral-nonneg)
qed

```

lemma *emeasure-bind-const*:

```

  space M ≠ {} ⇒ X ∈ sets N ⇒ subprob-space N ⇒
    emeasure (M ≫ (λx. N)) X = emeasure N X * emeasure M (space M)
  by (simp add: bind-nonempty emeasure-join nn-integral-distr
      space-subprob-algebra measurable-emeasure-subprob-algebra)

```

lemma *emeasure-bind-const'*:

```

  assumes subprob-space M subprob-space N
  shows emeasure (M ≫ (λx. N)) X = emeasure N X * emeasure M (space M)
  using assms
  proof (case-tac X ∈ sets N)
    fix X assume X ∈ sets N
    thus emeasure (M ≫ (λx. N)) X = emeasure N X * emeasure M (space M)
  using assms
    by (subst emeasure-bind-const)
    (simp-all add: subprob-space.subprob-not-empty subprob-space.emeasure-space-le-1)
  next
    fix X assume X ∉ sets N
    with assms show emeasure (M ≫ (λx. N)) X = emeasure N X * emeasure M
      (space M)
    by (simp add: sets-bind[of - - N] subprob-space.subprob-not-empty
        space-subprob-algebra emeasure-notin-sets)
  qed

```

lemma *emeasure-bind-const-prob-space*:

```

  assumes prob-space M subprob-space N
  shows emeasure (M ≫ (λx. N)) X = emeasure N X
  using assms by (simp add: emeasure-bind-const' prob-space-imp-subprob-space
      prob-space.emeasure-space-1)

```

lemma *bind-return*:

```

  assumes f ∈ measurable M (subprob-algebra N) and x ∈ space M
  shows bind (return M x) f = f x
  using sets-kernel[OF assms] assms

```


by (*simp-all add: distr-return join-return subprob-space-kernel bind-nonempty'*
cong: subprob-algebra-cong)

lemma *bind-return'*:

shows $\text{bind } M (\text{return } M) = M$

by (*cases space M = {}*)

(*simp-all add: bind-empty space-empty[symmetric] bind-nonempty join-return'*
cong: subprob-algebra-cong)

lemma *distr-bind*:

assumes $N: N \in \text{measurable } M (\text{subprob-algebra } K) \text{ space } M \neq \{\}$

assumes $f: f \in \text{measurable } K R$

shows $\text{distr } (M \gg N) R f = (M \gg (\lambda x. \text{distr } (N x) R f))$

unfolding *bind-nonempty''[OF N]*

apply (*subst bind-nonempty''[OF measurable-compose[OF N(1) measurable-distr]*
N(2)])

apply (*rule f*)

apply (*simp add: join-distr-distr[OF - f, symmetric]*)

apply (*subst distr-distr[OF measurable-distr, OF f N(1)]*)

apply (*simp add: comp-def*)

done

lemma *bind-distr*:

assumes $f[\text{measurable}]: f \in \text{measurable } M X$

assumes $N[\text{measurable}]: N \in \text{measurable } X (\text{subprob-algebra } K) \text{ and space } M \neq \{\}$

shows $(\text{distr } M X f \gg N) = (M \gg (\lambda x. N (f x)))$

proof –

have $\text{space } X \neq \{\} \text{ space } M \neq \{\}$

using $\langle \text{space } M \neq \{\} \rangle f[\text{THEN measurable-space}]$ **by** *auto*

then show *?thesis*

by (*simp add: bind-nonempty''[where N=K] distr-distr comp-def*)

qed

lemma *bind-count-space-singleton*:

assumes *subprob-space (f x)*

shows $\text{count-space } \{x\} \gg f = f x$

proof –

have $A: \bigwedge A. A \subseteq \{x\} \implies A = \{\} \vee A = \{x\}$ **by** *auto*

have $\text{count-space } \{x\} = \text{return } (\text{count-space } \{x\}) x$

by (*intro measure-eqI (auto dest: A)*)

also have $\dots \gg f = f x$

by (*subst bind-return[of - - f x] (auto simp: space-subprob-algebra assms)*)

finally show *?thesis* .

qed

lemma *restrict-space-bind*:

assumes $N: N \in \text{measurable } M (\text{subprob-algebra } K)$

assumes $\text{space } M \neq \{\}$

```

assumes  $X[simp]: X \in \text{sets } K \ X \neq \{\}$ 
shows  $\text{restrict-space } (\text{bind } M \ N) \ X = \text{bind } M \ (\lambda x. \text{restrict-space } (N \ x) \ X)$ 
proof (rule measure-eqI)
  note  $N\text{-sets} = \text{sets-bind}[OF \ \text{sets-kernel}[OF \ N] \ \text{assms}(2), \ \text{simp}]$ 
  note  $N\text{-space} = \text{sets-eq-imp-space-eq}[OF \ N\text{-sets}, \ \text{simp}]$ 
  show  $\text{sets } (\text{restrict-space } (\text{bind } M \ N) \ X) = \text{sets } (\text{bind } M \ (\lambda x. \text{restrict-space } (N \ x) \ X))$ 
    by (simp add: sets-restrict-space assms(2) sets-bind[OF sets-kernel[OF restrict-space-measurable[OF assms(4,3,1)]]])
  fix  $A$  assume  $A \in \text{sets } (\text{restrict-space } (M \gg N) \ X)$ 
  with  $X$  have  $A \in \text{sets } K \ A \subseteq X$ 
    by (auto simp: sets-restrict-space)
  then show  $\text{emeasure } (\text{restrict-space } (M \gg N) \ X) \ A = \text{emeasure } (M \gg (\lambda x. \text{restrict-space } (N \ x) \ X)) \ A$ 
    using assms
    apply (subst emeasure-restrict-space)
    apply (simp-all add: emeasure-bind[OF assms(2,1)])
    apply (subst emeasure-bind[OF - restrict-space-measurable[OF - - N]])
    apply (auto simp: sets-restrict-space emeasure-restrict-space space-subprob-algebra intro!: nn-integral-cong dest!: measurable-space)
  done
qed

```

lemma *bind-restrict-space*:

```

assumes  $A: A \cap \text{space } M \neq \{\}$   $A \cap \text{space } M \in \text{sets } M$ 
and  $f: f \in \text{measurable } (\text{restrict-space } M \ A) \ (\text{subprob-algebra } N)$ 
shows  $\text{restrict-space } M \ A \gg f = M \gg (\lambda x. \text{if } x \in A \ \text{then } f \ x \ \text{else } \text{null-measure } (f \ (\text{SOME } x. \ x \in A \ \wedge \ x \in \text{space } M)))$ 
  (is ?lhs = ?rhs is - = M >> ?f)
proof -
  let  $?P = \lambda x. \ x \in A \ \wedge \ x \in \text{space } M$ 
  let  $?x = \text{Eps } ?P$ 
  let  $?c = \text{null-measure } (f \ ?x)$ 
  from  $A$  have  $?P \ ?x$  by-(rule someI-ex, blast)
  hence  $?x \in \text{space } (\text{restrict-space } M \ A)$  by(simp add: space-restrict-space)
  with  $f$  have  $f \ ?x \in \text{space } (\text{subprob-algebra } N)$  by(rule measurable-space)
  hence  $\text{sps: subprob-space } (f \ ?x)$ 
    and  $\text{sets: sets } (f \ ?x) = \text{sets } N$ 
  by(simp-all add: space-subprob-algebra)
  have  $\text{space } (f \ ?x) \neq \{\}$  using sps by(rule subprob-space.subprob-not-empty)
  moreover have  $\text{sets } ?c = \text{sets } N$  by(simp add: null-measure-def)(simp add: sets)
  ultimately have  $c: ?c \in \text{space } (\text{subprob-algebra } N)$ 
    by(simp add: space-subprob-algebra subprob-space-null-measure)
  from  $f \ A \ c$  have  $f': ?f \in \text{measurable } M \ (\text{subprob-algebra } N)$ 
    by(simp add: measurable-restrict-space-iff)

from  $A$  have  $[simp]: \text{space } M \neq \{\}$  by blast

```

```

have ?lhs = join (distr (restrict-space M A) (subprob-algebra N) f)
  using assms by(simp add: space-restrict-space bind-nonempty'')
also have ... = join (distr M (subprob-algebra N) ?f)
  by(rule measure-eqI)(auto simp add: emeasure-join nn-integral-distr nn-integral-restrict-space
f f' A intro: nn-integral-cong)
also have ... = ?rhs using f' by(simp add: bind-nonempty'')
finally show ?thesis .
qed

```

```

lemma bind-const':  $\llbracket \text{prob-space } M; \text{subprob-space } N \rrbracket \implies M \ggg (\lambda x. N) = N$ 
by (intro measure-eqI)
  (simp-all add: space-subprob-algebra prob-space.not-empty emeasure-bind-const-prob-space)

```

```

lemma bind-return-distr:
   $\text{space } M \neq \{\}$   $\implies f \in \text{measurable } M N \implies \text{bind } M (\text{return } N \circ f) = \text{distr } M$ 
 $N f$ 
apply (simp add: bind-nonempty)
apply (subst subprob-algebra-cong)
apply (rule sets-return)
apply (subst distr-distr[symmetric])
apply (auto intro!: return-measurable simp: distr-distr[symmetric] join-return')
done

```

```

lemma bind-return-distr':
   $\text{space } M \neq \{\}$   $\implies f \in \text{measurable } M N \implies \text{bind } M (\lambda x. \text{return } N (f x)) = \text{distr}$ 
 $M N f$ 
using bind-return-distr[of M f N] by (simp add: comp-def)

```

```

lemma bind-assoc:
  fixes f :: 'a  $\Rightarrow$  'b measure and g :: 'b  $\Rightarrow$  'c measure
  assumes M1:  $f \in \text{measurable } M (\text{subprob-algebra } N)$  and M2:  $g \in \text{measurable}$ 
 $N (\text{subprob-algebra } R)$ 
  shows  $\text{bind } (\text{bind } M f) g = \text{bind } M (\lambda x. \text{bind } (f x) g)$ 
proof (cases space M = \{\})
  assume [simp]:  $\text{space } M \neq \{\}$ 
  from assms have [simp]:  $\text{space } N \neq \{\}$   $\text{space } R \neq \{\}$ 
    by (auto simp: measurable-empty-iff space-subprob-algebra-empty-iff)
  from assms have sets-fx[simp]:  $\bigwedge x. x \in \text{space } M \implies \text{sets } (f x) = \text{sets } N$ 
    by (simp add: sets-kernel)
  have ex-in:  $\bigwedge A. A \neq \{\} \implies \exists x. x \in A$  by blast
  note sets-some[simp] = sets-kernel[OF M1 someI-ex[OF ex-in[OF \langle space M \neq \{\} \rangle]]]]
     $\text{sets-kernel}$ [OF M2 someI-ex[OF ex-in[OF \langle space N \neq \{\} \rangle]]]]
  note space-some[simp] = sets-eq-imp-space-eq[OF this(1)] sets-eq-imp-space-eq[OF
this(2)]

```

```

have  $\text{bind } M (\lambda x. \text{bind } (f x) g) =$ 
   $\text{join } (\text{distr } M (\text{subprob-algebra } R) (\text{join} \circ (\lambda x. (\text{distr } x (\text{subprob-algebra } R)$ 
 $g)) \circ f))$ 

```

by (*simp add: sets-eq-imp-space-eq*[*OF sets-fx*] *bind-nonempty o-def*
cong: subprob-algebra-cong distr-cong)
also have *distr M (subprob-algebra R) (join ∘ (λx. (distr x (subprob-algebra R) g)) ∘ f) =*
distr (distr (distr M (subprob-algebra N) f)
(subprob-algebra (subprob-algebra R))
(λx. distr x (subprob-algebra R) g))
(subprob-algebra R) join
apply (*subst distr-distr,*
(blast intro: measurable-comp measurable-distr measurable-join M1
M2)+)+
apply (*simp add: o-assoc*)
done
also have *join ... = bind (bind M f) g*
by (*simp add: join-assoc join-distr-distr M2 bind-nonempty cong: subprob-algebra-cong*)
finally show *?thesis ..*
qed (*simp add: bind-empty*)

lemma *double-bind-assoc:*

assumes *Mg: g ∈ measurable N (subprob-algebra N')*
assumes *Mf: f ∈ measurable M (subprob-algebra M')*
assumes *Mh: case-prod h ∈ measurable (M ⊗_M M') N*
shows *do {x ← M; y ← f x; g (h x y)} = do {x ← M; y ← f x; return N (h x y)} ≫≧ g*
proof –
have *do {x ← M; y ← f x; return N (h x y)} ≫≧ g =*
do {x ← M; do {y ← f x; return N (h x y)} ≫≧ g}
using *Mh by (auto intro!: bind-assoc measurable-bind'[OF Mf] Mf Mg*
measurable-compose[OF - return-measurable] simp: measur-
able-split-conv)
also from *Mh have* $\bigwedge x. x \in \text{space } M \implies h \ x \in \text{measurable } M' \ N$ **by** *measurable*
hence *do {x ← M; do {y ← f x; return N (h x y)} ≫≧ g} =*
do {x ← M; y ← f x; return N (h x y)} ≫≧ g}
apply (*intro ballI bind-cong refl bind-assoc*)
apply (*subst measurable-cong-sets[OF sets-kernel[OF Mf] refl], simp*)
apply (*rule measurable-compose[OF - return-measurable], auto intro: Mg*)
done
also have $\bigwedge x. x \in \text{space } M \implies \text{space } (f \ x) = \text{space } M'$
by (*intro sets-eq-imp-space-eq sets-kernel[OF Mf]*)
with *measurable-space[OF Mh]*
have *do {x ← M; y ← f x; return N (h x y)} ≫≧ g = do {x ← M; y ← f x; g*
(h x y)}
by (*intro ballI bind-cong bind-return[OF Mg] (auto simp: space-pair-measure)*)
finally show *?thesis ..*
qed

lemma (**in** *prob-space*) *M-in-subprob*[*measurable (raw)*]: $M \in \text{space (subprob-algebra } M)$

by (*simp add: space-subprob-algebra*) *unfold-locales*

lemma (in *pair-prob-space*) *pair-measure-eq-bind*:

$(M1 \otimes_M M2) = (M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) (x, y))))$

proof (*rule measure-eqI*)

have *ps-M2*: *prob-space M2* **by** *unfold-locales*

note *return-measurable*[*measurable*]

show *sets* $(M1 \otimes_M M2) = \text{sets } (M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) (x, y))))$

by (*simp-all add: M1.not-empty M2.not-empty*)

fix *A* **assume** [*measurable*]: $A \in \text{sets } (M1 \otimes_M M2)$

show *emeasure* $(M1 \otimes_M M2) A = \text{emeasure } (M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) (x, y)))) A$

by (*auto simp: M2.emeasure-pair-measure M1.not-empty M2.not-empty emeasure-bind*[**where** $N=M1 \otimes_M M2$]
intro!: *nn-integral-cong*)

qed

lemma (in *pair-prob-space*) *bind-rotate*:

assumes *C*[*measurable*]: $(\lambda(x, y). C x y) \in \text{measurable } (M1 \otimes_M M2)$ (*subprob-algebra N*)

shows $(M1 \gg (\lambda x. M2 \gg (\lambda y. C x y))) = (M2 \gg (\lambda y. M1 \gg (\lambda x. C x y)))$

proof –

interpret *swap*: *pair-prob-space M2 M1* **by** *unfold-locales*

note *measurable-bind*[**where** $N=M2$, *measurable*]

note *measurable-bind*[**where** $N=M1$, *measurable*]

have [*simp*]: $M1 \in \text{space } (\text{subprob-algebra } M1)$ $M2 \in \text{space } (\text{subprob-algebra } M2)$

by (*auto simp: space-subprob-algebra*) *unfold-locales*

have $(M1 \gg (\lambda x. M2 \gg (\lambda y. C x y))) =$

$(M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) (x, y)))) \gg (\lambda(x, y). C x y)$

by (*auto intro!*: *bind-cong simp: bind-return*[**where** $N=N$] *space-pair-measure bind-assoc*[**where** $N=M1 \otimes_M M2$ **and** $R=N$])

also have $\dots = (\text{distr } (M2 \otimes_M M1) (M1 \otimes_M M2) (\lambda(x, y). (y, x))) \gg (\lambda(x, y). C x y)$

unfolding *pair-measure-eq-bind*[*symmetric*] *distr-pair-swap*[*symmetric*] ..

also have $\dots = (M2 \gg (\lambda x. M1 \gg (\lambda y. \text{return } (M2 \otimes_M M1) (x, y)))) \gg (\lambda(y, x). C x y)$

unfolding *swap.pair-measure-eq-bind*[*symmetric*]

by (*auto simp add: space-pair-measure M1.not-empty M2.not-empty bind-distr*[*OF - C*] *intro!*: *bind-cong*)

also have $\dots = (M2 \gg (\lambda y. M1 \gg (\lambda x. C x y)))$

by (*auto intro!*: *bind-cong simp: bind-return*[**where** $N=N$] *space-pair-measure bind-assoc*[**where** $N=M2 \otimes_M M1$ **and** $R=N$])

finally show *?thesis* .

qed

lemma *bind-return''*: $\text{sets } M = \text{sets } N \implies M \gg \text{return } N = M$

by (*cases space M = {}*)

(*simp-all add: bind-empty space-empty*[*symmetric*] *bind-nonempty join-return'*)

cong: subprob-algebra-cong)

lemma (*in prob-space*) *distr-const[simp]*:

$c \in \text{space } N \implies \text{distr } M \ N \ (\lambda x. c) = \text{return } N \ c$

by (*rule measure-eqI*) (*auto simp: emeasure-distr emeasure-space-1*)

lemma *return-count-space-eq-density*:

$\text{return } (\text{count-space } M) \ x = \text{density } (\text{count-space } M) \ (\text{indicator } \{x\})$

by (*rule measure-eqI*)

(*auto simp: indicator-inter-arith[symmetric] emeasure-density split: split-indicator*)

lemma *null-measure-in-space-subprob-algebra [simp]*:

$\text{null-measure } M \in \text{space } (\text{subprob-algebra } M) \iff \text{space } M \neq \{\}$

by (*simp add: space-subprob-algebra subprob-space-null-measure-iff*)

7.4 Giry monad on probability spaces

definition *prob-algebra* :: 'a measure \implies 'a measure measure **where**

prob-algebra $K = \text{restrict-space } (\text{subprob-algebra } K) \ \{M. \text{prob-space } M\}$

lemma *space-prob-algebra*: $\text{space } (\text{prob-algebra } M) = \{N. \text{sets } N = \text{sets } M \wedge \text{prob-space } N\}$

unfolding *prob-algebra-def* **by** (*auto simp: space-subprob-algebra space-restrict-space prob-space-imp-subprob-space*)

lemma *measurable-measure-prob-algebra[measurable]*:

$a \in \text{sets } A \implies (\lambda M. \text{Sigma-Algebra.measure } M \ a) \in \text{prob-algebra } A \rightarrow_M \text{borel}$

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space1 measurable-measure-subprob-algebra*)

lemma *measurable-prob-algebraD*:

$f \in N \rightarrow_M \text{prob-algebra } M \implies f \in N \rightarrow_M \text{subprob-algebra } M$

unfolding *prob-algebra-def measurable-restrict-space2-iff* **by** *auto*

lemma *measure-measurable-prob-algebra2*:

$\text{Sigma } (\text{space } M) \ A \in \text{sets } (M \otimes_M N) \implies L \in M \rightarrow_M \text{prob-algebra } N \implies$

$(\lambda x. \text{Sigma-Algebra.measure } (L \ x) \ (A \ x)) \in \text{borel-measurable } M$

using *measure-measurable-subprob-algebra2[of M A N L]* **by** (*auto intro: measurable-prob-algebraD*)

lemma *measurable-prob-algebraI*:

$(\bigwedge x. x \in \text{space } N \implies \text{prob-space } (f \ x)) \implies f \in N \rightarrow_M \text{subprob-algebra } M \implies$

$f \in N \rightarrow_M \text{prob-algebra } M$

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space2*) *auto*

lemma *measurable-distr-prob-space*:

assumes $f: f \in M \rightarrow_M N$

shows $(\lambda M'. \text{distr } M' \ N \ f) \in \text{prob-algebra } M \rightarrow_M \text{prob-algebra } N$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof (*intro conjI measurable-restrict-space1 measurable-distr f*)

show $(\lambda M'. \text{distr } M' N f) \in \text{space } (\text{restrict-space } (\text{subprob-algebra } M) (\text{Collect prob-space})) \rightarrow \text{Collect prob-space}$
using f **by** $(\text{auto simp: space-restrict-space space-subprob-algebra intro!: prob-space.prob-space-distr})$
qed

lemma *measurable-return-prob-space*[*measurable*]: *return* $N \in N \rightarrow_M \text{prob-algebra } N$

by $(\text{rule measurable-prob-algebraI}) (\text{auto simp: prob-space-return})$

lemma *measurable-distr-prob-space2*[*measurable (raw)*]:

assumes $f: g \in L \rightarrow_M \text{prob-algebra } M (\lambda(x, y). f x y) \in L \otimes_M M \rightarrow_M N$

shows $(\lambda x. \text{distr } (g x) N (f x)) \in L \rightarrow_M \text{prob-algebra } N$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof $(\text{intro conjI measurable-restrict-space1 measurable-distr2}[\text{where } M=M] f \text{ measurable-prob-algebraD})$

show $(\lambda x. \text{distr } (g x) N (f x)) \in \text{space } L \rightarrow \text{Collect prob-space}$

using f *subprob-measurableD[OF measurable-prob-algebraD[OF f(1)]]*

by $(\text{auto simp: measurable-restrict-space2-iff prob-algebra-def intro!: prob-space.prob-space-distr})$

qed

lemma *measurable-bind-prob-space*:

assumes $f: f \in M \rightarrow_M \text{prob-algebra } N$ **and** $g: g \in N \rightarrow_M \text{prob-algebra } R$

shows $(\lambda x. \text{bind } (f x) g) \in M \rightarrow_M \text{prob-algebra } R$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof $(\text{intro conjI measurable-restrict-space1 measurable-bind2}[\text{where } N=N] f g \text{ measurable-prob-algebraD})$

show $(\lambda x. f x \ggg g) \in \text{space } M \rightarrow \text{Collect prob-space}$

using $g f$ *subprob-measurableD[OF measurable-prob-algebraD[OF f]]*

by $(\text{auto simp: measurable-restrict-space2-iff prob-algebra-def intro!: prob-space.prob-space-bind}[\text{where } S=R] \text{AE-I2})$

qed

lemma *measurable-bind-prob-space2*[*measurable (raw)*]:

assumes $f: f \in M \rightarrow_M \text{prob-algebra } N$ **and** $g: (\lambda(x, y). g x y) \in (M \otimes_M N) \rightarrow_M \text{prob-algebra } R$

shows $(\lambda x. \text{bind } (f x) (g x)) \in M \rightarrow_M \text{prob-algebra } R$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof $(\text{intro conjI measurable-restrict-space1 measurable-bind}[\text{where } N=N] f g \text{ measurable-prob-algebraD})$

show $(\lambda x. f x \ggg g x) \in \text{space } M \rightarrow \text{Collect prob-space}$

using $g f$ *subprob-measurableD[OF measurable-prob-algebraD[OF f]]*

using *measurable-space[OF g]*

by $(\text{auto simp: measurable-restrict-space2-iff prob-algebra-def space-pair-measure Pi-iff})$

$\text{intro!: prob-space.prob-space-bind}[\text{where } S=R] \text{AE-I2})$

qed $(\text{insert } g, \text{simp})$

lemma *measurable-prob-algebra-generated*:

assumes *eq*: $\text{sets } N = \text{sigma-sets } \Omega \ G$ **and** *Int-stable* $G \subseteq \text{Pow } \Omega$

assumes *subsp*: $\bigwedge a. a \in \text{space } M \implies \text{prob-space } (K \ a)$

assumes *sets*: $\bigwedge a. a \in \text{space } M \implies \text{sets } (K \ a) = \text{sets } N$

assumes $\bigwedge A. A \in G \implies (\lambda a. \text{emeasure } (K \ a) \ A) \in \text{borel-measurable } M$

shows $K \in \text{measurable } M \ (\text{prob-algebra } N)$

unfolding *measurable-restrict-space2-iff prob-algebra-def*

proof

show $K \in M \rightarrow_M \text{subprob-algebra } N$

proof (*rule measurable-subprob-algebra-generated[OF assms(1,2,3) - assms(5,6)]*)

fix *a* **assume** $a \in \text{space } M$ **then show** *subprob-space* $(K \ a)$

using *subsp[of a]* **by** (*intro prob-space-imp-subprob-space*)

next

have $(\lambda a. \text{emeasure } (K \ a) \ \Omega) \in \text{borel-measurable } M \longleftrightarrow (\lambda a. 1::\text{ennreal}) \in$

borel-measurable M

using *sets-eq-imp-space-eq[of sigma $\Omega \ G \ N$] $\langle G \subseteq \text{Pow } \Omega \rangle$ eq sets-eq-imp-space-eq[OF sets]*

prob-space.emeasure-space-1[*OF subsp*]

by (*intro measurable-cong*) *auto*

then show $(\lambda a. \text{emeasure } (K \ a) \ \Omega) \in \text{borel-measurable } M$ **by** *simp*

qed

qed (*insert subsp, auto*)

lemma *in-space-prob-algebra*:

$x \in \text{space } (\text{prob-algebra } M) \implies \text{emeasure } x \ (\text{space } M) = 1$

unfolding *prob-algebra-def space-restrict-space space-subprob-algebra*

by (*auto dest!:* *prob-space.emeasure-space-1 sets-eq-imp-space-eq*)

lemma *prob-space-pair*:

assumes *prob-space* M *prob-space* N **shows** *prob-space* $(M \otimes_M N)$

proof –

interpret M : *prob-space* M **by** *fact*

interpret N : *prob-space* N **by** *fact*

interpret P : *pair-prob-space* $M \ N$ **proof** **qed**

show *?thesis*

by *unfold-locales*

qed

lemma *measurable-pair-prob[measurable]*:

$f \in M \rightarrow_M \text{prob-algebra } N \implies g \in M \rightarrow_M \text{prob-algebra } L \implies (\lambda x. f \ x \otimes_M g \ x) \in M \rightarrow_M \text{prob-algebra } (N \otimes_M L)$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

by (*auto intro!:* *measurable-pair-measure prob-space-pair*)

lemma *emeasure-bind-prob-algebra*:

assumes A : $A \in \text{space } (\text{prob-algebra } N)$

assumes B : $B \in N \rightarrow_M \text{prob-algebra } L$

assumes X : $X \in \text{sets } L$

shows $\text{emeasure } (\text{bind } A \ B) \ X = (\int^+ x. \text{emeasure } (B \ x) \ X \ \partial A)$

using $A B$
by (*intro emeasure-bind*[$OF - - X$])
 (*auto simp: space-prob-algebra measurable-prob-algebraD cong: measurable-cong-sets intro!: prob-space.not-empty*)

lemma *prob-space-bind'*:

assumes $A: A \in \text{space (prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows *prob-space* ($A \gg B$)
using *measurable-bind-prob-space*[$OF \text{ measurable-const, } OF A B$, *THEN measurable-space, of undefined count-space UNIV*]
by (*simp add: space-prob-algebra*)

lemma *sets-bind'*:

assumes $A: A \in \text{space (prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows *sets* ($A \gg B$) = *sets* N
using *measurable-bind-prob-space*[$OF \text{ measurable-const, } OF A B$, *THEN measurable-space, of undefined count-space UNIV*]
by (*simp add: space-prob-algebra*)

lemma *bind-cong-AE'*:

assumes $M: M \in \text{space (prob-algebra } L)$
and $f: f \in L \rightarrow_M \text{prob-algebra } N$ **and** $g: g \in L \rightarrow_M \text{prob-algebra } N$
and $ae: AE x \text{ in } M. f x = g x$
shows *bind* $M f = \text{bind } M g$
proof (*rule measure-eqI*)
show *sets* ($M \gg f$) = *sets* ($M \gg g$)
unfolding *sets-bind'*[$OF M f$] *sets-bind'*[$OF M g$] ..
show $A \in \text{sets } (M \gg f) \implies \text{emeasure } (M \gg f) A = \text{emeasure } (M \gg g) A$
for A
unfolding *sets-bind'*[$OF M f$]
using *emeasure-bind-prob-algebra*[$OF M f, \text{ of } A$] *emeasure-bind-prob-algebra*[$OF M g, \text{ of } A$] ae
by (*auto intro: nn-integral-cong-AE*)
qed

lemma *density-discrete*:

countable $A \implies \text{sets } N = \text{Set.Pow } A \implies (\bigwedge x. f x \geq 0) \implies (\bigwedge x. x \in A \implies f x = \text{emeasure } N \{x\}) \implies$
density (*count-space* A) $f = N$
by (*rule measure-eqI-countable*[$\text{of } - A$]) (*auto simp: emeasure-density*)

lemma *distr-density-discrete*:

fixes f'
assumes *countable* A
assumes $f' \in \text{borel-measurable } M$
assumes $g \in \text{measurable } M$ (*count-space* A)
defines $f \equiv \lambda x. \int^+ t. (\text{if } g t = x \text{ then } 1 \text{ else } 0) * f' t \partial M$
assumes $\bigwedge x. x \in \text{space } M \implies g x \in A$
shows *density* (*count-space* A) $(\lambda x. f x) = \text{distr (density } M f')$ (*count-space* A) g

proof (*rule density-discrete*)
fix x **assume** $x: x \in A$
have $f x = \int^{+t}. \text{indicator } (g - \{x\} \cap \text{space } M) t * f' t \partial M$ (**is** $- = ?I$)
unfolding $f\text{-def}$
by (*intro nn-integral-cong*) (*simp split: split-indicator*)
also from x **have** $\text{in-sets: } g - \{x\} \cap \text{space } M \in \text{sets } M$
by (*intro measurable-sets[OF assms(3)]*) *simp*
have $?I = \text{emeasure } (\text{density } M f') (g - \{x\} \cap \text{space } M)$ **unfolding** $f\text{-def}$
by (*subst emeasure-density[OF assms(2) in-sets]*, *subst mult.commute*) (*rule refl*)
also from $\text{assms}(3)$ x **have** $\dots = \text{emeasure } (\text{distr } (\text{density } M f') (\text{count-space } A) g) \{x\}$
by (*subst emeasure-distr*) *simp-all*
finally show $f x = \text{emeasure } (\text{distr } (\text{density } M f') (\text{count-space } A) g) \{x\}$.
qed (*insert assms, auto*)

lemma *bind-cong-AE*:

assumes $M = N$
assumes $f: f \in \text{measurable } N$ (*subprob-algebra* B)
assumes $g: g \in \text{measurable } N$ (*subprob-algebra* B)
assumes $ae: AE x \text{ in } N. f x = g x$
shows $\text{bind } M f = \text{bind } N g$

proof *cases*

assume $\text{space } N = \{\}$ **then show** $?thesis$
using $\langle M = N \rangle$ **by** (*simp add: bind-empty*)

next

assume $\text{space } N \neq \{\}$
show $?thesis$ **unfolding** $\langle M = N \rangle$
proof (*rule measure-eqI*)
have $*$: $\text{sets } (N \ggg f) = \text{sets } B$
using $\text{sets-bind}[OF \text{sets-kernel}[OF f] \langle \text{space } N \neq \{\} \rangle]$ **by** *simp*
then show $\text{sets } (N \ggg f) = \text{sets } (N \ggg g)$
using $\text{sets-bind}[OF \text{sets-kernel}[OF g] \langle \text{space } N \neq \{\} \rangle]$ **by** *auto*
fix A **assume** $A \in \text{sets } (N \ggg f)$
then have $A \in \text{sets } B$
unfolding $*$.
with $ae f g \langle \text{space } N \neq \{\} \rangle$ **show** $\text{emeasure } (N \ggg f) A = \text{emeasure } (N \ggg g) A$
by (*subst (1 2) emeasure-bind[where N=B]*) (*auto intro!: nn-integral-cong-AE*)
qed
qed

lemma *bind-cong-simp*: $M = N \implies (\bigwedge x. x \in \text{space } M = \text{simp} \implies f x = g x) \implies \text{bind } M f = \text{bind } N g$

by (*auto simp: simp-implies-def intro!: bind-cong*)

lemma *sets-bind-measurable*:

assumes $f: f \in \text{measurable } M$ (*subprob-algebra* B)
assumes $M: \text{space } M \neq \{\}$

shows $sets (M \gg f) = sets B$
using M **by** $(intro\ sets\ bind[OF\ sets\ kernel[OF\ f]])\ auto$

lemma *space-bind-measurable*:
assumes $f: f \in measurable\ M\ (subprob\ algebra\ B)$
assumes $M: space\ M \neq \{\}$
shows $space (M \gg f) = space\ B$
using M **by** $(intro\ space\ bind[OF\ sets\ kernel[OF\ f]])\ auto$

lemma *bind-distr-return*:
 $f \in M \rightarrow_M N \implies g \in N \rightarrow_M L \implies space\ M \neq \{\} \implies$
 $distr\ M\ N\ f \gg (\lambda x. return\ L\ (g\ x)) = distr\ M\ L\ (\lambda x. g\ (f\ x))$
by $(subst\ bind\ distr[OF - measurable\ compose[OF - return\ measurable]])$
 $(auto\ intro!: bind\ return\ distr')$

lemma $(in\ prob\ space)\ AE\ eq\ constD$:
assumes $AE\ x\ in\ M. x = y$
shows $M = return\ M\ y\ y \in space\ M$
proof –
have $AE\ x\ in\ M. x \in space\ M$
by *auto*
with *assms* **have** $AE\ x\ in\ M. y \in space\ M$
by *eventually-elim auto*
thus $y \in space\ M$
by *simp*

show $M = return\ M\ y$
proof $(rule\ measure\ eqI)$
fix X **assume** $X: X \in sets\ M$
have $AE\ x\ in\ M. (x \in X) = (x \in (if\ y \in X\ then\ space\ M\ else\ \{\}))$
using *assms* **by** *eventually-elim (use\ X\ <y \in space\ M> in\ auto)*
hence $emeasure\ M\ X = emeasure\ M\ (if\ y \in X\ then\ space\ M\ else\ \{\})$
using X **by** $(intro\ emeasure\ eq\ AE)\ auto$
also **have** $\dots = emeasure\ (return\ M\ y)\ X$
using X **by** $(auto\ simp: emeasure\ space\ 1)$
finally **show** $emeasure\ M\ X = \dots$

qed *auto*

qed

end

8 Projective Family

theory *Projective-Family*
imports *Giry-Monad*
begin

lemma *vimage-restrict-preseve-mono*:
assumes $J: J \subseteq I$

and sets: $A \subseteq (\prod_{E \ i \in J. S \ i})$ $B \subseteq (\prod_{E \ i \in J. S \ i})$ **and ne:** $(\prod_{E \ i \in I. S \ i}) \neq \{\}$
and eq: $(\lambda x. \text{restrict } x \ J) - ' A \cap (\prod_{E \ i \in I. S \ i}) \subseteq (\lambda x. \text{restrict } x \ J) - ' B \cap (\prod_{E \ i \in I. S \ i})$
shows $A \subseteq B$
proof (*intro subsetI*)
fix x **assume** $x \in A$
from ne **obtain** y **where** $y: \bigwedge i. i \in I \implies y \ i \in S \ i$ **by** *auto*
have $J \cap (I - J) = \{\}$ **by** *auto*
show $x \in B$
proof *cases*
assume $x: x \in (\prod_{E \ i \in J. S \ i})$
have $\text{merge } J \ (I - J) \ (x, y) \in (\lambda x. \text{restrict } x \ J) - ' A \cap (\prod_{E \ i \in I. S \ i})$
using $y \ x \ \langle J \subseteq I \rangle$ *PiE-cancel-merge*[*of J I - J x y S*] $\langle x \in A \rangle$
by (*auto simp del: PiE-cancel-merge simp add: Un-absorb1*)
also have $\dots \subseteq (\lambda x. \text{restrict } x \ J) - ' B \cap (\prod_{E \ i \in I. S \ i})$ **by** *fact*
finally show $x \in B$
using $y \ x \ \langle J \subseteq I \rangle$ *PiE-cancel-merge*[*of J I - J x y S*] $\langle x \in A \rangle$ *eq*
by (*auto simp del: PiE-cancel-merge simp add: Un-absorb1*)
qed (*insert* $\langle x \in A \rangle$ *sets, auto*)
qed

locale *projective-family* =

fixes $I :: 'i \text{ set}$ **and** $P :: 'i \text{ set} \implies ('i \implies 'a) \text{ measure}$ **and** $M :: 'i \implies 'a \text{ measure}$
assumes $P: \bigwedge J \ H. J \subseteq H \implies \text{finite } H \implies H \subseteq I \implies P \ J = \text{distr } (P \ H) \ (PiM \ J \ M)$ $(\lambda f. \text{restrict } f \ J)$
assumes *prob-space-P*: $\bigwedge J. \text{finite } J \implies J \subseteq I \implies \text{prob-space } (P \ J)$
begin

lemma *sets-P*: $\text{finite } J \implies J \subseteq I \implies \text{sets } (P \ J) = \text{sets } (PiM \ J \ M)$
by (*subst P*[*of J J*]) *simp-all*

lemma *space-P*: $\text{finite } J \implies J \subseteq I \implies \text{space } (P \ J) = \text{space } (PiM \ J \ M)$
using *sets-P* **by** (*rule sets-eq-imp-space-eq*)

lemma *not-empty-M*: $i \in I \implies \text{space } (M \ i) \neq \{\}$
using *prob-space-P*[*THEN prob-space.not-empty*] **by** (*auto simp: space-PiM space-P*)

lemma *not-empty*: $\text{space } (PiM \ I \ M) \neq \{\}$
by (*simp add: not-empty-M*)

abbreviation

$\text{emb } L \ K \equiv \text{prod-emb } L \ M \ K$

lemma *emb-preserve-mono*:

assumes $J \subseteq L \ L \subseteq I$ **and** *sets*: $X \in \text{sets } (PiM \ J \ M)$ $Y \in \text{sets } (PiM \ J \ M)$
assumes *emb L J X* \subseteq *emb L J Y*
shows $X \subseteq Y$

proof (*rule vimage-restrict-preseve-mono*)

show $X \subseteq (\prod_{E \ i \in J. \text{space } (M \ i)})$ $Y \subseteq (\prod_{E \ i \in J. \text{space } (M \ i)})$

using $\text{sets}[THEN \text{sets.sets-into-space}]$ **by** $(\text{auto simp: space-PiM})$
show $(\prod_E i \in L. \text{space } (M i)) \neq \{\}$
using $\langle L \subseteq I \rangle$ **by** $(\text{auto simp add: not-empty-M space-PiM[symmetric]})$
show $(\lambda x. \text{restrict } x J) - ' X \cap (\prod_E i \in L. \text{space } (M i)) \subseteq (\lambda x. \text{restrict } x J) - ' Y$
 $\cap (\prod_E i \in L. \text{space } (M i))$
using $\langle \text{prod-emb } L M J X \subseteq \text{prod-emb } L M J Y \rangle$ **by** $(\text{simp add: prod-emb-def})$
qed fact

lemma emb-injective:

assumes $L: J \subseteq L \subseteq I$ **and** $X: X \in \text{sets } (Pi_M J M)$ **and** $Y: Y \in \text{sets } (Pi_M J M)$
shows $\text{emb } L J X = \text{emb } L J Y \implies X = Y$
by $(\text{intro antisym emb-preserve-mono[OF L X Y] emb-preserve-mono[OF L Y X]})$ *auto*

lemma emeasure-P: $J \subseteq K \implies \text{finite } K \implies K \subseteq I \implies X \in \text{sets } (Pi_M J M)$
 $\implies P K (\text{emb } K J X) = P J X$

by $(\text{auto intro!: emeasure-distr-restrict[symmetric] simp: P sets-P})$

inductive-set generator :: $('i \Rightarrow 'a) \text{ set set where}$

$\text{finite } J \implies J \subseteq I \implies X \in \text{sets } (Pi_M J M) \implies \text{emb } I J X \in \text{generator}$

lemma algebra-generator: $\text{algebra } (\text{space } (Pi_M I M)) \text{ generator}$

unfolding algebra-iff-Int

proof $(\text{safe elim!: generator.cases})$

fix $J X$ **assume** $J: \text{finite } J \subseteq I$ **and** $X: X \in \text{sets } (Pi_M J M)$

from $X[THEN \text{sets.sets-into-space}] J$ **show** $x \in \text{emb } I J X \implies x \in \text{space } (Pi_M I M)$ **for** x

by $(\text{auto simp: prod-emb-def space-PiM})$

have $\text{emb } I J (\text{space } (Pi_M J M) - X) \in \text{generator}$

by $(\text{intro generator.intros } J \text{sets.Diff sets.top } X)$

with J **show** $\text{space } (Pi_M I M) - \text{emb } I J X \in \text{generator}$

by $(\text{simp add: space-PiM prod-emb-PiE})$

fix $K Y$ **assume** $K: \text{finite } K \subseteq I$ **and** $Y: Y \in \text{sets } (Pi_M K M)$

have $\text{emb } I (J \cup K) (\text{emb } (J \cup K) J X) \cap \text{emb } I (J \cup K) (\text{emb } (J \cup K) K Y)$
 $\in \text{generator}$

unfolding $\text{prod-emb-Int[symmetric]}$

by $(\text{intro generator.intros sets.Int measurable-prod-emb})$ $(\text{auto intro!: } J K X Y)$

with $J K X Y$ **show** $\text{emb } I J X \cap \text{emb } I K Y \in \text{generator}$

by simp

qed $(\text{force simp: generator.simps prod-emb-empty[symmetric]})$

interpretation generator: $\text{algebra } \text{space } (Pi_M I M) \text{ generator}$

by $(\text{rule algebra-generator})$

lemma sets-PiM-generator: $\text{sets } (Pi_M I M) = \text{sigma-sets } (\text{space } (Pi_M I M)) \text{ gen-}$

erator

proof (*intro antisym sets.sigma-sets-subset*)

show $\text{sets } (PiM I M) \subseteq \text{sigma-sets } (\text{space } (PiM I M))$ generator

unfolding *sets-PiM-single space-PiM[symmetric]*

proof (*intro sigma-sets-mono', safe*)

fix $i A$ **assume** $i \in I$ **and** $A: A \in \text{sets } (M i)$

then have $\{f \in \text{space } (PiM I M). f i \in A\} = \text{emb } I \{i\} (\prod_{E} j \in \{i\}. A)$

by (*auto simp: prod-emb-def space-PiM restrict-def Pi-iff PiE-iff extensional-def*)

with $\langle i \in I \rangle A$ **show** $\{f \in \text{space } (PiM I M). f i \in A\} \in \text{generator}$

by (*auto intro!: generator.intros sets-PiM-I-finite*)

qed

qed (*auto elim!: generator.cases*)

definition *mu-G* (μG) **where**

$\mu G A = (\text{THE } x. \forall J \subseteq I. \text{finite } J \longrightarrow (\forall X \in \text{sets } (PiM J M). A = \text{emb } I J X \longrightarrow x = \text{emeasure } (P J) X))$

definition *lim* :: $(i \Rightarrow 'a)$ *measure* **where**

$\text{lim} = \text{extend-measure } (\text{space } (PiM I M)) \text{ generator } (\lambda x. x) \mu G$

lemma *space-lim[simp]*: $\text{space } \text{lim} = \text{space } (PiM I M)$

using *generator.space-closed*

unfolding *lim-def* **by** (*intro space-extend-measure*) *simp*

lemma *sets-lim[simp, measurable]*: $\text{sets } \text{lim} = \text{sets } (PiM I M)$

using *generator.space-closed* **by** (*simp add: lim-def sets-PiM-generator sets-extend-measure*)

lemma *mu-G-spec*:

assumes $J: \text{finite } J J \subseteq I X \in \text{sets } (PiM J M)$

shows $\mu G (\text{emb } I J X) = \text{emeasure } (P J) X$

unfolding *mu-G-def*

proof (*intro the-equality allI impI ballI*)

fix $K Y$ **assume** $K: \text{finite } K K \subseteq I Y \in \text{sets } (PiM K M)$

and *[simp]*: $\text{emb } I J X = \text{emb } I K Y$

have $\text{emeasure } (P K) Y = \text{emeasure } (P (K \cup J)) (\text{emb } (K \cup J) K Y)$

using $K J$ **by** (*simp add: emeasure-P*)

also have $\text{emb } (K \cup J) K Y = \text{emb } (K \cup J) J X$

using $K J$ **by** (*simp add: emb-injective[of K U J I]*)

also have $\text{emeasure } (P (K \cup J)) (\text{emb } (K \cup J) J X) = \text{emeasure } (P J) X$

using $K J$ **by** (*subst emeasure-P*) *simp-all*

finally show $\text{emeasure } (P J) X = \text{emeasure } (P K) Y$..

qed (*insert J, force*)

lemma *positive-mu-G*: *positive generator* μG

proof –

show *?thesis*

proof (*safe intro!: positive-def[THEN iffD2]*)

obtain J **where** $\text{finite } J J \subseteq I$ **by** *auto*

then have $\mu G (emb I J \{\}) = 0$
by (*subst mu-G-spec*) *auto*
then show $\mu G \{\} = 0$ **by** *simp*
qed
qed

lemma *additive-mu-G: additive generator μG*

proof (*safe intro!: additive-def[THEN iffD2] elim!: generator.cases*)

fix $J X K Y$ **assume** $J: finite J J \subseteq I X \in sets (PiM J M)$
and $K: finite K K \subseteq I Y \in sets (PiM K M)$
and $emb I J X \cap emb I K Y = \{\}$
then have *JK-disj*: $emb (J \cup K) J X \cap emb (J \cup K) K Y = \{\}$
by (*intro emb-injective[of J \cup K I - $\{\}$]*) (*auto simp: sets.Int prod-emb-Int*)
have $\mu G (emb I J X \cup emb I K Y) = \mu G (emb I (J \cup K) (emb (J \cup K) J X \cup emb (J \cup K) K Y))$
using *J K by simp*
also have $\dots = emeasure (P (J \cup K)) (emb (J \cup K) J X \cup emb (J \cup K) K Y)$
using *J K by (simp add: mu-G-spec sets.Un del: prod-emb-Un)*
also have $\dots = \mu G (emb I J X) + \mu G (emb I K Y)$
using *J K JK-disj by (simp add: plus-emeasure[symmetric] mu-G-spec emeasure-P sets-P)*
finally show $\mu G (emb I J X \cup emb I K Y) = \mu G (emb I J X) + \mu G (emb I K Y)$.
qed

lemma *emeasure-lim:*

assumes *JX: finite J J \subseteq I X \in sets (PiM J M)*
assumes *cont: $\bigwedge J X. (\bigwedge i. J i \subseteq I) \implies incseq J \implies (\bigwedge i. finite (J i)) \implies (\bigwedge i. X i \in sets (PiM (J i) M)) \implies decseq (\lambda i. emb I (J i) (X i)) \implies 0 < (INF i. P (J i) (X i)) \implies (\bigcap i. emb I (J i) (X i)) \neq \{\}$*
shows *emeasure lim (emb I J X) = P J X*
proof –
have $\exists \mu. (\forall s \in generator. \mu s = \mu G s) \wedge measure-space (space (PiM I M)) (sigma-sets (space (PiM I M)) generator) \mu$
proof (*rule generator.caratheodory-empty-continuous[OF positive-mu-G additive-mu-G]*)
show $\bigwedge A. A \in generator \implies \mu G A \neq \infty$
proof (*clarsimp elim!: generator.cases simp: mu-G-spec del: notI*)
fix J **assume** *finite J J \subseteq I*
then interpret *prob-space P J by (rule prob-space-P)*
show $\bigwedge X. X \in sets (PiM J M) \implies emeasure (P J) X \neq top$
by *simp*
qed
next
fix A **assume** *range A \subseteq generator and decseq A $(\bigcap i. A i) = \{\}$*
then have $\forall i. \exists J X. A i = emb I J X \wedge finite J \wedge J \subseteq I \wedge X \in sets (PiM J M)$

unfolding *image-subset-iff generator.simps* **by** *blast*
then obtain $J X$ **where** $A: \bigwedge i. A i = \text{emb } I (J i) (X i)$
and $*$: $\bigwedge i. \text{finite } (J i) \wedge i. J i \subseteq I \wedge i. X i \in \text{sets } (PiM (J i) M)$
by *metis*
have $(INF i. P (J i) (X i)) = 0$
proof (*rule ccontr*)
assume $INF\text{-}P: (INF i. P (J i) (X i)) \neq 0$
have $(\bigcap i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i))) \neq \{\}$
proof (*rule cont*)
show $\text{decseq } (\lambda i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$
using $*$ $\langle \text{decseq } A \rangle$ **by** (*subst prod-emb-trans*) (*auto simp: A[abs-def]*)
show $0 < (INF i. P (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$
using $*$ $INF\text{-}P$ **by** (*subst emeasure-P*) (*auto simp: less-le intro!*:
 $INF\text{-}greatest$)
show $\text{incseq } (\lambda i. \bigcup n \leq i. J n)$
by (*force simp: incseq-def*)
qed (*insert *, auto*)
with $\langle (\bigcap i. A i) = \{\} \rangle$ **show** *False*
by (*subst (asm) prod-emb-trans*) (*auto simp: A[abs-def]*)
qed
moreover have $(\lambda i. P (J i) (X i)) \longrightarrow (INF i. P (J i) (X i))$
proof (*intro LIMSEQ-INF antimonoI*)
fix $x y :: \text{nat}$ **assume** $x \leq y$
have $P (J y \cup J x) (\text{emb } (J y \cup J x) (J y) (X y)) \leq P (J y \cup J x) (\text{emb } (J y \cup J x) (J x) (X x))$
using $\langle \text{decseq } A \rangle [THEN \text{decseqD}, OF \langle x \leq y \rangle]$ $*$
by (*auto simp: A sets-P del: subsetI intro!: emeasure-mono \langle x \leq y \rangle*
 $\text{emb-preserve-mono[of } J y \cup J x I, \text{ where } X = \text{emb } (J y \cup J x) (J y)$
 $(X y)]$)
then show $P (J y) (X y) \leq P (J x) (X x)$
using $*$ **by** (*simp add: emeasure-P*)
qed
ultimately show $(\lambda i. \mu G (A i)) \longrightarrow 0$
by (*auto simp: A[abs-def] mu-G-spec **)
qed
then obtain μ **where** $\text{eq: } \forall s \in \text{generator}. \mu s = \mu G s$
and $ms: \text{measure-space } (\text{space } (PiM I M)) (\text{sets } (PiM I M)) \mu$
by (*metis sets-PiM-generator*)
show *?thesis*
proof (*subst emeasure-extend-measure[OF lim-def]*)
show $A \in \text{generator} \implies \mu A = \mu G A$ **for** A
using eq **by** *simp*
show *positive (sets lim) \mu countably-additive (sets lim) \mu*
using ms **by** (*auto simp add: measure-space-def*)
show $(\lambda x. x) \text{ 'generator } \subseteq \text{Pow } (\text{space } (PiM I M))$
using *generator.space-closed* **by** *simp*
show $\text{emb } I J X \in \text{generator } \mu G (\text{emb } I J X) = \text{emeasure } (P J) X$
using JX **by** (*auto intro: generator.intros simp: mu-G-spec*)
qed

qed

end

sublocale *product-prob-space* \subseteq *projective-family* $I \lambda J. PiM J M M$
unfolding *projective-family-def*
proof (*intro conjI allI impI distr-restrict*)
show $\bigwedge J. \text{finite } J \implies \text{prob-space } (PiM J M)$
by (*intro prob-spaceI*) (*simp add: space-PiM emeasure-PiM emeasure-space-1*)
qed *auto*

Proof due to Ionescu Tulcea.

locale *Ionescu-Tulcea* =
fixes $P :: \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow 'a \text{ measure}$ **and** $M :: \text{nat} \Rightarrow 'a \text{ measure}$
assumes $P[\text{measurable}]$: $\bigwedge i. P i \in \text{measurable } (PiM \{0..<i\} M)$ (*subprob-algebra*
 $(M i)$)
assumes *prob-space-P*: $\bigwedge i x. x \in \text{space } (PiM \{0..<i\} M) \implies \text{prob-space } (P i$
 $x)$
begin

lemma *non-empty[simp]*: $\text{space } (M i) \neq \{\}$
proof (*induction i rule: less-induct*)
case (*less i*)
then obtain x **where** $\bigwedge j. j < i \implies x j \in \text{space } (M j)$
unfolding *ex-in-conv[symmetric]* **by** *metis*
then have $*$: $\text{restrict } x \{0..<i\} \in \text{space } (PiM \{0..<i\} M)$
by (*auto simp: space-PiM PiE-iff*)
then interpret *prob-space P i* (*restrict x \{0..<i\}*)
by (*rule prob-space-P*)
show *?case*
using *not-empty subprob-measurableD(1)[OF P, OF *]* **by** *simp*
qed

lemma *space-PiM-not-empty[simp]*: $\text{space } (PiM UNIV M) \neq \{\}$
unfolding *space-PiM-empty-iff* **by** *auto*

lemma *space-P*: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{space } (P n x) = \text{space } (M n)$
by (*simp add: P[THEN subprob-measurableD(1)]*)

lemma *sets-P[measurable-cong]*: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{sets } (P n x) =$
 $\text{sets } (M n)$
by (*simp add: P[THEN subprob-measurableD(2)]*)

definition $eP :: \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \text{ measure}$ **where**
 $eP n \omega = \text{distr } (P n \omega) (PiM \{0..<Suc n\} M) (\text{fun-upd } \omega n)$

lemma *measurable-eP[measurable]*:
 $eP n \in \text{measurable } (PiM \{0..<n\} M)$ (*subprob-algebra* $(PiM \{0..<Suc n\} M)$)
by (*auto simp: eP-def[abs-def] measurable-split-conv*)

intro!: measurable-fun-upd[**where** $J=\{0..<n\}$] measurable-distr2[OF - P])

lemma *space-eP*:

$x \in \text{space } (PiM \{0..<n\} M) \implies \text{space } (eP n x) = \text{space } (PiM \{0..<Suc n\} M)$
by (*simp add: measurable-eP[THEN subprob-measurableD(1)]*)

lemma *sets-eP[measurable]*:

$x \in \text{space } (PiM \{0..<n\} M) \implies \text{sets } (eP n x) = \text{sets } (PiM \{0..<Suc n\} M)$
by (*simp add: measurable-eP[THEN subprob-measurableD(2)]*)

lemma *prob-space-eP*: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{prob-space } (eP n x)$

unfolding *eP-def*

by (*intro prob-space.prob-space-distr prob-space-P measurable-fun-upd[where $J=\{0..<n\}$]*)
auto

lemma *nn-integral-eP*:

$\omega \in \text{space } (PiM \{0..<n\} M) \implies f \in \text{borel-measurable } (PiM \{0..<Suc n\} M)$
 \implies

$(\int^+ x. f x \partial eP n \omega) = (\int^+ x. f (\omega(n := x)) \partial P n \omega)$

unfolding *eP-def*

by (*subst nn-integral-distr*) (*auto intro!*: measurable-fun-upd[**where** $J=\{0..<n\}$])
simp: space-PiM PiE-iff)

lemma *emeasure-eP*:

assumes $\omega[\text{simp}] : \omega \in \text{space } (PiM \{0..<n\} M)$ **and** $A[\text{measurable}] : A \in \text{sets } (PiM \{0..<Suc n\} M)$

shows $eP n \omega A = P n \omega ((\lambda x. \omega(n := x)) -' A \cap \text{space } (M n))$

using *nn-integral-eP[of ωn indicator A]*

apply (*simp add: sets-eP nn-integral-indicator[symmetric] sets-P del: nn-integral-indicator*)

apply (*subst nn-integral-indicator[symmetric]*)

using *measurable-sets[OF measurable-fun-upd[OF - measurable-const[OF ω] measurable-id] A , of n]*

apply (*auto simp add: sets-P atLeastLessThanSuc space-P simp del: nn-integral-indicator intro!: nn-integral-cong split: split-indicator*)

done

primrec $C :: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \text{ measure where}$

$C n 0 \omega = \text{return } (PiM \{0..<n\} M) \omega$

| $C n (Suc m) \omega = C n m \omega \gg eP (n + m)$

lemma *measurable-C[measurable]*:

$C n m \in \text{measurable } (PiM \{0..<n\} M) (\text{subprob-algebra } (PiM \{0..<n + m\} M))$

by (*induction m*) *auto*

lemma *space-C*:

$x \in \text{space } (PiM \{0..<n\} M) \implies \text{space } (C n m x) = \text{space } (PiM \{0..<n + m\} M)$

by (*simp add: measurable-C[THEN subprob-measurableD(1)]*)

lemma *sets-C[measurable-cong]*:

$x \in \text{space } (PiM \{0..<n\} M) \implies \text{sets } (C n m x) = \text{sets } (PiM \{0..<n+m\} M)$
by (*simp add: measurable-C[THEN subprob-measurableD(2)]*)

lemma *prob-space-C*: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{prob-space } (C n m x)$

proof (*induction m*)

case (*Suc m*) **then show** *?case*

by (*auto intro!: prob-space.prob-space-bind[where S=PiM {0..<Suc (n+m)} M]*)

simp: space-C prob-space-eP)

qed (*auto intro!: prob-space-return simp: space-PiM*)

lemma *split-C*:

assumes $\omega: \omega \in \text{space } (PiM \{0..<n\} M)$ **shows** $(C n m \omega \gg C (n+m) l) = C n (m+l) \omega$

proof (*induction l*)

case *0*

with ω **show** *?case*

by (*simp add: bind-return-distr' prob-space-C[THEN prob-space.not-empty]*
distr-cong[OF refl sets-C[symmetric, OF ω]])

next

case (*Suc l*) **with** ω **show** *?case*

by (*simp add: bind-assoc[symmetric, OF - measurable-eP]*) (*simp add: ac-simps*)
qed

lemma *nn-integral-C*:

assumes $m \leq m'$ **and** *f[measurable]*: $f \in \text{borel-measurable } (PiM \{0..<n+m\} M)$

and *nonneg*: $\bigwedge x. x \in \text{space } (PiM \{0..<n+m\} M) \implies 0 \leq f x$

and $x: x \in \text{space } (PiM \{0..<n\} M)$

shows $(\int^{+x}. f x \partial C n m x) = (\int^{+x}. f (\text{restrict } x \{0..<n+m\}) \partial C n m' x)$

using $\langle m \leq m' \rangle$

proof (*induction rule: dec-induct*)

case (*step i*)

let $?E = \lambda x. f (\text{restrict } x \{0..<n+m\})$ **and** $?C = \lambda i f. \int^{+x}. f x \partial C n i x$

from $\langle m \leq i \rangle x$ **have** $?C i ?E = ?C (\text{Suc } i) ?E$

by (*auto simp: nn-integral-bind[where B=PiM {0..<Suc (n+i)} M] space-C nn-integral-eP*)

intro!: nn-integral-cong)

(*simp add: space-PiM PiE-iff nonneg prob-space.emmeasure-space-1[OF prob-space-P]*)

with *step* **show** *?case* **by** (*simp del: restrict-apply*)

qed (*auto simp: space-PiM space-C[OF x] simp del: restrict-apply intro!: nn-integral-cong*)

lemma *emeasure-C*:

assumes $m \leq m'$ **and** *A[measurable]*: $A \in \text{sets } (PiM \{0..<n+m\} M)$ **and** [*simp*]:
 $x \in \text{space } (PiM \{0..<n\} M)$

shows $\text{emeasure } (C n m' x) (\text{prod-emb } \{0..<n+m'\} M \{0..<n+m\} A) =$

emeasure (C n m x) A

using *assms*

by (*subst* (1 2) *nn-integral-indicator*[*symmetric*])

(*auto intro!*: *nn-integral-cong split: split-indicator simp del: nn-integral-indicator simp: nn-integral-C*[*of m m' - n*] *prod-emb-iff space-PiM PiE-iff sets-C*

space-C)

lemma *distr-C*:

assumes $m \leq m'$ **and** [*simp*]: $x \in \text{space } (PiM \{0..<n\} M)$

shows $C \ n \ m \ x = \text{distr } (C \ n \ m' \ x) \ (PiM \ \{0..<n+m\} \ M) \ (\lambda x. \text{restrict } x \ \{0..<n+m\})$

proof (*rule measure-eqI*)

fix A **assume** $A \in \text{sets } (C \ n \ m \ x)$

with $\langle m \leq m' \rangle$ **show** *emeasure* (C n m x) A = *emeasure* (distr (C n m' x) (PiM {0..<n+m} M) ($\lambda x. \text{restrict } x \ \{0..<n+m\}$)) A

by (*subst emeasure-C*[*symmetric, OF* $\langle m \leq m' \rangle$]) (*auto intro!*: *emeasure-distr-restrict*[*symmetric*]

simp: sets-C)

qed (*simp add: sets-C*)

definition *up-to* :: *nat set* \Rightarrow *nat* **where**

up-to J = (*LEAST* n. $\forall i \geq n. i \notin J$)

lemma *up-to-less*: *finite* J $\Rightarrow i \in J \Rightarrow i < \text{up-to } J$

unfolding *up-to-def*

by (*rule LeastI2*[*of - Suc (Max J)*]) (*auto simp: Suc-le-eq not-le*[*symmetric*])

lemma *up-to-iff*: *finite* J $\Rightarrow \text{up-to } J \leq n \iff (\forall i \in J. i < n)$

proof *safe*

show *finite* J $\Rightarrow \text{up-to } J \leq n \Rightarrow i \in J \Rightarrow i < n$ **for** i

using *up-to-less*[*of J i*] **by** *auto*

qed (*auto simp: up-to-def intro!: Least-le*)

lemma *up-to-iff-Ico*: *finite* J $\Rightarrow \text{up-to } J \leq n \iff J \subseteq \{0..<n\}$

by (*auto simp: up-to-iff*)

lemma *up-to*: *finite* J $\Rightarrow J \subseteq \{0..< \text{up-to } J\}$

by (*auto simp: up-to-less*)

lemma *up-to-mono*: $J \subseteq H \Rightarrow \text{finite } H \Rightarrow \text{up-to } J \leq \text{up-to } H$

by (*auto simp add: up-to-iff finite-subset up-to-less*)

definition *CI* :: *nat set* \Rightarrow (*nat* \Rightarrow 'a) *measure* **where**

CI J = *distr* (C 0 (up-to J) ($\lambda x. \text{undefined}$)) (PiM J M) ($\lambda f. \text{restrict } f \ J$)

sublocale *PF*: *projective-family UNIV CI*

unfolding *projective-family-def*

proof *safe*

show *finite* J $\Rightarrow \text{prob-space } (CI \ J)$ **for** J

using *up-to*[*of J*] **unfolding** *CI-def*

by (*intro prob-space.prob-space-distr prob-space-C measurable-restrict*) *auto*
note *measurable-cong-sets*[*OF sets-C, simp*]
have [*simp*]: $J \subseteq H \implies (\lambda f. \text{restrict } f J) \in \text{measurable } (Pi_M H M) (Pi_M J M)$
for $H J$
by (*auto intro!: measurable-restrict*)

show $J \subseteq H \implies \text{finite } H \implies CI J = \text{distr } (CI H) (Pi_M J M) (\lambda f. \text{restrict } f J)$ **for** $J H$
by (*simp add: CI-def distr-C*[*OF up-to-mono*[*of J H*]] *up-to up-to-mono distr-distr comp-def*
inf.absorb2 finite-subset)
qed

lemma *emeasure-CI'*:

$\text{finite } J \implies X \in \text{sets } (Pi_M J M) \implies CI J X = C 0 (\text{up-to } J) (\lambda-. \text{undefined})$
(*PF.emb* $\{0..<n\}$ *up-to* J) $J X$)
unfolding *CI-def* **using** *up-to*[*of J*] **by** (*rule emeasure-distr-restrict*) (*auto simp: sets-C*)

lemma *emeasure-CI*:

$J \subseteq \{0..<n\} \implies X \in \text{sets } (Pi_M J M) \implies CI J X = C 0 n (\lambda-. \text{undefined})$
(*PF.emb* $\{0..<n\}$ $J X$)
apply (*subst emeasure-CI', simp-all add: finite-subset*)
apply (*subst emeasure-C*[*symmetric, of up-to J n*])
apply (*auto simp: finite-subset up-to-iff-Ico up-to-less*)
apply (*subst prod-emb-trans*)
apply (*auto simp: up-to-less finite-subset up-to-iff-Ico*)
done

lemma *lim*:

assumes J : *finite J* **and** X : $X \in \text{sets } (Pi_M J M)$
shows *emeasure PF.lim* (*PF.emb UNIV J X*) = *emeasure* ($CI J$) X
proof (*rule PF.emeasure-lim*[*OF J subset-UNIV X*])
fix $J X'$ **assume** J [*simp*]: $\bigwedge i. \text{finite } (J i)$ **and** X' [*measurable*]: $\bigwedge i. X' i \in \text{sets } (Pi_M (J i) M)$
and *dec: decseq* ($\lambda i. PF.emb UNIV (J i) (X' i)$)
define X **where** $X n =$
 $(\bigcap_{i \in \{i. J i \subseteq \{0..<n\}\}} PF.emb \{0..<n\} (J i) (X' i)) \cap \text{space } (Pi_M \{0..<n\} M)$ **for** n

have *sets-X*[*measurable*]: $X n \in \text{sets } (Pi_M \{0..<n\} M)$ **for** n
by (*cases* $\{i. J i \subseteq \{0..<n\}\} = \{\}$)
(*simp-all add: X-def, auto intro!: sets.countable-INT' sets.Int*)

have *dec-X*: $n \leq m \implies X m \subseteq PF.emb \{0..<m\} \{0..<n\} (X n)$ **for** $n m$
unfolding *X-def* **using** *ivl-subset*[*of 0 n 0 m*]
by (*cases* $\{i. J i \subseteq \{0..<n\}\} = \{\}$)
(*auto simp add: prod-emb-Int prod-emb-PiE space-PiM simp del: ivl-subset*)

```

have  $dec\text{-}X'$ :  $PF.emb \{0..<n\} (J j) (X' j) \subseteq PF.emb \{0..<n\} (J i) (X' i)$ 
if  $[simp]$ :  $J i \subseteq \{0..<n\} J j \subseteq \{0..<n\} i \leq j$  for  $n i j$ 
by (rule  $PF.emb\text{-}preserve\text{-}mono$ [of  $\{0..<n\}$   $UNIV$ ]) (auto del: subsetI intro:
 $dec[THEN antimonoD]$ )

assume  $0 < (INF i. CI (J i) (X' i))$ 
also have  $\dots \leq (INF i. C 0 i (\lambda x. undefined) (X i))$ 
proof (intro  $INF\text{-}greatest$ )
  fix  $n$ 
  interpret  $C$ : prob-space  $C 0 n (\lambda x. undefined)$ 
  by (rule  $prob\text{-}space\text{-}C$ ) simp
  show  $(INF i. CI (J i) (X' i)) \leq C 0 n (\lambda x. undefined) (X n)$ 
  proof cases
    assume  $\{i. J i \subseteq \{0..<n\}\} = \{\}$  with  $C.emeasure\text{-}space\text{-}1$  show ?thesis
    by (auto simp add: X-def space-C intro!: INF-lower2[of 0] prob-space.measure-le-1
 $PF.emb\text{-}space\text{-}P$ )
  next
    assume  $*$ :  $\{i. J i \subseteq \{0..<n\}\} \neq \{\}$ 
    have  $(INF i. CI (J i) (X' i)) \leq$ 
       $(INF i \in \{i. J i \subseteq \{0..<n\}\}. C 0 n (\lambda x. undefined) (PF.emb \{0..<n\} (J i)$ 
 $(X' i)))$ 
    by (intro  $INF\text{-}superset\text{-}mono$ ) (auto simp: emeasure-CI)
    also have  $\dots = C 0 n (\lambda x. undefined) (\bigcap i \in \{i. J i \subseteq \{0..<n\}\}. (PF.emb$ 
 $\{0..<n\} (J i) (X' i)))$ 
    using  $*$  by (intro  $emeasure\text{-}INT\text{-}decseq\text{-}subset[symmetric]$ ) (auto intro!:
 $dec\text{-}X'$  del: subsetI simp: sets-C)
    also have  $\dots = C 0 n (\lambda x. undefined) (X n)$ 
    using  $*$  by (auto simp add: X-def INT-extend-simps)
    finally show  $(INF i. CI (J i) (X' i)) \leq C 0 n (\lambda x. undefined) (X n)$  .
  qed
qed
finally have  $pos$ :  $0 < (INF i. C 0 i (\lambda x. undefined) (X i))$  .
from  $less\text{-}INF\text{-}D[OF this, of 0]$  have  $X 0 \neq \{\}$ 
by auto

{ fix  $\omega n$  assume  $\omega$ :  $\omega \in space (PiM \{0..<n\} M)$ 
  let  $?C = \lambda i. emeasure (C n i \omega) (X (n + i))$ 
  let  $?C' = \lambda i x. emeasure (C (Suc n) i (\omega(n:=x))) (X (Suc n + i))$ 
  have  $M$ :  $\bigwedge i. ?C' i \in borel\text{-}measurable (P n \omega)$ 
  using  $\omega$  [measurable, simp] measurable-fun-upd[where J={0..<n}] by mea-
surable auto

assume  $0 < (INF i. ?C i)$ 
also have  $\dots \leq (INF i. emeasure (C n (1 + i) \omega) (X (n + (1 + i))))$ 
by (intro  $INF\text{-}greatest INF\text{-}lower$ ) auto
also have  $\dots = (INF i. \int^+ x. ?C' i x \partial P n \omega)$ 
using  $\omega$  measurable-C[of Suc n]
apply (intro  $INF\text{-}cong refl$ )
apply (subst split-C[symmetric, OF  $\omega$ ])

```

```

apply (subst emeasure-bind[OF - - sets-X])
apply (simp-all del: C.simps add: space-C)
apply measurable
apply simp
apply (simp add: bind-return[OF measurable-eP] nn-integral-eP)
done
also have ... = ( $\int^+ x. (\text{INF } i. ?C' i x) \partial P n \omega$ )
proof (rule nn-integral-monotone-convergence-INF-AE[symmetric])
  have ( $\int^+ x. ?C' 0 x \partial P n \omega$ )  $\leq$  ( $\int^+ x. 1 \partial P n \omega$ )
    by (intro nn-integral-mono) (auto split: split-indicator)
  also have ...  $< \infty$ 
    using prob-space-P[OF  $\omega$ , THEN prob-space.emeasure-space-1] by simp
  finally show ( $\int^+ x. ?C' 0 x \partial P n \omega$ )  $< \infty$  .
next
show AE  $x$  in  $P n \omega$ .  $?C' (\text{Suc } i) x \leq ?C' i x$  for  $i$ 
proof (rule AE-I2)
  fix  $x$  assume  $x \in \text{space } (P n \omega)$ 
  with  $\omega$  have  $\omega' : \omega(n := x) \in \text{space } (PiM \{0..< \text{Suc } n\} M)$ 
    by (auto simp: space-P[OF  $\omega$ ] space-PiM PiE-iff extensional-def)
  with  $\omega$  show  $?C' (\text{Suc } i) x \leq ?C' i x$ 
    apply (subst emeasure-C[symmetric, of  $i \text{ Suc } i$ ])
    apply (auto intro!: emeasure-mono[OF dec-X] del: subsetI
      simp: sets-C space-P)
    apply (subst sets-bind[OF sets-eP])
    apply (simp-all add: space-C space-P)
  done
qed
qed fact
finally have ( $\int^+ x. (\text{INF } i. ?C' i x) \partial P n \omega$ )  $\neq 0$ 
  by simp
with  $M$  have  $\exists_F x$  in ae-filter  $(P n \omega)$ .  $0 < (\text{INF } i. ?C' i x)$ 
  by (subst (asm) nn-integral-0-iff-AE)
    (auto intro!: borel-measurable-INF simp: Filter.not-eventually not-le
  zero-less-iff-neq-zero)
  then have  $\exists_F x$  in ae-filter  $(P n \omega)$ .  $x \in \text{space } (M n) \wedge 0 < (\text{INF } i. ?C' i x)$ 
    by (rule frequently-mp[rotated]) (auto simp: space-P  $\omega$ )
  then obtain  $x$  where  $x \in \text{space } (M n) \wedge 0 < (\text{INF } i. ?C' i x)$ 
    by (auto dest: frequently-ex)
  from this(2)[THEN less-INF-D, of 0] this(2)
  have  $\exists x. \omega(n := x) \in X (\text{Suc } n) \wedge 0 < (\text{INF } i. ?C' i x)$ 
    by (intro exI[of -  $x$ ]) (simp split: split-indicator-asm) }
note step = this

let  $?\omega = \lambda \omega n x. (\text{restrict } \omega \{0..<n\})(n := x)$ 
let  $?L = \lambda \omega n r. \text{INF } i. \text{emeasure } (C (\text{Suc } n) i (\?\omega \omega n r)) (X (\text{Suc } n + i))$ 
have *: ( $\bigwedge i. i < n \implies ?\omega \omega i (\omega i) \in X (\text{Suc } i)$ )  $\implies$ 
   $\text{restrict } \omega \{0..<n\} \in \text{space } (PiM \{0..<n\} M)$  for  $\omega n$ 
using sets.sets-into-space[OF sets-X, of  $n$ ]
by (cases  $n$ ) (auto simp: atLeastLessThanSuc restrict-def[of - {}])

```

```

have  $\exists \omega. \forall n. ?\omega \omega n (\omega n) \in X (Suc n) \wedge 0 < ?L \omega n (\omega n)$ 
proof (rule dependent-wellorder-choice)
  fix  $n \omega$  assume  $IH: \bigwedge i. i < n \implies ?\omega \omega i (\omega i) \in X (Suc i) \wedge 0 < ?L \omega i (\omega$ 
i)
  show  $\exists r. ?\omega \omega n r \in X (Suc n) \wedge 0 < ?L \omega n r$ 
  proof (rule step)
    show  $restrict \omega \{0..<n\} \in space (Pi_M \{0..<n\} M)$ 
      using  $IH[THEN conjunct1]$  by (rule *)
    show  $0 < (INF i. emeasure (C n i (restrict \omega \{0..<n\}))) (X (n + i))$ 
    proof (cases n)
      case 0 with pos show ?thesis
        by (simp add: CI-def restrict-def)
      next
        case (Suc i) then show ?thesis
          using  $IH[of i, THEN conjunct2]$  by (simp add: atLeastLessThanSuc)
    qed
  qed
qed (simp cong: restrict-cong)
then obtain  $\omega$  where  $\omega: \bigwedge n. ?\omega \omega n (\omega n) \in X (Suc n)$ 
by auto
from this[THEN *] have  $\omega\text{-space}: \omega \in space (Pi_M UNIV M)$ 
by (auto simp: space-PiM PiE-iff Ball-def)
have  $*$ :  $\omega \in PF.emb UNIV \{0..<n\} (X n)$  for  $n$ 
proof (cases n)
  case 0 with  $\omega\text{-space} \langle X 0 \neq \{\} \rangle$  sets.sets-into-space[OF sets-X, of 0] show
?thesis
  by (auto simp add: space-PiM prod-emb-def restrict-def PiE-iff)
  next
    case (Suc i) then show ?thesis
      using  $\omega[of i] \omega\text{-space}$  by (auto simp: prod-emb-def space-PiM PiE-iff atLeast-
LessThanSuc)
    qed
  have  $**$ :  $\{i. J i \subseteq \{0..<up\text{-to } (J n)\}\} \neq \{\}$  for  $n$ 
    by (auto intro!: exI[of - n] up-to J)
  have  $\omega \in PF.emb UNIV (J n) (X' n)$  for  $n$ 
    using  $*[of up\text{-to } (J n)] up\text{-to}[of J n]$  by (simp add: X-def prod-emb-Int prod-emb-INT[OF
**])
  then show  $(\bigcap i. PF.emb UNIV (J i) (X' i)) \neq \{\}$ 
    by auto
qed

lemma distr-lim: assumes  $J[simp]$ : finite  $J$  shows  $distr PF.lim (Pi_M J M) (\lambda x.$ 
 $restrict x J) = CI J$ 
  apply (rule measure-eqI)
  apply (simp add: CI-def)
  apply (simp add: emeasure-distr measurable-cong-sets[OF PF.sets-lim] lim[symmetric]
prod-emb-def space-PiM)
  done

```


end

lemma (in *product-prob-space*) *emeasure-lim-emb*:

assumes *: *finite* J $J \subseteq I$ $X \in \text{sets}$ ($\text{PiM } J M$)

shows *emeasure* *lim* (*emb* $I J X$) = *emeasure* ($\text{PiM } J M$) X

proof (*rule* *emeasure-lim*[*OF* *], *goal-cases*)

case ($1 J X$)

have $\exists Q. (\forall i. \text{sets } Q = \text{PiM } (\bigcup i. J i) M \wedge \text{distr } Q (\text{PiM } (J i) M) (\lambda x. \text{restrict } x (J i)) = \text{PiM } (J i) M)$

proof *cases*

assume *finite* ($\bigcup i. J i$)

then have *distr* ($\text{PiM } (\bigcup i. J i) M$) ($\text{PiM } (J i) M$) ($\lambda x. \text{restrict } x (J i)$) = $\text{PiM } (J i) M$ **for** i

by (*intro* *distr-restrict*[*symmetric*]) *auto*

then show *?thesis*

by *auto*

next

assume *inf*: *infinite* ($\bigcup i. J i$)

moreover have *count*: *countable* ($\bigcup i. J i$)

using $1(3)$ **by** (*auto* *intro*: *countable-finite*)

define f **where** $f = \text{from-nat-into } (\bigcup i. J i)$

define t **where** $t = \text{to-nat-on } (\bigcup i. J i)$

have *ft*[*simp*]: $x \in J i \implies f (t x) = x$ **for** $x i$

unfolding *f-def* *t-def* **using** *inf* *count* **by** (*intro* *from-nat-into-to-nat-on*) *auto*

have *tf*[*simp*]: $t (f i) = i$ **for** i

unfolding *t-def* *f-def* **by** (*intro* *to-nat-on-from-nat-into-infinite* *inf* *count*)

have *inj-t*: *inj-on* t ($\bigcup i. J i$)

using *count* **by** (*auto* *simp*: *t-def*)

then have *inj-t-J*: *inj-on* t ($J i$) **for** i

by (*rule* *subset-inj-on*) *auto*

interpret *IT*: *Ionescu-Tulcea* $\lambda i \omega. M (f i) \lambda i. M (f i)$

by *standard* *auto*

interpret *Mf*: *product-prob-space* $\lambda x. M (f x) \text{UNIV}$

by *standard*

have *C-eq-PiM*: *IT.C* $0 n (\lambda \cdot. \text{undefined}) = \text{PiM } \{0..<n\} (\lambda x. M (f x))$ **for** n

proof (*induction* n)

case 0 **then show** *?case*

by (*auto* *simp*: *PiM-empty* *intro!*: *measure-eqI* *dest!*: *subset-singletonD*)

next

case (*Suc* n) **then show** *?case*

apply (*auto* *intro!*: *measure-eqI* *simp*: *sets-bind*[*OF* *IT.sets-eP*] *emeasure-bind*[*OF* - *IT.measurable-eP*])

apply (*auto* *simp*: *Mf.product-nn-integral-insert* *nn-integral-indicator*[*symmetric*] *atLeastLessThanSuc* *IT.emeasure-eP* *space-PiM*

split: *split-indicator* *simp* *del*: *nn-integral-indicator* *intro!*:

nn-integral-cong)

done

qed

have *CI-eq-PiM*: $IT.CI\ X = PiM\ X\ (\lambda x. M\ (f\ x))$ **if** X : *finite X for X*
by (*auto simp*: $IT.up-to-less\ X\ IT.CI-def\ C-eq-PiM\ intro!$: $Mf.distr-restrict[symmetric]$)

let $?Q = distr\ IT.PF.lim\ (PiM\ (\bigcup i. J\ i)\ M)\ (\lambda\omega. \lambda x \in \bigcup i. J\ i. \omega\ (t\ x))$
{ fix i
have $distr\ ?Q\ (Pi_M\ (J\ i)\ M)\ (\lambda x. restrict\ x\ (J\ i)) =$
 $distr\ IT.PF.lim\ (Pi_M\ (J\ i)\ M)\ ((\lambda\omega. \lambda n \in J\ i. \omega\ (t\ n)) \circ (\lambda\omega. restrict\ \omega\ (t^J\ i)))$
proof (*subst distr-distr*)
have $(\lambda\omega. \omega\ (t\ x)) \in measurable\ (Pi_M\ UNIV\ (\lambda x. M\ (f\ x)))\ (M\ x)$ **if** $x: x \in J\ i$ **for** $x\ i$
using *measurable-component-singleton*[*of t x UNIV $\lambda x. M\ (f\ x)$*] **unfolding**
ft[OF x] **by** *simp*
then show $(\lambda\omega. \lambda x \in \bigcup i. J\ i. \omega\ (t\ x)) \in measurable\ IT.PF.lim\ (Pi_M\ (\bigcup (J\ ' UNIV))\ M)$
by (*auto intro!*: *measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim refl]*)
qed (*auto intro!*: *distr-cong measurable-restrict measurable-component-singleton*)
also have $\dots = distr\ (distr\ IT.PF.lim\ (Pi_M\ (t^J\ i)\ (\lambda x. M\ (f\ x)))\ (\lambda\omega. restrict\ \omega\ (t^J\ i)))\ (Pi_M\ (J\ i)\ M)\ (\lambda\omega. \lambda n \in J\ i. \omega\ (t\ n))$
proof (*intro distr-distr[symmetric]*)
have $(\lambda\omega. \omega\ (t\ x)) \in measurable\ (Pi_M\ (t^J\ i)\ (\lambda x. M\ (f\ x)))\ (M\ x)$ **if** $x: x \in J\ i$ **for** x
using *measurable-component-singleton*[*of t x t^J i $\lambda x. M\ (f\ x)$*] x **unfolding**
ft[OF x] **by** *auto*
then show $(\lambda\omega. \lambda n \in J\ i. \omega\ (t\ n)) \in measurable\ (Pi_M\ (t^J\ i)\ (\lambda x. M\ (f\ x)))\ (Pi_M\ (J\ i)\ M)$
by (*auto intro!*: *measurable-restrict*)
qed (*auto intro!*: *measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim refl]*)
also have $\dots = distr\ (Pi_M\ (t^J\ i)\ (\lambda x. M\ (f\ x)))\ (Pi_M\ (J\ i)\ M)\ (\lambda\omega. \lambda n \in J\ i. \omega\ (t\ n))$
using $\langle finite\ (J\ i) \rangle$ **by** (*subst IT.distr-lim*) (*auto simp: CI-eq-PiM*)
also have $\dots = Pi_M\ (J\ i)\ M$
using *Mf.distr-reorder*[*of t J i*] **by** (*simp add: 1 inj-t-J cong: PiM-cong*)
finally have $distr\ ?Q\ (Pi_M\ (J\ i)\ M)\ (\lambda x. restrict\ x\ (J\ i)) = Pi_M\ (J\ i)\ M$ **}**
then show $\exists Q. \forall i. sets\ Q = PiM\ (\bigcup i. J\ i)\ M \wedge distr\ Q\ (Pi_M\ (J\ i)\ M)\ (\lambda x. restrict\ x\ (J\ i)) = Pi_M\ (J\ i)\ M$
by (*intro exI*[*of - ?Q*]) *auto*
qed
then obtain Q **where** *sets-Q*: $sets\ Q = PiM\ (\bigcup i. J\ i)\ M$
and $Q: \bigwedge i. distr\ Q\ (Pi_M\ (J\ i)\ M)\ (\lambda x. restrict\ x\ (J\ i)) = Pi_M\ (J\ i)\ M$ **by**
blast

from 1 interpret Q : *prob-space Q*
by (*intro prob-space-distrD*[*of $\lambda x. restrict\ x\ (J\ 0)\ Q\ PiM\ (J\ 0)\ M$*])
(*auto simp: Q measurable-cong-sets[OF sets-Q]*)
intro!: *prob-space-P measurable-restrict measurable-component-singleton*)

have $0 < (\text{INF } i. \text{emeasure } (Pi_M (J i) M) (X i))$ **by fact**
also have $\dots = (\text{INF } i. \text{emeasure } Q (\text{emb } (\bigcup i. J i) (J i) (X i)))$
by (*simp add: emeasure-distr-restrict[OF - sets-Q 1(4), symmetric] SUP-upper Q*)
also have $\dots = \text{emeasure } Q (\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X i))$
proof (*rule INF-emeasure-decseq*)
from 1 show *decseq* $(\lambda n. \text{emb } (\bigcup i. J i) (J n) (X n))$
by (*intro antimonoI emb-preserve-mono[where X=emb (\bigcup i. J i) (J n) (X n) and L=I and J=\bigcup i. J i for n] measurable-prod-emb*)
(auto simp: SUP-least SUP-upper antimono-def)
qed (*insert 1, auto simp: sets-Q*)
finally have $(\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X i)) \neq \{\}$
by auto
moreover have $(\bigcap i. \text{emb } I (J i) (X i)) = \{\} \implies (\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X i)) = \{\}$
using 1 by (*intro emb-injective[of \bigcup i. J i I - \{\}] sets.countable-INT*) (*auto simp: SUP-least SUP-upper*)
ultimately show ?case by auto
qed
end

9 Infinite Product Measure

theory *Infinite-Product-Measure*

imports *Probability-Measure Projective-Family*

begin

lemma (*in product-prob-space*) *distr-PiM-restrict-finite*:

assumes *finite J J \subseteq I*

shows *distr (PiM I M) (PiM J M) (\lambda x. restrict x J) = PiM J M*

proof (*rule PiM-eqI*)

fix *X* **assume** *X: \bigwedge i. i \in J \implies X i \in sets (M i)*

{ fix J X assume J: J \neq \{\} \vee I = \{\} finite J J \subseteq I and X: \bigwedge i. i \in J \implies X i \in sets (M i)

have *emeasure (PiM I M) (emb I J (Pi_E J X)) = (\prod i \in J. M i (X i))*

proof (*subst emeasure-extend-measure-Pair[OF PiM-def, where \mu' = lim], goal-cases*)

case 1 then show ?case

by (*simp add: M.emeasure-space-1 emeasure-PiM Pi-iff sets-PiM-I-finite emeasure-lim-emb*)

next

case $(2 J X)$

then have *emb I J (Pi_E J X) \in sets (PiM I M)*

by (*intro measurable-prod-emb sets-PiM-I-finite*) *auto*

from this[THEN sets.sets-into-space] show ?case

by (*simp add: space-PiM*)

qed (*insert assms J X, simp-all del: sets-lim*

add: M.emeasure-space-1 sets-lim[symmetric] emeasure-countably-additive)

```

emeasure-positive) }
  note * = this

  have emeasure (PiM I M) (emb I J (PiE J X)) = ( $\prod_{i \in J}. M\ i\ (X\ i)$ )
  proof (cases J ≠ {} ∨ I = {})
    case False
      then obtain i where i: J = {} i ∈ I by auto
      then have emb I {} {λx. undefined} = emb I {i} (ΠE i ∈ {i}. space (M i))
        by (auto simp: space-PiM prod-emb-def)
      with i show ?thesis
        by (simp add: * M.emeasure-space-1)
    next
      case True
      then show ?thesis
        by (simp add: *[OF - assms X])
  qed
  with assms show emeasure (distr (PiM I M) (PiM J M) (λx. restrict x J))
(PiE J X) = ( $\prod_{i \in J}. \text{emeasure } (M\ i)\ (X\ i)$ )
  by (subst emeasure-distr-restrict[OF - refl]) (auto intro!: sets-PiM-I-finite X)
qed (insert assms, auto)

```

lemma (*in product-prob-space*) *emeasure-PiM-emb'*:

```

J ⊆ I ⇒ finite J ⇒ X ∈ sets (PiM J M) ⇒ emeasure (PiM I M) (emb I J X)
= PiM J M X
  by (subst distr-PiM-restrict-finite[symmetric, of J])
  (auto intro!: emeasure-distr-restrict[symmetric])

```

lemma (*in product-prob-space*) *emeasure-PiM-emb*:

```

J ⊆ I ⇒ finite J ⇒ ( $\bigwedge i. i \in J \Rightarrow X\ i \in \text{sets } (M\ i)$ ) ⇒
  emeasure (PiM I M) (emb I J (PiE J X)) = ( $\prod_{i \in J}. \text{emeasure } (M\ i)\ (X\ i)$ )
  by (subst emeasure-PiM-emb') (auto intro!: emeasure-PiM)

```

sublocale *product-prob-space* ⊆ *P?*: *prob-space PiM I M*

```

proof
  have *: emb I {} {λx. undefined} = space (PiM I M)
    by (auto simp: prod-emb-def space-PiM)
  show emeasure (PiM I M) (space (PiM I M)) = 1
    using emeasure-PiM-emb[of {} λ-. {}] by (simp add: *)
qed

```

lemma *prob-space-PiM*:

```

assumes M:  $\bigwedge i. i \in I \Rightarrow \text{prob-space } (M\ i)$  shows prob-space (PiM I M)
proof –
  let ?M = λi. if i ∈ I then M i else count-space {undefined}
  interpret M': prob-space ?M i for i
    using M by (cases i ∈ I) (auto intro!: prob-spaceI)
  interpret product-prob-space ?M I
    by unfold-locales
  have prob-space (ΠM i ∈ I. ?M i)

```

by *unfold-locales*
 also have $(\prod_{M \ i \in I}. ?M \ i) = (\prod_{M \ i \in I}. M \ i)$
 by (*intro PiM-cong*) *auto*
 finally show *?thesis* .
 qed

lemma (in *product-prob-space*) *emeasure-PiM-Collect*:
 assumes $X: J \subseteq I$ finite $J \wedge i. i \in J \implies X \ i \in \text{sets } (M \ i)$
 shows $\text{emeasure } (Pi_M \ I \ M) \ \{x \in \text{space } (Pi_M \ I \ M). \forall i \in J. x \ i \in X \ i\} = (\prod_{i \in J}. \text{emeasure } (M \ i) \ (X \ i))$
proof –
 have $\{x \in \text{space } (Pi_M \ I \ M). \forall i \in J. x \ i \in X \ i\} = \text{emb } I \ J \ (Pi_E \ J \ X)$
 unfolding *prod-emb-def* using *assms* by (*auto simp: space-PiM Pi-iff*)
 with *emeasure-PiM-emb[OF assms]* show *?thesis* by *simp*
 qed

lemma (in *product-prob-space*) *emeasure-PiM-Collect-single*:
 assumes $X: i \in I \ A \in \text{sets } (M \ i)$
 shows $\text{emeasure } (Pi_M \ I \ M) \ \{x \in \text{space } (Pi_M \ I \ M). x \ i \in A\} = \text{emeasure } (M \ i) \ A$
 using *emeasure-PiM-Collect[of \{i\} \lambda i. A]* *assms*
 by *simp*

lemma (in *product-prob-space*) *measure-PiM-emb*:
 assumes $J \subseteq I$ finite $J \wedge i. i \in J \implies X \ i \in \text{sets } (M \ i)$
 shows $\text{measure } (Pi_M \ I \ M) \ (\text{emb } I \ J \ (Pi_E \ J \ X)) = (\prod_{i \in J}. \text{measure } (M \ i) \ (X \ i))$
 using *emeasure-PiM-emb[OF assms]*
 unfolding *emeasure-eq-measure* *M.emeasure-eq-measure*
 by (*simp add: prod-ennreal measure-nonneg prod-nonneg*)

lemma *sets-Collect-single'*:
 $i \in I \implies \{x \in \text{space } (M \ i). P \ x\} \in \text{sets } (M \ i) \implies \{x \in \text{space } (Pi_M \ I \ M). P \ (x \ i)\} \in \text{sets } (Pi_M \ I \ M)$
 using *sets-Collect-single[of i I \{x \in \text{space } (M \ i). P \ x\} M]*
 by (*simp add: space-PiM PiE-iff cong: conj-cong*)

lemma (in *finite-product-prob-space*) *finite-measure-PiM-emb*:
 $(\wedge i. i \in I \implies A \ i \in \text{sets } (M \ i)) \implies \text{measure } (Pi_M \ I \ M) \ (Pi_E \ I \ A) = (\prod_{i \in I}. \text{measure } (M \ i) \ (A \ i))$
 using *measure-PiM-emb[of I A]* *finite-index prod-emb-PiE-same-index[OF sets.sets-into-space, of I A M]*
 by *auto*

lemma (in *product-prob-space*) *PiM-component*:
 assumes $i \in I$
 shows $\text{distr } (Pi_M \ I \ M) \ (M \ i) \ (\lambda \omega. \omega \ i) = M \ i$
proof (*rule measure-eqI[symmetric]*)
 fix A assume $A \in \text{sets } (M \ i)$
 moreover have $((\lambda \omega. \omega \ i) -' A \cap \text{space } (Pi_M \ I \ M)) = \{x \in \text{space } (Pi_M \ I \ M). x$

$i \in A$
 by *auto*
 ultimately show $\text{emeasure } (M \ i) \ A = \text{emeasure } (\text{distr } (PiM \ I \ M) \ (M \ i) \ (\lambda\omega. \omega \ i)) \ A$
 by (*auto simp: $\langle i \in I \rangle$ emeasure-distr measurable-component-singleton emeasure-PiM-Collect-single*)
 qed *simp*

lemma (in *product-prob-space*) *PiM-eq*:
 assumes M' : *sets* $M' = \text{sets } (PiM \ I \ M)$
 assumes *eq*: $\bigwedge J \ F. \text{finite } J \implies J \subseteq I \implies (\bigwedge j. j \in J \implies F \ j \in \text{sets } (M \ j)) \implies$
 $\text{emeasure } M' \ (\text{prod-emb } I \ M \ J \ (\prod_{E \ j \in J}. F \ j)) = (\prod_{j \in J}. \text{emeasure } (M \ j) \ (F \ j))$
 shows $M' = (PiM \ I \ M)$
proof (*rule measure-eqI-PiM-infinite[symmetric, OF refl M']*)
 show *finite-measure* $(PiM \ I \ M)$
 by *standard* (*simp add: P.emeasure-space-1*)
 qed (*simp add: eq emeasure-PiM-emb*)

lemma (in *product-prob-space*) *AE-component*: $i \in I \implies AE \ x \ \text{in } M \ i. \ P \ x \implies$
 $AE \ x \ \text{in } PiM \ I \ M. \ P \ (x \ i)$
 apply (*rule AE-distrD[of $\lambda\omega. \omega \ i \ PiM \ I \ M \ M \ i \ P$]*)
 apply *simp*
 apply (*subst PiM-component*)
 apply *simp-all*
 done

lemma *emeasure-PiM-emb*:
 assumes M : $\bigwedge i. i \in I \implies \text{prob-space } (M \ i)$
 assumes J : $J \subseteq I$ *finite* J and A : $\bigwedge i. i \in J \implies A \ i \in \text{sets } (M \ i)$
 shows $\text{emeasure } (PiM \ I \ M) \ (\text{prod-emb } I \ M \ J \ (PiE \ J \ A)) = (\prod_{i \in J}. \text{emeasure } (M \ i) \ (A \ i))$
proof –
 let $?M = \lambda i. \text{if } i \in I \ \text{then } M \ i \ \text{else } \text{count-space } \{\text{undefined}\}$
 interpret M' : *prob-space* $?M \ i$ for i
 using M by (*cases $i \in I$*) (*auto intro!: prob-spaceI*)
 interpret P : *product-prob-space* $?M \ I$
 by *unfold-locales*
 have $\text{emeasure } (PiM \ I \ M) \ (\text{prod-emb } I \ M \ J \ (PiE \ J \ A)) = \text{emeasure } (PiM \ I \ ?M) \ (P.\text{emb } I \ J \ (PiE \ J \ A))$
 by (*auto simp: prod-emb-def PiE-iff intro!: arg-cong2[where $f = \text{emeasure}$]*)
PiM-cong)
 also have $\dots = (\prod_{i \in J}. \text{emeasure } (M \ i) \ (A \ i))$
 using $J \ A$ by (*subst P.emeasure-PiM-emb[OF J]*) (*auto intro!: prod.cong*)
 finally show *?thesis* .
 qed

lemma *distr-pair-PiM-eq-PiM*:
 fixes $i' :: 'i$ and $I :: 'i \ \text{set}$ and $M :: 'i \Rightarrow 'a \ \text{measure}$

assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M\ i) \text{ prob-space } (M\ i')$
shows $\text{distr } (M\ i' \otimes_M (\prod_{M\ i \in I. M\ i})) (\prod_{M\ i \in \text{insert } i' I. M\ i}) (\lambda(x, X). X(i' := x)) =$
 $(\prod_{M\ i \in \text{insert } i' I. M\ i}) (\text{is } ?L = -)$
proof (*rule measure-eqI-PiM-infinite[symmetric, OF refl]*)
interpret M' : *prob-space* $M\ i'$ **by fact**
interpret I : *prob-space* $(\prod_{M\ i \in I. M\ i})$
using M **by** (*intro prob-space-PiM*) *auto*
interpret I' : *prob-space* $(\prod_{M\ i \in \text{insert } i' I. M\ i})$
using M **by** (*intro prob-space-PiM*) *auto*
show *finite-measure* $(\prod_{M\ i \in \text{insert } i' I. M\ i})$
by *unfold-locales*
fix $J\ A$ **assume** J : *finite* $J\ J \subseteq \text{insert } i' I$ **and** A : $\bigwedge i. i \in J \implies A\ i \in \text{sets } (M\ i)$
let $?X = \text{prod-emb } (\text{insert } i' I)\ M\ J\ (Pi_E\ J\ A)$
have $Pi_M (\text{insert } i' I)\ M\ ?X = (\prod_{i \in J. M\ i\ (A\ i)})$
using $M\ J\ A$ **by** (*intro emeasure-PiM-emb*) *auto*
also have $\dots = M\ i'$ (*if* $i' \in J$ *then* $(A\ i')$ *else* $\text{space } (M\ i')$) $*$ $(\prod_{i \in J - \{i'\}. M\ i\ (A\ i)})$
using *prod.insert-remove[of J λi. M i (A i) i'] J M'.emeasure-space-1*
by (*cases* $i' \in J$) (*auto simp: insert-absorb*)
also have $(\prod_{i \in J - \{i'\}. M\ i\ (A\ i)}) = Pi_M\ I\ M\ (\text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A))$
using $M\ J\ A$ **by** (*intro emeasure-PiM-emb[symmetric]*) *auto*
also have $M\ i'$ (*if* $i' \in J$ *then* $(A\ i')$ *else* $\text{space } (M\ i')$) $*$ $\dots =$
 $(M\ i' \otimes_M Pi_M\ I\ M) ((\text{if } i' \in J \text{ then } (A\ i') \text{ else } \text{space } (M\ i')) \times \text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A))$
using $J\ A$ **by** (*intro I.emeasure-pair-measure-Times[symmetric] sets-PiM-I*) *auto*
also have $((\text{if } i' \in J \text{ then } (A\ i') \text{ else } \text{space } (M\ i')) \times \text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A)) =$
 $(\lambda(x, X). X(i' := x)) - ' ?X \cap \text{space } (M\ i' \otimes_M Pi_M\ I\ M)$
using A [*of* i' , *THEN sets.sets-into-space*] **unfolding** *set-eq-iff*
by (*simp add: prod-emb-def space-pair-measure space-PiM PiE-fun-upd ac-simps cong: conj-cong*)
(auto simp add: Pi-iff Ball-def all-conj-distrib)
finally show $Pi_M (\text{insert } i' I)\ M\ ?X = ?L\ ?X$
using $J\ A$ **by** (*simp add: emeasure-distr*)
qed *simp*

lemma *distr-PiM-reindex*:

assumes $M: \bigwedge i. i \in K \implies \text{prob-space } (M\ i)$
assumes f : *inj-on* $f\ I\ f \in I \rightarrow K$
shows $\text{distr } (Pi_M\ K\ M) (\prod_{M\ i \in I. M\ (f\ i)}) (\lambda\omega. \lambda n \in I. \omega\ (f\ n)) = (\prod_{M\ i \in I. M\ (f\ i)})$
(is distr ?K ?I ?t = ?I)
proof (*rule measure-eqI-PiM-infinite[symmetric, OF refl]*)
interpret *prob-space* $?I$
using $f\ M$ **by** (*intro prob-space-PiM*) *auto*

show *finite-measure ?I*
by *unfold-locales*
fix $A J$ **assume** J : *finite* $J J \subseteq I$ **and** A : $\bigwedge i. i \in J \implies A i \in \text{sets } (M (f i))$
have [*simp*]: $i \in J \implies \text{the-inv-into } I f (f i) = i$ **for** i
using $J f$ **by** (*intro the-inv-into-f-f*) *auto*
have $?I$ (*prod-emb* $I (\lambda i. M (f i)) J (Pi_E J A)$) = $(\prod_{j \in J}. M (f j) (A j))$
using $f J A$ **by** (*intro emeasure-PiM-emb M*) *auto*
also have $\dots = (\prod_{j \in f'J}. M j (A (\text{the-inv-into } I f j)))$
using $f J$ **by** (*subst prod.reindex*) (*auto intro!*: *prod.cong intro: inj-on-subset simp: the-inv-into-f-f*)
also have $\dots = ?K$ (*prod-emb* $K M (f'J) (\prod_{j \in f'J}. A (\text{the-inv-into } I f j))$)
using $f J A$ **by** (*intro emeasure-PiM-emb[symmetric] M*) (*auto simp: the-inv-into-f-f*)
also have *prod-emb* $K M (f'J) (\prod_{j \in f'J}. A (\text{the-inv-into } I f j)) = ?t - ' \text{prod-emb}$
 $I (\lambda i. M (f i)) J (Pi_E J A) \cap \text{space } ?K$
using $f J A$ **by** (*auto simp: prod-emb-def space-PiM Pi-iff PiE-iff Int-absorb1*)
also have $?K \dots = \text{distr } ?K ?I ?t$ (*prod-emb* $I (\lambda i. M (f i)) J (Pi_E J A)$)
using $f J A$ **by** (*intro emeasure-distr[symmetric] sets-PiM-I*) (*auto simp: Pi-iff*)
finally show $?I$ (*prod-emb* $I (\lambda i. M (f i)) J (Pi_E J A)$) = *distr* $?K ?I ?t$
(*prod-emb* $I (\lambda i. M (f i)) J (Pi_E J A)$) .
qed *simp*

lemma *distr-PiM-component*:

assumes M : $\bigwedge i. i \in I \implies \text{prob-space } (M i)$
assumes $i \in I$
shows *distr* ($Pi_M I M$) ($M i$) $(\lambda \omega. \omega i) = M i$
proof –
have $*$: $(\lambda \omega. \omega i) - ' A \cap \text{space } (Pi_M I M) = \text{prod-emb } I M \{i\} (\prod_{i' \in \{i\}}. A)$
for A
by (*auto simp: prod-emb-def space-PiM*)
show *?thesis*
apply (*intro measure-eqI*)
apply (*auto simp add: emeasure-distr <i∈I> * emeasure-PiM-emb M*)
apply (*subst emeasure-PiM-emb*)
apply (*simp-all add: M <i∈I>*)
done

qed

lemma *AE-PiM-component*:

$(\bigwedge i. i \in I \implies \text{prob-space } (M i)) \implies i \in I \implies \text{AE } x \text{ in } M i. P x \implies \text{AE } x \text{ in}$
 $Pi_M I M. P (x i)$
using *AE-distrD*[*of* $\lambda x. x i Pi_M I M M i$]
by (*subst (asm) distr-PiM-component*[*of* $I - i$]) (*auto intro: AE-distrD*[*of* $\lambda x. x i$
– – P])

lemma *decseq-emb-PiE*:

$\text{incseq } J \implies \text{decseq } (\lambda i. \text{prod-emb } I M (J i) (\prod_{j \in J} i. X j))$
by (*fastforce simp: decseq-def prod-emb-def incseq-def Pi-iff*)

9.1 Sequence space

definition *comb-seq* :: $\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a)$ where
 $\text{comb-seq } i \ \omega \ \omega' \ j = (\text{if } j < i \text{ then } \omega \ j \text{ else } \omega' (j - i))$

lemma *split-comb-seq*: $P (\text{comb-seq } i \ \omega \ \omega' \ j) \longleftrightarrow (j < i \longrightarrow P (\omega \ j)) \wedge (\forall k. j = i + k \longrightarrow P (\omega' \ k))$

by (*auto simp: comb-seq-def not-less*)

lemma *split-comb-seq-asm*: $P (\text{comb-seq } i \ \omega \ \omega' \ j) \longleftrightarrow \neg ((j < i \wedge \neg P (\omega \ j)) \vee (\exists k. j = i + k \wedge \neg P (\omega' \ k)))$

by (*auto simp: comb-seq-def*)

lemma *measurable-comb-seq*:

$(\lambda(\omega, \omega'). \text{comb-seq } i \ \omega \ \omega') \in \text{measurable } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M)) (\prod_M i \in \text{UNIV}. M)$

proof (*rule measurable-PiM-single*)

show $(\lambda(\omega, \omega'). \text{comb-seq } i \ \omega \ \omega') \in \text{space } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M)) \rightarrow (\text{UNIV} \rightarrow_E \text{space } M)$

by (*auto simp: space-pair-measure space-PiM PiE-iff split: split-comb-seq*)

fix $j :: \text{nat}$ **and** A **assume** $A: A \in \text{sets } M$

then have $*$: $\{\omega \in \text{space } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M)). \text{case-prod } (\text{comb-seq } i) \ \omega \ j \in A\} =$

$(\text{if } j < i \text{ then } \{\omega \in \text{space } (\prod_M i \in \text{UNIV}. M). \omega \ j \in A\} \times \text{space } (\prod_M i \in \text{UNIV}. M)$

$\text{else } \text{space } (\prod_M i \in \text{UNIV}. M) \times \{\omega \in \text{space } (\prod_M i \in \text{UNIV}. M). \omega (j - i) \in A\})$

by (*auto simp: space-PiM space-pair-measure comb-seq-def dest: sets.sets-into-space*)

show $\{\omega \in \text{space } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M)). \text{case-prod } (\text{comb-seq } i) \ \omega \ j \in A\} \in \text{sets } ((\prod_M i \in \text{UNIV}. M) \otimes_M (\prod_M i \in \text{UNIV}. M))$

unfolding $*$ **by** (*auto simp: A intro!: sets-Collect-single*)

qed

lemma *measurable-comb-seq'*[*measurable (raw)*]:

assumes $f: f \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$ **and** $g: g \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$

shows $(\lambda x. \text{comb-seq } i (f \ x) (g \ x)) \in \text{measurable } N (\prod_M i \in \text{UNIV}. M)$

using *measurable-compose[OF measurable-Pair[OF f g] measurable-comb-seq]* **by** *simp*

lemma *comb-seq-0*: $\text{comb-seq } 0 \ \omega \ \omega' = \omega'$

by (*auto simp add: comb-seq-def*)

lemma *comb-seq-Suc*: $\text{comb-seq } (\text{Suc } n) \ \omega \ \omega' = \text{comb-seq } n \ \omega \ (\text{case-nat } (\omega \ n) \ \omega')$

by (*auto simp add: comb-seq-def not-less less-Suc-eq le-imp-diff-is-add intro!: ext split: nat.split*)

lemma *comb-seq-Suc-0*[*simp*]: $\text{comb-seq } (\text{Suc } 0) \ \omega = \text{case-nat } (\omega \ 0)$

by (*intro ext*) (*simp add: comb-seq-Suc comb-seq-0*)

lemma *comb-seq-less*: $i < n \implies \text{comb-seq } n \ \omega \ \omega' \ i = \omega \ i$
by (*auto split: split-comb-seq*)

lemma *comb-seq-add*: $\text{comb-seq } n \ \omega \ \omega' \ (i + n) = \omega' \ i$
by (*auto split: nat.split split-comb-seq*)

lemma *case-nat-comb-seq*: $\text{case-nat } s' \ (\text{comb-seq } n \ \omega \ \omega') \ (i + n) = \text{case-nat}$
 $(\text{case-nat } s' \ \omega \ n) \ \omega' \ i$
by (*auto split: nat.split split-comb-seq*)

lemma *case-nat-comb-seq'*:
 $\text{case-nat } s \ (\text{comb-seq } i \ \omega \ \omega') = \text{comb-seq } (\text{Suc } i) \ (\text{case-nat } s \ \omega) \ \omega'$
by (*auto split: split-comb-seq nat.split*)

locale *sequence-space* = *product-prob-space* $\lambda i. M \ UNIV :: \text{nat set}$ **for** M
begin

abbreviation $S \equiv \prod_M \ i \in UNIV :: \text{nat set}. M$

lemma *infprod-in-sets*[*intro*]:
fixes $E :: \text{nat} \Rightarrow 'a \ \text{set}$ **assumes** $E: \bigwedge i. E \ i \in \text{sets } M$
shows $Pi \ UNIV \ E \in \text{sets } S$
proof –
have $Pi \ UNIV \ E = (\bigcap i. \text{emb } UNIV \ \{..i\} \ (\prod_E \ j \in \{..i\}. E \ j))$
using $E \ E$ [*THEN sets.sets-into-space*]
by (*auto simp: prod-emb-def Pi-iff extensional-def*)
with E **show** *?thesis* **by** *auto*
qed

lemma *measure-PiM-countable*:
fixes $E :: \text{nat} \Rightarrow 'a \ \text{set}$ **assumes** $E: \bigwedge i. E \ i \in \text{sets } M$
shows $(\lambda n. \prod_{i \leq n}. \text{measure } M \ (E \ i)) \longrightarrow \text{measure } S \ (Pi \ UNIV \ E)$
proof –
let $?E = \lambda n. \text{emb } UNIV \ \{..n\} \ (Pi_E \ \{..n\} \ E)$
have $\bigwedge n. (\prod_{i \leq n}. \text{measure } M \ (E \ i)) = \text{measure } S \ (?E \ n)$
using E **by** (*simp add: measure-PiM-emb*)
moreover **have** $Pi \ UNIV \ E = (\bigcap n. ?E \ n)$
using $E \ E$ [*THEN sets.sets-into-space*]
by (*auto simp: prod-emb-def extensional-def Pi-iff*)
moreover **have** $\text{range } ?E \subseteq \text{sets } S$
using E **by** *auto*
moreover **have** *decseq* $?E$
by (*auto simp: prod-emb-def Pi-iff decseq-def*)
ultimately **show** *?thesis*
by (*simp add: finite-Lim-measure-decseq*)
qed

lemma *nat-eq-diff-eq*:
fixes $a \ b \ c :: \text{nat}$

shows $c \leq b \implies a = b - c \iff a + c = b$
by *auto*

lemma *PiM-comb-seq*:

distr $(S \otimes_M S) S (\lambda(\omega, \omega'). \text{comb-seq } i \omega \omega') = S$ (**is** $?D = -$)

proof (*rule PiM-eq*)

let $?I = \text{UNIV}::\text{nat set}$ **and** $?M = \lambda n. M$

let *distr* - - $?f = ?D$

fix $J E$ **assume** $J: \text{finite } J \ J \subseteq ?I \ \bigwedge j. j \in J \implies E j \in \text{sets } M$

let $?X = \text{prod-emb } ?I ?M J (\prod_E j \in J. E j)$

have $\bigwedge j x. j \in J \implies x \in E j \implies x \in \text{space } M$

using $J(\beta)[\text{THEN sets.sets-into-space}]$ **by** (*auto simp: space-PiM Pi-iff subset-eq*)

with J **have** $?f - ' ?X \cap \text{space } (S \otimes_M S) =$

$(\text{prod-emb } ?I ?M (J \cap \{..<i\})) (\prod_E j \in J \cap \{..<i\}. E j) \times$

$(\text{prod-emb } ?I ?M ((+) i) - ' J) (\prod_E j \in ((+) i) - ' J. E (i + j))$ (**is** $- = ?E \times ?F$)

by (*auto simp: space-pair-measure space-PiM prod-emb-def all-conj-distrib PiE-iff split: split-comb-seq split-comb-seq-asm*)

then have *emeasure* $?D ?X = \text{emeasure } (S \otimes_M S) (?E \times ?F)$

by (*subst emeasure-distr[OF measurable-comb-seq]*)

(*auto intro!: sets-PiM-I simp: split-beta' J*)

also have $\dots = \text{emeasure } S ?E * \text{emeasure } S ?F$

using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I finite-vimageI simp: inj-on-def*)

also have *emeasure* $S ?F = (\prod j \in ((+) i) - ' J. \text{emeasure } M (E (i + j)))$

using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI inj-on-def*)

also have $\dots = (\prod j \in J - (J \cap \{..<i\}). \text{emeasure } M (E j))$

by (*rule prod.reindex-cong [of $\lambda x. x - i$]*)

(*auto simp: image-iff ac-simps nat-eq-diff-eq cong: conj-cong intro!: inj-onI*)

also have *emeasure* $S ?E = (\prod j \in J \cap \{..<i\}. \text{emeasure } M (E j))$

using J **by** (*intro emeasure-PiM-emb simp-all*)

also have $(\prod j \in J \cap \{..<i\}. \text{emeasure } M (E j)) * (\prod j \in J - (J \cap \{..<i\}). \text{emeasure } M (E j)) = (\prod j \in J. \text{emeasure } M (E j))$

by (*subst mult commute*) (*auto simp: J prod.subset-diff[symmetric]*)

finally show *emeasure* $?D ?X = (\prod j \in J. \text{emeasure } M (E j))$.

qed *simp-all*

lemma *PiM-iter*:

distr $(M \otimes_M S) S (\lambda(s, \omega). \text{case-nat } s \omega) = S$ (**is** $?D = -$)

proof (*rule PiM-eq*)

let $?I = \text{UNIV}::\text{nat set}$ **and** $?M = \lambda n. M$

let *distr* - - $?f = ?D$

fix $J E$ **assume** $J: \text{finite } J \ J \subseteq ?I \ \bigwedge j. j \in J \implies E j \in \text{sets } M$

let $?X = \text{prod-emb } ?I ?M J (\prod_E j \in J. E j)$

have $\bigwedge j x. j \in J \implies x \in E j \implies x \in \text{space } M$

using $J(\beta)[\text{THEN sets.sets-into-space}]$ **by** (*auto simp: space-PiM Pi-iff sub-*

set-eq
with J **have** $?f - ' ?X \cap \text{space } (M \otimes_M S) = (\text{if } 0 \in J \text{ then } E 0 \text{ else } \text{space } M)$
 \times
 (*prod-emb* $?I ?M (\text{Suc } - ' J) (\prod_{E j \in \text{Suc } - ' J} E (\text{Suc } j))$) (**is** $- = ?E \times ?F$)
by (*auto simp: space-pair-measure space-PiM PiE-iff prod-emb-def all-conj-distrib split: nat.split nat.split-asm*)
then have *emeasure* $?D ?X = \text{emeasure } (M \otimes_M S) (?E \times ?F)$
by (*subst emeasure-distr*)
 (*auto intro!: sets-PiM-I simp: split-beta' J*)
also have $\dots = \text{emeasure } M ?E * \text{emeasure } S ?F$
using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I finite-vimageI*)
also have *emeasure* $S ?F = (\prod_{j \in \text{Suc } - ' J} \text{emeasure } M (E (\text{Suc } j)))$
using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI*)
also have $\dots = (\prod_{j \in J - \{0\}} \text{emeasure } M (E j))$
by (*rule prod.reindex-cong [of $\lambda x. x - 1$]*)
 (*auto simp: image-iff nat-eq-diff-eq ac-simps cong: conj-cong intro!: inj-onI*)
also have *emeasure* $M ?E * (\prod_{j \in J - \{0\}} \text{emeasure } M (E j)) = (\prod_{j \in J} \text{emeasure } M (E j))$
emeasure $M (E j)$
by (*auto simp: M.emeasure-space-1 prod.remove J*)
finally show *emeasure* $?D ?X = (\prod_{j \in J} \text{emeasure } M (E j))$.
qed *simp-all*

end

lemma *PiM-return*:

assumes *finite I*
assumes [*measurable*]: $\bigwedge i. i \in I \implies \{a i\} \in \text{sets } (M i)$
shows $\text{PiM } I (\lambda i. \text{return } (M i) (a i)) = \text{return } (\text{PiM } I M) (\text{restrict } a I)$
proof –
have [*simp*]: $a i \in \text{space } (M i)$ **if** $i \in I$ **for** i
using *assms(2)[OF that]* **by** (*meson insert-subset sets.sets-into-space*)
interpret *prob-space* $\text{PiM } I (\lambda i. \text{return } (M i) (a i))$
by (*intro prob-space-PiM prob-space-return*) *auto*
have $\text{AE } x \text{ in } \text{PiM } I (\lambda i. \text{return } (M i) (a i)). \forall i \in I. x i = \text{restrict } a I i$
by (*intro eventually-ball-finite ballI AE-PiM-component prob-space-return assms*)
 (*auto simp: AE-return*)
moreover have $\text{AE } x \text{ in } \text{PiM } I (\lambda i. \text{return } (M i) (a i)). x \in \text{space } (\text{PiM } I (\lambda i. \text{return } (M i) (a i)))$
by *simp*
ultimately have $\text{AE } x \text{ in } \text{PiM } I (\lambda i. \text{return } (M i) (a i)). x = \text{restrict } a I$
by *eventually-elim* (*auto simp: fun-eq-iff space-PiM*)
hence $\text{PiM } I (\lambda i. \text{return } (M i) (a i)) = \text{return } (\text{PiM } I (\lambda i. \text{return } (M i) (a i)))$
 (*restrict a I*)
by (*rule AE-eq-constD*)
also have $\dots = \text{return } (\text{PiM } I M) (\text{restrict } a I)$
by (*intro return-cong sets-PiM-cong*) *auto*
finally show *thesis* .
qed

```

lemma distr-PiM-finite-prob-space':
  assumes fin: finite I
  assumes  $\bigwedge i. i \in I \implies \text{prob-space } (M\ i)$ 
  assumes  $\bigwedge i. i \in I \implies \text{prob-space } (M'\ i)$ 
  assumes [measurable]:  $\bigwedge i. i \in I \implies f \in \text{measurable } (M\ i) (M'\ i)$ 
  shows  $\text{distr } (PiM\ I\ M) (PiM\ I\ M') (\text{compose } I\ f) = PiM\ I\ (\lambda i. \text{distr } (M\ i) (M'\ i) f)$ 
proof –
  define N where  $N = (\lambda i. \text{if } i \in I \text{ then } M\ i \text{ else return } (\text{count-space } UNIV))$ 
  undefined
  define N' where  $N' = (\lambda i. \text{if } i \in I \text{ then } M'\ i \text{ else return } (\text{count-space } UNIV))$ 
  undefined
  have [simp]:  $PiM\ I\ N = PiM\ I\ M\ PiM\ I\ N' = PiM\ I\ M'$ 
    by (intro PiM-cong; simp add: N-def N'-def)+

  have  $\text{distr } (PiM\ I\ N) (PiM\ I\ N') (\text{compose } I\ f) = PiM\ I\ (\lambda i. \text{distr } (N\ i) (N'\ i) f)$ 
proof (rule distr-PiM-finite-prob-space)
  show product-prob-space N
    by (rule product-prob-spaceI) (auto simp: N-def intro!: prob-space-return assms)
  show product-prob-space N'
    by (rule product-prob-spaceI) (auto simp: N'-def intro!: prob-space-return assms)
  qed (auto simp: N-def N'-def fin)
  also have  $PiM\ I\ (\lambda i. \text{distr } (N\ i) (N'\ i) f) = PiM\ I\ (\lambda i. \text{distr } (M\ i) (M'\ i) f)$ 
    by (intro PiM-cong) (simp-all add: N-def N'-def)
  finally show ?thesis by simp
qed
end

```

10 Independent families of events, event sets, and random variables

```

theory Independent-Family
  imports Infinite-Product-Measure
begin

```

```

definition (in prob-space)
  indep-sets  $F\ I \longleftrightarrow (\forall i \in I. F\ i \subseteq \text{events}) \wedge$ 
     $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow (\forall A \in Pi\ J\ F. \text{prob } (\bigcap_{j \in J}. A\ j) = (\prod_{j \in J}. \text{prob } (A\ j))))$ 

```

```

definition (in prob-space)
  indep-set  $A\ B \longleftrightarrow \text{indep-sets } (\text{case-bool } A\ B)\ UNIV$ 

```

```

definition (in prob-space)

```

indep-events-def-alt: $\text{indep-events } A \ I \longleftrightarrow \text{indep-sets } (\lambda i. \{A \ i\}) \ I$

lemma (in *prob-space*) *indep-events-def*:
 $\text{indep-events } A \ I \longleftrightarrow (A \ I \subseteq \text{events}) \wedge$
 $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{prob } (\bigcap_{j \in J}. A \ j) = (\prod_{j \in J}. \text{prob } (A \ j)))$
unfolding *indep-events-def-alt indep-sets-def*
apply (*simp add: Ball-def Pi-iff image-subset-iff-funcset*)
apply (*intro conj-cong refl arg-cong[where f=All] ext imp-cong*)
apply *auto*
done

lemma (in *prob-space*) *indep-eventsI*:
 $(\bigwedge i. i \in I \Longrightarrow F \ i \in \text{sets } M) \Longrightarrow (\bigwedge J. J \subseteq I \Longrightarrow \text{finite } J \Longrightarrow J \neq \{\} \Longrightarrow \text{prob}$
 $(\bigcap_{i \in J}. F \ i) = (\prod_{i \in J}. \text{prob } (F \ i))) \Longrightarrow \text{indep-events } F \ I$
by (*auto simp: indep-events-def*)

definition (in *prob-space*)
 $\text{indep-event } A \ B \longleftrightarrow \text{indep-events } (\text{case-bool } A \ B) \ \text{UNIV}$

lemma (in *prob-space*) *indep-sets-cong*:
 $I = J \Longrightarrow (\bigwedge i. i \in I \Longrightarrow F \ i = G \ i) \Longrightarrow \text{indep-sets } F \ I \longleftrightarrow \text{indep-sets } G \ J$
by (*simp add: indep-sets-def, intro conj-cong all-cong imp-cong ball-cong*) *blast+*

lemma (in *prob-space*) *indep-events-finite-index-events*:
 $\text{indep-events } F \ I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-events } F \ J)$
by (*auto simp: indep-events-def*)

lemma (in *prob-space*) *indep-sets-finite-index-sets*:
 $\text{indep-sets } F \ I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F \ J)$
proof (*intro iffI allI impI*)
assume *: $\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F \ J$
show $\text{indep-sets } F \ I$ **unfolding** *indep-sets-def*
proof (*intro conjI ballI allI impI*)
fix *i* **assume** $i \in I$
with * [*THEN spec, of {i}*] **show** $F \ i \subseteq \text{events}$
by (*auto simp: indep-sets-def*)
qed (*insert *, auto simp: indep-sets-def*)
qed (*auto simp: indep-sets-def*)

lemma (in *prob-space*) *indep-sets-mono-index*:
 $J \subseteq I \Longrightarrow \text{indep-sets } F \ I \Longrightarrow \text{indep-sets } F \ J$
unfolding *indep-sets-def* **by** *auto*

lemma (in *prob-space*) *indep-sets-mono-sets*:
assumes *indep*: $\text{indep-sets } F \ I$
assumes *mono*: $\bigwedge i. i \in I \Longrightarrow G \ i \subseteq F \ i$
shows $\text{indep-sets } G \ I$
proof –
have $(\forall i \in I. F \ i \subseteq \text{events}) \Longrightarrow (\forall i \in I. G \ i \subseteq \text{events})$

using *mono* **by** *auto*
moreover have $\bigwedge A J. J \subseteq I \implies A \in (\prod_{j \in J}. G j) \implies A \in (\prod_{j \in J}. F j)$
using *mono* **by** (*auto simp: Pi-iff*)
ultimately show *?thesis*
using *indep* **by** (*auto simp: indep-sets-def*)
qed

lemma (*in prob-space*) *indep-sets-mono*:
assumes *indep: indep-sets F I*
assumes *mono: J ⊆ I ∧ i. i ∈ J ⟹ G i ⊆ F i*
shows *indep-sets G J*
apply (*rule indep-sets-mono-sets*)
apply (*rule indep-sets-mono-index*)
apply (*fact +*)
done

lemma (*in prob-space*) *indep-setsI*:
assumes $\bigwedge i. i \in I \implies F i \subseteq \text{events}$
and $\bigwedge A J. J \neq \{\} \implies J \subseteq I \implies \text{finite } J \implies (\forall j \in J. A j \in F j) \implies \text{prob}$
 $(\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j))$
shows *indep-sets F I*
using *assms unfolding indep-sets-def* **by** (*auto simp: Pi-iff*)

lemma (*in prob-space*) *indep-setsD*:
assumes *indep-sets F I* **and** $J \subseteq I \ J \neq \{\} \ \text{finite } J \ \forall j \in J. A j \in F j$
shows $\text{prob } (\bigcap_{j \in J}. A j) = (\prod_{j \in J}. \text{prob } (A j))$
using *assms unfolding indep-sets-def* **by** *auto*

lemma (*in prob-space*) *indep-setI*:
assumes *ev: A ⊆ events B ⊆ events*
and *indep: ∧ a b. a ∈ A ⟹ b ∈ B ⟹ prob (a ∩ b) = prob a * prob b*
shows *indep-set A B*
unfolding *indep-set-def*
proof (*rule indep-setsI*)
fix *F J* **assume** $J \neq \{\} \ J \subseteq \text{UNIV}$
and $F: \forall j \in J. F j \in (\text{case } j \text{ of True} \Rightarrow A \mid \text{False} \Rightarrow B)$
have $J \in \text{Pow UNIV}$ **by** *auto*
with $F \langle J \neq \{\} \rangle$ *indep[of F True F False]*
show $\text{prob } (\bigcap_{j \in J}. F j) = (\prod_{j \in J}. \text{prob } (F j))$
unfolding *UNIV-bool Pow-insert* **by** (*auto simp: ac-simps*)
qed (*auto split: bool.split simp: ev*)

lemma (*in prob-space*) *indep-setD*:
assumes *indep: indep-set A B* **and** *ev: a ∈ A b ∈ B*
shows $\text{prob } (a \cap b) = \text{prob } a * \text{prob } b$
using *indep[unfolded indep-set-def, THEN indep-setsD, of UNIV case-bool a b]*
ev
by (*simp add: ac-simps UNIV-bool*)

lemma (in *prob-space*)
assumes *indep*: *indep-set* $A B$
shows *indep-setD-ev1*: $A \subseteq \text{events}$
and *indep-setD-ev2*: $B \subseteq \text{events}$
using *indep unfolding indep-set-def indep-sets-def UNIV-bool* **by** *auto*

lemma (in *prob-space*) *indep-sets-Dynkin*:
assumes *indep*: *indep-sets* $F I$
shows *indep-sets* ($\lambda i. \text{Dynkin (space } M) (F i) I$) I
(is *indep-sets ?F I*)

proof (*subst indep-sets-finite-index-sets, intro allI impI ballI*)
fix J **assume** *finite* $J J \subseteq I J \neq \{\}$
with *indep* **have** *indep-sets* $F J$
by (*subst (asm) indep-sets-finite-index-sets*) *auto*
{ **fix** $J K$ **assume** *indep-sets* $F K$
let $?G = \lambda S i. \text{if } i \in S \text{ then } ?F i \text{ else } F i$
assume *finite* $J J \subseteq K$
then have *indep-sets* ($?G J$) K
proof *induct*
case (*insert j J*)
moreover define G **where** $G = ?G J$
ultimately have $G: \text{indep-sets } G K \wedge i. i \in K \implies G i \subseteq \text{events}$ **and** $j \in K$
by (*auto simp: indep-sets-def*)
let $?D = \{E \in \text{events}. \text{indep-sets } (G(j := \{E\})) K\}$
{ **fix** X **assume** $X: X \in \text{events}$
assume *indep*: $\bigwedge J A. J \neq \{\} \implies J \subseteq K \implies \text{finite } J \implies j \notin J \implies (\forall i \in J. A i \in G i)$
 $\implies \text{prob } ((\bigcap i \in J. A i) \cap X) = \text{prob } X * (\prod i \in J. \text{prob } (A i))$
have *indep-sets* ($G(j := \{X\}) K$)
proof (*rule indep-setsI*)
fix i **assume** $i \in K$ **then show** ($G(j := \{X\}) i \subseteq \text{events}$)
using $G X$ **by** *auto*
next
fix $A J$ **assume** $J: J \neq \{\} J \subseteq K \text{ finite } J \forall i \in J. A i \in (G(j := \{X\})) i$
show $\text{prob } (\bigcap j \in J. A j) = (\prod j \in J. \text{prob } (A j))$
proof *cases*
assume $j \in J$
with J **have** $A j = X$ **by** *auto*
show *?thesis*
proof *cases*
assume $J = \{j\}$ **then show** *?thesis* **by** *simp*
next
assume $J \neq \{j\}$
have $\text{prob } (\bigcap i \in J. A i) = \text{prob } ((\bigcap i \in J - \{j\}. A i) \cap X)$
using $\langle j \in J \rangle \langle A j = X \rangle$ **by** (*auto intro!: arg-cong[where f=prob]*)
split: if-split-asm
also have $\dots = \text{prob } X * (\prod i \in J - \{j\}. \text{prob } (A i))$
proof (*rule indep*)
show $J - \{j\} \neq \{\} J - \{j\} \subseteq K \text{ finite } (J - \{j\}) j \notin J - \{j\}$


```

    using J ⟨J ≠ {j}⟩ ⟨j ∈ J⟩ by auto
    show ∀ i ∈ J - {j}. A i ∈ G i
    using J by auto
  qed
  also have ... = prob (A j) * (∏ i ∈ J - {j}. prob (A i))
    using ⟨A j = X⟩ by simp
  also have ... = (∏ i ∈ J. prob (A i))
    unfolding prod.insert-remove[OF ⟨finite J⟩, symmetric, of λi. prob
(A i)]
    using ⟨j ∈ J⟩ by (simp add: insert-absorb)
    finally show ?thesis .
  qed
next
  assume j ∉ J
  with J have ∀ i ∈ J. A i ∈ G i by (auto split: if-split-asm)
  with J show ?thesis
    by (intro indep-setsD[OF G(1)]) auto
  qed
qed }
note indep-sets-insert = this
have Dynkin-system (space M) ?D
proof (rule Dynkin-systemI', simp-all cong del: indep-sets-cong, safe)
  show indep-sets (G(j := {{}))) K
    by (rule indep-sets-insert) auto
next
fix X assume X: X ∈ events and G': indep-sets (G(j := {X})) K
show indep-sets (G(j := {space M - X})) K
proof (rule indep-sets-insert)
  fix J A assume J: J ≠ {} J ⊆ K finite J j ∉ J and A: ∀ i ∈ J. A i ∈ G i
  then have A-sets: ∧ i. i ∈ J ⇒ A i ∈ events
    using G by auto
  have prob ((∩ j ∈ J. A j) ∩ (space M - X)) =
    prob ((∩ j ∈ J. A j) - (∩ i ∈ insert j J. (A(j := X)) i))
    using A-sets sets.sets-into-space[of - M] X ⟨J ≠ {}⟩
    by (auto intro!: arg-cong[where f=prob] split: if-split-asm)
  also have ... = prob (∩ j ∈ J. A j) - prob (∩ i ∈ insert j J. (A(j := X)) i)
    using J ⟨J ≠ {}⟩ ⟨j ∉ J⟩ A-sets X sets.sets-into-space
    by (auto intro!: finite-measure-Diff sets.finite-INT split: if-split-asm)
  finally have prob ((∩ j ∈ J. A j) ∩ (space M - X)) =
    prob (∩ j ∈ J. A j) - prob (∩ i ∈ insert j J. (A(j := X)) i) .
  moreover {
    have prob (∩ j ∈ J. A j) = (∏ j ∈ J. prob (A j))
      using J A ⟨finite J⟩ by (intro indep-setsD[OF G(1)]) auto
    then have prob (∩ j ∈ J. A j) = prob (space M) * (∏ i ∈ J. prob (A i))
      using prob-space by simp }
  moreover {
    have prob (∩ i ∈ insert j J. (A(j := X)) i) = (∏ i ∈ insert j J. prob ((A(j
:= X)) i))
      using J A ⟨j ∈ K⟩ by (intro indep-setsD[OF G']) auto

```

```

then have prob ( $\bigcap i \in \text{insert } j \ J. (A(j := X)) \ i$ ) = prob  $X * (\prod i \in J. \text{prob } (A \ i))$ 
using  $\langle \text{finite } J \rangle \langle j \notin J \rangle$  by (auto intro!: prod.cong) }
ultimately have prob ( $(\bigcap j \in J. A \ j) \cap (\text{space } M - X)$ ) = (prob (space
 $M) - \text{prob } X) * (\prod i \in J. \text{prob } (A \ i))$ 
by (simp add: field-simps)
also have ... = prob (space  $M - X) * (\prod i \in J. \text{prob } (A \ i))$ 
using  $X \ A$  by (simp add: finite-measure-compl)
finally show prob ( $(\bigcap j \in J. A \ j) \cap (\text{space } M - X)$ ) = prob (space  $M -$ 
 $X) * (\prod i \in J. \text{prob } (A \ i))$  .
qed (insert  $X$ , auto)
next
fix  $F :: \text{nat} \Rightarrow 'a \ \text{set}$  assume disj: disjoint-family  $F$  and range  $F \subseteq ?D$ 
then have  $F: \bigwedge i. F \ i \in \text{events} \ \bigwedge i. \text{indep-sets } (G(j := \{F \ i\})) \ K$  by auto
show indep-sets ( $G(j := \{\bigcup k. F \ k\})$ )  $K$ 
proof (rule indep-sets-insert)
fix  $J \ A$  assume  $J: j \notin J \ J \neq \{\}$   $J \subseteq K$  finite  $J$  and  $A: \forall i \in J. A \ i \in G \ i$ 
then have  $A\text{-sets}: \bigwedge i. i \in J \implies A \ i \in \text{events}$ 
using  $G$  by auto
have prob ( $(\bigcap j \in J. A \ j) \cap (\bigcup k. F \ k)$ ) = prob ( $\bigcup k. (\bigcap i \in \text{insert } j \ J. (A(j$ 
 $:= F \ k)) \ i)$ )
using  $\langle J \neq \{\} \rangle \langle j \notin J \rangle \langle j \in K \rangle$  by (auto intro!: arg-cong[where  $f = \text{prob}$ ]
split: if-split-asm)
moreover have ( $\lambda k. \text{prob } (\bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i)$ ) sums prob
( $\bigcup k. (\bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i)$ )
proof (rule finite-measure-UNION)
show disjoint-family ( $\lambda k. \bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i$ )
using disj by (rule disjoint-family-on-bisimulation) auto
show range ( $\lambda k. \bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i$ )  $\subseteq \text{events}$ 
using  $A\text{-sets } F \ \langle \text{finite } J \rangle \langle J \neq \{\} \rangle \langle j \notin J \rangle$  by (auto intro!: sets.Int)
qed
moreover { fix  $k$ 
from  $J \ A \ \langle j \in K \rangle$  have prob ( $\bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i$ ) = prob
( $F \ k) * (\prod i \in J. \text{prob } (A \ i))$ 
by (subst indep-setsD[OF  $F(2)$ ]) (auto intro!: prod.cong split: if-split-asm)
also have ... = prob ( $F \ k) * \text{prob } (\bigcap i \in J. A \ i)$ 
using  $J \ A \ \langle j \in K \rangle$  by (subst indep-setsD[OF  $G(1)$ ]) auto
finally have prob ( $\bigcap i \in \text{insert } j \ J. (A(j := F \ k)) \ i$ ) = prob ( $F \ k) * \text{prob}$ 
( $\bigcap i \in J. A \ i$ ) . }
ultimately have ( $\lambda k. \text{prob } (F \ k) * \text{prob } (\bigcap i \in J. A \ i)$ ) sums (prob ( $(\bigcap j \in J.$ 
 $A \ j) \cap (\bigcup k. F \ k)$ ))
by simp
moreover
have ( $\lambda k. \text{prob } (F \ k) * \text{prob } (\bigcap i \in J. A \ i)$ ) sums (prob ( $\bigcup k. F \ k) * \text{prob}$ 
( $\bigcap i \in J. A \ i$ ))
using disj  $F(1)$  by (intro finite-measure-UNION sums-mult2) auto
then have ( $\lambda k. \text{prob } (F \ k) * \text{prob } (\bigcap i \in J. A \ i)$ ) sums (prob ( $\bigcup k. F \ k) *$ 
( $\prod i \in J. \text{prob } (A \ i)$ ))
using  $J \ A \ \langle j \in K \rangle$  by (subst indep-setsD[OF  $G(1)$ , symmetric]) auto

```

```

ultimately
show prob (( $\bigcap_{j \in J}. A j$ )  $\cap$  ( $\bigcup k. F k$ )) = prob ( $\bigcup k. F k$ ) * ( $\prod_{j \in J}. prob$ 
(A j))
  by (auto dest!: sums-unique)
qed (insert F, auto)
qed (insert sets.sets-into-space, auto)
then have mono: Dynkin (space M) (G j)  $\subseteq$  {E  $\in$  events. indep-sets (G(j
:= {E})) K}
proof (rule Dynkin-system.Dynkin-subset, safe)
  fix X assume X  $\in$  G j
  then show X  $\in$  events using G  $\langle j \in K \rangle$  by auto
  from  $\langle indep-sets G K \rangle$ 
  show indep-sets (G(j := {X})) K
    by (rule indep-sets-mono-sets) (insert  $\langle X \in G j \rangle$ , auto)
qed
have indep-sets (G(j:=?D)) K
proof (rule indep-setsI)
  fix i assume i  $\in$  K then show (G(j := ?D)) i  $\subseteq$  events
    using G(2) by auto
next
fix A J assume J: J $\neq\{\}$  J  $\subseteq$  K finite J and A:  $\forall i \in J. A i \in$  (G(j := ?D))
i
  show prob ( $\bigcap_{j \in J}. A j$ ) = ( $\prod_{j \in J}. prob$  (A j))
  proof cases
    assume j  $\in$  J
    with A have indep: indep-sets (G(j := {A j})) K by auto
    from J A show ?thesis
      by (intro indep-setsD[OF indep]) auto
    next
    assume j  $\notin$  J
    with J A have  $\forall i \in J. A i \in G i$  by (auto split: if-split-asm)
    with J show ?thesis
      by (intro indep-setsD[OF G(1)]) auto
  qed
qed
then have indep-sets (G(j := Dynkin (space M) (G j))) K
  by (rule indep-sets-mono-sets) (insert mono, auto)
then show ?case
  by (rule indep-sets-mono-sets) (insert  $\langle j \in K \rangle \langle j \notin J \rangle$ , auto simp: G-def)
qed (insert  $\langle indep-sets F K \rangle$ , simp) }
from this[OF  $\langle indep-sets F J \rangle \langle finite J \rangle$  subset-refl]
show indep-sets ?F J
  by (rule indep-sets-mono-sets) auto
qed

```

lemma (in prob-space) indep-sets-sigma:

```

assumes indep: indep-sets F I
assumes stable:  $\bigwedge i. i \in I \implies Int-stable (F i)$ 
shows indep-sets ( $\lambda i. sigma-sets (space M) (F i)$ ) I

```

proof –

from *indep-sets-Dynkin*[*OF indep*]
show *?thesis*
proof (*rule indep-sets-mono-sets, subst sigma-eq-Dynkin, simp-all add: stable*)
fix *i* **assume** $i \in I$
with *indep* **have** $F\ i \subseteq \text{events}$ **by** (*auto simp: indep-sets-def*)
with *sets.sets-into-space* **show** $F\ i \subseteq \text{Pow}(\text{space } M)$ **by** *auto*
qed
qed

lemma (*in prob-space*) *indep-sets-sigma-sets-iff*:

assumes $\bigwedge i. i \in I \implies \text{Int-stable}(F\ i)$
shows $\text{indep-sets}(\lambda i. \text{sigma-sets}(\text{space } M)(F\ i))\ I \longleftrightarrow \text{indep-sets } F\ I$

proof

assume $\text{indep-sets } F\ I$ **then show** $\text{indep-sets}(\lambda i. \text{sigma-sets}(\text{space } M)(F\ i))\ I$
by (*rule indep-sets-sigma*) *fact*
next
assume $\text{indep-sets}(\lambda i. \text{sigma-sets}(\text{space } M)(F\ i))\ I$ **then show** $\text{indep-sets } F\ I$
by (*rule indep-sets-mono-sets*) (*intro subsetI sigma-sets.Basic*)
qed

definition (*in prob-space*)

indep-vars-def2: $\text{indep-vars } M'\ X\ I \longleftrightarrow$
 $(\forall i \in I. \text{random-variable}(M'\ i)(X\ i)) \wedge$
 $\text{indep-sets}(\lambda i. \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets}(M'\ i)\})\ I$

definition (*in prob-space*)

indep-var $Ma\ A\ Mb\ B \longleftrightarrow \text{indep-vars}(\text{case-bool } Ma\ Mb)(\text{case-bool } A\ B)\ \text{UNIV}$

lemma (*in prob-space*) *indep-vars-def*:

indep-vars $M'\ X\ I \longleftrightarrow$
 $(\forall i \in I. \text{random-variable}(M'\ i)(X\ i)) \wedge$
 $\text{indep-sets}(\lambda i. \text{sigma-sets}(\text{space } M)\{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets}(M'\ i)\})\ I$

unfolding *indep-vars-def2*

apply (*rule conj-cong*[*OF refl*])
apply (*rule indep-sets-sigma-sets-iff*[*symmetric*])
apply (*auto simp: Int-stable-def*)
apply (*rule-tac* $x=A \cap Aa$ **in** exI)
apply *auto*
done

lemma (*in prob-space*) *indep-var-eq*:

indep-var $S\ X\ T\ Y \longleftrightarrow$
 $(\text{random-variable } S\ X \wedge \text{random-variable } T\ Y) \wedge$
indep-set
 $(\text{sigma-sets}(\text{space } M)\{X - 'A \cap \text{space } M \mid A. A \in \text{sets } S\})$
 $(\text{sigma-sets}(\text{space } M)\{Y - 'A \cap \text{space } M \mid A. A \in \text{sets } T\})$
unfolding *indep-var-def indep-vars-def indep-set-def UNIV-bool*

by (intro arg-cong2[**where** $f=(\wedge)$] arg-cong2[**where** $f=indep\text{-}sets$] ext)
 (auto split: bool.split)

lemma (in prob-space) indep-sets2-eq:

$indep\text{-}set\ A\ B \longleftrightarrow A \subseteq events \wedge B \subseteq events \wedge (\forall a \in A. \forall b \in B. prob\ (a \cap b) = prob\ a * prob\ b)$

unfolding indep-set-def

proof (intro iffI ballI conjI)

assume indep: indep-sets (case-bool A B) UNIV

{ **fix** a b **assume** $a \in A\ b \in B$

with indep-setsD[OF indep, of UNIV case-bool a b]

show $prob\ (a \cap b) = prob\ a * prob\ b$

unfolding UNIV-bool **by** (simp add: ac-simps) }

from indep **show** $A \subseteq events\ B \subseteq events$

unfolding indep-sets-def UNIV-bool **by** auto

next

assume *: $A \subseteq events \wedge B \subseteq events \wedge (\forall a \in A. \forall b \in B. prob\ (a \cap b) = prob\ a * prob\ b)$

show indep-sets (case-bool A B) UNIV

proof (rule indep-setsI)

fix i **show** (case i of True $\Rightarrow A \mid$ False $\Rightarrow B$) $\subseteq events$

using * **by** (auto split: bool.split)

next

fix J X **assume** $J \neq \{\}$ $J \subseteq UNIV$ **and** $X: \forall j \in J. X\ j \in (case\ j\ of\ True \Rightarrow A \mid False \Rightarrow B)$

then have $J = \{True\} \vee J = \{False\} \vee J = \{True, False\}$

by (auto simp: UNIV-bool)

then show $prob\ (\bigcap j \in J. X\ j) = (\prod j \in J. prob\ (X\ j))$

using X * **by** auto

qed

qed

lemma (in prob-space) indep-set-sigma-sets:

assumes indep-set A B

assumes A: Int-stable A **and** B: Int-stable B

shows indep-set (sigma-sets (space M) A) (sigma-sets (space M) B)

proof –

have indep-sets ($\lambda i. sigma\text{-}sets\ (space\ M)\ (case\ i\ of\ True \Rightarrow A \mid False \Rightarrow B)$) UNIV

proof (rule indep-sets-sigma)

show indep-sets (case-bool A B) UNIV

by (rule <indep-set A B>[unfolded indep-set-def])

fix i **show** Int-stable (case i of True $\Rightarrow A \mid$ False $\Rightarrow B$)

using A B **by** (cases i) auto

qed

then show ?thesis

unfolding indep-set-def

by (rule indep-sets-mono-sets) (auto split: bool.split)

qed

lemma (in *prob-space*) *indep-eventsI-indep-vars*:
assumes *indep*: *indep-vars* $N X I$
assumes $P: \bigwedge i. i \in I \implies \{x \in \text{space } (N i). P i x\} \in \text{sets } (N i)$
shows *indep-events* $(\lambda i. \{x \in \text{space } M. P i (X i x)\}) I$
proof –
have *indep-sets* $(\lambda i. \{X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}) I$
using *indep unfolding indep-vars-def2* **by** *auto*
then show *?thesis*
unfolding *indep-events-def-alt*
proof (*rule indep-sets-mono-sets*)
fix i **assume** $i \in I$
then have $\{x \in \text{space } M. P i (X i x)\} = \{X i - ' \{x \in \text{space } (N i). P i x\} \cap \text{space } M\}$
using *indep* **by** (*auto simp: indep-vars-def dest: measurable-space*)
also have $\dots \subseteq \{X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}$
using $P[OF \langle i \in I \rangle]$ **by** *blast*
finally show $\{x \in \text{space } M. P i (X i x)\} \subseteq \{X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}$.
qed
qed

lemma (in *prob-space*) *indep-sets-collect-sigma*:
fixes $I :: 'j \Rightarrow 'i \text{ set}$ **and** $J :: 'j \text{ set}$ **and** $E :: 'i \Rightarrow 'a \text{ set set}$
assumes *indep*: *indep-sets* $E (\bigcup j \in J. I j)$
assumes *Int-stable*: $\bigwedge i j. j \in J \implies i \in I j \implies \text{Int-stable } (E i)$
assumes *disjoint*: *disjoint-family-on* $I J$
shows *indep-sets* $(\lambda j. \text{sigma-sets } (\text{space } M) (\bigcup i \in I j. E i)) J$
proof –
let $?E = \lambda j. \{\bigcap k \in K. E' k \mid E' K. \text{finite } K \wedge K \neq \{\} \wedge K \subseteq I j \wedge (\forall k \in K. E' k \in E k)\}$

from *indep* **have** $E: \bigwedge j i. j \in J \implies i \in I j \implies E i \subseteq \text{events}$
unfolding *indep-sets-def* **by** *auto*
{ fix j
let $?S = \text{sigma-sets } (\text{space } M) (\bigcup i \in I j. E i)$
assume $j \in J$
from $E[OF \text{ this}]$ **interpret** $S: \text{sigma-algebra } \text{space } M ?S$
using *sets.sets-into-space[of - M]* **by** (*intro sigma-algebra-sigma-sets*) *auto*
have *sigma-sets* $(\text{space } M) (\bigcup i \in I j. E i) = \text{sigma-sets } (\text{space } M) (?E j)$
proof (*rule sigma-sets-eqI*)
fix A **assume** $A \in (\bigcup i \in I j. E i)$
then obtain i **where** $i \in I j \wedge A \in E i$..
then show $A \in \text{sigma-sets } (\text{space } M) (?E j)$
by (*auto intro!: sigma-sets.intros(2-)*) *exI[of - {i}] exI[of - $\lambda i. A$]*
next
fix A **assume** $A \in ?E j$
then obtain $E' K$ **where** *finite* $K \wedge K \neq \{\} \wedge K \subseteq I j \wedge k \in K \implies E' k \in$

```

E k
  and A: A = ( $\bigcap k \in K. E' k$ )
  by auto
  then have A  $\in$  ?S unfolding A
  by (safe intro!: S.finite-INT) auto
  then show A  $\in$  sigma-sets (space M) ( $\bigcup i \in I j. E i$ )
  by simp
qed }
moreover have indep-sets ( $\lambda j. \text{sigma-sets (space M) (?E j)}$ ) J
proof (rule indep-sets-sigma)
  show indep-sets ?E J
  proof (intro indep-setsI)
    fix j assume j  $\in$  J with E show ?E j  $\subseteq$  events by (force intro!: sets.finite-INT)
  next
    fix K A assume K: K  $\neq$  {} K  $\subseteq$  J finite K
    and  $\forall j \in K. A j \in ?E j$ 
    then have  $\forall j \in K. \exists E' L. A j = (\bigcap l \in L. E' l) \wedge \text{finite } L \wedge L \neq \{\} \wedge L \subseteq I$ 
    j  $\wedge (\forall l \in L. E' l \in E l)$ 
    by simp
    from bchoice[OF this] obtain E'
    where  $\forall x \in K. \exists L. A x = \bigcap (E' x ' L) \wedge \text{finite } L \wedge L \neq \{\} \wedge L \subseteq I x \wedge$ 
    ( $\forall l \in L. E' x l \in E l$ )
    ..
    from bchoice[OF this] obtain L
    where A:  $\bigwedge j. j \in K \implies A j = (\bigcap l \in L j. E' j l)$ 
    and L:  $\bigwedge j. j \in K \implies \text{finite } (L j) \bigwedge j. j \in K \implies L j \neq \{\} \bigwedge j. j \in K \implies L j$ 
     $\subseteq I j$ 
    and E':  $\bigwedge j l. j \in K \implies l \in L j \implies E' j l \in E l$ 
    by auto
    { fix k l j assume k  $\in$  K j  $\in$  K l  $\in$  L j l  $\in$  L k
      have k = j
      proof (rule ccontr)
        assume k  $\neq$  j
        with disjoint  $\langle K \subseteq J \rangle \langle k \in K \rangle \langle j \in K \rangle$  have  $I k \cap I j = \{\}$ 
        unfolding disjoint-family-on-def by auto
        with L(2,3)[OF  $\langle j \in K \rangle$ ] L(2,3)[OF  $\langle k \in K \rangle$ ]
        show False using  $\langle l \in L k \rangle \langle l \in L j \rangle$  by auto
      qed }
    note L-inj = this

  define k where k l = (SOME k. k  $\in$  K  $\wedge$  l  $\in$  L k) for l
  { fix x j l assume *: j  $\in$  K l  $\in$  L j
    have k l = j unfolding k-def
    proof (rule some-equality)
      fix k assume k  $\in$  K  $\wedge$  l  $\in$  L k
      with * L-inj show k = j by auto
    qed (insert *, simp) }
  note k-simp[simp] = this
  let ?E' =  $\lambda l. E' (k l) l$ 

```

have $\text{prob} (\bigcap_{j \in K}. A j) = \text{prob} (\bigcap_{l \in (\bigcup_{k \in K}. L k)}. ?E' l)$
by (*auto simp: A intro!: arg-cong[where f=prob]*)
also have $\dots = (\prod_{l \in (\bigcup_{k \in K}. L k)}. \text{prob} (?E' l))$
using $L K E'$ **by** (*intro indep-setsD[OF indep] (simp-all add: UN-mono)*)
also have $\dots = (\prod_{j \in K}. \prod_{l \in L j}. \text{prob} (E' j l))$
using $L K L\text{-inj}$ **by** (*subst prod.UNION-disjoint auto*)
also have $\dots = (\prod_{j \in K}. \text{prob} (A j))$
using $K L E'$ **by** (*auto simp add: A intro!: prod.cong indep-setsD[OF indep, symmetric]*) *blast*
finally show $\text{prob} (\bigcap_{j \in K}. A j) = (\prod_{j \in K}. \text{prob} (A j))$.
qed
next
fix j **assume** $j \in J$
show *Int-stable* $(?E j)$
proof (*rule Int-stableI*)
fix a **assume** $a \in ?E j$ **then obtain** $Ka Ea$
where $a: a = (\bigcap_{k \in Ka}. Ea k)$ *finite* $Ka Ka \neq \{\}$ $Ka \subseteq I j \wedge k. k \in Ka \implies$
 $Ea k \in E k$ **by** *auto*
fix b **assume** $b \in ?E j$ **then obtain** $Kb Eb$
where $b: b = (\bigcap_{k \in Kb}. Eb k)$ *finite* $Kb Kb \neq \{\}$ $Kb \subseteq I j \wedge k. k \in Kb \implies$
 $Eb k \in E k$ **by** *auto*
let $?f = \lambda k. (if k \in Ka \cap Kb \text{ then } Ea k \cap Eb k \text{ else if } k \in Kb \text{ then } Eb k \text{ else if } k \in Ka \text{ then } Ea k \text{ else } \{\})$
have $Ka \cup Kb = (Ka \cap Kb) \cup (Kb - Ka) \cup (Ka - Kb)$
by *blast*
moreover have $(\bigcap_{x \in Ka \cap Kb}. Ea x \cap Eb x) \cap$
 $(\bigcap_{x \in Kb - Ka}. Eb x) \cap (\bigcap_{x \in Ka - Kb}. Ea x) = (\bigcap_{k \in Ka}. Ea k) \cap (\bigcap_{k \in Kb}. Eb k)$
 $Eb k)$
by *auto*
ultimately have $(\bigcap_{k \in Ka \cup Kb}. ?f k) = (\bigcap_{k \in Ka}. Ea k) \cap (\bigcap_{k \in Kb}. Eb k)$
(is ?lhs = ?rhs)
by (*simp only: image-Un Inter-Un-distrib simp*)
then have $a \cap b = (\bigcap_{k \in Ka \cup Kb}. ?f k)$
by (*simp only: a(1) b(1)*)
with $a b \langle j \in J \rangle$ *Int-stableD[OF Int-stable]* **show** $a \cap b \in ?E j$
by (*intro CollectI exI[of - Ka \cup Kb] exI[of - ?f]*) *auto*
qed
qed
ultimately show *?thesis*
by (*simp cong: indep-sets-cong*)
qed

lemma (*in prob-space*) *indep-vars-restrict*:

assumes *ind: indep-vars* $M' X I$ **and** $K: \bigwedge j. j \in L \implies K j \subseteq I$ **and** J : *disjoint-family-on* $K L$

shows *indep-vars* $(\lambda j. Pi_M (K j) M') (\lambda j \omega. \text{restrict} (\lambda i. X i \omega) (K j)) L$

unfolding *indep-vars-def*

proof *safe*

fix j **assume** $j \in L$ **then show** *random-variable* $(Pi_M (K j) M') (\lambda \omega. \lambda i \in K j.$


```

X i ω
  using K ind by (auto simp: indep-vars-def intro!: measurable-restrict)
next
  have X:  $\bigwedge i. i \in I \implies X i \in \text{measurable } M (M' i)$ 
  using ind by (auto simp: indep-vars-def)
  let ?proj =  $\lambda j S. \{(\lambda \omega. \lambda i \in K j. X i \omega) - ' A \cap \text{space } M \mid A. A \in S\}$ 
  let ?UN =  $\lambda j. \text{sigma-sets } (\text{space } M) (\bigcup i \in K j. \{ X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i) \})$ 
  show indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (?proj i (\text{sets } (Pi_M (K i) M')))) L$ 
  proof (rule indep-sets-mono-sets)
    fix j assume j:  $j \in L$ 
    have sigma-sets ( $\text{space } M$ ) (?proj j ( $\text{sets } (Pi_M (K j) M')$ )) =
      sigma-sets ( $\text{space } M$ ) (sigma-sets ( $\text{space } M$ ) (?proj j (prod-algebra ( $K j$ )  $M'$ ))))
    using j K X [THEN measurable-space] unfolding sets-PiM
    by (subst sigma-sets-vimage-commute) (auto simp add: Pi-iff)
    also have ... = sigma-sets ( $\text{space } M$ ) (?proj j (prod-algebra ( $K j$ )  $M'$ )))
    by (rule sigma-sets-sigma-sets-eq) auto
    also have ...  $\subseteq$  ?UN j
  proof (rule sigma-sets-mono, safe del: disjE elim!: prod-algebraE)
    fix J E assume J: finite J  $J \neq \{\}$   $\vee K j = \{\}$   $J \subseteq K j$  and E:  $\forall i. i \in J \implies E i \in \text{sets } (M' i)$ 
    show  $(\lambda \omega. \lambda i \in K j. X i \omega) - ' \text{prod-emb } (K j) M' J (Pi_E J E) \cap \text{space } M \in ?UN j$ 
  proof cases
    assume  $K j = \{\}$  with J show ?thesis
    by (auto simp add: sigma-sets-empty-eq prod-emb-def)
  next
    assume  $K j \neq \{\}$  with J have  $J \neq \{\}$ 
    by auto
    { interpret sigma-algebra  $\text{space } M$  ?UN j
      by (rule sigma-algebra-sigma-sets) auto
      have  $\bigwedge A. (\bigwedge i. i \in J \implies A i \in ?UN j) \implies \bigcap (A - ' J) \in ?UN j$ 
      using  $\langle \text{finite } J \rangle \langle J \neq \{\} \rangle$  by (rule finite-INT) blast }
    note INT = this

  from  $\langle J \neq \{\} \rangle J K E$  [rule-format, THEN sets.sets-into-space] j
  have  $(\lambda \omega. \lambda i \in K j. X i \omega) - ' \text{prod-emb } (K j) M' J (Pi_E J E) \cap \text{space } M$ 
    =  $(\bigcap i \in J. X i - ' E i \cap \text{space } M)$ 
  apply (subst prod-emb-PiE [OF -])
  apply auto []
  apply auto []
  apply (auto simp add: Pi-iff intro!: X [THEN measurable-space])
  apply (erule-tac x=i in ballE)
  apply auto
  done
  also have ...  $\in$  ?UN j
  apply (rule INT)
  apply (rule sigma-sets.Basic)
  using  $\langle J \subseteq K j \rangle E$ 

```

```

      apply auto
      done
      finally show ?thesis .
    qed
  qed
  finally show sigma-sets (space M) (?proj j (sets (Pi_M (K j) M'))) ⊆ ?UN j .
next
show indep-sets ?UN L
proof (rule indep-sets-collect-sigma)
  show indep-sets (λi. {X i -' A ∩ space M | A. A ∈ sets (M' i)}) (⋃ j ∈ L. K
j)
  proof (rule indep-sets-mono-index)
    show indep-sets (λi. {X i -' A ∩ space M | A. A ∈ sets (M' i)}) I
      using ind unfolding indep-vars-def2 by auto
    show (⋃ l ∈ L. K l) ⊆ I
      using K by auto
  qed
next
fix l i assume l ∈ L i ∈ K l
show Int-stable {X i -' A ∩ space M | A. A ∈ sets (M' i)}
  apply (auto simp: Int-stable-def)
  apply (rule-tac x=A ∩ Aa in exI)
  apply auto
  done
qed fact
qed
qed

```

```

lemma (in prob-space) indep-var-restrict:
  assumes ind: indep-vars M' X I and AB: A ∩ B = {} A ⊆ I B ⊆ I
  shows indep-var (Pi_M A M') (λω. restrict (λi. X i ω) A) (Pi_M B M') (λω.
restrict (λi. X i ω) B)
proof -
  have *:
    case-bool (Pi_M A M') (Pi_M B M') = (λb. Pi_M (case-bool A B b) M')
    case-bool (λω. λi ∈ A. X i ω) (λω. λi ∈ B. X i ω) = (λb ω. λi ∈ case-bool A B b.
X i ω)
  by (simp-all add: fun-eq-iff split: bool.split)
  show ?thesis
  unfolding indep-var-def * using AB
  by (intro indep-vars-restrict[OF ind]) (auto simp: disjoint-family-on-def split:
bool.split)
qed

```

```

lemma (in prob-space) indep-vars-subset:
  assumes indep-vars M' X I J ⊆ I
  shows indep-vars M' X J
  using assms unfolding indep-vars-def indep-sets-def
  by auto

```

lemma (in *prob-space*) *indep-vars-cong*:
 $I = J \implies (\bigwedge i. i \in I \implies X i = Y i) \implies (\bigwedge i. i \in I \implies M' i = N' i) \implies$
indep-vars $M' X I \longleftrightarrow \text{indep-vars } N' Y J$
unfolding *indep-vars-def2* **by** (*intro conj-cong indep-sets-cong*) *auto*

definition (in *prob-space*) *tail-events where*
tail-events $A = (\bigcap n. \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\})))$

lemma (in *prob-space*) *tail-events-sets*:
assumes $A: \bigwedge i::\text{nat}. A i \subseteq \text{events}$
shows *tail-events* $A \subseteq \text{events}$
proof
fix X **assume** $X: X \in \text{tail-events } A$
let $?A = (\bigcap n. \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\})))$
from X **have** $\bigwedge n::\text{nat}. X \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\}))$ **by** (*auto simp: tail-events-def*)
from *this*[of 0] **have** $X \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \text{UNIV}))$ **by** *simp*
then show $X \in \text{events}$
by *induct (insert A, auto)*
qed

lemma (in *prob-space*) *sigma-algebra-tail-events*:
assumes $\bigwedge i::\text{nat}. \text{sigma-algebra } (\text{space } M) (A i)$
shows *sigma-algebra* $(\text{space } M) (\text{tail-events } A)$
unfolding *tail-events-def*
proof (*simp add: sigma-algebra-iff2, safe*)
let $?A = (\bigcap n. \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\})))$
interpret $A: \text{sigma-algebra } \text{space } M A i$ **for** i **by** *fact*
{ fix $X x$ **assume** $X \in ?A x \in X$
then have $\bigwedge n. X \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\}))$ **by** *auto*
from *this*[of 0] **have** $X \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \text{UNIV}))$ **by** *simp*
then have $X \subseteq \text{space } M$
by *induct (insert A.sets-into-space, auto)*
with $\langle x \in X \rangle$ **show** $x \in \text{space } M$ **by** *auto* }
{ fix $F :: \text{nat} \Rightarrow \text{'a set}$ **and** n **assume** $\text{range } F \subseteq ?A$
then show $(\bigcup (F \text{ ' } \text{UNIV})) \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\}))$
by (*intro sigma-sets.Union*) *auto* }
qed (*auto intro!: sigma-sets.Compl sigma-sets.Empty*)

lemma (in *prob-space*) *kolmogorov-0-1-law*:
fixes $A :: \text{nat} \Rightarrow \text{'a set set}$
assumes $\bigwedge i::\text{nat}. \text{sigma-algebra } (\text{space } M) (A i)$
assumes *indep: indep-sets* $A \text{ UNIV}$
and $X: X \in \text{tail-events } A$
shows $\text{prob } X = 0 \vee \text{prob } X = 1$
proof –
have $A: \bigwedge i. A i \subseteq \text{events}$
using *indep unfolding indep-sets-def* **by** *simp*

```

let ?D = {D ∈ events. prob (X ∩ D) = prob X * prob D}
interpret A: sigma-algebra space M A i for i by fact
interpret T: sigma-algebra space M tail-events A
  by (rule sigma-algebra-tail-events) fact
have X ⊆ space M using T.space-closed X by auto

have X-in: X ∈ events
  using tail-events-sets A X by auto

interpret D: Dynkin-system space M ?D
proof (rule Dynkin-systemI)
  fix D assume D ∈ ?D then show D ⊆ space M
    using sets.sets-into-space by auto
next
  show space M ∈ ?D
    using prob-space ⟨X ⊆ space M⟩ by (simp add: Int-absorb2)
next
  fix A assume A: A ∈ ?D
  have prob (X ∩ (space M - A)) = prob (X - (X ∩ A))
    using ⟨X ⊆ space M⟩ by (auto intro!: arg-cong[where f=prob])
  also have ... = prob X - prob (X ∩ A)
    using X-in A by (intro finite-measure-Diff) auto
  also have ... = prob X * prob (space M) - prob X * prob A
    using A prob-space by auto
  also have ... = prob X * prob (space M - A)
    using X-in A sets.sets-into-space
    by (subst finite-measure-Diff) (auto simp: field-simps)
  finally show space M - A ∈ ?D
    using A ⟨X ⊆ space M⟩ by auto
next
  fix F :: nat ⇒ 'a set assume dis: disjoint-family F and range F ⊆ ?D
  then have F: range F ⊆ events ∧ i. prob (X ∩ F i) = prob X * prob (F i)
    by auto
  have (λi. prob (X ∩ F i)) sums prob (⋃ i. X ∩ F i)
  proof (rule finite-measure-UNION)
    show range (λi. X ∩ F i) ⊆ events
      using F X-in by auto
    show disjoint-family (λi. X ∩ F i)
      using dis by (rule disjoint-family-on-bisimulation) auto
  qed
  with F have (λi. prob X * prob (F i)) sums prob (X ∩ (⋃ i. F i))
    by simp
  moreover have (λi. prob X * prob (F i)) sums (prob X * prob (⋃ i. F i))
    by (intro sums-mult finite-measure-UNION F dis)
  ultimately have prob (X ∩ (⋃ i. F i)) = prob X * prob (⋃ i. F i)
    by (auto dest!: sums-unique)
  with F show (⋃ i. F i) ∈ ?D
    by auto

```

```

qed

{ fix  $n$ 
  have  $\text{indep-sets } (\lambda b. \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \text{case-bool } \{..n\}} \{ \text{Suc } n.. \} b. A m)) \text{ UNIV}$ 
  proof (rule indep-sets-collect-sigma)
    have  $*$ :  $(\bigcup b. \text{case } b \text{ of True } \Rightarrow \{..n\} \mid \text{False } \Rightarrow \{ \text{Suc } n.. \}) = \text{UNIV}$  (is  $?U = -$ )
    by (simp split: bool.split add: set-eq-iff) (metis not-less-eq-eq)
    with  $\text{indep}$  show  $\text{indep-sets } A \text{ ?U}$  by simp
    show  $\text{disjoint-family } (\text{case-bool } \{..n\} \{ \text{Suc } n.. \})$ 
    unfolding  $\text{disjoint-family-on-def}$  by (auto split: bool.split)
    fix  $m$ 
    show  $\text{Int-stable } (A m)$ 
    unfolding  $\text{Int-stable-def}$  using  $A.\text{Int}$  by auto
  qed
  also have  $(\lambda b. \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \text{case-bool } \{..n\}} \{ \text{Suc } n.. \} b. A m))$ 
  =
     $\text{case-bool } (\text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m)) (\text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{ \text{Suc } n.. \}}. A m))$ 
    by (auto intro!: ext split: bool.split)
    finally have  $\text{indep: indep-set } (\text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m)) (\text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{ \text{Suc } n.. \}}. A m))$ 
    unfolding  $\text{indep-set-def}$  by simp

  have  $\text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m) \subseteq ?D$ 
  proof (simp add: subset-eq, rule)
    fix  $D$  assume  $D: D \in \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m)$ 
    have  $X \in \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{ \text{Suc } n.. \}}. A m)$ 
    using  $X$  unfolding  $\text{tail-events-def}$  by simp
    from  $\text{indep-setD}[OF \text{ indep } D \text{ this}] \text{indep-setD-ev1}[OF \text{ indep}] D$ 
    show  $D \in \text{events} \wedge \text{prob } (X \cap D) = \text{prob } X * \text{prob } D$ 
    by (auto simp add: ac-simps)
  qed }
then have  $(\bigcup n. \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m)) \subseteq ?D$  (is  $?A \subseteq -$ )
by auto

note  $\langle X \in \text{tail-events } A \rangle$ 
also {
  have  $\bigwedge n. \text{sigma-sets } (\text{space } M) (\bigcup_{i \in \{n.. \}}. A i) \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
  by (intro sigma-sets-subseteq UN-mono) auto
  then have  $\text{tail-events } A \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
  unfolding  $\text{tail-events-def}$  by auto }
also have  $\text{sigma-sets } (\text{space } M) ?A = \text{Dynkin } (\text{space } M) ?A$ 
proof (rule sigma-eq-Dynkin)
  { fix  $B n$  assume  $B \in \text{sigma-sets } (\text{space } M) (\bigcup_{m \in \{..n\}}. A m)$ 
    then have  $B \subseteq \text{space } M$ 
    by induct (insert A sets.sets-into-space[of - M], auto) }
  then show  $?A \subseteq \text{Pow } (\text{space } M)$  by auto

```

```

show Int-stable ?A
proof (rule Int-stableI)
  fix a b assume  $a \in ?A$   $b \in ?A$  then obtain  $n\ m$ 
    where  $a: n \in UNIV$   $a \in \text{sigma-sets (space } M)$   $(\bigcup (A \text{ ' } \{..n\}))$ 
      and  $b: m \in UNIV$   $b \in \text{sigma-sets (space } M)$   $(\bigcup (A \text{ ' } \{..m\}))$  by auto
    interpret Amn: sigma-algebra space  $M$  sigma-sets (space  $M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$ 
    using A sets.sets-into-space[of - M] by (intro sigma-algebra-sigma-sets) auto
    have  $\text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..n\}} A\ i) \subseteq \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$ 
    by (intro sigma-sets-subseteq UN-mono) auto
    with a have  $a \in \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$  by auto
    moreover
    have  $\text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..m\}} A\ i) \subseteq \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$ 
    by (intro sigma-sets-subseteq UN-mono) auto
    with b have  $b \in \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$  by auto
    ultimately have  $a \cap b \in \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..max\ m\ n\}} A\ i)$ 
    using Amn.Int[of a b] by simp
    then show  $a \cap b \in (\bigcup n. \text{sigma-sets (space } M)$   $(\bigcup_{i \in \{..n\}} A\ i))$  by auto
  qed
qed
also have Dynkin (space  $M)$   $?A \subseteq ?D$ 
  using  $\langle ?A \subseteq ?D \rangle$  by (auto intro!: D.Dynkin-subset)
finally show ?thesis by auto
qed

```

lemma (*in prob-space*) *borel-0-1-law*:

```

fixes  $F :: nat \Rightarrow 'a\ \text{set}$ 
assumes F2: indep-events  $F$  UNIV
shows  $\text{prob} (\bigcap n. \bigcup_{m \in \{n..\}} F\ m) = 0 \vee \text{prob} (\bigcap n. \bigcup_{m \in \{n..\}} F\ m) = 1$ 
proof (rule kolmogorov-0-1-law[of  $\lambda i. \text{sigma-sets (space } M)$   $\{F\ i\}$  ])
  have F1:  $\text{range } F \subseteq \text{events}$ 
    using F2 by (simp add: indep-events-def subset-eq)
  { fix i show sigma-algebra (space  $M)$   $(\text{sigma-sets (space } M)$   $\{F\ i\})$ 
    using sigma-algebra-sigma-sets[of  $\{F\ i\}$  space  $M]$  F1 sets.sets-into-space
    by auto }
show indep-sets  $(\lambda i. \text{sigma-sets (space } M)$   $\{F\ i\})$  UNIV
proof (rule indep-sets-sigma)
  show indep-sets  $(\lambda i. \{F\ i\})$  UNIV
    unfolding indep-events-def-alt[symmetric] by fact
  fix i show Int-stable  $\{F\ i\}$ 
    unfolding Int-stable-def by simp
  qed
let  $?Q = \lambda n. \bigcup_{i \in \{n..\}} F\ i$ 
show  $(\bigcap n. \bigcup_{m \in \{n..\}} F\ m) \in \text{tail-events} (\lambda i. \text{sigma-sets (space } M)$   $\{F\ i\})$ 
  unfolding tail-events-def
proof
  fix j

```

interpret S : *sigma-algebra space M sigma-sets (space M) ($\bigcup i \in \{j.. \}$). sigma-sets (space M) {F i}*
using *order-trans[OF F1 sets.space-closed]*
by (*intro sigma-algebra-sigma-sets*) (*simp add: sigma-sets-singleton subset-eq*)
have $(\bigcap n. ?Q n) = (\bigcap n \in \{j.. \}. ?Q n)$
by (*intro decseq-SucI INT-decseq-offset UN-mono*) *auto*
also have $\dots \in \text{sigma-sets (space M) } (\bigcup i \in \{j.. \}. \text{sigma-sets (space M) } \{F i\})$
using *order-trans[OF F1 sets.space-closed]*
by (*safe intro!: S.countable-INT S.countable-UN*)
(auto simp: sigma-sets-singleton intro!: sigma-sets.Basic bexI)
finally show $(\bigcap n. ?Q n) \in \text{sigma-sets (space M) } (\bigcup i \in \{j.. \}. \text{sigma-sets (space M) } \{F i\})$
by *simp*
qed
qed

lemma (*in prob-space*) *borel-0-1-law-AE*:

fixes $P :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$
assumes *indep-events* $(\lambda m. \{x \in \text{space } M. P m x\})$ *UNIV* (**is** *indep-events* $?P$ -)
shows $(AE x \text{ in } M. \text{infinite } \{m. P m x\}) \vee (AE x \text{ in } M. \text{finite } \{m. P m x\})$
proof -
have $[\text{measurable}]: \bigwedge m. \{x \in \text{space } M. P m x\} \in \text{sets } M$
using *assms* **by** (*auto simp: indep-events-def*)
have $*$: $(\bigcap n. \bigcup m \in \{n.. \}. \{x \in \text{space } M. P m x\}) \in \text{events}$
by *simp*
from *assms* **have** $\text{prob } (\bigcap n. \bigcup m \in \{n.. \}. ?P m) = 0 \vee \text{prob } (\bigcap n. \bigcup m \in \{n.. \}. ?P m) = 1$
by (*rule borel-0-1-law*)
also have $\text{prob } (\bigcap n. \bigcup m \in \{n.. \}. ?P m) = 1 \iff (AE x \text{ in } M. \text{infinite } \{m. P m x\})$
using $*$ **by** (*simp add: prob-eq-1*)
(simp add: Bex-def infinite-nat-iff-unbounded-le)
also have $\text{prob } (\bigcap n. \bigcup m \in \{n.. \}. ?P m) = 0 \iff (AE x \text{ in } M. \text{finite } \{m. P m x\})$
using $*$ **by** (*simp add: prob-eq-0*)
(auto simp add: Ball-def finite-nat-iff-bounded not-less [symmetric])
finally show *?thesis*
by *blast*
qed

lemma (*in prob-space*) *indep-sets-finite*:

assumes $I: I \neq \{\}$ *finite I*
and $F: \bigwedge i. i \in I \implies F i \subseteq \text{events} \bigwedge i. i \in I \implies \text{space } M \in F i$
shows *indep-sets* $F I \iff (\forall A \in \text{Pi } I F. \text{prob } (\bigcap j \in I. A j) = (\prod j \in I. \text{prob } (A j)))$
proof
assume $*$: *indep-sets* $F I$
from I **show** $\forall A \in \text{Pi } I F. \text{prob } (\bigcap j \in I. A j) = (\prod j \in I. \text{prob } (A j))$
by (*intro indep-setsD[OF *] ballI*) *auto*

next

assume *indep*: $\forall A \in Pi\ I\ F. \text{prob} (\bigcap_{j \in I}. A\ j) = (\prod_{j \in I}. \text{prob} (A\ j))$
show *indep-sets* $F\ I$
proof (*rule indep-setsI*[*OF F(1)*])
fix $A\ J$ **assume** $J: J \neq \{\}$ $J \subseteq I$ *finite* J
assume $A: \forall j \in J. A\ j \in F\ j$
let $?A = \lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M$
have $\text{prob} (\bigcap_{j \in I}. ?A\ j) = \text{prob} (\bigcap_{j \in J}. A\ j)$
using *subset-trans*[*OF F(1) sets.space-closed*] $J\ A$
by (*auto intro!*: *arg-cong*[**where** $f = \text{prob}$] *split: if-split-asm*) *blast*
also
from $A\ F$ **have** $(\lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M) \in Pi\ I\ F$ (**is** $?A \in -$)
by (*auto split: if-split-asm*)
with *indep* **have** $\text{prob} (\bigcap_{j \in I}. ?A\ j) = (\prod_{j \in I}. \text{prob} (?A\ j))$
by *auto*
also have $\dots = (\prod_{j \in J}. \text{prob} (A\ j))$
unfolding *if-distrib prod.If-cases*[*OF <finite I>*]
using *prob-space <J ⊆ I>* **by** (*simp add: Int-absorb1 prod.neutral-const*)
finally show $\text{prob} (\bigcap_{j \in J}. A\ j) = (\prod_{j \in J}. \text{prob} (A\ j))$..
qed
qed

lemma (*in prob-space*) *indep-vars-finite*:

fixes $I :: 'i\ \text{set}$
assumes $I: I \neq \{\}$ *finite* I
and M' : $\bigwedge i. i \in I \implies \text{sets} (M'\ i) = \text{sigma-sets} (\text{space} (M'\ i)) (E\ i)$
and rv : $\bigwedge i. i \in I \implies \text{random-variable} (M'\ i) (X\ i)$
and *Int-stable*: $\bigwedge i. i \in I \implies \text{Int-stable} (E\ i)$
and *space*: $\bigwedge i. i \in I \implies \text{space} (M'\ i) \in E\ i$ **and** *closed*: $\bigwedge i. i \in I \implies E\ i \subseteq$
Pow (*space* ($M'\ i$))
shows *indep-vars* $M'\ X\ I \longleftrightarrow$
 $(\forall A \in (Pi\ i \in I. E\ i). \text{prob} (\bigcap_{j \in I}. X\ j - 'A\ j \cap \text{space } M) = (\prod_{j \in I}. \text{prob} (X\ j - 'A\ j \cap \text{space } M)))$

proof –

from *rv* **have** $X: \bigwedge i. i \in I \implies X\ i \in \text{space } M \rightarrow \text{space} (M'\ i)$
unfolding *measurable-def* **by** *simp*

{ **fix** i **assume** $i \in I$
from *closed*[*OF <i ∈ I>*]
have *sigma-sets* (*space* M) $\{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets} (M'\ i)\}$
 $= \text{sigma-sets} (\text{space } M) \{X\ i - 'A \cap \text{space } M \mid A. A \in E\ i\}$
unfolding *sigma-sets-vimage-commute*[*OF X, OF <i ∈ I>, symmetric*] M' [*OF*
 $<i \in I>$]
by (*subst sigma-sets-sigma-sets-eq*) *auto* }
note *sigma-sets-X* = *this*

{ **fix** i **assume** $i \in I$
have *Int-stable* $\{X\ i - 'A \cap \text{space } M \mid A. A \in E\ i\}$
proof (*rule Int-stableI*)

fix a **assume** $a \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$
then obtain A **where** $a = X\ i - ' A \cap \text{space } M \mid A \in E\ i$ **by auto**
moreover
fix b **assume** $b \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$
then obtain B **where** $b = X\ i - ' B \cap \text{space } M \mid B \in E\ i$ **by auto**
moreover
have $(X\ i - ' A \cap \text{space } M) \cap (X\ i - ' B \cap \text{space } M) = X\ i - ' (A \cap B) \cap$
 $\text{space } M$ **by auto**
moreover note $\text{Int-stable}[OF\ \langle i \in I \rangle]$
ultimately
show $a \cap b \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$
by $(\text{auto simp del: vimage-Int intro!: exI[of - } A \cap B] \text{ dest: Int-stableD})$
qed }
note $\text{indep-sets-}X = \text{indep-sets-sigma-sets-iff}[OF\ \text{this}]$

{ fix i **assume** $i \in I$
{ fix A **assume** $A \in E\ i$
with $M'[OF\ \langle i \in I \rangle]$ **have** $A \in \text{sets } (M'\ i)$ **by auto**
moreover
from $rv[OF\ \langle i \in I \rangle]$ **have** $X\ i \in \text{measurable } M\ (M'\ i)$ **by auto**
ultimately
have $X\ i - ' A \cap \text{space } M \in \text{sets } M$ **by** $(\text{auto intro: measurable-sets})$ }
with $X[OF\ \langle i \in I \rangle]$ $\text{space}[OF\ \langle i \in I \rangle]$
have $\{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\} \subseteq \text{events}$
 $\text{space } M \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$
by $(\text{auto intro!: exI[of - } \text{space } (M'\ i)])$ }
note $\text{indep-sets-finite-}X = \text{indep-sets-finite}[OF\ I\ \text{this}]$

have $(\forall A \in \Pi\ i \in I. \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}. \text{prob } (\bigcap (A\ ' I)) = (\prod_{j \in I. \text{prob } (A\ j)})) =$
 $\text{prob } (A\ j)) =$
 $(\forall A \in \Pi\ i \in I. E\ i. \text{prob } ((\bigcap_{j \in I. X\ j - ' A\ j) \cap \text{space } M) = (\prod_{x \in I. \text{prob } (X\ x - ' A\ x \cap \text{space } M))))$
(is $?L = ?R$
proof safe
fix A **assume** $?L$ **and** $A: A \in (\Pi\ i \in I. E\ i)$
from $\langle ?L \rangle [THEN\ \text{bspec, of } \lambda i. X\ i - ' A\ i \cap \text{space } M] A\ \langle I \neq \{\} \rangle$
show $\text{prob } ((\bigcap_{j \in I. X\ j - ' A\ j) \cap \text{space } M) = (\prod_{x \in I. \text{prob } (X\ x - ' A\ x \cap \text{space } M)))$
by $(\text{auto simp add: Pi-iff})$
next
fix A **assume** $?R$ **and** $A: A \in (\Pi\ i \in I. \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\})$
from A **have** $\forall i \in I. \exists B. A\ i = X\ i - ' B \cap \text{space } M \wedge B \in E\ i$ **by auto**
from $\text{bchoice}[OF\ \text{this}]$ **obtain** B **where** $B: \forall i \in I. A\ i = X\ i - ' B\ i \cap \text{space } M$
 $B \in (\Pi\ i \in I. E\ i)$ **by auto**
from $\langle ?R \rangle [THEN\ \text{bspec, OF } B(2)] B(1)\ \langle I \neq \{\} \rangle$
show $\text{prob } (\bigcap (A\ ' I)) = (\prod_{j \in I. \text{prob } (A\ j)})$
by simp
qed
then show $?thesis$ **using** $\langle I \neq \{\} \rangle$

by (simp add: rv indep-vars-def indep-sets-X sigma-sets-X indep-sets-finite-X cong: indep-sets-cong)

qed

lemma (in prob-space) indep-vars-compose:

assumes indep-vars $M' X I$

assumes rv: $\bigwedge i. i \in I \implies Y i \in \text{measurable } (M' i) (N i)$

shows indep-vars $N (\lambda i. Y i \circ X i) I$

unfolding indep-vars-def

proof

from rv $\langle \text{indep-vars } M' X I \rangle$

show $\forall i \in I. \text{random-variable } (N i) (Y i \circ X i)$

by (auto simp: indep-vars-def)

have indep-sets $(\lambda i. \text{sigma-sets } (\text{space } M) \{X i -' A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}) I$

using $\langle \text{indep-vars } M' X I \rangle$ by (simp add: indep-vars-def)

then show indep-sets $(\lambda i. \text{sigma-sets } (\text{space } M) \{(Y i \circ X i) -' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\}) I$

proof (rule indep-sets-mono-sets)

fix i assume $i \in I$

with $\langle \text{indep-vars } M' X I \rangle$ have $X: X i \in \text{space } M \rightarrow \text{space } (M' i)$

unfolding indep-vars-def measurable-def by auto

{ fix A assume $A \in \text{sets } (N i)$

then have $\exists B. (Y i \circ X i) -' A \cap \text{space } M = X i -' B \cap \text{space } M \wedge B \in \text{sets } (M' i)$

by (intro exI[of - $Y i -' A \cap \text{space } (M' i)$])

(auto simp: vimage-comp intro!: measurable-sets rv $\langle i \in I \rangle \text{funcset-mem}[OF$

$X]$) }

then show $\text{sigma-sets } (\text{space } M) \{(Y i \circ X i) -' A \cap \text{space } M \mid A. A \in \text{sets } (N i)\} \subseteq$

$\text{sigma-sets } (\text{space } M) \{X i -' A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$

by (intro sigma-sets-subseteq) (auto simp: vimage-comp)

qed

qed

lemma (in prob-space) indep-vars-compose2:

assumes indep-vars $M' X I$

assumes rv: $\bigwedge i. i \in I \implies Y i \in \text{measurable } (M' i) (N i)$

shows indep-vars $N (\lambda i x. Y i (X i x)) I$

using indep-vars-compose [OF assms] by (simp add: comp-def)

lemma (in prob-space) indep-var-compose:

assumes indep-var $M1 X1 M2 X2 Y1 \in \text{measurable } M1 N1 Y2 \in \text{measurable } M2 N2$

shows indep-var $N1 (Y1 \circ X1) N2 (Y2 \circ X2)$

proof –

have indep-vars (case-bool $N1 N2$) $(\lambda b. \text{case-bool } Y1 Y2 b \circ \text{case-bool } X1 X2 b)$
UNIV

using *assms*
by (*intro indep-vars-compose*[**where** $M' = \text{case-bool } M1 \ M2$])
 (*auto simp: indep-var-def split: bool.split*)
also have $(\lambda b. \text{case-bool } Y1 \ Y2 \ b \circ \text{case-bool } X1 \ X2 \ b) = \text{case-bool } (Y1 \circ X1)$
 $(Y2 \circ X2)$
by (*simp add: fun-eq-iff split: bool.split*)
finally show *?thesis*
unfolding *indep-var-def* .
qed

lemma (*in prob-space*) *indep-vars-Min*:

fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes I : *finite* I $i \notin I$ **and** *indep*: *indep-vars* $(\lambda-. \text{borel}) \ X$ (*insert* $i \ I$)
shows *indep-var borel* $(X \ i)$ *borel* $(\lambda\omega. \text{Min } ((\lambda i. X \ i \ \omega)'I))$
proof –
have *indep-var*
 $\text{borel } ((\lambda f. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ \{i\}))$
 $\text{borel } ((\lambda f. \text{Min } (f'I)) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ I))$
using I **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]] *borel-measurable-Min*)
auto
also have $((\lambda f. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ \{i\})) = X \ i$
by *auto*
also have $((\lambda f. \text{Min } (f'I)) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ I)) = (\lambda\omega. \text{Min } ((\lambda i. X \ i$
 $\omega)'I))$
by (*auto cong: rev-conj-cong*)
finally show *?thesis*
unfolding *indep-var-def* .
qed

lemma (*in prob-space*) *indep-vars-sum*:

fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes I : *finite* I $i \notin I$ **and** *indep*: *indep-vars* $(\lambda-. \text{borel}) \ X$ (*insert* $i \ I$)
shows *indep-var borel* $(X \ i)$ *borel* $(\lambda\omega. \sum_{i \in I}. X \ i \ \omega)$
proof –
have *indep-var*
 $\text{borel } ((\lambda f. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ \{i\}))$
 $\text{borel } ((\lambda f. \sum_{i \in I}. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ I))$
using I **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]]) *auto*
also have $((\lambda f. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ \{i\})) = X \ i$
by *auto*
also have $((\lambda f. \sum_{i \in I}. f \ i) \circ (\lambda\omega. \text{restrict } (\lambda i. X \ i \ \omega) \ I)) = (\lambda\omega. \sum_{i \in I}. X \ i \ \omega)$
by (*auto cong: rev-conj-cong*)
finally show *?thesis* .
qed

lemma (*in prob-space*) *indep-vars-prod*:

fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes I : *finite* I $i \notin I$ **and** *indep*: *indep-vars* $(\lambda-. \text{borel}) \ X$ (*insert* $i \ I$)
shows *indep-var borel* $(X \ i)$ *borel* $(\lambda\omega. \prod_{i \in I}. X \ i \ \omega)$

proof –

have *indep-var*

borel $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega)\ \{i\}))$

borel $((\lambda f. \prod_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega)\ I))$

using *I* **by** $(\text{intro indep-var-compose}[OF\ \text{indep-var-restrict}[OF\ \text{indep}]])\ \text{auto}$

also have $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega)\ \{i\})) = X\ i$

by *auto*

also have $((\lambda f. \prod_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega)\ I)) = (\lambda \omega. \prod_{i \in I}. X\ i\ \omega)$

by $(\text{auto cong; rev-conj-cong})$

finally show *?thesis* .

qed

lemma $(\text{in prob-space})\ \text{indep-varsD-finite}$:

assumes *X*: *indep-vars* *M'* *X* *I*

assumes *I*: $I \neq \{\}$ *finite* *I* $\bigwedge i. i \in I \implies A\ i \in \text{sets } (M'\ i)$

shows $\text{prob } (\bigcap_{i \in I}. X\ i - 'A\ i \cap \text{space } M) = (\prod_{i \in I}. \text{prob } (X\ i - 'A\ i \cap \text{space } M))$

proof $(\text{rule indep-setsD})$

show *indep-sets* $(\lambda i. \text{sigma-sets } (\text{space } M)\ \{X\ i - 'A\ i \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\})\ I$

using *X* **by** $(\text{auto simp: indep-vars-def})$

show $I \subseteq I\ I \neq \{\}$ *finite* *I* **using** *I* **by** *auto*

show $\forall i \in I. X\ i - 'A\ i \cap \text{space } M \in \text{sigma-sets } (\text{space } M)\ \{X\ i - 'A\ i \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\}$

using *I* **by** *auto*

qed

lemma $(\text{in prob-space})\ \text{indep-varsD}$:

assumes *X*: *indep-vars* *M'* *X* *I*

assumes *I*: $J \neq \{\}$ *finite* $J \subseteq I \bigwedge i. i \in J \implies A\ i \in \text{sets } (M'\ i)$

shows $\text{prob } (\bigcap_{i \in J}. X\ i - 'A\ i \cap \text{space } M) = (\prod_{i \in J}. \text{prob } (X\ i - 'A\ i \cap \text{space } M))$

proof $(\text{rule indep-setsD})$

show *indep-sets* $(\lambda i. \text{sigma-sets } (\text{space } M)\ \{X\ i - 'A\ i \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\})\ I$

using *X* **by** $(\text{auto simp: indep-vars-def})$

show $\forall i \in J. X\ i - 'A\ i \cap \text{space } M \in \text{sigma-sets } (\text{space } M)\ \{X\ i - 'A\ i \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\}$

using *I* **by** *auto*

qed *fact+*

lemma $(\text{in prob-space})\ \text{indep-vars-iff-distr-eq-PiM}$:

fixes *I* :: *'i* *set* **and** *X* :: *'i* \Rightarrow *'a* \Rightarrow *'b*

assumes $I \neq \{\}$

assumes *rv*: $\bigwedge i. \text{random-variable } (M'\ i)\ (X\ i)$

shows *indep-vars* *M'* *X* *I* \longleftrightarrow

$\text{distr } M\ (\prod_M\ i \in I. M'\ i)\ (\lambda x. \lambda i \in I. X\ i\ x) = (\prod_M\ i \in I. \text{distr } M\ (M'\ i)\ (X\ i))$

proof –

let *?P* = $\prod_M\ i \in I. M'\ i$

```

let ?X = λx. λi∈I. X i x
let ?D = distr M ?P ?X
have X: random-variable ?P ?X by (intro measurable-restrict rv)
interpret D: prob-space ?D by (intro prob-space-distr X)

let ?D' = λi. distr M (M' i) (X i)
let ?P' = ΠM i∈I. distr M (M' i) (X i)
interpret D': prob-space ?D' i for i by (intro prob-space-distr rv)
interpret P: product-prob-space ?D' I ..

show ?thesis
proof
  assume indep-vars M' X I
  show ?D = ?P'
  proof (rule measure-eqI-generator-eq)
    show Int-stable (prod-algebra I M')
      by (rule Int-stable-prod-algebra)
    show prod-algebra I M' ⊆ Pow (space ?P)
      using prod-algebra-sets-into-space by (simp add: space-PiM)
    show sets ?D = sigma-sets (space ?P) (prod-algebra I M')
      by (simp add: sets-PiM space-PiM)
    show sets ?P' = sigma-sets (space ?P) (prod-algebra I M')
      by (simp add: sets-PiM space-PiM cong: prod-algebra-cong)
    let ?A = λi. ΠE i∈I. space (M' i)
    show range ?A ⊆ prod-algebra I M' (⋃ i. ?A i) = space (PiM I M')
    by (auto simp: space-PiM intro!: space-in-prod-algebra cong: prod-algebra-cong)
    { fix i show emeasure ?D (ΠE i∈I. space (M' i)) ≠ ∞ by auto }
  next
  fix E assume E: E ∈ prod-algebra I M'
  from prod-algebraE[OF E] obtain J Y
  where J:
    E = prod-emb I M' J (PiE J Y)
    finite J
    J ≠ {} ∨ I = {}
    J ⊆ I
    ∧ i. i ∈ J ⇒ Y i ∈ sets (M' i)
  by auto
  from E have E ∈ sets ?P by (auto simp: sets-PiM)
  then have emeasure ?D E = emeasure M (?X -' E ∩ space M)
  by (simp add: emeasure-distr X)
  also have ?X -' E ∩ space M = (∩ i∈J. X i -' Y i ∩ space M)
  using J ⟨I ≠ {}⟩ measurable-space[OF rv] by (auto simp: prod-emb-def
  PiE-iff split: if-split-asm)
  also have emeasure M (∩ i∈J. X i -' Y i ∩ space M) = (∏ i∈J. emeasure
  M (X i -' Y i ∩ space M))
  using ⟨indep-vars M' X I⟩ J ⟨I ≠ {}⟩ using indep-varsD[of M' X I J]
  by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
  also have ... = (∏ i∈J. emeasure (?D' i) (Y i))
  using rv J by (simp add: emeasure-distr)

```

```

also have ... = emeasure ?P' E
  using P.emeasure-PiM-emb[of J Y] J by (simp add: prod-emb-def)
  finally show emeasure ?D E = emeasure ?P' E .
qed
next
  assume ?D = ?P'
  show indep-vars M' X I unfolding indep-vars-def
  proof (intro conjI indep-setsI ballI rv)
    fix i show sigma-sets (space M) {X i -' A ∩ space M | A. A ∈ sets (M' i)}
  ⊆ events
    by (auto intro!: sets.sigma-sets-subset measurable-sets rv)
  next
    fix J Y' assume J: J ≠ {} J ⊆ I finite J
    assume Y': ∀j∈J. Y' j ∈ sigma-sets (space M) {X j -' A ∩ space M | A. A
  ∈ sets (M' j)}
    have ∀j∈J. ∃Y. Y' j = X j -' Y ∩ space M ∧ Y ∈ sets (M' j)
    proof
      fix j assume j ∈ J
      from Y'[rule-format, OF this] rv[of j]
      show ∃Y. Y' j = X j -' Y ∩ space M ∧ Y ∈ sets (M' j)
      by (subst (asm) sigma-sets-vimage-commute[symmetric, of - - space (M'
j)])
      (auto dest: measurable-space simp: sets.sigma-sets-eq)
    qed
  from bchoice[OF this] obtain Y where
    Y: ∧j. j ∈ J ⇒ Y' j = X j -' Y j ∩ space M ∧ j. j ∈ J ⇒ Y j ∈ sets
(M' j) by auto
    let ?E = prod-emb I M' J (PiE J Y)
    from Y have (∩j∈J. Y' j) = ?X -' ?E ∩ space M
      using J <I ≠ {}> measurable-space[OF rv] by (auto simp: prod-emb-def
PiE-iff split: if-split-asm)
    then have emeasure M (∩j∈J. Y' j) = emeasure M (?X -' ?E ∩ space M)
      by simp
    also have ... = emeasure ?D ?E
      using Y J by (intro emeasure-distr[symmetric] X sets-PiM-I) auto
    also have ... = emeasure ?P' ?E
      using <?D = ?P'> by simp
    also have ... = (∏ i∈J. emeasure (?D' i) (Y i))
      using P.emeasure-PiM-emb[of J Y] J Y by (simp add: prod-emb-def)
    also have ... = (∏ i∈J. emeasure M (Y' i))
      using rv J Y by (simp add: emeasure-distr)
    finally have emeasure M (∩j∈J. Y' j) = (∏ i∈J. emeasure M (Y' i)) .
    then show prob (∩j∈J. Y' j) = (∏ i∈J. prob (Y' i))
      by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
    qed
  qed
qed

```

lemma (*in prob-space*) *indep-vars-iff-distr-eq-PiM'*:

fixes $I :: 'i \text{ set}$ **and** $X :: 'i \Rightarrow 'a \Rightarrow 'b$
assumes $I \neq \{\}$
assumes $rv: \bigwedge i. i \in I \Longrightarrow \text{random-variable } (M' i) (X i)$
shows $\text{indep-vars } M' X I \longleftrightarrow$
 $\text{distr } M (\prod_M i \in I. M' i) (\lambda x. \lambda i \in I. X i x) = (\prod_M i \in I. \text{distr } M (M' i)$
 $(X i))$
proof –
from *assms* **obtain** j **where** $j: j \in I$
by *auto*
define N' **where** $N' = (\lambda i. \text{if } i \in I \text{ then } M' i \text{ else } M' j)$
define Y **where** $Y = (\lambda i. \text{if } i \in I \text{ then } X i \text{ else } X j)$
have $rv: \text{random-variable } (N' i) (Y i)$ **for** i
using j **by** (*auto simp: N'-def Y-def intro: assms*)

have $\text{indep-vars } M' X I = \text{indep-vars } N' Y I$
by (*intro indep-vars-cong*) (*auto simp: N'-def Y-def*)
also have $\dots \longleftrightarrow \text{distr } M (\prod_M i \in I. N' i) (\lambda x. \lambda i \in I. Y i x) = (\prod_M i \in I. \text{distr}$
 $M (N' i) (Y i))$
by (*intro indep-vars-iff-distr-eq-PiM rv assms*)
also have $(\prod_M i \in I. N' i) = (\prod_M i \in I. M' i)$
by (*intro PiM-cong*) (*simp-all add: N'-def*)
also have $(\lambda x. \lambda i \in I. Y i x) = (\lambda x. \lambda i \in I. X i x)$
by (*simp-all add: Y-def fun-eq-iff*)
also have $(\prod_M i \in I. \text{distr } M (N' i) (Y i)) = (\prod_M i \in I. \text{distr } M (M' i) (X i))$
by (*intro PiM-cong distr-cong*) (*simp-all add: N'-def Y-def*)
finally show *?thesis* .
qed

lemma (*in prob-space*) *indep-varD*:
assumes *indep: indep-var Ma A Mb B*
assumes *sets: Xa ∈ sets Ma Xb ∈ sets Mb*
shows $\text{prob } ((\lambda x. (A x, B x)) -' (Xa \times Xb) \cap \text{space } M) =$
 $\text{prob } (A -' Xa \cap \text{space } M) * \text{prob } (B -' Xb \cap \text{space } M)$
proof –
have $\text{prob } ((\lambda x. (A x, B x)) -' (Xa \times Xb) \cap \text{space } M) =$
 $\text{prob } (\bigcap i \in UNIV. (\text{case-bool } A B i -' \text{case-bool } Xa Xb i \cap \text{space } M))$
by (*auto intro!: arg-cong[where f=prob] simp: UNIV-bool*)
also have $\dots = (\prod i \in UNIV. \text{prob } (\text{case-bool } A B i -' \text{case-bool } Xa Xb i \cap \text{space}$
 $M))$
using *indep unfolding indep-var-def*
by (*rule indep-varsD*) (*auto split: bool.split intro: sets*)
also have $\dots = \text{prob } (A -' Xa \cap \text{space } M) * \text{prob } (B -' Xb \cap \text{space } M)$
unfolding *UNIV-bool* **by** *simp*
finally show *?thesis* .
qed

lemma (*in prob-space*) *prob-indep-random-variable*:
assumes *ind[simp]: indep-var N X N Y*
assumes *[simp]: A ∈ sets N B ∈ sets N*

shows $\mathcal{P}(x \text{ in } M. X x \in A \wedge Y x \in B) = \mathcal{P}(x \text{ in } M. X x \in A) * \mathcal{P}(x \text{ in } M. Y x \in B)$

proof –

have $\mathcal{P}(x \text{ in } M. (X x) \in A \wedge (Y x) \in B) = \text{prob } ((\lambda x. (X x, Y x)) - ' (A \times B) \cap \text{space } M)$

by (*auto intro!*: *arg-cong*[**where** $f = \text{prob}$])

also have $\dots = \text{prob } (X - ' A \cap \text{space } M) * \text{prob } (Y - ' B \cap \text{space } M)$

by (*auto intro!*: *indep-varD*[**where** $Ma=N$ **and** $Mb=N$])

also have $\dots = \mathcal{P}(x \text{ in } M. X x \in A) * \mathcal{P}(x \text{ in } M. Y x \in B)$

by (*auto intro!*: *arg-cong2*[**where** $f = (*)$] *arg-cong*[**where** $f = \text{prob}$])

finally show *?thesis* .

qed

lemma (*in prob-space*)

assumes *indep-var S X T Y*

shows *indep-var-rv1: random-variable S X*

and *indep-var-rv2: random-variable T Y*

proof –

have $\forall i \in \text{UNIV}. \text{random-variable } (\text{case-bool } S \ T \ i) \ (\text{case-bool } X \ Y \ i)$

using *assms unfolding indep-var-def indep-vars-def* **by** *auto*

then show *random-variable S X random-variable T Y*

unfolding *UNIV-bool* **by** *auto*

qed

lemma (*in prob-space*) *indep-var-distribution-eq*:

indep-var S X T Y \longleftrightarrow random-variable S X \wedge random-variable T Y \wedge

*distr M S X \otimes_M distr M T Y = distr M (S \otimes_M T) ($\lambda x. (X x, Y x)$) (**is** - $\longleftrightarrow - \wedge - \wedge ?S \otimes_M ?T = ?J$)*

proof *safe*

assume *indep-var S X T Y*

then show *rvs: random-variable S X random-variable T Y*

by (*blast dest: indep-var-rv1 indep-var-rv2*)+

then have *XY: random-variable (S \otimes_M T) ($\lambda x. (X x, Y x)$)*

by (*rule measurable-Pair*)

interpret *X: prob-space ?S* **by** (*rule prob-space-distr*) *fact*

interpret *Y: prob-space ?T* **by** (*rule prob-space-distr*) *fact*

interpret *XY: pair-prob-space ?S ?T ..*

show *?S \otimes_M ?T = ?J*

proof (*rule pair-measure-eqI*)

show *sigma-finite-measure ?S ..*

show *sigma-finite-measure ?T ..*

fix *A B* **assume** *A: A \in sets ?S and B: B \in sets ?T*

have *emeasure ?J (A \times B) = emeasure M (($\lambda x. (X x, Y x)$) - ' (A \times B) \cap space M)*

using *A B* **by** (*intro emeasure-distr[OF XY]*) *auto*

also have $\dots = \text{emeasure } M \ (X - ' A \cap \text{space } M) * \text{emeasure } M \ (Y - ' B \cap \text{space } M)$


```

    using indep-varD[OF ‹indep-var S X T Y›, of A B] A B
    by (simp add: emeasure-eq-measure measure-nonneg ennreal-mult)
  also have ... = emeasure ?S A * emeasure ?T B
    using rvs A B by (simp add: emeasure-distr)
  finally show emeasure ?S A * emeasure ?T B = emeasure ?J (A × B) by
simp
qed simp
next
assume rvs: random-variable S X random-variable T Y
then have XY: random-variable (S ⊗M T) (λx. (X x, Y x))
  by (rule measurable-Pair)

let ?S = distr M S X and ?T = distr M T Y
interpret X: prob-space ?S by (rule prob-space-distr) fact
interpret Y: prob-space ?T by (rule prob-space-distr) fact
interpret XY: pair-prob-space ?S ?T ..

assume ?S ⊗M ?T = ?J

{ fix S and X
  have Int-stable {X -‘ A ∩ space M | A. A ∈ sets S}
  proof (safe intro!: Int-stableI)
    fix A B assume A ∈ sets S B ∈ sets S
    then show ∃ C. (X -‘ A ∩ space M) ∩ (X -‘ B ∩ space M) = (X -‘ C ∩
space M) ∧ C ∈ sets S
    by (intro exI[of - A ∩ B]) auto
  qed }
note Int-stable = this

show indep-var S X T Y unfolding indep-var-eq
proof (intro conjI indep-set-sigma-sets Int-stable rvs)
  show indep-set {X -‘ A ∩ space M | A. A ∈ sets S} {Y -‘ A ∩ space M | A.
A ∈ sets T}
  proof (safe intro!: indep-setI)
    { fix A assume A ∈ sets S then show X -‘ A ∩ space M ∈ sets M
      using ‹X ∈ measurable M S› by (auto intro: measurable-sets) }
    { fix A assume A ∈ sets T then show Y -‘ A ∩ space M ∈ sets M
      using ‹Y ∈ measurable M T› by (auto intro: measurable-sets) }
  next
  fix A B assume ab: A ∈ sets S B ∈ sets T
  then have prob ((X -‘ A ∩ space M) ∩ (Y -‘ B ∩ space M)) = emeasure
?J (A × B)
    using XY by (auto simp add: emeasure-distr emeasure-eq-measure mea-
sure-nonneg intro!: arg-cong[where f=prob])
  also have ... = emeasure (?S ⊗M ?T) (A × B)
    unfolding ‹?S ⊗M ?T = ?J› ..
  also have ... = emeasure ?S A * emeasure ?T B
    using ab by (simp add: Y.emeasure-pair-measure-Times)
  finally show prob ((X -‘ A ∩ space M) ∩ (Y -‘ B ∩ space M)) =

```

```

    prob (X -‘ A ∩ space M) * prob (Y -‘ B ∩ space M)
    using rvs ab by (simp add: emeasure-eq-measure emeasure-distr mea-
sure-nonneg ennreal-mult[symmetric])
  qed
  qed
  qed

```

```

lemma (in prob-space) distributed-joint-indep:
  assumes S: sigma-finite-measure S and T: sigma-finite-measure T
  assumes X: distributed M S X Px and Y: distributed M T Y Py
  assumes indep: indep-var S X T Y
  shows distributed M (S ⊗M T) (λx. (X x, Y x)) (λ(x, y). Px x * Py y)
  using indep-var-distribution-eq[of S X T Y] indep
  by (intro distributed-joint-indep'[OF S T X Y]) auto

```

```

lemma (in prob-space) indep-vars-nn-integral:
  assumes I: finite I indep-vars (λ-. borel) X I ∧ i ω. i ∈ I ⇒ 0 ≤ X i ω
  shows (∫+ω. (∏ i∈I. X i ω) ∂M) = (∏ i∈I. ∫+ω. X i ω ∂M)

```

proof cases

```

  assume I ≠ {}
  define Y where [abs-def]: Y i ω = (if i ∈ I then X i ω else 0) for i ω
  { fix i have i ∈ I ⇒ random-variable borel (X i)
    using I(2) by (cases i∈I) (auto simp: indep-vars-def) }
  note rv-X = this

```

```

  { fix i have random-variable borel (Y i)
    using I(2) by (cases i∈I) (auto simp: Y-def rv-X) }
  note rv-Y = this[measurable]

```

```

interpret Y: prob-space distr M borel (Y i) for i
  using I(2) by (cases i ∈ I) (auto intro!: prob-space-distr simp: indep-vars-def
prob-space-return)
interpret product-sigma-finite λi. distr M borel (Y i)

```

..

```

have indep-Y: indep-vars (λi. borel) Y I
  by (rule indep-vars-cong[THEN iffD1, OF - - - I(2)]) (auto simp: Y-def)

```

```

have (∫+ω. (∏ i∈I. X i ω) ∂M) = (∫+ω. (∏ i∈I. Y i ω) ∂M)
  using I(3) by (auto intro!: nn-integral-cong prod.cong simp add: Y-def max-def)
also have ... = (∫+ω. (∏ i∈I. ω i) ∂distr M (Pi_M I (λi. borel)) (λx. λi∈I. Y
i x))

```

by (subst nn-integral-distr) auto

```

also have ... = (∫+ω. (∏ i∈I. ω i) ∂Pi_M I (λi. distr M borel (Y i)))
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF ⟨I ≠ {}⟩ rv-Y indep-Y]

```

..

```

also have ... = (∏ i∈I. (∫+ω. ω ∂distr M borel (Y i)))
  by (rule product-nn-integral-prod) (auto intro: ⟨finite I⟩)
also have ... = (∏ i∈I. ∫+ω. X i ω ∂M)

```

by (intro prod.cong nn-integral-cong) (auto simp: nn-integral-distr Y-def rv-X)
 finally show ?thesis .
 qed (simp add: emeasure-space-1)

lemma (in prob-space)

fixes $X :: 'i \Rightarrow 'a \Rightarrow 'b :: \{\text{real-normed-field, banach, second-countable-topology}\}$
 assumes I : finite I indep-vars $(\lambda i. \text{borel } X i \wedge i. i \in I \implies \text{integrable } M (X i))$
 shows indep-vars-lebesgue-integral: $(\int \omega. (\prod_{i \in I}. X i \omega) \partial M) = (\prod_{i \in I}. \int \omega. X i \omega \partial M)$ (is ?eq)

and indep-vars-integrable: integrable $M (\lambda \omega. (\prod_{i \in I}. X i \omega))$ (is ?int)
 proof (induct rule: case-split)

assume $I \neq \{\}$

define Y where [abs-def]: $Y i \omega = (\text{if } i \in I \text{ then } X i \omega \text{ else } 0)$ for $i \omega$
 { fix i have $i \in I \implies \text{random-variable borel } (X i)$
 using $I(2)$ by (cases $i \in I$) (auto simp: indep-vars-def) }
 note $rv-X = \text{this}[\text{measurable}]$

{ fix i have random-variable borel $(Y i)$
 using $I(2)$ by (cases $i \in I$) (auto simp: Y-def rv-X) }
 note $rv-Y = \text{this}[\text{measurable}]$

{ fix i have integrable $M (Y i)$
 using $I(3)$ by (cases $i \in I$) (auto simp: Y-def) }
 note $int-Y = \text{this}$

interpret Y : prob-space distr M borel $(Y i)$ for i

using $I(2)$ by (cases $i \in I$) (auto intro!: prob-space-distr simp: indep-vars-def prob-space-return)

interpret product-sigma-finite $\lambda i. \text{distr } M \text{ borel } (Y i)$

..

have indep-Y: indep-vars $(\lambda i. \text{borel } Y i)$

by (rule indep-vars-cong[THEN iffD1, OF - - - $I(2)$]) (auto simp: Y-def)

have $(\int \omega. (\prod_{i \in I}. X i \omega) \partial M) = (\int \omega. (\prod_{i \in I}. Y i \omega) \partial M)$

using $I(3)$ by (simp add: Y-def)

also have $\dots = (\int \omega. (\prod_{i \in I}. \omega i) \partial \text{distr } M (Pi_M I (\lambda i. \text{borel } (Y i))))$ ($\lambda x. \lambda i \in I. Y i x$)

by (subst integral-distr) auto

also have $\dots = (\int \omega. (\prod_{i \in I}. \omega i) \partial Pi_M I (\lambda i. \text{distr } M \text{ borel } (Y i)))$

unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF $\langle I \neq \{\} \rangle$ rv-Y indep-Y]

..

also have $\dots = (\prod_{i \in I}. (\int \omega. \omega \partial \text{distr } M \text{ borel } (Y i)))$

by (rule product-integral-prod) (auto intro: $\langle \text{finite } I \rangle$ simp: integrable-distr-eq int-Y)

also have $\dots = (\prod_{i \in I}. \int \omega. X i \omega \partial M)$

by (intro prod.cong integral-cong)

(auto simp: integral-distr Y-def rv-X)

finally show ?eq .

```

have integrable (distr M (PiM I ( $\lambda i$ . borel)) ( $\lambda x$ .  $\lambda i \in I$ . Y i x)) ( $\lambda \omega$ . ( $\prod i \in I$ .  $\omega$ 
i))
  unfolding indep-vars-iff-distr-eq-PiM [THEN iffD1, OF  $\langle I \neq \{\} \rangle$  rv-Y indep-Y]
  by (intro product-integrable-prod [OF  $\langle$  finite I  $\rangle$ ])
    (simp add: integrable-distr-eq int-Y)
  then show ?int
    by (simp add: integrable-distr-eq Y-def)
qed (simp-all add: prob-space)

```

```

lemma (in prob-space)
  fixes X1 X2 :: 'a  $\Rightarrow$  'b::{real-normed-field, banach, second-countable-topology}
  assumes indep-var borel X1 borel X2 integrable M X1 integrable M X2
  shows indep-var-lebesgue-integral: ( $\int \omega$ . X1  $\omega$  * X2  $\omega$   $\partial M$ ) = ( $\int \omega$ . X1  $\omega$   $\partial M$ )
  * ( $\int \omega$ . X2  $\omega$   $\partial M$ ) (is ?eq)
    and indep-var-integrable: integrable M ( $\lambda \omega$ . X1  $\omega$  * X2  $\omega$ ) (is ?int)
unfolding indep-var-def
proof –
  have *: ( $\lambda \omega$ . X1  $\omega$  * X2  $\omega$ ) = ( $\lambda \omega$ .  $\prod i \in UNIV$ . (case-bool X1 X2 i  $\omega$ ))
    by (simp add: UNIV-bool mult.commute)
  have **: ( $\lambda$  -. borel) = case-bool borel borel
    by (rule ext, metis (full-types) bool.simps(3) bool.simps(4))
  show ?eq
    apply (subst *)
    apply (subst indep-vars-lebesgue-integral)
    apply (auto)
    apply (subst **, subst indep-var-def [symmetric], rule assms)
    apply (simp split: bool.split add: assms)
    by (simp add: UNIV-bool mult.commute)
  show ?int
    apply (subst *)
    apply (rule indep-vars-integrable)
    apply auto
    apply (subst **, subst indep-var-def [symmetric], rule assms)
    by (simp split: bool.split add: assms)
qed

```

end

11 Convolution Measure

theory *Convolution*

imports *Independent-Family*

begin

lemma (**in** *finite-measure*) *sigma-finite-measure: sigma-finite-measure M*

..

definition *convolution* :: ('a :: *ordered-euclidean-space*) *measure* \Rightarrow 'a *measure* \Rightarrow

'a measure (infix \star 50) where
convolution $M N = \text{distr } (M \otimes_M N) \text{ borel } (\lambda(x, y). x + y)$

lemma

shows *space-convolution*[simp]: *space* (convolution $M N$) = *space borel*
and *sets-convolution*[simp]: *sets* (convolution $M N$) = *sets borel*
and *measurable-convolution1*[simp]: *measurable* A (convolution $M N$) = *measurable* A *borel*
and *measurable-convolution2*[simp]: *measurable* (convolution $M N$) B = *measurable borel* B
by (*simp-all add: convolution-def*)

lemma *nn-integral-convolution*:

assumes *finite-measure* M *finite-measure* N
assumes [*measurable-cong*]: *sets* N = *sets borel* *sets* M = *sets borel*
assumes [*measurable*]: $f \in$ *borel-measurable borel*
shows $(\int^{+x}. f x \partial \text{convolution } M N) = (\int^{+x}. \int^{+y}. f (x + y) \partial N \partial M)$

proof –

interpret M : *finite-measure* M **by fact**

interpret N : *finite-measure* N **by fact**

interpret *pair-sigma-finite* $M N$..

show *?thesis*

unfolding *convolution-def*

by (*simp add: nn-integral-distr N.nn-integral-fst[symmetric]*)

qed

lemma *convolution-emeasure*:

assumes $A \in$ *sets borel* *finite-measure* M *finite-measure* N
assumes [*simp*]: *sets* N = *sets borel* *sets* M = *sets borel*
assumes [*simp*]: *space* M = *space* N *space* N = *space borel*
shows *emeasure* $(M \star N)$ $A = \int^{+x}. (\text{emeasure } N \{a. a + x \in A\}) \partial M$
using *assms* **by** (*auto intro!: nn-integral-cong simp del: nn-integral-indicator*
simp: nn-integral-convolution
nn-integral-indicator [symmetric] ac-simps split:split-indicator)

lemma *convolution-emeasure'*:

assumes [*simp*]: $A \in$ *sets borel*

assumes [*simp*]: *finite-measure* M *finite-measure* N

assumes [*simp*]: *sets* N = *sets borel* *sets* M = *sets borel*

shows *emeasure* $(M \star N)$ $A = \int^{+x}. \int^{+y}. (\text{indicator } A (x + y)) \partial N \partial M$

by (*auto simp del: nn-integral-indicator simp: nn-integral-convolution*

nn-integral-indicator[symmetric] borel-measurable-indicator)

lemma *convolution-finite*:

assumes [*simp*]: *finite-measure* M *finite-measure* N

assumes [*measurable-cong*]: *sets* N = *sets borel* *sets* M = *sets borel*

shows *finite-measure* $(M \star N)$

unfolding *convolution-def*

by (*intro finite-measure-pair-measure finite-measure.finite-measure-distr*) *auto*

lemma *convolution-emeasure-3*:

assumes [*simp*, *measurable*]: $A \in \text{sets borel}$
assumes [*simp*]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [*simp*]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$ *sets* $L = \text{sets borel}$
shows *emeasure* $(L \star (M \star N)) A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z)$
 $\partial N \partial M \partial L$
apply (*subst nn-integral-indicator[symmetric]*, *simp*)
apply (*subst nn-integral-convolution*,
auto intro!: *borel-measurable-indicator borel-measurable-indicator' convolution-finite*)
by (*rule nn-integral-cong*)
+ (*auto simp: semigroup-add-class.add.assoc*)

lemma *convolution-emeasure-3'*:

assumes [*simp*, *measurable*]: $A \in \text{sets borel}$
assumes [*simp*]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [*measurable-cong*, *simp*]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$ *sets* $L = \text{sets borel}$
shows *emeasure* $((L \star M) \star N) A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z)$
 $\partial N \partial M \partial L$
apply (*subst nn-integral-indicator[symmetric]*, *simp*)
apply (*subst nn-integral-convolution*)
apply (*simp-all add: convolution-finite*)
apply (*subst nn-integral-convolution*)
apply (*simp-all add: finite-measure.sigma-finite-measure sigma-finite-measure.borel-measurable-nn-integral*)
done

lemma *convolution-commutative*:

assumes [*simp*]: *finite-measure* M *finite-measure* N
assumes [*measurable-cong*, *simp*]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$
shows $(M \star N) = (N \star M)$
proof (*rule measure-eqI*)
interpret M : *finite-measure* M **by fact**
interpret N : *finite-measure* N **by fact**
interpret *pair-sigma-finite* $M N$..

show *sets* $(M \star N) = \text{sets } (N \star M)$ **by** *simp*

fix A **assume** $A \in \text{sets } (M \star N)$

then have $1[\text{measurable}]$: $A \in \text{sets borel}$ **by** *simp*

have *emeasure* $(M \star N) A = \int^+ x. \int^+ y. \text{indicator } A (x + y)$ $\partial N \partial M$ **by** (*auto intro!*: *convolution-emeasure'*)

also have $\dots = \int^+ x. \int^+ y. (\lambda(x,y). \text{indicator } A (x + y)) (x, y)$ $\partial N \partial M$ **by** (*auto intro!*: *nn-integral-cong*)

also have $\dots = \int^+ y. \int^+ x. (\lambda(x,y). \text{indicator } A (x + y)) (x, y)$ $\partial M \partial N$ **by** (*rule Fubini[symmetric]*) *simp*

also have $\dots = \text{emeasure } (N \star M) A$ **by** (*auto intro!*: *nn-integral-cong simp: add commute convolution-emeasure'*)

finally show *emeasure* $(M \star N) A = \text{emeasure } (N \star M) A$ **by** *simp*

qed

lemma *convolution-associative:*

assumes [*simp*]: *finite-measure M finite-measure N finite-measure L*
assumes [*simp*]: *sets N = sets borel sets M = sets borel sets L = sets borel*
shows $(L \star (M \star N)) = ((L \star M) \star N)$
by (*auto intro!: measure-eqI simp: convolution-emeasure-3 convolution-emeasure-3'*)

lemma (*in prob-space*) *sum-indep-random-variable:*

assumes *ind: indep-var borel X borel Y*
assumes [*simp, measurable*]: *random-variable borel X*
assumes [*simp, measurable*]: *random-variable borel Y*
shows *distr M borel* $(\lambda x. X x + Y x) = \text{convolution } (\text{distr } M \text{ borel } X) (\text{distr } M \text{ borel } Y)$
using *ind unfolding indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr intro!: arg-cong[where f = distr M borel]*)

lemma (*in prob-space*) *sum-indep-random-variable-lborel:*

assumes *ind: indep-var borel X borel Y*
assumes [*simp, measurable*]: *random-variable lborel X*
assumes [*simp, measurable*]: *random-variable lborel Y*
shows *distr M lborel* $(\lambda x. X x + Y x) = \text{convolution } (\text{distr } M \text{ lborel } X) (\text{distr } M \text{ lborel } Y)$
using *ind unfolding indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr o-def intro!: arg-cong[where f = distr M borel] cong: distr-cong*)

lemma *convolution-density:*

fixes $f g :: \text{real} \Rightarrow \text{ennreal}$
assumes [*measurable*]: $f \in \text{borel-measurable borel } g \in \text{borel-measurable borel}$
assumes [*simp*]: *finite-measure (density lborel f) finite-measure (density lborel g)*
shows *density lborel f* \star *density lborel g* = *density lborel* $(\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel})$
(is ?l = ?r)

proof (*intro measure-eqI*)

fix A **assume** $A \in \text{sets } ?l$
then have [*measurable*]: $A \in \text{sets borel}$
by *simp*

have $(\int^+ x. f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) \partial \text{lborel}) =$
 $(\int^+ x. (\int^+ y. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel}) \partial \text{lborel})$

proof (*intro nn-integral-cong-AE, eventually-elim*)

fix x

have $f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) =$
 $(\int^+ y. f x * (g y * \text{indicator } A (x + y)) \partial \text{lborel})$

by (*intro nn-integral-cmult[symmetric] auto*)

then show $f x * (\int^+ y. g y * \text{indicator } A (x + y) \partial \text{lborel}) =$
 $(\int^+ y. g y * (f x * \text{indicator } A (x + y)) \partial \text{lborel})$

by (*simp add: ac-simps*)

qed
also have $\dots = (\int^{+y}. (\int^{+x}. g\ y * (f\ x * \text{indicator}\ A\ (x + y))\ \partial\text{lborel})\ \partial\text{lborel})$
by (*intro lborel-pair.Fubini'*) *simp*
also have $\dots = (\int^{+y}. (\int^{+x}. f\ (x - y) * g\ y * \text{indicator}\ A\ x\ \partial\text{lborel})\ \partial\text{lborel})$
proof (*intro nn-integral-cong-AE, eventually-elim*)
fix y
have $(\int^{+x}. g\ y * (f\ x * \text{indicator}\ A\ (x + y))\ \partial\text{lborel}) =$
 $g\ y * (\int^{+x}. f\ x * \text{indicator}\ A\ (x + y)\ \partial\text{lborel})$
by (*intro nn-integral-cmult*) *auto*
also have $\dots = g\ y * (\int^{+x}. f\ (x - y) * \text{indicator}\ A\ x\ \partial\text{lborel})$
by (*subst nn-integral-real-affine[where c=1 and t=-y]*)
(auto simp add: one-ennreal-def[symmetric])
also have $\dots = (\int^{+x}. g\ y * (f\ (x - y) * \text{indicator}\ A\ x)\ \partial\text{lborel})$
by (*intro nn-integral-cmult[symmetric]*) *auto*
finally show $(\int^{+x}. g\ y * (f\ x * \text{indicator}\ A\ (x + y))\ \partial\text{lborel}) =$
 $(\int^{+x}. f\ (x - y) * g\ y * \text{indicator}\ A\ x\ \partial\text{lborel})$
by (*simp add: ac-simps*)
qed
also have $\dots = (\int^{+x}. (\int^{+y}. f\ (x - y) * g\ y * \text{indicator}\ A\ x\ \partial\text{lborel})\ \partial\text{lborel})$
by (*intro lborel-pair.Fubini'*) *simp*
finally show $\text{emeasure}\ ?l\ A = \text{emeasure}\ ?r\ A$
by (*auto simp: convolution-emeasure' nn-integral-density emeasure-density*
nn-integral-multc)
qed simp

lemma (*in prob-space*) *distributed-finite-measure-density*:
distributed M N X f \implies finite-measure (density N f)
using *finite-measure-distr[of X N] distributed-distr-eq-density[of M N X f]* **by**
simp

lemma (*in prob-space*) *distributed-convolution*:
fixes $f :: \text{real} \Rightarrow -$
fixes $g :: \text{real} \Rightarrow -$
assumes *indep: indep-var borel X borel Y*
assumes $X: \text{distributed}\ M\ \text{lborel}\ X\ f$
assumes $Y: \text{distributed}\ M\ \text{lborel}\ Y\ g$
shows *distributed M lborel $(\lambda x. X\ x + Y\ x) (\lambda x. \int^{+y}. f\ (x - y) * g\ y\ \partial\text{lborel})$*
unfolding *distributed-def*
proof safe
have $fg[\text{measurable}]: f \in \text{borel-measurable}\ \text{borel}\ g \in \text{borel-measurable}\ \text{borel}$
using *distributed-borel-measurable[OF X] distributed-borel-measurable[OF Y]*
by *simp-all*

show $(\lambda x. \int^{+x}. f\ (x - xa) * g\ xa\ \partial\text{lborel}) \in \text{borel-measurable}\ \text{lborel}$
by *measurable*

have $\text{distr}\ M\ \text{borel}\ (\lambda x. X\ x + Y\ x) = (\text{distr}\ M\ \text{borel}\ X\ \star\ \text{distr}\ M\ \text{borel}\ Y)$
using *distributed-measurable[OF X] distributed-measurable[OF Y]*


```

  by (intro sum-indep-random-variable) (auto simp: indep)
  also have ... = (density lborel f ★ density lborel g)
  using distributed-distr-eq-density[OF X] distributed-distr-eq-density[OF Y]
  by (simp cong: distr-cong)
  also have ... = density lborel (λx. ∫+ y. f (x - y) * g y ∂lborel)
  proof (rule convolution-density)
    show finite-measure (density lborel f)
      using X by (rule distributed-finite-measure-density)
    show finite-measure (density lborel g)
      using Y by (rule distributed-finite-measure-density)
  qed fact+
  finally show distr M lborel (λx. X x + Y x) = density lborel (λx. ∫+ y. f (x -
y) * g y ∂lborel)
  by (simp cong: distr-cong)
  show random-variable lborel (λx. X x + Y x)
  using distributed-measurable[OF X] distributed-measurable[OF Y] by simp
qed

```

lemma *prob-space-convolution-density*:

```

  fixes f:: real ⇒ -
  fixes g:: real ⇒ -
  assumes [measurable]: f ∈ borel-measurable borel
  assumes [measurable]: g ∈ borel-measurable borel
  assumes gt-0[simp]: ∧x. 0 ≤ f x ∧x. 0 ≤ g x
  assumes prob-space (density lborel f) (is prob-space ?F)
  assumes prob-space (density lborel g) (is prob-space ?G)
  shows prob-space (density lborel (λx. ∫+ y. f (x - y) * g y ∂lborel)) (is prob-space
?D)
  proof (subst convolution-density[symmetric])
    interpret F: prob-space ?F by fact
    show finite-measure ?F by unfold-locales
    interpret G: prob-space ?G by fact
    show finite-measure ?G by unfold-locales
    interpret FG: pair-prob-space ?F ?G ..

    show prob-space (density lborel f ★ density lborel g)
      unfolding convolution-def by (rule FG.prob-space-distr) simp
  qed simp-all

```

end

12 Information theory

```

theory Information
imports
  Independent-Family
begin

```

lemma *log-le*: $1 < a \implies 0 < x \implies x \leq y \implies \log a x \leq \log a y$

by (subst log-le-cancel-iff) auto

lemma *log-less*: $1 < a \implies 0 < x \implies x < y \implies \log a x < \log a y$
by (subst log-less-cancel-iff) auto

lemma *sum-cartesian-product'*:
 $(\sum x \in A \times B. f x) = (\sum x \in A. \text{sum } (\lambda y. f (x, y)) B)$
unfolding *sum.cartesian-product* **by** *simp*

lemma *split-pairs*:
 $((A, B) = X) \longleftrightarrow (\text{fst } X = A \wedge \text{snd } X = B)$ **and**
 $(X = (A, B)) \longleftrightarrow (\text{fst } X = A \wedge \text{snd } X = B)$ **by** *auto*

12.1 Information theory

locale *information-space* = *prob-space* +
fixes *b* :: *real* **assumes** *b-gt-1*: $1 < b$

context *information-space*
begin

Introduce some simplification rules for logarithm of base *b*.

lemma *log-neg-const*:
assumes $x \leq 0$
shows $\log b x = \log b 0$
proof –
 { **fix** *u* :: *real*
 have $x \leq 0$ **by** *fact*
 also have $0 < \exp u$
 using *exp-gt-zero* .
 finally have $\exp u \neq x$
 by *auto* }
 then show $\log b x = \log b 0$
 by (*simp add: log-def ln-real-def*)
qed

lemma *log-mult-eq*:
 $\log b (A * B) = (\text{if } 0 < A * B \text{ then } \log b |A| + \log b |B| \text{ else } \log b 0)$
using *log-mult*[of *b* *|A|* *|B|*] *b-gt-1* *log-neg-const*[of *A* * *B*]
by (*auto simp: zero-less-mult-iff mult-le-0-iff*)

lemma *log-inverse-eq*:
 $\log b (\text{inverse } B) = (\text{if } 0 < B \text{ then } - \log b B \text{ else } \log b 0)$
using *log-inverse*[of *b* *B*] *log-neg-const*[of *inverse B*] *b-gt-1* **by** *simp*

lemma *log-divide-eq*:
 $\log b (A / B) = (\text{if } 0 < A * B \text{ then } \log b |A| - \log b |B| \text{ else } \log b 0)$
unfolding *divide-inverse* *log-mult-eq* *log-inverse-eq* *abs-inverse*
by (*auto simp: zero-less-mult-iff mult-le-0-iff*)

lemmas *log-simps* = *log-mult-eq* *log-inverse-eq* *log-divide-eq*

end

12.2 Kullback–Leibler divergence

The Kullback–Leibler divergence is also known as relative entropy or Kullback–Leibler distance.

definition

entropy-density b M N = $\log b \circ \text{enn2real} \circ \text{RN-deriv } M$ N

definition

KL-divergence b M N = $\text{integral}^L N$ (*entropy-density* b M N)

lemma *measurable-entropy-density[measurable]*: *entropy-density* b M N \in *borel-measurable* M

unfolding *entropy-density-def* **by** *auto*

lemma (**in** *sigma-finite-measure*) *KL-density*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $1 < b$

assumes $f[\text{measurable}]$: $f \in \text{borel-measurable } M$ **and** nn : AE x *in* M . $0 \leq f$ x

shows *KL-divergence* b M (*density* M f) = $(\int x. f$ x * $\log b$ (f x) ∂M)

unfolding *KL-divergence-def*

proof (*subst integral-real-density*)

show $[\text{measurable}]$: *entropy-density* b M (*density* M $(\lambda x. \text{ennreal } (f$ $x))$) \in *borel-measurable* M

using f

by (*auto simp: comp-def entropy-density-def*)

have *density* M (*RN-deriv* M (*density* M f)) = *density* M f

using f nn **by** (*intro density-RN-deriv-density*) *auto*

then have *eq*: AE x *in* M . *RN-deriv* M (*density* M f) x = f x

using f nn **by** (*intro density-unique*) *auto*

show $(\int x. f$ x * *entropy-density* b M (*density* M $(\lambda x. \text{ennreal } (f$ $x))$) x ∂M) = $(\int x. f$ x * $\log b$ (f x) ∂M)

apply (*intro integral-cong-AE*)

apply *measurable*

using *eq nn*

apply *eventually-elim*

apply (*auto simp: entropy-density-def*)

done

qed *fact+*

lemma (**in** *sigma-finite-measure*) *KL-density-density*:

fixes f $g :: 'a \Rightarrow \text{real}$

assumes $1 < b$

assumes f : $f \in \text{borel-measurable } M$ AE x *in* M . $0 \leq f$ x

assumes g : $g \in \text{borel-measurable } M$ AE x *in* M . $0 \leq g$ x

assumes ac : $AE\ x\ in\ M.\ f\ x = 0 \longrightarrow g\ x = 0$
shows $KL\text{-divergence}\ b\ (density\ M\ f)\ (density\ M\ g) = (\int\ x.\ g\ x * \log\ b\ (g\ x / f\ x)\ \partial M)$
proof –
interpret Mf : $sigma\text{-finite-measure}\ density\ M\ f$
using f **by** $(subst\ sigma\text{-finite-iff-density-finite})\ auto$
have $KL\text{-divergence}\ b\ (density\ M\ f)\ (density\ M\ g) =$
 $KL\text{-divergence}\ b\ (density\ M\ f)\ (density\ (density\ M\ f)\ (\lambda x.\ g\ x / f\ x))$
using $f\ g\ ac$ **by** $(subst\ density\text{-density-divide})\ simp\text{-all}$
also have $\dots = (\int\ x.\ (g\ x / f\ x) * \log\ b\ (g\ x / f\ x)\ \partial density\ M\ f)$
using $f\ g\ \langle 1 < b \rangle$ **by** $(intro\ Mf.KL\text{-density})\ (auto\ simp:\ AE\text{-density})$
also have $\dots = (\int\ x.\ g\ x * \log\ b\ (g\ x / f\ x)\ \partial M)$
using $ac\ f\ g\ \langle 1 < b \rangle$ **by** $(subst\ integral\text{-density})\ (auto\ intro!\ : integral\text{-cong-AE})$
finally show $?thesis$.
qed

lemma (in $information\text{-space}$) $KL\text{-gt-0}$:

fixes $D :: 'a \Rightarrow real$
assumes $prob\text{-space}\ (density\ M\ D)$
assumes D : $D \in borel\text{-measurable}\ M\ AE\ x\ in\ M.\ 0 \leq D\ x$
assumes int : $integrable\ M\ (\lambda x.\ D\ x * \log\ b\ (D\ x))$
assumes A : $density\ M\ D \neq M$
shows $0 < KL\text{-divergence}\ b\ M\ (density\ M\ D)$

proof –

interpret N : $prob\text{-space}\ density\ M\ D$ **by** $fact$

obtain A **where** $A \in sets\ M\ emeasure\ (density\ M\ D)\ A \neq emeasure\ M\ A$
using $measure\text{-eqI}[of\ density\ M\ D\ M]\ \langle density\ M\ D \neq M \rangle$ **by** $auto$

let $?D\text{-set} = \{x \in space\ M.\ D\ x \neq 0\}$

have $[simp,\ intro]$: $?D\text{-set} \in sets\ M$

using D **by** $auto$

have $D\text{-neg}$: $(\int^+ x.\ ennreal\ (-\ D\ x)\ \partial M) = 0$

using D **by** $(subst\ nn\text{-integral-0-iff-AE})\ (auto\ simp:\ ennreal\text{-neg})$

have $(\int^+ x.\ ennreal\ (D\ x)\ \partial M) = emeasure\ (density\ M\ D)\ (space\ M)$

using D **by** $(simp\ add:\ emeasure\text{-density}\ cong:\ nn\text{-integral-cong})$

then have $D\text{-pos}$: $(\int^+ x.\ ennreal\ (D\ x)\ \partial M) = 1$

using $N.emeasure\text{-space-1}$ **by** $simp$

have $integrable\ M\ D$

using $D\ D\text{-pos}\ D\text{-neg}$ **unfolding** $real\text{-integrable-def}\ real\text{-lebesgue-integral-def}$ **by** $simp\text{-all}$

then have $integral^L\ M\ D = 1$

using $D\ D\text{-pos}\ D\text{-neg}$ **by** $(simp\ add:\ real\text{-lebesgue-integral-def})$

have $0 \leq 1 - measure\ M\ ?D\text{-set}$

using $prob\text{-le-1}$ **by** $(auto\ simp:\ field\text{-simps})$

```

also have ... = (∫ x. D x - indicator ?D-set x ∂M)
  using ⟨integrable M D⟩ ⟨integralL M D = 1⟩
  by (simp add: emeasure-eq-measure)
also have ... < (∫ x. D x * (ln b * log b (D x)) ∂M)
proof (rule integral-less-AE)
  show integrable M (λx. D x - indicator ?D-set x)
    using ⟨integrable M D⟩ by (auto simp: less-top[symmetric])
next
from integrable-mult-left(1)[OF int, of ln b]
show integrable M (λx. D x * (ln b * log b (D x)))
  by (simp add: ac-simps)
next
show emeasure M {x∈space M. D x ≠ 1 ∧ D x ≠ 0} ≠ 0
proof
  assume eq-0: emeasure M {x∈space M. D x ≠ 1 ∧ D x ≠ 0} = 0
  then have disj: AE x in M. D x = 1 ∨ D x = 0
    using D(1) by (auto intro!: AE-I[OF subset-refl] sets.sets-Collect)

  have emeasure M {x∈space M. D x = 1} = (∫+ x. indicator {x∈space M.
D x = 1} x ∂M)
    using D(1) by auto
  also have ... = (∫+ x. ennreal (D x) ∂M)
  using disj by (auto intro!: nn-integral-cong-AE simp: indicator-def one-ennreal-def)
  finally have AE x in M. D x = 1
    using D D-pos by (intro AE-I-eq-1) auto
  then have (∫+ x. indicator A x ∂M) = (∫+ x. ennreal (D x) * indicator A
x ∂M)
    by (intro nn-integral-cong-AE) (auto simp: one-ennreal-def[symmetric])
  also have ... = density M D A
    using ⟨A ∈ sets M⟩ D by (simp add: emeasure-density)
  finally show False using ⟨A ∈ sets M⟩ ⟨emeasure (density M D) A ≠
emeasure M A⟩ by simp
qed
show {x∈space M. D x ≠ 1 ∧ D x ≠ 0} ∈ sets M
  using D(1) by (auto intro: sets.sets-Collect-conj)

show AE t in M. t ∈ {x∈space M. D x ≠ 1 ∧ D x ≠ 0} →
  D t - indicator ?D-set t ≠ D t * (ln b * log b (D t))
  using D(2)
proof (eventually-elim, safe)
  fix t assume Dt: t ∈ space M D t ≠ 1 D t ≠ 0 0 ≤ D t
    and eq: D t - indicator ?D-set t = D t * (ln b * log b (D t))

  have D t - 1 = D t - indicator ?D-set t
    using Dt by simp
  also note eq
  also have D t * (ln b * log b (D t)) = - D t * ln (1 / D t)
    using b-gt-1 ⟨D t ≠ 0⟩ ⟨0 ≤ D t⟩
    by (simp add: log-def ln-div less-le)

```

```

finally have  $\ln (1 / D t) = 1 / D t - 1$ 
  using  $\langle D t \neq 0 \rangle$  by (auto simp: field-simps)
from ln-eq-minus-one[OF - this]  $\langle D t \neq 0 \rangle \langle 0 \leq D t \rangle \langle D t \neq 1 \rangle$ 
show False by auto
qed

show AE  $t$  in  $M$ .  $D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
  using  $D(2)$  AE-space
proof eventually-elim
  fix  $t$  assume  $t \in \text{space } M$   $0 \leq D t$ 
  show  $D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
  proof cases
    assume asm:  $D t \neq 0$ 
    then have  $0 < D t$  using  $\langle 0 \leq D t \rangle$  by auto
    then have  $0 < 1 / D t$  by auto
    have  $D t - \text{indicator } ?D\text{-set } t \leq - D t * (1 / D t - 1)$ 
      using asm  $\langle t \in \text{space } M \rangle$  by (simp add: field-simps)
    also have  $- D t * (1 / D t - 1) \leq - D t * \ln (1 / D t)$ 
      using ln-le-minus-one  $\langle 0 < 1 / D t \rangle$  by (intro mult-left-mono-neg) auto
    also have  $\dots = D t * (\ln b * \log b (D t))$ 
      using  $\langle 0 < D t \rangle$  b-gt-1
      by (simp-all add: log-def ln-div)
    finally show ?thesis by simp
  qed simp
qed
also have  $\dots = (\int x. \ln b * (D x * \log b (D x)) \partial M)$ 
  by (simp add: ac-simps)
also have  $\dots = \ln b * (\int x. D x * \log b (D x) \partial M)$ 
  using int by simp
finally show ?thesis
  using b-gt-1  $D$  by (subst KL-density) (auto simp: zero-less-mult-iff)
qed

lemma (in sigma-finite-measure) KL-same-eq-0: KL-divergence  $b$   $M$   $M = 0$ 
proof -
  have AE  $x$  in  $M$ .  $1 = \text{RN-deriv } M$   $M$   $x$ 
  proof (rule RN-deriv-unique)
    show density  $M$   $(\lambda x. 1) = M$ 
    apply (auto intro!: measure-eqI emeasure-density)
    apply (subst emeasure-density)
    apply auto
    done
  qed auto
  then have AE  $x$  in  $M$ .  $\log b (\text{enn2real } (\text{RN-deriv } M$   $M$   $x)) = 0$ 
    by (elim AE-mp) simp
  from integral-cong-AE[OF - - this]
  have integralL  $M$  (entropy-density  $b$   $M$   $M) = 0$ 
    by (simp add: entropy-density-def comp-def)

```

then show *KL-divergence* $b M M = 0$
unfolding *KL-divergence-def*
by *auto*
qed

lemma (in *information-space*) *KL-eq-0-iff-eq*:
fixes $D :: 'a \Rightarrow \text{real}$
assumes *prob-space* (*density* $M D$)
assumes $D: D \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq D x$
assumes *int*: *integrable* $M (\lambda x. D x * \log b (D x))$
shows *KL-divergence* $b M (\text{density } M D) = 0 \longleftrightarrow \text{density } M D = M$
using *KL-same-eq-0[of b]* *KL-gt-0[OF assms]*
by (*auto simp: less-le*)

lemma (in *information-space*) *KL-eq-0-iff-eq-ac*:
fixes $D :: 'a \Rightarrow \text{real}$
assumes *prob-space* N
assumes *ac*: *absolutely-continuous* $M N$ *sets* $N = \text{sets } M$
assumes *int*: *integrable* $N (\text{entropy-density } b M N)$
shows *KL-divergence* $b M N = 0 \longleftrightarrow N = M$

proof –

interpret N : *prob-space* N **by** *fact*
have *finite-measure* N **by** *unfold-locales*
from *real-RN-deriv[OF this ac]* **obtain** D
where D :
random-variable borel D
AE $x \text{ in } M. \text{RN-deriv } M N x = \text{ennreal } (D x)$
AE $x \text{ in } N. 0 < D x$
 $\bigwedge x. 0 \leq D x$
by *this auto*

have $N = \text{density } M (\text{RN-deriv } M N)$
using *ac* **by** (*rule density-RN-deriv[symmetric]*)
also have $\dots = \text{density } M D$
using D **by** (*auto intro!: density-cong*)
finally have $N: N = \text{density } M D$.

from *absolutely-continuous-AE[OF ac(2,1) D(2)]* D *b-gt-1 ac measurable-entropy-density*
have *integrable* $N (\lambda x. \log b (D x))$
by (*intro integrable-cong-AE[THEN iffD2, OF - - - int]*)
(auto simp: N entropy-density-def)
with D *b-gt-1* **have** *integrable* $M (\lambda x. D x * \log b (D x))$
by (*subst integrable-real-density[symmetric]*) (*auto simp: N[symmetric] comp-def*)
with $\langle \text{prob-space } N \rangle D$ **show** *?thesis*
unfolding N
by (*intro KL-eq-0-iff-eq*) *auto*
qed

lemma (in *information-space*) *KL-nonneg*:

assumes *prob-space* (*density* $M D$)
assumes $D: D \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq D x$
assumes *int*: *integrable* $M (\lambda x. D x * \log b (D x))$
shows $0 \leq \text{KL-divergence } b M (\text{density } M D)$
using *KL-gt-0*[*OF assms*] **by** (*cases density* $M D = M$) (*auto simp: KL-same-eq-0*)

lemma (*in sigma-finite-measure*) *KL-density-density-nonneg*:
fixes $f g :: 'a \Rightarrow \text{real}$
assumes $1 < b$
assumes $f: f \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq f x \text{ prob-space } (\text{density } M f)$
assumes $g: g \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq g x \text{ prob-space } (\text{density } M g)$
assumes *ac*: $\text{AE } x \text{ in } M. f x = 0 \longrightarrow g x = 0$
assumes *int*: *integrable* $M (\lambda x. g x * \log b (g x / f x))$
shows $0 \leq \text{KL-divergence } b (\text{density } M f) (\text{density } M g)$
proof –
interpret Mf : *prob-space density* $M f$ **by** *fact*
interpret Mf : *information-space density* $M f b$ **by** *standard fact*
have *eq*: $\text{density } (\text{density } M f) (\lambda x. g x / f x) = \text{density } M g$ (**is** $?DD = -$)
using $f g ac$ **by** (*subst density-density-divide*) *simp-all*

have $0 \leq \text{KL-divergence } b (\text{density } M f) (\text{density } (\text{density } M f) (\lambda x. g x / f x))$
proof (*rule* $Mf.KL-nonneg$)
show *prob-space* $?DD$ **unfolding** *eq* **by** *fact*
from $f g$ **show** $(\lambda x. g x / f x) \in \text{borel-measurable } (\text{density } M f)$
by *auto*
show $\text{AE } x \text{ in } \text{density } M f. 0 \leq g x / f x$
using $f g$ **by** (*auto simp: AE-density*)
show *integrable* $(\text{density } M f) (\lambda x. g x / f x * \log b (g x / f x))$
using $\langle 1 < b \rangle f g ac$
by (*subst integrable-density*)
(auto intro!: integrable-cong-AE[THEN iffD2, OF - - - int] measurable-If)

qed
also have $\dots = \text{KL-divergence } b (\text{density } M f) (\text{density } M g)$
using $f g ac$ **by** (*subst density-density-divide*) *simp-all*
finally show *thesis* .
qed

12.3 Finite Entropy

definition (*in information-space*) *finite-entropy* $:: 'b \text{ measure} \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow \text{real}) \Rightarrow \text{bool}$

where

finite-entropy $S X f \longleftrightarrow$
distributed $M S X f \wedge$
integrable $S (\lambda x. f x * \log b (f x)) \wedge$
 $(\forall x \in \text{space } S. 0 \leq f x)$

lemma (in *information-space*) *finite-entropy-simple-function*:
assumes *X*: *simple-function* *M* *X*
shows *finite-entropy* (*count-space* (*X*'*space* *M*)) *X* (λa . *measure* *M* {*x* \in *space* *M*. *X* *x* = *a*})
unfolding *finite-entropy-def*
proof *safe*
have [*simp*]: *finite* (*X* '*space* *M*)
using *X* **by** (*auto simp: simple-function-def*)
then show *integrable* (*count-space* (*X* '*space* *M*))
(λx . *prob* {*xa* \in *space* *M*. *X* *xa* = *x*} * *log* *b* (*prob* {*xa* \in *space* *M*. *X* *xa* = *x*}))
by (*rule integrable-count-space*)
have *d*: *distributed* *M* (*count-space* (*X* '*space* *M*)) *X* (λx . *ennreal* (*if* *x* \in *X*'*space* *M* *then* *prob* {*xa* \in *space* *M*. *X* *xa* = *x*} *else* 0))
by (*rule distributed-simple-function-superset*[*OF X*]) (*auto intro!*: *arg-cong*[**where** *f*=*prob*])
show *distributed* *M* (*count-space* (*X* '*space* *M*)) *X* (λx . *ennreal* (*prob* {*xa* \in *space* *M*. *X* *xa* = *x*}))
by (*rule distributed-cong-density*[*THEN iffD1*, *OF* - - - *d*]) *auto*
qed (*rule measure-nonneg*)

lemma *ac-fst*:
assumes *sigma-finite-measure* *T*
shows *absolutely-continuous* *S* (*distr* (*S* \otimes_M *T*) *S* *fst*)
proof -
interpret *sigma-finite-measure* *T* **by** *fact*
{ **fix** *A* **assume** *A*: *A* \in *sets* *S* *emeasure* *S* *A* = 0
then have *fst* -' *A* \cap *space* (*S* \otimes_M *T*) = *A* \times *space* *T*
by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)
with *A* **have** *emeasure* (*S* \otimes_M *T*) (*fst* -' *A* \cap *space* (*S* \otimes_M *T*)) = 0
by (*simp add: emeasure-pair-measure-Times*) }
then show ?*thesis*
unfolding *absolutely-continuous-def*
apply (*auto simp: null-sets-distr-iff*)
apply (*auto simp: null-sets-def intro!: measurable-sets*)
done
qed

lemma *ac-snd*:
assumes *sigma-finite-measure* *T*
shows *absolutely-continuous* *T* (*distr* (*S* \otimes_M *T*) *T* *snd*)
proof -
interpret *sigma-finite-measure* *T* **by** *fact*
{ **fix** *A* **assume** *A*: *A* \in *sets* *T* *emeasure* *T* *A* = 0
then have *snd* -' *A* \cap *space* (*S* \otimes_M *T*) = *space* *S* \times *A*
by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)
with *A* **have** *emeasure* (*S* \otimes_M *T*) (*snd* -' *A* \cap *space* (*S* \otimes_M *T*)) = 0
by (*simp add: emeasure-pair-measure-Times*) }
then show ?*thesis*
unfolding *absolutely-continuous-def*

```

apply (auto simp: null-sets-distr-iff)
apply (auto simp: null-sets-def intro!: measurable-sets)
done
qed

```

```

lemma (in information-space) finite-entropy-integrable:
  finite-entropy  $S X Px \implies$  integrable  $S (\lambda x. Px x * \log b (Px x))$ 
unfolding finite-entropy-def by auto

```

```

lemma (in information-space) finite-entropy-distributed:
  finite-entropy  $S X Px \implies$  distributed  $M S X Px$ 
unfolding finite-entropy-def by auto

```

```

lemma (in information-space) finite-entropy-nn:
  finite-entropy  $S X Px \implies x \in \text{space } S \implies 0 \leq Px x$ 
by (auto simp: finite-entropy-def)

```

```

lemma (in information-space) finite-entropy-measurable:
  finite-entropy  $S X Px \implies Px \in S \rightarrow_M \text{borel}$ 
using distributed-real-measurable[of  $S Px M X$ ]
  finite-entropy-nn[of  $S X Px$ ] finite-entropy-distributed[of  $S X Px$ ] by auto

```

```

lemma (in information-space) subdensity-finite-entropy:
  fixes  $g :: 'b \Rightarrow \text{real}$  and  $f :: 'c \Rightarrow \text{real}$ 
  assumes  $T: T \in \text{measurable } P Q$ 
  assumes  $f: \text{finite-entropy } P X f$ 
  assumes  $g: \text{finite-entropy } Q Y g$ 
  assumes  $Y: Y = T \circ X$ 
  shows  $\text{AE } x \text{ in } P. g (T x) = 0 \longrightarrow f x = 0$ 
using subdensity[OF  $T$ , of  $M X \lambda x. \text{ennreal } (f x) Y \lambda x. \text{ennreal } (g x)$ ]
  finite-entropy-distributed[OF  $f$ ] finite-entropy-distributed[OF  $g$ ]
  finite-entropy-nn[OF  $f$ ] finite-entropy-nn[OF  $g$ ]
  assms
by auto

```

```

lemma (in information-space) finite-entropy-integrable-transform:
  finite-entropy  $S X Px \implies$  distributed  $M T Y Py \implies (\bigwedge x. x \in \text{space } T \implies 0 \leq Py x) \implies$ 
   $X = (\lambda x. f (Y x)) \implies f \in \text{measurable } T S \implies$  integrable  $T (\lambda x. Py x * \log b (Px (f x)))$ 
using distributed-transform-integrable[of  $M T Y Py S X Px f \lambda x. \log b (Px x)$ ]
using distributed-real-measurable[of  $S Px M X$ ]
by (auto simp: finite-entropy-def)

```

12.4 Mutual Information

```

definition (in prob-space)
  mutual-information  $b S T X Y =$ 
  KL-divergence  $b (\text{distr } M S X \otimes_M \text{distr } M T Y) (\text{distr } M (S \otimes_M T) (\lambda x.$ 

```

($X x, Y x$))

lemma (in *information-space*) *mutual-information-indep-vars*:

fixes $S T X Y$

defines $P \equiv \text{distr } M S X \otimes_M \text{distr } M T Y$

defines $Q \equiv \text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x))$

shows *indep-var* $S X T Y \longleftrightarrow$

(*random-variable* $S X \wedge$ *random-variable* $T Y \wedge$

absolutely-continuous $P Q \wedge$ *integrable* Q (*entropy-density* $b P Q$) \wedge

mutual-information $b S T X Y = 0$)

unfolding *indep-var-distribution-eq*

proof *safe*

assume *rv[measurable]*: *random-variable* $S X$ *random-variable* $T Y$

interpret X : *prob-space* $\text{distr } M S X$

by (*rule prob-space-distr*) *fact*

interpret Y : *prob-space* $\text{distr } M T Y$

by (*rule prob-space-distr*) *fact*

interpret XY : *pair-prob-space* $\text{distr } M S X \text{distr } M T Y$ **by** *standard*

interpret P : *information-space* $P b$ **unfolding** P -*def* **by** *standard* (*rule b-gt-1*)

interpret Q : *prob-space* Q **unfolding** Q -*def*

by (*rule prob-space-distr*) *simp*

{ **assume** $\text{distr } M S X \otimes_M \text{distr } M T Y = \text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x))$

then have [*simp*]: $Q = P$ **unfolding** Q -*def* P -*def* **by** *simp*

show *ac*: *absolutely-continuous* $P Q$ **by** (*simp add*: *absolutely-continuous-def*)

then have *ed*: *entropy-density* $b P Q \in$ *borel-measurable* P

by *simp*

have *AE* x *in* P . $1 =$ *RN-deriv* $P Q x$

proof (*rule P.RN-deriv-unique*)

show *density* $P (\lambda x. 1) = Q$

unfolding $\langle Q = P \rangle$ **by** (*intro measure-eqI*) (*auto simp*: *emeasure-density*)

qed *auto*

then have *ae-0*: *AE* x *in* P . *entropy-density* $b P Q x = 0$

by *eventually-elim* (*auto simp*: *entropy-density-def*)

then have *integrable* P (*entropy-density* $b P Q$) \longleftrightarrow *integrable* $Q (\lambda x. 0::\text{real})$

using *ed* **unfolding** $\langle Q = P \rangle$ **by** (*intro integrable-cong-AE*) *auto*

then show *integrable* Q (*entropy-density* $b P Q$) **by** *simp*

from *ae-0* **have** *mutual-information* $b S T X Y = (\int x. 0 \partial P)$

unfolding *mutual-information-def* *KL-divergence-def* P -*def*[*symmetric*] Q -*def*[*symmetric*]

$\langle Q = P \rangle$

by (*intro integral-cong-AE*) *auto*

then show *mutual-information* $b S T X Y = 0$

by *simp* }

```

{ assume ac: absolutely-continuous  $P Q$ 
  assume int: integrable  $Q$  (entropy-density  $b P Q$ )
  assume I-eq-0: mutual-information  $b S T X Y = 0$ 

  have eq:  $Q = P$ 
  proof (rule P.KL-eq-0-iff-eq-ac[THEN iffD1])
    show prob-space  $Q$  by unfold-locales
    show absolutely-continuous  $P Q$  by fact
    show integrable  $Q$  (entropy-density  $b P Q$ ) by fact
    show sets  $Q = sets P$  by (simp add: P-def Q-def sets-pair-measure)
    show KL-divergence  $b P Q = 0$ 
    using I-eq-0 unfolding mutual-information-def by (simp add: P-def Q-def)
  qed
  then show distr  $M S X \otimes_M distr M T Y = distr M (S \otimes_M T) (\lambda x. (X x,$ 
Y x))
    unfolding P-def Q-def .. }
qed

```

abbreviation (in *information-space*)

mutual-information-Pow ($\mathcal{I}'(-; -)$) **where**
 $\mathcal{I}(X; Y) \equiv$ *mutual-information* b (*count-space* (X 'space M)) (*count-space* (Y 'space M)) $X Y$

lemma (in *information-space*)

fixes $Pxy :: 'b \times 'c \Rightarrow real$ **and** $Px :: 'b \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes Fx : *finite-entropy* $S X Px$ **and** Fy : *finite-entropy* $T Y Py$
assumes Fxy : *finite-entropy* ($S \otimes_M T$) ($\lambda x. (X x, Y x)$) Pxy
defines $f \equiv \lambda x. Pxy x * \log b (Pxy x / (Px (fst x) * Py (snd x)))$
shows *mutual-information-distr'*: *mutual-information* $b S T X Y = \text{integral}^L (S \otimes_M T) f$ (**is** $?M = ?R$)
and *mutual-information-nonneg'*: $0 \leq$ *mutual-information* $b S T X Y$

proof –

```

have  $Px$ : distributed  $M S X Px$  and  $Px$ -nn:  $\bigwedge x. x \in space S \Longrightarrow 0 \leq Px x$ 
  using  $Fx$  by (auto simp: finite-entropy-def)
have  $Py$ : distributed  $M T Y Py$  and  $Py$ -nn:  $\bigwedge x. x \in space T \Longrightarrow 0 \leq Py x$ 
  using  $Fy$  by (auto simp: finite-entropy-def)
have  $Pxy$ : distributed  $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$ 
  and  $Pxy$ -nn:  $\bigwedge x. x \in space (S \otimes_M T) \Longrightarrow 0 \leq Pxy x$ 
   $\bigwedge x y. x \in space S \Longrightarrow y \in space T \Longrightarrow 0 \leq Pxy (x, y)$ 
  using  $Fxy$  by (auto simp: finite-entropy-def space-pair-measure)

```

have [*measurable*]: $Px \in S \rightarrow_M borel$

using $Px Px$ -*nn* **by** (*intro distributed-real-measurable*)

have [*measurable*]: $Py \in T \rightarrow_M borel$

using $Py Py$ -*nn* **by** (*intro distributed-real-measurable*)

have *measurable-Pxy*[*measurable*]: $Pxy \in (S \otimes_M T) \rightarrow_M borel$

using $Pxy Pxy$ -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

```

have X[measurable]: random-variable S X
  using Px by auto
have Y[measurable]: random-variable T Y
  using Py by auto
interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..
interpret X: prob-space distr M S X using X by (rule prob-space-distr)
interpret Y: prob-space distr M T Y using Y by (rule prob-space-distr)
interpret XY: pair-prob-space distr M S X distr M T Y ..
let ?P = S  $\otimes_M$  T
let ?D = distr M ?P ( $\lambda x. (X x, Y x)$ )

{ fix A assume A  $\in$  sets S
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M S X) A = emeasure ?D (A  $\times$  space T)
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq1 = this
{ fix A assume A  $\in$  sets T
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M T Y) A = emeasure ?D (space S  $\times$  A)
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq2 = this

have distr-eq: distr M S X  $\otimes_M$  distr M T Y = density ?P ( $\lambda x. \text{ennreal } (Px (\text{fst } x) * Py (\text{snd } x))$ )
  unfolding Px(1)[THEN distributed-distr-eq-density] Py(1)[THEN distributed-distr-eq-density]
  proof (subst pair-measure-density)
    show ( $\lambda x. \text{ennreal } (Px x) \in \text{borel-measurable } S (\lambda y. \text{ennreal } (Py y)) \in \text{borel-measurable } T$ )
      using Px Py by (auto simp: distributed-def)
    show sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density, symmetric] ..
    show density (S  $\otimes_M$  T) ( $\lambda(x, y). \text{ennreal } (Px x) * \text{ennreal } (Py y) = \text{density } (S \otimes_M T) (\lambda x. \text{ennreal } (Px (\text{fst } x) * Py (\text{snd } x)))$ )
      using Px-nn Py-nn by (auto intro!: density-cong simp: distributed-def ennreal-mult space-pair-measure)
  qed fact

have M: ?M = KL-divergence b (density ?P ( $\lambda x. \text{ennreal } (Px (\text{fst } x) * Py (\text{snd } x))$ )) (density ?P ( $\lambda x. \text{ennreal } (Pxy x)$ ))
  unfolding mutual-information-def distr-eq Pxy(1)[THEN distributed-distr-eq-density]
  ..

from Px Py have f: ( $\lambda x. Px (\text{fst } x) * Py (\text{snd } x) \in \text{borel-measurable } ?P$ )
  by (intro borel-measurable-times) (auto intro: distributed-real-measurable measurable-fst'' measurable-snd'')
have PxPy-nonneg:  $\text{AE } x \text{ in } ?P. 0 \leq Px (\text{fst } x) * Py (\text{snd } x)$ 

```

using $Px\text{-}nn$ $Py\text{-}nn$ **by** (*auto simp: space-pair-measure*)

have A : ($AE\ x\ in\ ?P. Px\ (fst\ x) = 0 \longrightarrow Pxy\ x = 0$)
by (*rule subdensity-real[OF measurable-fst Pxy Px]*) (*insert Px-nn Pxy-nn, auto simp: space-pair-measure*)

moreover

have B : ($AE\ x\ in\ ?P. Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$)
by (*rule subdensity-real[OF measurable-snd Pxy Py]*) (*insert Py-nn Pxy-nn, auto simp: space-pair-measure*)

ultimately have ac : $AE\ x\ in\ ?P. Px\ (fst\ x) * Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
by *eventually-elim auto*

show $?M = ?R$

unfolding $M\ f\text{-}def$ **using** $Pxy\text{-}nn\ Px\text{-}nn\ Py\text{-}nn$
by (*intro ST.KL-density-density b-gt-1 f PxPy-nonneg ac*) (*auto simp: space-pair-measure*)

have X : $X = fst \circ (\lambda x. (X\ x, Y\ x))$ **and** Y : $Y = snd \circ (\lambda x. (X\ x, Y\ x))$
by *auto*

have *integrable* ($S \otimes_M T$) ($\lambda x. Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Px\ (fst\ x)) - Pxy\ x * \log\ b\ (Py\ (snd\ x))$)
using *finite-entropy-integrable[OF Fxy]*
using *finite-entropy-integrable-transform[OF Fx Pxy, of fst]*
using *finite-entropy-integrable-transform[OF Fy Pxy, of snd]*
by (*simp add: Pxy-nn*)

moreover have $f \in \text{borel-measurable}\ (S \otimes_M T)$
unfolding $f\text{-}def$ **using** $Px\ Py\ Pxy$
by (*auto intro: distributed-real-measurable measurable-fst'' measurable-snd'' intro!: borel-measurable-times borel-measurable-log borel-measurable-divide*)

ultimately have int : *integrable* ($S \otimes_M T$) f
apply (*rule integrable-cong-AE-imp*)
using $A\ B\ AE\text{-}space$
by *eventually-elim*
(auto simp: f-def log-divide-eq log-mult-eq field-simps space-pair-measure Px-nn Py-nn Pxy-nn less-le)

show $0 \leq ?M$ **unfolding** M

proof (*intro ST.KL-density-density-nonneg*)

show *prob-space* (*density* ($S \otimes_M T$) ($\lambda x. ennreal\ (Pxy\ x)$))
unfolding *distributed-distr-eq-density*[*OF Pxy, symmetric*]
using *distributed-measurable*[*OF Pxy*] **by** (*rule prob-space-distr*)

show *prob-space* (*density* ($S \otimes_M T$) ($\lambda x. ennreal\ (Px\ (fst\ x) * Py\ (snd\ x))$))
unfolding *distr-eq*[*symmetric*] **by** *unfold-locales*

show *integrable* ($S \otimes_M T$) ($\lambda x. Pxy\ x * \log\ b\ (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x)))$)
using int **unfolding** $f\text{-}def$.

qed (*insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure*)
qed

lemma (in *information-space*)

fixes $Pxy :: 'b \times 'c \Rightarrow \text{real}$ **and** $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$
assumes *sigma-finite-measure* S *sigma-finite-measure* T
assumes Px : *distributed* $M S X Px$ **and** Px -*nn*: $\bigwedge x. x \in \text{space } S \Longrightarrow 0 \leq Px x$
and Py : *distributed* $M T Y Py$ **and** Py -*nn*: $\bigwedge y. y \in \text{space } T \Longrightarrow 0 \leq Py y$
and Pxy : *distributed* $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$
and Pxy -*nn*: $\bigwedge x y. x \in \text{space } S \Longrightarrow y \in \text{space } T \Longrightarrow 0 \leq Pxy (x, y)$
defines $f \equiv \lambda x. Pxy x * \log b (Pxy x / (Px (fst x) * Py (snd x)))$
shows *mutual-information-distr*: *mutual-information* $b S T X Y = \text{integral}^L (S \otimes_M T) f$ (is $?M = ?R$)
and *mutual-information-nonneg*: *integrable* $(S \otimes_M T) f \Longrightarrow 0 \leq \text{mutual-information } b S T X Y$

proof –

have X [*measurable*]: *random-variable* $S X$
using Px **by** (*auto simp: distributed-def*)
have Y [*measurable*]: *random-variable* $T Y$
using Py **by** (*auto simp: distributed-def*)
have [*measurable*]: $Px \in S \rightarrow_M \text{borel}$
using Px Px -*nn* **by** (*intro distributed-real-measurable*)
have [*measurable*]: $Py \in T \rightarrow_M \text{borel}$
using Py Py -*nn* **by** (*intro distributed-real-measurable*)
have *measurable-Pxy*[*measurable*]: $Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$
using Pxy Pxy -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

interpret S : *sigma-finite-measure* S **by** *fact*

interpret T : *sigma-finite-measure* T **by** *fact*

interpret ST : *pair-sigma-finite* $S T$..

interpret X : *prob-space distr* $M S X$ **using** X **by** (*rule prob-space-distr*)

interpret Y : *prob-space distr* $M T Y$ **using** Y **by** (*rule prob-space-distr*)

interpret XY : *pair-prob-space distr* $M S X distr M T Y$..

let $?P = S \otimes_M T$

let $?D = \text{distr } M ?P (\lambda x. (X x, Y x))$

{ **fix** A **assume** $A \in \text{sets } S$

with X [*THEN measurable-space*] Y [*THEN measurable-space*]

have *emeasure* $(\text{distr } M S X) A = \text{emeasure } ?D (A \times \text{space } T)$

by (*auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]*) }

note *marginal-eq1* = *this*

{ **fix** A **assume** $A \in \text{sets } T$

with X [*THEN measurable-space*] Y [*THEN measurable-space*]

have *emeasure* $(\text{distr } M T Y) A = \text{emeasure } ?D (\text{space } S \times A)$

by (*auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]*) }

note *marginal-eq2* = *this*

have *distr-eq*: *distr* $M S X \otimes_M \text{distr } M T Y = \text{density } ?P (\lambda x. \text{ennreal } (Px (fst x) * Py (snd x)))$

unfolding $Px(1)$ [*THEN distributed-distr-eq-density*] $Py(1)$ [*THEN distributed-distr-eq-density*]

proof (*subst pair-measure-density*)

show $(\lambda x. \text{ennreal } (Px \ x)) \in \text{borel-measurable } S \ (\lambda y. \text{ennreal } (Py \ y)) \in$
borel-measurable } T
using $Px \ Py$ **by** *(auto simp: distributed-def)*
show *sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density, symmetric] ..*
show $\text{density } (S \otimes_M T) \ (\lambda(x, y). \text{ennreal } (Px \ x) * \text{ennreal } (Py \ y)) =$
 $\text{density } (S \otimes_M T) \ (\lambda x. \text{ennreal } (Px \ (\text{fst } x) * Py \ (\text{snd } x)))$
using $Px\text{-nn } Py\text{-nn}$ **by** *(auto intro!: density-cong simp: distributed-def en-
nreal-mult space-pair-measure)*
qed fact

have $M: ?M = \text{KL-divergence } b \ (\text{density } ?P \ (\lambda x. \text{ennreal } (Px \ (\text{fst } x) * Py \ (\text{snd } x))))$
 $(\text{density } ?P \ (\lambda x. \text{ennreal } (Pxy \ x)))$
unfolding *mutual-information-def distr-eq Pxy(1)[THEN distributed-distr-eq-density]*
..

from $Px \ Py$ **have** $f: (\lambda x. Px \ (\text{fst } x) * Py \ (\text{snd } x)) \in \text{borel-measurable } ?P$
by *(intro borel-measurable-times) (auto intro: distributed-real-measurable mea-
surable-fst'' measurable-snd'')*
have $PxPy\text{-nonneg}: AE \ x \ \text{in } ?P. 0 \leq Px \ (\text{fst } x) * Py \ (\text{snd } x)$
using $Px\text{-nn } Py\text{-nn}$ **by** *(auto simp: space-pair-measure)*

have $(AE \ x \ \text{in } ?P. Px \ (\text{fst } x) = 0 \longrightarrow Pxy \ x = 0)$
by *(rule subdensity-real[OF measurable-fst Pxy Px]) (insert Px-nn Pxy-nn, auto
simp: space-pair-measure)*
moreover
have $(AE \ x \ \text{in } ?P. Py \ (\text{snd } x) = 0 \longrightarrow Pxy \ x = 0)$
by *(rule subdensity-real[OF measurable-snd Pxy Py]) (insert Py-nn Pxy-nn,
auto simp: space-pair-measure)*
ultimately have $ac: AE \ x \ \text{in } ?P. Px \ (\text{fst } x) * Py \ (\text{snd } x) = 0 \longrightarrow Pxy \ x = 0$
by *eventually-elim auto*

show $?M = ?R$
unfolding $M \ f\text{-def}$
using $b\text{-gt-1 } f \ PxPy\text{-nonneg } ac \ Pxy\text{-nn}$
by *(intro ST.KL-density-density) (auto simp: space-pair-measure)*

assume $int: \text{integrable } (S \otimes_M T) \ f$
show $0 \leq ?M$ **unfolding** M
proof *(intro ST.KL-density-density-nonneg)*
show $\text{prob-space } (\text{density } (S \otimes_M T) \ (\lambda x. \text{ennreal } (Pxy \ x)))$
unfolding *distributed-distr-eq-density[OF Pxy, symmetric]*
using *distributed-measurable[OF Pxy] by (rule prob-space-distr)*
show $\text{prob-space } (\text{density } (S \otimes_M T) \ (\lambda x. \text{ennreal } (Px \ (\text{fst } x) * Py \ (\text{snd } x))))$
unfolding *distr-eq[symmetric] by unfold-locales*
show $\text{integrable } (S \otimes_M T) \ (\lambda x. Pxy \ x * \log b \ (Pxy \ x / (Px \ (\text{fst } x) * Py \ (\text{snd } x))))$
using int **unfolding** $f\text{-def}$.
qed *(insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure)*

qed

lemma (in information-space)

fixes $Pxy :: 'b \times 'c \Rightarrow \text{real}$ **and** $Px :: 'b \Rightarrow \text{real}$ **and** $P_y :: 'c \Rightarrow \text{real}$

assumes $\text{sigma-finite-measure } S$ $\text{sigma-finite-measure } T$

assumes $Px[\text{measurable}]$: distributed $M S X Px$ **and** $Px\text{-nn}$: $\bigwedge x. x \in \text{space } S \Rightarrow 0 \leq Px x$

and $P_y[\text{measurable}]$: distributed $M T Y P_y$ **and** $P_y\text{-nn}$: $\bigwedge x. x \in \text{space } T \Rightarrow 0 \leq P_y x$

and $Pxy[\text{measurable}]$: distributed $M (S \otimes_M T) (\lambda x. (X x, Y x)) Pxy$

and $Pxy\text{-nn}$: $\bigwedge x. x \in \text{space } (S \otimes_M T) \Rightarrow 0 \leq Pxy x$

assumes ae : $AE x$ in S . $AE y$ in T . $Pxy (x, y) = Px x * P_y y$

shows $\text{mutual-information-eq-0}$: $\text{mutual-information } b S T X Y = 0$

proof –

interpret S : $\text{sigma-finite-measure } S$ **by fact**

interpret T : $\text{sigma-finite-measure } T$ **by fact**

interpret ST : $\text{pair-sigma-finite } S T$..

note

$\text{distributed-real-measurable}[OF Px\text{-nn } Px, \text{measurable}]$

$\text{distributed-real-measurable}[OF P_y\text{-nn } P_y, \text{measurable}]$

$\text{distributed-real-measurable}[OF Pxy\text{-nn } Pxy, \text{measurable}]$

have $AE x$ in $S \otimes_M T$. $Px (\text{fst } x) = 0 \longrightarrow Pxy x = 0$

by ($\text{rule subdensity-real}[OF \text{measurable-fst } Pxy Px]$) ($\text{auto simp: } Px\text{-nn } Pxy\text{-nn}$ $\text{space-pair-measure}$)

moreover

have $AE x$ in $S \otimes_M T$. $P_y (\text{snd } x) = 0 \longrightarrow Pxy x = 0$

by ($\text{rule subdensity-real}[OF \text{measurable-snd } Pxy P_y]$) ($\text{auto simp: } P_y\text{-nn } Pxy\text{-nn}$ $\text{space-pair-measure}$)

moreover

have $AE x$ in $S \otimes_M T$. $Pxy x = Px (\text{fst } x) * P_y (\text{snd } x)$

by ($\text{intro } ST.AE\text{-pair-measure}$) ($\text{auto simp: } ae \text{ intro!}: \text{measurable-snd'' measurable-fst''}$)

ultimately have $AE x$ in $S \otimes_M T$. $Pxy x * \log b (Pxy x / (Px (\text{fst } x) * P_y (\text{snd } x))) = 0$

by $\text{eventually-elim simp}$

then have $(\int x. Pxy x * \log b (Pxy x / (Px (\text{fst } x) * P_y (\text{snd } x)))) \partial(S \otimes_M T) = (\int x. 0 \partial(S \otimes_M T))$

by ($\text{intro integral-cong-AE}$) auto

then show $?thesis$

by ($\text{subst mutual-information-distr}[OF \text{assms}(1-8)]$) ($\text{auto simp add: space-pair-measure}$)

qed

lemma (in information-space) $\text{mutual-information-simple-distributed}$:

assumes X : $\text{simple-distributed } M X Px$ **and** Y : $\text{simple-distributed } M Y P_y$

assumes XY : $\text{simple-distributed } M (\lambda x. (X x, Y x)) Pxy$

shows $\mathcal{I}(X ; Y) = (\sum (x, y) \in (\lambda x. (X x, Y x)) \text{space } M. Pxy (x, y) * \log b (Pxy (x, y) / (Px x * P_y y)))$

proof ($\text{subst mutual-information-distr}[OF - - \text{simple-distributed}[OF X] - \text{simple-distributed}[OF$

$Y]$ - *simple-distributed-joint*[*OF XY*]]
note $fin = \text{simple-distributed-joint-finite}[OF XY, simp]$
show *sigma-finite-measure* (*count-space* (X ‘ *space* M))
by (*simp add: sigma-finite-measure-count-space-finite*)
show *sigma-finite-measure* (*count-space* (Y ‘ *space* M))
by (*simp add: sigma-finite-measure-count-space-finite*)
let $?Pxy = \lambda x. (\text{if } x \in (\lambda x. (X x, Y x)) \text{ ‘ space } M \text{ then } Pxy \ x \text{ else } 0)$
let $?f = \lambda x. ?Pxy \ x * \log b \ (?Pxy \ x / (Px \ (fst \ x) * Py \ (snd \ x)))$
have $\bigwedge x. ?f \ x = (\text{if } x \in (\lambda x. (X x, Y x)) \text{ ‘ space } M \text{ then } Pxy \ x * \log b \ (Pxy \ x / (Px \ (fst \ x) * Py \ (snd \ x))) \text{ else } 0)$
by *auto*
with fin **show** $(\int x. ?f \ x \ \partial(\text{count-space } (X \text{ ‘ space } M) \otimes_M \text{count-space } (Y \text{ ‘ space } M))) =$
 $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. Pxy \ (x, y) * \log b \ (Pxy \ (x, y) / (Px \ x * Py \ y)))$
by (*auto simp add: pair-measure-count-space lebesgue-integral-count-space-finite sum.If-cases split-beta'*
intro!: sum.cong)
qed (*insert X Y XY, auto simp: simple-distributed-def*)

lemma (*in information-space*)

fixes $Pxy :: 'b \times 'c \Rightarrow \text{real}$ **and** $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$
assumes Px : *simple-distributed* $M \ X \ Px$ **and** Py : *simple-distributed* $M \ Y \ Py$
assumes Pxy : *simple-distributed* $M \ (\lambda x. (X x, Y x)) \ Pxy$
assumes ae : $\forall x \in \text{space } M. Pxy \ (X x, Y x) = Px \ (X x) * Py \ (Y x)$
shows *mutual-information-eq-0-simple*: $\mathcal{I}(X ; Y) = 0$
proof (*subst mutual-information-simple-distributed[OF Px Py Pxy]*)
have $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. Pxy \ (x, y) * \log b \ (Pxy \ (x, y) / (Px \ x * Py \ y))) =$
 $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. 0)$
by (*intro sum.cong*) (*auto simp: ae*)
then show $(\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. Pxy \ (x, y) * \log b \ (Pxy \ (x, y) / (Px \ x * Py \ y))) = 0$ **by** *simp*
qed

12.5 Entropy

definition (*in prob-space*) $entropy :: \text{real} \Rightarrow 'b \text{ measure} \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{real}$ **where**
 $entropy \ b \ S \ X = - \text{KL-divergence } b \ S \ (\text{distr } M \ S \ X)$

abbreviation (*in information-space*)

$entropy\text{-Pow } (\mathcal{H}'(-))$ **where**
 $\mathcal{H}(X) \equiv \text{entropy } b \ (\text{count-space } (X \text{ ‘ space } M)) \ X$

lemma (*in prob-space*) *distributed-RN-deriv*:

assumes X : *distributed* $M \ S \ X \ Px$

shows $AE \ x \ \text{in } S. \text{RN-deriv } S \ (\text{density } S \ Px) \ x = Px \ x$

proof –

note $D = \text{distributed-measurable}[OF X] \ \text{distributed-borel-measurable}[OF X]$

```

interpret X: prob-space distr M S X
  using D(1) by (rule prob-space-distr)

have sf: sigma-finite-measure (distr M S X) by standard
show ?thesis
  using D
  apply (subst eq-commute)
  apply (intro RN-deriv-unique-sigma-finite)
  apply (auto simp: distributed-distr-eq-density[symmetric, OF X] sf)
done
qed

lemma (in information-space)
  fixes X :: 'a  $\Rightarrow$  'b
  assumes X[measurable]: distributed M MX X f and nn:  $\bigwedge x. x \in \text{space } MX \implies 0 \leq f x$ 
  shows entropy-distr: entropy b MX X =  $-\left(\int x. f x * \log b (f x) \partial MX\right)$  (is ?eq)
proof -
  note D = distributed-measurable[OF X] distributed-borel-measurable[OF X]
  note ae = distributed-RN-deriv[OF X]
  note distributed-real-measurable[OF nn X, measurable]

  have ae-eq: AE x in distr M MX X.  $\log b (enn2real (RN-deriv MX (distr M MX X) x)) = \log b (f x)$ 
    unfolding distributed-distr-eq-density[OF X]
    apply (subst AE-density)
    using D apply simp
    using ae apply eventually-elim
    apply auto
    done

  have int-eq:  $\left(\int x. f x * \log b (f x) \partial MX\right) = \left(\int x. \log b (f x) \partial \text{distr } M \text{ } MX \text{ } X\right)$ 
    unfolding distributed-distr-eq-density[OF X]
    using D
    by (subst integral-density) (auto simp: nn)

  show ?eq
    unfolding entropy-def KL-divergence-def entropy-density-def comp-def int-eq
    neg-equal-iff-equal
    using ae-eq by (intro integral-cong-AE) (auto simp: nn)
qed

lemma (in information-space) entropy-le:
  fixes Px :: 'b  $\Rightarrow$  real and MX :: 'b measure
  assumes X[measurable]: distributed M MX X Px and Px-nn[simp]:  $\bigwedge x. x \in \text{space } MX \implies 0 \leq Px x$ 
  and fin: emeasure MX  $\{x \in \text{space } MX. Px x \neq 0\} \neq \text{top}$ 
  and int: integrable MX  $(\lambda x. - Px x * \log b (Px x))$ 

```

shows $\text{entropy } b \text{ } MX \ X \leq \log b \text{ (measure } MX \ \{x \in \text{space } MX. \ Px \ x \neq 0\})$

proof –

note $Px = \text{distributed-borel-measurable}[OF \ X]$

interpret $X: \text{prob-space } \text{distr } M \ MX \ X$

using $\text{distributed-measurable}[OF \ X]$ **by** (rule prob-space-distr)

have $-\log b \text{ (measure } MX \ \{x \in \text{space } MX. \ Px \ x \neq 0\}) =$
 $-\log b \text{ (}\int x. \text{indicator } \{x \in \text{space } MX. \ Px \ x \neq 0\} \ x \ \partial MX)$

using $Px \ Px\text{-nn } \text{fin}$ **by** (auto simp: measure-def)

also have $-\log b \text{ (}\int x. \text{indicator } \{x \in \text{space } MX. \ Px \ x \neq 0\} \ x \ \partial MX) = -\log$
 $b \text{ (}\int x. \ 1 / Px \ x \ \partial \text{distr } M \ MX \ X)$

proof –

have $\text{integral}^L \ MX \ (\text{indicator } \{x \in \text{space } MX. \ Px \ x \neq 0\}) = \text{LINT } x | MX. \ Px$
 $x \ *_R \ (1 / Px \ x)$

by (rule $\text{Bochner-Integration.integral-cong}$) *auto*

also have $\dots = \text{LINT } x | \text{density } MX \ (\lambda x. \ \text{ennreal } (Px \ x)). \ 1 / Px \ x$

by (rule $\text{integral-density [symmetric]}$) (use $Px \ Px\text{-nn}$ **in** *auto*)

finally show *?thesis*

unfolding $\text{distributed-distr-eq-density}[OF \ X]$ **by** *simp*

qed

also have $\dots \leq \text{(}\int x. \ -\log b \text{ (} 1 / Px \ x \text{) } \partial \text{distr } M \ MX \ X)$

proof (rule $X.\text{jensens-inequality}[of \ \lambda x. \ 1 / Px \ x \ \{0 < ..\} \ 0 \ 1 \ \lambda x. \ -\log b \ x]$)

show $AE \ x \ \text{in } \text{distr } M \ MX \ X. \ 1 / Px \ x \in \{0 < ..\}$

unfolding $\text{distributed-distr-eq-density}[OF \ X]$

using Px **by** (auto simp: AE-density)

have $[\text{simp}]: \bigwedge x. \ x \in \text{space } MX \implies \text{ennreal } (\text{if } Px \ x = 0 \ \text{then } 0 \ \text{else } 1) =$
 $\text{indicator } \{x \in \text{space } MX. \ Px \ x \neq 0\} \ x$

by (auto $\text{simp: one-ennreal-def}$)

have $(\int^+ x. \ \text{ennreal } (- \text{(if } Px \ x = 0 \ \text{then } 0 \ \text{else } 1))) \ \partial MX) = (\int^+ x. \ 0 \ \partial MX)$

by (intro nn-integral-cong) (auto simp: ennreal-neg)

then show $\text{integrable } (\text{distr } M \ MX \ X) \ (\lambda x. \ 1 / Px \ x)$

unfolding $\text{distributed-distr-eq-density}[OF \ X]$ **using** Px

by (auto $\text{simp: nn-integral-density real-integrable-def fin ennreal-neg en-}$
 $\text{nreal-mult[symmetric]}$

$\text{cong: nn-integral-cong}$)

have $\text{integrable } MX \ (\lambda x. \ Px \ x \ * \ \log b \text{ (} 1 / Px \ x)) =$
 $\text{integrable } MX \ (\lambda x. \ - Px \ x \ * \ \log b \text{ (} Px \ x))$

using Px

by (intro $\text{integrable-cong-AE}$) (auto $\text{simp: log-divide-eq less-le}$)

then show $\text{integrable } (\text{distr } M \ MX \ X) \ (\lambda x. \ -\log b \text{ (} 1 / Px \ x))$

unfolding $\text{distributed-distr-eq-density}[OF \ X]$

using $Px \ \text{int}$

by (subst $\text{integrable-real-density}$) *auto*

qed (auto $\text{simp: minus-log-convex}[OF \ b\text{-gt-1}]$)

also have $\dots = \text{(}\int x. \ \log b \text{ (} Px \ x \text{) } \partial \text{distr } M \ MX \ X)$

unfolding $\text{distributed-distr-eq-density}[OF \ X]$ **using** Px

by (intro integral-cong-AE) (auto $\text{simp: AE-density log-divide-eq}$)

also have $\dots = -\text{entropy } b \text{ } MX \ X$

unfolding $\text{distributed-distr-eq-density}[OF \ X]$ **using** Px

by (*subst entropy-distr*[*OF X*]) (*auto simp: integral-density*)
finally show *?thesis*
 by *simp*
qed

lemma (*in information-space*) *entropy-le-space*:

fixes *Px* :: 'b \Rightarrow real **and** *MX* :: 'b *measure*
assumes *X*: *distributed M MX X Px* **and** *Px-nn*[*simp*]: $\bigwedge x. x \in \text{space } MX \implies 0 \leq Px\ x$
and *fin*: *finite-measure MX*
and *int*: *integrable MX* ($\lambda x. - Px\ x * \log b (Px\ x)$)
shows *entropy b MX X* $\leq \log b (\text{measure } MX (\text{space } MX))$
proof –
note *Px = distributed-borel-measurable*[*OF X*]
interpret *finite-measure MX* **by** *fact*
have *entropy b MX X* $\leq \log b (\text{measure } MX \{x \in \text{space } MX. Px\ x \neq 0\})$
using *int X* **by** (*intro entropy-le*) *auto*
also have $\dots \leq \log b (\text{measure } MX (\text{space } MX))$
using *Px distributed-imp-emeasure-nonzero*[*OF X*]
by (*intro log-le*)
 (*auto intro!*: *finite-measure-mono b-gt-1 less-le*[*THEN iffD2*])
simp: emeasure-eq-measure cong: conj-cong)
finally show *?thesis* .
qed

lemma (*in information-space*) *entropy-uniform*:

assumes *X*: *distributed M MX X* ($\lambda x. \text{indicator } A\ x / \text{measure } MX\ A$) (*is distributed - - - ?f*)
shows *entropy b MX X* $= \log b (\text{measure } MX\ A)$
proof (*subst entropy-distr*[*OF X*])
have [*simp*]: *emeasure MX A* $\neq \infty$
using *uniform-distributed-params*[*OF X*] **by** (*auto simp add: measure-def*)
have *eq*: $(\int x. \text{indicator } A\ x / \text{measure } MX\ A * \log b (\text{indicator } A\ x / \text{measure } MX\ A) \partial MX) =$
 $(\int x. (- \log b (\text{measure } MX\ A) / \text{measure } MX\ A) * \text{indicator } A\ x \partial MX)$
using *uniform-distributed-params*[*OF X*]
by (*intro Bochner-Integration.integral-cong*) (*auto split: split-indicator simp: log-divide-eq zero-less-measure-iff*)
show $-(\int x. \text{indicator } A\ x / \text{measure } MX\ A * \log b (\text{indicator } A\ x / \text{measure } MX\ A) \partial MX) =$
 $\log b (\text{measure } MX\ A)$
unfolding *eq* **using** *uniform-distributed-params*[*OF X*]
by (*subst Bochner-Integration.integral-mult-right*) (*auto simp: measure-def less-top*[*symmetric*])
intro!: *integrable-real-indicator*)
qed *simp*

lemma (*in information-space*) *entropy-simple-distributed*:

simple-distributed M X f $\implies \mathcal{H}(X) = - (\sum_{x \in X \text{ space } M. f\ x * \log b (f\ x)})$
by (*subst entropy-distr*[*OF simple-distributed*])

(*auto simp add: lebesgue-integral-count-space-finite*)

lemma (in *information-space*) *entropy-le-card-not-0*:
assumes X : *simple-distributed* M X f
shows $\mathcal{H}(X) \leq \log b$ (*card* (X ‘ *space* $M \cap \{x. f\ x \neq 0\}$))
proof –
let $?X =$ *count-space* (X ‘ *space* M)
have $\mathcal{H}(X) \leq \log b$ (*measure* $?X$ $\{x \in$ *space* $?X. f\ x \neq 0\}$)
by (*rule* *entropy-le*[*OF* *simple-distributed*[*OF* X]])
(*insert* X , *auto simp: simple-distributed-finite*[*OF* X] *subset-eq integrable-count-space*
emeasure-count-space)
also have *measure* $?X$ $\{x \in$ *space* $?X. f\ x \neq 0\} =$ *card* (X ‘ *space* $M \cap \{x. f\ x$
 $\neq 0\}$)
by (*simp-all add: simple-distributed-finite*[*OF* X] *subset-eq emeasure-count-space*
measure-def Int-def)
finally show *thesis* .
qed

lemma (in *information-space*) *entropy-le-card*:
assumes X : *simple-distributed* M X f
shows $\mathcal{H}(X) \leq \log b$ (*real* (*card* (X ‘ *space* M)))
proof –
let $?X =$ *count-space* (X ‘ *space* M)
have $\mathcal{H}(X) \leq \log b$ (*measure* $?X$ (*space* $?X$))
by (*rule* *entropy-le-space*[*OF* *simple-distributed*[*OF* X]])
(*insert* X , *auto simp: simple-distributed-finite*[*OF* X] *subset-eq integrable-count-space*
emeasure-count-space finite-measure-count-space)
also have *measure* $?X$ (*space* $?X$) = *card* (X ‘ *space* M)
by (*simp-all add: simple-distributed-finite*[*OF* X] *subset-eq emeasure-count-space*
measure-def)
finally show *thesis* .
qed

12.6 Conditional Mutual Information

definition (in *prob-space*)

conditional-mutual-information b MX MY MZ X Y $Z \equiv$
mutual-information b MX ($MY \otimes_M MZ$) X ($\lambda x. (Y\ x, Z\ x)$) –
mutual-information b MX MZ X Z

abbreviation (in *information-space*)

conditional-mutual-information-Pow ($\mathcal{I}'(-; - | -')$) **where**
 $\mathcal{I}(X ; Y | Z) \equiv$ *conditional-mutual-information* b
(*count-space* (X ‘ *space* M)) (*count-space* (Y ‘ *space* M)) (*count-space* (Z ‘ *space*
 M)) X Y Z

lemma (in *information-space*)

assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T **and** P :
sigma-finite-measure P

assumes Px [*measurable*]: *distributed* $M S X Px$
and Px -*nn*[*simp*]: $\bigwedge x. x \in \text{space } S \implies 0 \leq Px x$
assumes Pz [*measurable*]: *distributed* $M P Z Pz$
and Pz -*nn*[*simp*]: $\bigwedge z. z \in \text{space } P \implies 0 \leq Pz z$
assumes Pyz [*measurable*]: *distributed* $M (T \otimes_M P) (\lambda x. (Y x, Z x)) Pyz$
and Pyz -*nn*[*simp*]: $\bigwedge y z. y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pyz (y, z)$
assumes Pxz [*measurable*]: *distributed* $M (S \otimes_M P) (\lambda x. (X x, Z x)) Pxz$
and Pxz -*nn*[*simp*]: $\bigwedge x z. x \in \text{space } S \implies z \in \text{space } P \implies 0 \leq Pxz (x, z)$
assumes $Pxyz$ [*measurable*]: *distributed* $M (S \otimes_M T \otimes_M P) (\lambda x. (X x, Y x, Z x)) Pxyz$
and $Pxyz$ -*nn*[*simp*]: $\bigwedge x y z. x \in \text{space } S \implies y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pxyz (x, y, z)$
assumes $I1$: *integrable* $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Px x * Pyz (y, z))))$
assumes $I2$: *integrable* $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxz (x, z) / (Px x * Pz z)))$
shows *conditional-mutual-information-generic-eq: conditional-mutual-information*
 $b S T P X Y Z$
 $= (\int (x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))) \partial(S \otimes_M T \otimes_M P))$ (*is ?eq*)
and *conditional-mutual-information-generic-nonneg: $0 \leq$ conditional-mutual-information*
 $b S T P X Y Z$ (*is ?nonneg*)
proof –
have [*measurable*]: $Px \in S \rightarrow_M \text{borel}$
using Px Px -*nn* **by** (*intro distributed-real-measurable*)
have [*measurable*]: $Pz \in P \rightarrow_M \text{borel}$
using Pz Pz -*nn* **by** (*intro distributed-real-measurable*)
have *measurable- Pyz* [*measurable*]: $Pyz \in (T \otimes_M P) \rightarrow_M \text{borel}$
using Pyz Pyz -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)
have *measurable- Pxz* [*measurable*]: $Pxz \in (S \otimes_M P) \rightarrow_M \text{borel}$
using Pxz Pxz -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)
have *measurable- $Pxyz$* [*measurable*]: $Pxyz \in (S \otimes_M T \otimes_M P) \rightarrow_M \text{borel}$
using $Pxyz$ $Pxyz$ -*nn* **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

interpret S : *sigma-finite-measure* S **by** *fact*
interpret T : *sigma-finite-measure* T **by** *fact*
interpret P : *sigma-finite-measure* P **by** *fact*
interpret TP : *pair-sigma-finite* $T P$..
interpret SP : *pair-sigma-finite* $S P$..
interpret ST : *pair-sigma-finite* $S T$..
interpret SPT : *pair-sigma-finite* $S \otimes_M P T$..
interpret STP : *pair-sigma-finite* $S T \otimes_M P$..
interpret TPS : *pair-sigma-finite* $T \otimes_M P S$..
have TP : *sigma-finite-measure* $(T \otimes_M P)$..
have SP : *sigma-finite-measure* $(S \otimes_M P)$..
have YZ : *random-variable* $(T \otimes_M P) (\lambda x. (Y x, Z x))$
using Pyz **by** (*simp add: distributed-measurable*)

from Pxz $Pxyz$ **have** *distr-eq: distr* $M (S \otimes_M P) (\lambda x. (X x, Z x)) =$

$distr (distr M (S \otimes_M T \otimes_M P) (\lambda x. (X x, Y x, Z x))) (S \otimes_M P) (\lambda(x, y, z). (x, z))$

by (*simp add: comp-def distr-distr*)

have *mutual-information b S P X Z =*

$(\int x. Pxz x * \log b (Pxz x / (Px (fst x) * Pz (snd x)))) \partial(S \otimes_M P)$

by (*rule mutual-information-distr[OF S P Px Px-nn Pz Pz-nn Pxz Pxz-nn]*)

also have $\dots = (\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) / (Px x * Pz z))) \partial(S \otimes_M T \otimes_M P)$

using *b-gt-1 Pxz Px Pz*

by (*subst distributed-transform-integral[OF Pxyz - Pxz -, where T= $\lambda(x, y, z).$ (x, z)]*)

(*auto simp: split-beta' space-pair-measure*)

finally have *mi-eq:*

$mutual-information b S P X Z = (\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) / (Px x * Pz z))) \partial(S \otimes_M T \otimes_M P)$.

have *ae1: AE x in S \otimes_M T \otimes_M P. Px (fst x) = 0 \longrightarrow Pxyz x = 0*

by (*intro subdensity-real[of fst, OF - Pxyz Px]*) (*auto simp: space-pair-measure*)

moreover have *ae2: AE x in S \otimes_M T \otimes_M P. Pz (snd (snd x)) = 0 \longrightarrow Pxyz x = 0*

by (*intro subdensity-real[of $\lambda x. snd (snd x)$, OF - Pxyz Pz]*) (*auto simp: space-pair-measure*)

moreover have *ae3: AE x in S \otimes_M T \otimes_M P. Pxz (fst x, snd (snd x)) = 0 \longrightarrow Pxyz x = 0*

by (*intro subdensity-real[of $\lambda x. (fst x, snd (snd x))$, OF - Pxyz Pxz]*) (*auto simp: space-pair-measure*)

moreover have *ae4: AE x in S \otimes_M T \otimes_M P. Pyz (snd x) = 0 \longrightarrow Pxyz x = 0*

by (*intro subdensity-real[of snd, OF - Pxyz Pyz]*) (*auto simp: space-pair-measure*)

ultimately have *ae: AE x in S \otimes_M T \otimes_M P.*

$Pxyz x * \log b (Pxyz x / (Px (fst x) * Pyz (snd x))) -$

$Pxyz x * \log b (Pxz (fst x, snd (snd x)) / (Px (fst x) * Pz (snd (snd x)))) =$

$Pxyz x * \log b (Pxyz x * Pz (snd (snd x)) / (Pxz (fst x, snd (snd x)) * Pyz (snd x)))$

using *AE-space*

proof *eventually-elim*

case (*elim x*)

show *?case*

proof *cases*

assume *Pxyz x \neq 0*

with *elim* **have** $0 < Px (fst x) \ 0 < Pz (snd (snd x)) \ 0 < Pxz (fst x, snd (snd x))$

$0 < Pyz (snd x) \ 0 < Pxyz x$

by (*auto simp: space-pair-measure less-le*)

then show *?thesis*

using *b-gt-1* **by** (*simp add: log-simps less-imp-le field-simps*)

qed *simp*

qed


```

with I1 I2 show ?eq
  unfolding conditional-mutual-information-def
  apply (subst mi-eq)
  apply (subst mutual-information-distr[OF S TP Px Px-nn Pyz - Pxyz])
  apply (auto simp: space-pair-measure)
  apply (subst Bochner-Integration.integral-diff[symmetric])
  apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
done

let ?P = density (S  $\otimes_M$  T  $\otimes_M$  P) Pxyz
interpret P: prob-space ?P
  unfolding distributed-distr-eq-density[OF Pxyz, symmetric]
  by (rule prob-space-distr) simp

let ?Q = density (T  $\otimes_M$  P) Pyz
interpret Q: prob-space ?Q
  unfolding distributed-distr-eq-density[OF Pyz, symmetric]
  by (rule prob-space-distr) simp

let ?f =  $\lambda(x, y, z). Pxz(x, z) * (Pyz(y, z) / Pz z) / Pxyz(x, y, z)$ 

from subdensity-real[of snd, OF - Pyz Pz - AE-I2 AE-I2]
have aeX1: AE x in T  $\otimes_M$  P. Pz (snd x) = 0  $\longrightarrow$  Pyz x = 0
  by (auto simp: comp-def space-pair-measure)
have aeX2: AE x in T  $\otimes_M$  P. 0  $\leq$  Pz (snd x)
  using Pz by (intro TP.AE-pair-measure) (auto simp: comp-def)

have aeX3: AE y in T  $\otimes_M$  P. ( $\int^+ x. ennreal (Pxz(x, snd y)) \partial S$ ) = ennreal
(Pz (snd y))
  using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
  by (intro TP.AE-pair-measure) auto

have ( $\int^+ x. ?f x \partial ?P$ )  $\leq$  ( $\int^+ (x, y, z). Pxz(x, z) * (Pyz(y, z) / Pz z) \partial(S$ 
 $\otimes_M T \otimes_M P)$ )
  by (subst nn-integral-density)
  (auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
also have ... = ( $\int^+(y, z). (\int^+ x. ennreal (Pxz(x, z)) * ennreal (Pyz(y, z) /$ 
Pz z)  $\partial S$ )  $\partial(T \otimes_M P)$ )
  by (subst STP.nn-integral-snd[symmetric])
  (auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!:
nn-integral-cong)
also have ... = ( $\int^+ x. ennreal (Pyz x) * 1 \partial T \otimes_M P$ )
  apply (rule nn-integral-cong-AE)
  using aeX1 aeX2 aeX3 AE-space
  apply eventually-elim
proof (case-tac x, simp add: space-pair-measure)
fix a b assume Pz b = 0  $\longrightarrow$  Pyz (a, b) = 0 a  $\in$  space T  $\wedge$  b  $\in$  space P
  ( $\int^+ x. ennreal (Pxz(x, b)) \partial S$ ) = ennreal (Pz b)
  then show ( $\int^+ x. ennreal (Pxz(x, b)) * ennreal (Pyz(a, b) / Pz b) \partial S$ ) =

```

```

ennreal (Pyz (a, b))
  by (subst nn-integral-multc) (auto split: prod.split simp: ennreal-mult[symmetric])
qed
also have ... = 1
  using Q.emeasure-space-1 distributed-distr-eq-density[OF Pyz]
  by (subst nn-integral-density[symmetric]) auto
finally have le1:  $(\int^+ x. ?f x \partial?P) \leq 1$  .
also have ... <  $\infty$  by simp
finally have fin:  $(\int^+ x. ?f x \partial?P) \neq \infty$  by simp

have pos:  $(\int^+ x. ?f x \partial?P) \neq 0$ 
  apply (subst nn-integral-density)
  apply (simp-all add: split-beta')
proof
  let ?g =  $\lambda x. \text{ennreal} (Pxyz x) * (Pxz (fst x, snd (snd x)) * Pyz (snd x) / (Pz (snd (snd x)) * Pxyz x))$ 
  assume  $(\int^+ x. ?g x \partial(S \otimes_M T \otimes_M P)) = 0$ 
  then have AE x in  $S \otimes_M T \otimes_M P. ?g x = 0$ 
    by (intro nn-integral-0-iff-AE[THEN iffD1]) auto
  then have AE x in  $S \otimes_M T \otimes_M P. Pxyz x = 0$ 
    using ae1 ae2 ae3 ae4 AE-space
    by eventually-elim (auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff
space-pair-measure)
  then have  $(\int^+ x. \text{ennreal} (Pxyz x) \partial S \otimes_M T \otimes_M P) = 0$ 
    by (subst nn-integral-cong-AE[of -  $\lambda x. 0$ ]) auto
  with P.emeasure-space-1 show False
    by (subst (asm) emeasure-density) (auto cong: nn-integral-cong)
qed

have neg:  $(\int^+ x. - ?f x \partial?P) = 0$ 
  apply (rule nn-integral-0-iff-AE[THEN iffD2])
  apply simp
  apply (subst AE-density)
  apply (auto simp: space-pair-measure ennreal-neg)
done

have I3: integrable  $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))))$ 
  apply (rule integrable-cong-AE[THEN iffD1, OF - - - Bochner-Integration.integrable-diff[OF I1 I2]])
  using ae
  apply (auto simp: split-beta')
done

have  $-\log b 1 \leq -\log b (\text{integral}^L ?P ?f)$ 
proof (intro le-imp-neg-le log-le[OF b-gt-1])
  have If: integrable ?P ?f
    unfolding real-integrable-def
  proof (intro conjI)

```

```

from neg show  $(\int^+ x. - ?f x \partial ?P) \neq \infty$ 
  by simp
from fin show  $(\int^+ x. ?f x \partial ?P) \neq \infty$ 
  by simp
qed simp
then have  $(\int^+ x. ?f x \partial ?P) = (\int x. ?f x \partial ?P)$ 
  apply (rule nn-integral-eq-integral)
  apply (subst AE-density)
  apply simp
  apply (auto simp: space-pair-measure ennreal-neg)
  done
with pos le1
show  $0 < (\int x. ?f x \partial ?P) (\int x. ?f x \partial ?P) \leq 1$ 
  by (simp-all add: one-ennreal.rep-eq zero-less-iff-neq-zero[symmetric])
qed
also have  $-\log b (\text{integral}^L ?P ?f) \leq (\int x. -\log b (?f x) \partial ?P)$ 
proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
  show AE x in ?P. ?f x  $\in \{0<..\}$ 
    unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
    using ae1 ae2 ae3 ae4 AE-space
    by eventually-elim (auto simp: space-pair-measure less-le)
  show integrable ?P ?f
    unfolding real-integrable-def
    using fin neg by (auto simp: split-beta')
  show integrable ?P  $(\lambda x. -\log b (?f x))$ 
    apply (subst integrable-real-density)
    apply simp
    apply (auto simp: space-pair-measure) []
    apply simp
    apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
    apply simp
    apply simp
    using ae1 ae2 ae3 ae4 AE-space
    apply eventually-elim
    apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
zero-less-divide-iff field-simps
less-le space-pair-measure)
    done
qed (auto simp: b-gt-1 minus-log-convex)
also have  $\dots = \text{conditional-mutual-information } b \ S \ T \ P \ X \ Y \ Z$ 
  unfolding  $\langle ?eq \rangle$ 
  apply (subst integral-real-density)
  apply simp
  apply (auto simp: space-pair-measure) []
  apply simp
  apply (intro integral-cong-AE)
  using ae1 ae2 ae3 ae4
  apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff field-simps
space-pair-measure less-le)

```

done
 finally show ?nonneg
 by simp
 qed

lemma (in information-space)

fixes $Px :: - \Rightarrow \text{real}$
 assumes S : sigma-finite-measure S and T : sigma-finite-measure T and P :
 sigma-finite-measure P
 assumes Fx : finite-entropy S X Px
 assumes Fz : finite-entropy P Z Pz
 assumes Fyz : finite-entropy $(T \otimes_M P)$ $(\lambda x. (Y\ x, Z\ x))$ Pyz
 assumes Fxz : finite-entropy $(S \otimes_M P)$ $(\lambda x. (X\ x, Z\ x))$ Pxz
 assumes $Fxyz$: finite-entropy $(S \otimes_M T \otimes_M P)$ $(\lambda x. (X\ x, Y\ x, Z\ x))$ $Pxyz$
 shows conditional-mutual-information-generic-eq': conditional-mutual-information
 $b\ S\ T\ P\ X\ Y\ Z$
 $= (\int (x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Pxz\ (x, z) * (Pyz\ (y, z) / Pz\ z))) \partial(S \otimes_M T \otimes_M P))$ (is ?eq)
 and conditional-mutual-information-generic-nonneg': $0 \leq$ conditional-mutual-information
 $b\ S\ T\ P\ X\ Y\ Z$ (is ?nonneg)

proof –

note $Px = Fx$ [THEN finite-entropy-distributed, measurable]
 note $Pz = Fz$ [THEN finite-entropy-distributed, measurable]
 note $Pyz = Fyz$ [THEN finite-entropy-distributed, measurable]
 note $Pxz = Fxz$ [THEN finite-entropy-distributed, measurable]
 note $Pxyz = Fxyz$ [THEN finite-entropy-distributed, measurable]

note $Px\text{-nn} = Fx$ [THEN finite-entropy-nn]
 note $Pz\text{-nn} = Fz$ [THEN finite-entropy-nn]
 note $Pyz\text{-nn} = Fyz$ [THEN finite-entropy-nn]
 note $Pxz\text{-nn} = Fxz$ [THEN finite-entropy-nn]
 note $Pxyz\text{-nn} = Fxyz$ [THEN finite-entropy-nn]

note $Px' = Fx$ [THEN finite-entropy-measurable, measurable]
 note $Pz' = Fz$ [THEN finite-entropy-measurable, measurable]
 note $Pyz' = Fyz$ [THEN finite-entropy-measurable, measurable]
 note $Pxz' = Fxz$ [THEN finite-entropy-measurable, measurable]
 note $Pxyz' = Fxyz$ [THEN finite-entropy-measurable, measurable]

interpret S : sigma-finite-measure S by fact
 interpret T : sigma-finite-measure T by fact
 interpret P : sigma-finite-measure P by fact
 interpret TP : pair-sigma-finite $T\ P$..
 interpret SP : pair-sigma-finite $S\ P$..
 interpret ST : pair-sigma-finite $S\ T$..
 interpret SPT : pair-sigma-finite $S \otimes_M P\ T$..
 interpret STP : pair-sigma-finite $S\ T \otimes_M P$..
 interpret TPS : pair-sigma-finite $T \otimes_M P\ S$..
 have TP : sigma-finite-measure $(T \otimes_M P)$..

have SP : *sigma-finite-measure* $(S \otimes_M P)$..

from Pxz $Pxyz$ **have** *distr-eq*: $distr\ M\ (S \otimes_M P)\ (\lambda x. (X\ x, Z\ x)) =$
 $distr\ (distr\ M\ (S \otimes_M T \otimes_M P)\ (\lambda x. (X\ x, Y\ x, Z\ x)))\ (S \otimes_M P)\ (\lambda(x, y,$
 $z). (x, z))$
by (*simp add: distr-distr comp-def*)

have *mutual-information* $b\ S\ P\ X\ Z =$
 $(\int x. Pxz\ x * \log b\ (Pxz\ x / (Px\ (fst\ x) * Pz\ (snd\ x))))\ \partial(S \otimes_M P)$
using $Px\ Px\text{-nn}\ Pz\ Pz\text{-nn}\ Pxz\ Pxz\text{-nn}$
by (*rule mutual-information-distr[OF S P]*) (*auto simp: space-pair-measure*)
also have $\dots = (\int (x,y,z). Pxyz\ (x,y,z) * \log b\ (Pxz\ (x,z) / (Px\ x * Pz\ z)))\ \partial(S$
 $\otimes_M T \otimes_M P)$
using $b\text{-gt-1}\ Pxz\ Pxz\text{-nn}\ Pxyz\ Pxyz\text{-nn}$
by (*subst distributed-transform-integral[OF Pxyz - Pxz, where T= $\lambda(x, y, z).$*
 $(x, z)]$)
(auto simp: split-beta')

finally have *mi-eq*:
 $mutual\text{-information}\ b\ S\ P\ X\ Z = (\int (x,y,z). Pxyz\ (x,y,z) * \log b\ (Pxz\ (x,z) /$
 $(Px\ x * Pz\ z)))\ \partial(S \otimes_M T \otimes_M P)$.

have $ae1$: $AE\ x\ in\ S \otimes_M T \otimes_M P. Px\ (fst\ x) = 0 \longrightarrow Pxyz\ x = 0$
by (*intro subdensity-finite-entropy[of fst, OF - Fxyz Fx]*) *auto*
moreover have $ae2$: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pz\ (snd\ (snd\ x)) = 0 \longrightarrow Pxyz$
 $x = 0$
by (*intro subdensity-finite-entropy[of $\lambda x. snd\ (snd\ x)$, OF - Fxyz Fz]*) *auto*
moreover have $ae3$: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pxz\ (fst\ x, snd\ (snd\ x)) = 0$
 $\longrightarrow Pxyz\ x = 0$
by (*intro subdensity-finite-entropy[of $\lambda x. (fst\ x, snd\ (snd\ x))$, OF - Fxyz Fxz]*) *auto*
moreover have $ae4$: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pyz\ (snd\ x) = 0 \longrightarrow Pxyz\ x$
 $= 0$
by (*intro subdensity-finite-entropy[of snd, OF - Fxyz Fyz]*) *auto*

ultimately have ae : $AE\ x\ in\ S \otimes_M T \otimes_M P.$
 $Pxyz\ x * \log b\ (Pxyz\ x / (Px\ (fst\ x) * Pyz\ (snd\ x))) -$
 $Pxyz\ x * \log b\ (Pxz\ (fst\ x, snd\ (snd\ x)) / (Px\ (fst\ x) * Pz\ (snd\ (snd\ x)))) =$
 $Pxyz\ x * \log b\ (Pxyz\ x * Pz\ (snd\ (snd\ x)) / (Pxz\ (fst\ x, snd\ (snd\ x)) * Pyz$
 $(snd\ x)))$
using *AE-space*
proof *eventually-elim*
case (*elim* x)
show *?case*
proof *cases*
assume $Pxyz\ x \neq 0$
with *elim* **have** $0 < Px\ (fst\ x)\ 0 < Pz\ (snd\ (snd\ x))\ 0 < Pxz\ (fst\ x, snd$
 $(snd\ x))$
 $0 < Pyz\ (snd\ x)\ 0 < Pxyz\ x$
using $Px\text{-nn}\ Pz\text{-nn}\ Pxz\text{-nn}\ Pyz\text{-nn}\ Pxyz\text{-nn}$
by (*auto simp: space-pair-measure less-le*)

```

then show ?thesis
  using b-gt-1 by (simp add: log-simps less-imp-le field-simps)
qed simp
qed

have integrable (S  $\otimes_M$  T  $\otimes_M$  P)
  ( $\lambda x. Pxyz\ x * \log b (Pxyz\ x) - Pxyz\ x * \log b (Px\ (fst\ x)) - Pxyz\ x * \log b (Pyz\ (snd\ x))$ )
  using finite-entropy-integrable[OF Fxyz]
  using finite-entropy-integrable-transform[OF Fx Pxyz Pxyz-nn, of fst]
  using finite-entropy-integrable-transform[OF Fyz Pxyz Pxyz-nn, of snd]
  by simp
moreover have ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z)))$ )  $\in$  borel-measurable (S  $\otimes_M$  T  $\otimes_M$  P)
  using Pxyz Px Pyz by simp
ultimately have I1: integrable (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z)))$ )
  apply (rule integrable-cong-AE-imp)
  using ae1 ae4 AE-space
  by eventually-elim
  (insert Px-nn Pyz-nn Pxyz-nn,
   auto simp: log-divide-eq log-mult-eq field-simps zero-less-mult-iff space-pair-measure less-le)

have integrable (S  $\otimes_M$  T  $\otimes_M$  P)
  ( $\lambda x. Pxyz\ x * \log b (Pxz\ (fst\ x, snd\ (snd\ x))) - Pxyz\ x * \log b (Px\ (fst\ x)) - Pxyz\ x * \log b (Pz\ (snd\ (snd\ x)))$ )
  using finite-entropy-integrable-transform[OF Fxz Pxyz Pxyz-nn, of  $\lambda x. (fst\ x, snd\ (snd\ x))$ ]
  using finite-entropy-integrable-transform[OF Fx Pxyz Pxyz-nn, of fst]
  using finite-entropy-integrable-transform[OF Fz Pxyz Pxyz-nn, of  $snd \circ snd$ ]
  by simp
moreover have ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b (Pxz\ (x, z) / (Px\ x * Pz\ z))$ )  $\in$  borel-measurable (S  $\otimes_M$  T  $\otimes_M$  P)
  using Pxyz Px Pz
  by auto
ultimately have I2: integrable (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b (Pxz\ (x, z) / (Px\ x * Pz\ z))$ )
  apply (rule integrable-cong-AE-imp)
  using ae1 ae2 ae3 ae4 AE-space
  by eventually-elim
  (insert Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn,
   auto simp: log-divide-eq log-mult-eq field-simps zero-less-mult-iff less-le space-pair-measure)

from ae I1 I2 show ?eq
  unfolding conditional-mutual-information-def
  apply (subst mi-eq)
  apply (subst mutual-information-distr[OF S TP Px Px-nn Pyz Pyz-nn Pxyz

```

```

Pxyz-nn])
  apply simp
  apply simp
  apply (simp add: space-pair-measure)
  apply (subst Bochner-Integration.integral-diff[symmetric])
  apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
  done

let ?P = density (S  $\otimes_M$  T  $\otimes_M$  P) Pxyz
interpret P: prob-space ?P
  unfolding distributed-distr-eq-density[OF Pxyz, symmetric] by (rule prob-space-distr)
simp

let ?Q = density (T  $\otimes_M$  P) Pyz
interpret Q: prob-space ?Q
  unfolding distributed-distr-eq-density[OF Pyz, symmetric] by (rule prob-space-distr)
simp

let ?f =  $\lambda(x, y, z). Pxz(x, z) * (Pyz(y, z) / Pz z) / Pxyz(x, y, z)$ 

from subdensity-finite-entropy[of snd, OF - Fyz Fz]
have aeX1: AE x in T  $\otimes_M$  P. Pz (snd x) = 0  $\longrightarrow$  Pyz x = 0 by (auto simp: comp-def)
have aeX2: AE x in T  $\otimes_M$  P. 0  $\leq$  Pz (snd x)
  using Pz by (intro TP.AE-pair-measure) (auto intro: Pz-nn)

have aeX3: AE y in T  $\otimes_M$  P. ( $\int^+$  x. ennreal (Pxz (x, snd y))  $\partial S$ ) = ennreal (Pz (snd y))
  using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
  by (intro TP.AE-pair-measure) (auto)
have ( $\int^+$  x. ?f x  $\partial ?P$ )  $\leq$  ( $\int^+$  (x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z)  $\partial (S \otimes_M T \otimes_M P)$ )
  using Pz-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  by (subst nn-integral-density)
  (auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
also have ... = ( $\int^+$  (y, z).  $\int^+$  x. ennreal (Pxz (x, z)) * ennreal (Pyz (y, z) / Pz z)  $\partial S \partial T \otimes_M P$ )
  using Pz-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  by (subst STP.nn-integral-snd[symmetric])
  (auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!: nn-integral-cong)
also have ... = ( $\int^+$  x. ennreal (Pyz x) * 1  $\partial T \otimes_M P$ )
  apply (rule nn-integral-cong-AE)
  using aeX1 aeX2 aeX3 AE-space
  apply eventually-elim
proof (case-tac x, simp add: space-pair-measure)
  fix a b assume Pz b = 0  $\longrightarrow$  Pyz (a, b) = 0 0  $\leq$  Pz b a  $\in$  space T  $\wedge$  b  $\in$  space P
  ( $\int^+$  x. ennreal (Pxz (x, b))  $\partial S$ ) = ennreal (Pz b)

```

```

then show ( $\int^+ x. \text{ennreal } (Pxz (x, b)) * \text{ennreal } (Pyz (a, b) / Pz b) \partial S =$ 
 $\text{ennreal } (Pyz (a, b))$ )
  using  $Pyz\text{-nn}[of (a,b)]$ 
  by ( $\text{subst nn-integral-multc}$ ) ( $\text{auto simp: space-pair-measure ennreal-mult[symmetric]}$ )
qed
also have  $\dots = 1$ 
  using  $Q.\text{emeasure-space-1 } Pyz\text{-nn distributed-distr-eq-density[OF } Pyz]$ 
  by ( $\text{subst nn-integral-density[symmetric]}$ )  $\text{auto}$ 
finally have  $le1: (\int^+ x. ?f x \partial P) \leq 1$  .
also have  $\dots < \infty$  by  $\text{simp}$ 
finally have  $fin: (\int^+ x. ?f x \partial P) \neq \infty$  by  $\text{simp}$ 

have ( $\int^+ x. ?f x \partial P \neq 0$ )
  using  $Pxyz\text{-nn}$ 
  apply ( $\text{subst nn-integral-density}$ )
  apply ( $\text{simp-all add: split-beta' ennreal-mult'[symmetric] cong: nn-integral-cong}$ )
proof
  let  $?g = \lambda x. \text{ennreal } (if Pxz x = 0 \text{ then } 0 \text{ else } Pxz (fst x, snd (snd x)) * Pyz$ 
 $(snd x) / Pz (snd (snd x)))$ 
  assume ( $\int^+ x. ?g x \partial (S \otimes_M T \otimes_M P) = 0$ )
  then have  $AE x \text{ in } S \otimes_M T \otimes_M P. ?g x = 0$ 
    by ( $\text{intro nn-integral-0-iff-AE[THEN iffD1]}$ )  $\text{auto}$ 
  then have  $AE x \text{ in } S \otimes_M T \otimes_M P. Pxz x = 0$ 
    using  $ae1 ae2 ae3 ae4 AE\text{-space}$ 
    by  $\text{eventually-elim}$ 
    ( $\text{insert } Px\text{-nn } Pz\text{-nn } Pxz\text{-nn } Pyz\text{-nn},$ 
 $\text{auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff space-pair-measure}$ )
  then have ( $\int^+ x. \text{ennreal } (Pxyz x) \partial S \otimes_M T \otimes_M P = 0$ )
    by ( $\text{subst nn-integral-cong-AE}[of - \lambda x. 0]$ )  $\text{auto}$ 
  with  $P.\text{emeasure-space-1}$  show  $\text{False}$ 
    by ( $\text{subst (asm) emeasure-density}$ ) ( $\text{auto cong: nn-integral-cong}$ )
qed
then have  $pos: 0 < (\int^+ x. ?f x \partial P)$ 
  by ( $\text{simp add: zero-less-iff-neq-zero}$ )

have  $neg: (\int^+ x. - ?f x \partial P) = 0$ 
  using  $Pz\text{-nn } Pxz\text{-nn } Pyz\text{-nn } Pxyz\text{-nn}$ 
  by ( $\text{intro nn-integral-0-iff-AE[THEN iffD2]}$ )
  ( $\text{auto simp: split-beta' AE-density space-pair-measure intro!: AE-I2 en-$ 
 $\text{nreal-neg}$ )

have  $I3: \text{integrable } (S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz$ 
 $(x, y, z) / (Pxz (x, z) * (Pyz (y,z) / Pz z))))$ 
  apply ( $\text{rule integrable-cong-AE[THEN iffD1, OF - - - Bochner-Integration.integrable-diff[OF$ 
 $I1 I2]}$ )
  using  $ae$ 
  apply ( $\text{auto simp: split-beta'}$ )
done

```



```

have  $-\log b \ 1 \leq -\log b \ (\text{integral}^L \ ?P \ ?f)$ 
proof (intro le-imp-neg-le log-le[OF b-gt-1])
  have  $I f: \text{integrable } ?P \ ?f$ 
  unfolding real-integrable-def
proof (intro conjI)
  from neg show  $(\int^+ x. - ?f x \ \partial ?P) \neq \infty$ 
  by simp
  from fin show  $(\int^+ x. ?f x \ \partial ?P) \neq \infty$ 
  by simp
qed simp
then have  $(\int^+ x. ?f x \ \partial ?P) = (\int x. ?f x \ \partial ?P)$ 
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  by (intro nn-integral-eq-integral)
  (auto simp: AE-density space-pair-measure)
with pos le1
show  $0 < (\int x. ?f x \ \partial ?P) \ (\int x. ?f x \ \partial ?P) \leq 1$ 
  by (simp-all add: )
qed
also have  $-\log b \ (\text{integral}^L \ ?P \ ?f) \leq (\int x. -\log b \ (\ ?f x) \ \partial ?P)$ 
proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
  show  $AE \ x \ \text{in } ?P. \ ?f \ x \in \{0<..\}$ 
  unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
  using ae1 ae2 ae3 ae4 AE-space
  by eventually-elim (insert Pxyz-nn Pyz-nn Pz-nn Pxz-nn, auto simp: space-pair-measure
less-le)
  show integrable ?P ?f
  unfolding real-integrable-def
  using fin neg by (auto simp: split-beta')
show integrable ?P  $(\lambda x. -\log b \ (\ ?f x))$ 
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  apply (subst integrable-real-density)
  apply simp
  apply simp
  apply simp
  apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
  apply simp
  apply simp
  using ae1 ae2 ae3 ae4 AE-space
  apply eventually-elim
  apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
zero-less-divide-iff field-simps space-pair-measure less-le)
done
qed (auto simp: b-gt-1 minus-log-convex)
also have  $\dots = \text{conditional-mutual-information } b \ S \ T \ P \ X \ Y \ Z$ 
  unfolding <?eq>
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  apply (subst integral-real-density)
  apply simp
  apply simp

```

```

apply simp
apply (intro integral-cong-AE)
using ae1 ae2 ae3 ae4 AE-space
apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff
        field-simps space-pair-measure less-le)
done
finally show ?nonneg
  by simp
qed

```

lemma (*in information-space*) *conditional-mutual-information-eq*:

```

assumes Pz: simple-distributed M Z Pz
assumes Pyz: simple-distributed M (λx. (Y x, Z x)) Pyz
assumes Pxz: simple-distributed M (λx. (X x, Z x)) Pxz
assumes Pxyz: simple-distributed M (λx. (X x, Y x, Z x)) Pxyz
shows  $\mathcal{I}(X ; Y \mid Z) =$ 
   $(\sum_{(x, y, z) \in (\lambda x. (X x, Y x, Z x))' \text{space } M} Pxyz(x, y, z) * \log b(Pxyz(x, y,$ 
   $z) / (Pxz(x, z) * (Pyz(y, z) / Pz z))))$ 
proof (subst conditional-mutual-information-generic-eq[OF - - - - -
        simple-distributed[OF Pz] - simple-distributed-joint[OF Pyz] - simple-distributed-joint[OF
        Pxz] -
        simple-distributed-joint2[OF Pxyz]])
note simple-distributed-joint2-finite[OF Pxyz, simp]
show sigma-finite-measure (count-space (X ' space M))
  by (simp add: sigma-finite-measure-count-space-finite)
show sigma-finite-measure (count-space (Y ' space M))
  by (simp add: sigma-finite-measure-count-space-finite)
show sigma-finite-measure (count-space (Z ' space M))
  by (simp add: sigma-finite-measure-count-space-finite)
have count-space (X ' space M)  $\otimes_M$  count-space (Y ' space M)  $\otimes_M$  count-space
(Z ' space M) =
  count-space (X' space M  $\times$  Y' space M  $\times$  Z' space M)
  (is ?P = ?C)
  by (simp add: pair-measure-count-space)

let ?Px = λx. measure M (X -' {x}  $\cap$  space M)
have  $(\lambda x. (X x, Z x)) \in \text{measurable } M$  (count-space (X ' space M)  $\otimes_M$  count-space
(Z ' space M))
  using simple-distributed-joint[OF Pxz] by (rule distributed-measurable)
from measurable-comp[OF this measurable-fst]
have random-variable (count-space (X ' space M)) X
  by (simp add: comp-def)
then have simple-function M X
  unfolding simple-function-def by (auto simp: measurable-count-space-eq2)
then have simple-distributed M X ?Px
  by (rule simple-distributedI) (auto simp: measure-nonneg)
then show distributed M (count-space (X ' space M)) X ?Px
  by (rule simple-distributed)

```

```

let ?f = ( $\lambda x$ . if  $x \in (\lambda x$ . ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pxyz$   $x$  else  $0$ )
let ?g = ( $\lambda x$ . if  $x \in (\lambda x$ . ( $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pyz$   $x$  else  $0$ )
let ?h = ( $\lambda x$ . if  $x \in (\lambda x$ . ( $X$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then  $Pxz$   $x$  else  $0$ )
show
  integrable ?P ( $\lambda(x, y, z)$ . ?f ( $x, y, z$ ) * log b (?f ( $x, y, z$ ) / (?P $x$   $x$  * ?g ( $y, z$ )))
  integrable ?P ( $\lambda(x, y, z)$ . ?f ( $x, y, z$ ) * log b (?h ( $x, z$ ) / (?P $x$   $x$  * P $z$   $z$ )))
  by (auto intro!: integrable-count-space simp: pair-measure-count-space)
  let ?i =  $\lambda x y z$ . ?f ( $x, y, z$ ) * log b (?f ( $x, y, z$ ) / (?h ( $x, z$ ) * (?g ( $y, z$ ) / P $z$   $z$ )))
  let ?j =  $\lambda x y z$ .  $Pxyz$  ( $x, y, z$ ) * log b ( $Pxyz$  ( $x, y, z$ ) / (P $xz$  ( $x, z$ ) * (P $yz$  ( $y, z$ ) / P $z$   $z$ )))
  have ( $\lambda(x, y, z)$ . ?i  $x y z$ ) = ( $\lambda x$ . if  $x \in (\lambda x$ . ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$  then ?j (fst  $x$ ) (fst (snd  $x$ )) (snd (snd  $x$ )) else  $0$ )
  by (auto intro!: ext)
  then show ( $\int (x, y, z)$ . ?i  $x y z$   $\partial ?P$ ) = ( $\sum (x, y, z) \in (\lambda x$ . ( $X$   $x$ ,  $Y$   $x$ ,  $Z$   $x$ )) ‘ space  $M$ . ?j  $x y z$ )
  by (auto intro!: sum.cong simp add: ?P = ?C) lebesgue-integral-count-space-finite simple-distributed-finite sum.If-cases split-beta)
qed (insert P $z$  P $yz$  P $xz$  P $xyz$ , auto intro: measure-nonneg)

```

lemma (*in information-space*) *conditional-mutual-information-nonneg*:

assumes X : *simple-function* M X **and** Y : *simple-function* M Y **and** Z : *simple-function* M Z

shows $0 \leq \mathcal{I}(X ; Y | Z)$

proof –

have [*simp*]: *count-space* (X ‘ *space* M) \otimes_M *count-space* (Y ‘ *space* M) \otimes_M *count-space* (Z ‘ *space* M) =

count-space (X ‘ *space* M \times Y ‘ *space* M \times Z ‘ *space* M)

by (*simp add: pair-measure-count-space X Y Z simple-functionD*)

note sf = *sigma-finite-measure-count-space-finite*[*OF simple-functionD*(1)]

note sd = *simple-distributedI*[*OF - - refl*]

note sp = *simple-function-Pair*

show ?*thesis*

apply (*rule conditional-mutual-information-generic-nonneg*[*OF sf*[*OF X*] *sf*[*OF Y*] *sf*[*OF Z*]])

apply (*rule simple-distributed*[*OF sd*[*OF X*]])

apply *simp*

apply *simp*

apply (*rule simple-distributed*[*OF sd*[*OF Z*]])

apply *simp*

apply *simp*

apply (*rule simple-distributed-joint*[*OF sd*[*OF sp*[*OF Y Z*]])

apply *simp*

apply *simp*

apply (*rule simple-distributed-joint*[*OF sd*[*OF sp*[*OF X Z*]])

apply *simp*

apply *simp*

apply (*rule simple-distributed-joint2*[*OF sd*[*OF sp*[*OF X sp*[*OF Y Z*]]]])

```

apply simp
apply simp
apply (auto intro!: integrable-count-space simp: X Y Z simple-functionD)
done
qed

```

12.7 Conditional Entropy

definition (in *prob-space*)

conditional-entropy $b S T X Y = - (\int (x, y). \log b (\text{enn2real} (\text{RN-deriv} (S \otimes_M T) (\text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x)))) (x, y)) / \text{enn2real} (\text{RN-deriv } T (\text{distr } M T Y) y)) \partial \text{distr } M (S \otimes_M T) (\lambda x. (X x, Y x))$

abbreviation (in *information-space*)

conditional-entropy-Pow $(\mathcal{H}'(- | -'))$ **where**
 $\mathcal{H}(X | Y) \equiv \text{conditional-entropy } b (\text{count-space } (X \text{'space } M)) (\text{count-space } (Y \text{'space } M)) X Y$

lemma (in *information-space*) *conditional-entropy-generic-eq*:

```

fixes Pxy :: - => real and Py :: 'c => real
assumes S: sigma-finite-measure S and T: sigma-finite-measure T
assumes Py[measurable]: distributed M T Y Py and Py-nn[simp]:  $\bigwedge x. x \in \text{space } T \implies 0 \leq Py x$ 
assumes Pxy[measurable]: distributed M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ ) Pxy
and Pxy-nn[simp]:  $\bigwedge x y. x \in \text{space } S \implies y \in \text{space } T \implies 0 \leq Pxy (x, y)$ 
shows conditional-entropy b S T X Y = - ( $\int (x, y). Pxy (x, y) * \log b (Pxy (x, y) / Py y) \partial (S \otimes_M T)$ )
proof -
interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..

```

```

have [measurable]: Py  $\in T \rightarrow_M$  borel
using Py Py-nn by (intro distributed-real-measurable)
have measurable-Pxy[measurable]: Pxy  $\in (S \otimes_M T) \rightarrow_M$  borel
using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

```

```

have AE x in density (S  $\otimes_M$  T) ( $\lambda x. \text{ennreal} (Pxy x)$ ). Pxy x = enn2real
(RN-deriv (S  $\otimes_M$  T) (distr M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ )) x)
unfolding AE-density[OF distributed-borel-measurable, OF Pxy]
unfolding distributed-distr-eq-density[OF Pxy]
using distributed-RN-deriv[OF Pxy]
by auto

```

moreover

```

have AE x in density (S  $\otimes_M$  T) ( $\lambda x. \text{ennreal} (Pxy x)$ ). Py (snd x) = enn2real
(RN-deriv T (distr M T Y) (snd x))
unfolding AE-density[OF distributed-borel-measurable, OF Pxy]
unfolding distributed-distr-eq-density[OF Py]

```

```

apply (rule ST.AE-pair-measure)
apply auto
using distributed-RN-deriv[OF Py]
apply auto
done
ultimately
have conditional-entropy b S T X Y = - (∫ x. Pxy x * log b (Pxy x / Py (snd x)) ∂(S ⊗M T))
unfolding conditional-entropy-def neg-equal-iff-equal
apply (subst integral-real-density[symmetric])
apply (auto simp: distributed-distr-eq-density[OF Pxy] space-pair-measure intro!: integral-cong-AE)
done
then show ?thesis by (simp add: split-beta^)
qed

```

lemma (in *information-space*) *conditional-entropy-eq-entropy*:

```

fixes Px :: 'b ⇒ real and Py :: 'c ⇒ real
assumes S: sigma-finite-measure S and T: sigma-finite-measure T
assumes Py[measurable]: distributed M T Y Py
and Py-nn[simp]: ∫ x. x ∈ space T ⇒ 0 ≤ Py x
assumes Pxy[measurable]: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
and Pxy-nn[simp]: ∫ x y. x ∈ space S ⇒ y ∈ space T ⇒ 0 ≤ Pxy (x, y)
assumes I1: integrable (S ⊗M T) (λx. Pxy x * log b (Pxy x))
assumes I2: integrable (S ⊗M T) (λx. Pxy x * log b (Py (snd x)))
shows conditional-entropy b S T X Y = entropy b (S ⊗M T) (λx. (X x, Y x))
– entropy b T Y

```

proof –

```

interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..

```

```

have [measurable]: Py ∈ T →M borel
using Py Py-nn by (intro distributed-real-measurable)
have measurable-Pxy[measurable]: Pxy ∈ (S ⊗M T) →M borel
using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

```

```

have entropy b T Y = - (∫ y. Py y * log b (Py y) ∂T)

```

```

by (rule entropy-distr[OF Py Py-nn])

```

```

also have ... = - (∫ (x,y). Pxy (x,y) * log b (Py y) ∂(S ⊗M T))

```

```

using b-gt-1

```

```

by (subst distributed-transform-integral[OF Pxy - Py, where T=snd])

```

```

(auto intro!: Bochner-Integration.integral-cong simp: space-pair-measure)

```

```

finally have e-eq: entropy b T Y = - (∫ (x,y). Pxy (x,y) * log b (Py y) ∂(S ⊗M T)) .

```

```

have **: ∫ x. x ∈ space (S ⊗M T) ⇒ 0 ≤ Pxy x

```

```

by (auto simp: space-pair-measure)

```

have *ae2*: $AE\ x\ in\ S\ \otimes_M\ T.\ Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
by (*intro subdensity-real[of snd, OF - Pxy Py]*)
(auto intro: measurable-Pair simp: space-pair-measure)
moreover have *ae4*: $AE\ x\ in\ S\ \otimes_M\ T.\ 0 \leq Py\ (snd\ x)$
using *Py* **by** (*intro ST.AE-pair-measure*) (*auto simp: comp-def intro!: measurable-snd'*)
ultimately have $AE\ x\ in\ S\ \otimes_M\ T.\ 0 \leq Pxy\ x \wedge 0 \leq Py\ (snd\ x) \wedge$
 $(Pxy\ x = 0 \vee (Pxy\ x \neq 0 \longrightarrow 0 < Pxy\ x \wedge 0 < Py\ (snd\ x)))$
using *AE-space* **by** *eventually-elim* (*auto simp: space-pair-measure less-le*)
then have *ae*: $AE\ x\ in\ S\ \otimes_M\ T.$
 $Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Py\ (snd\ x)) = Pxy\ x * \log\ b\ (Pxy\ x /$
 $Py\ (snd\ x))$
by *eventually-elim* (*auto simp: log-simps field-simps b-gt-1*)
have *conditional-entropy b S T X Y =*
 $-(\int x.\ Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Py\ (snd\ x))\ \partial(S\ \otimes_M\ T))$
unfolding *conditional-entropy-generic-eq[OF S T Py Py-nn Pxy Pxy-nn, simplified]* *neg-equal-iff-equal*
apply (*intro integral-cong-AE*)
using *ae*
apply *auto*
done
also have $\dots = -(\int x.\ Pxy\ x * \log\ b\ (Pxy\ x)\ \partial(S\ \otimes_M\ T)) - -(\int x.\ Pxy\ x$
 $* \log\ b\ (Py\ (snd\ x))\ \partial(S\ \otimes_M\ T))$
by (*simp add: Bochner-Integration.integral-diff[OF I1 I2]*)
finally show *?thesis*
using *conditional-entropy-generic-eq[OF S T Py Py-nn Pxy Pxy-nn, simplified]*
*entropy-distr[OF Pxy **, simplified]* *e-eq*
by (*simp add: split-beta'*)
qed

lemma (*in information-space*) *conditional-entropy-eq-entropy-simple*:

assumes *X*: *simple-function M X* **and** *Y*: *simple-function M Y*

shows $\mathcal{H}(X\ |Y) = \text{entropy}\ b\ (\text{count-space}\ (X\ \text{space}\ M) \otimes_M \text{count-space}\ (Y\ \text{space}\ M))\ (\lambda x.\ (X\ x,\ Y\ x)) - \mathcal{H}(Y)$

proof –

have $\text{count-space}\ (X\ \text{space}\ M) \otimes_M \text{count-space}\ (Y\ \text{space}\ M) = \text{count-space}\ (X\ \text{space}\ M \times Y\ \text{space}\ M)$

(*is ?P = ?C*) **using** *X Y* **by** (*simp add: simple-functionD pair-measure-count-space*)

show *?thesis*

by (*rule conditional-entropy-eq-entropy sigma-finite-measure-count-space-finite*
simple-functionD X Y simple-distributed simple-distributedI[OF - - refl]
simple-distributed-joint simple-function-Pair integrable-count-space

measure-nonneg)+

(*auto simp: <?P = ?C> measure-nonneg intro!: integrable-count-space simple-functionD X Y*)

qed

lemma (*in information-space*) *conditional-entropy-eq*:

assumes *Y*: *simple-distributed M Y Py*

assumes XY : *simple-distributed* M $(\lambda x. (X\ x, Y\ x))$ Pxy
shows $\mathcal{H}(X \mid Y) = - (\sum (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ 'space } M. Pxy\ (x, y) * \log b\ (Pxy\ (x, y) / Py\ y))$
proof (*subst conditional-entropy-generic-eq*[OF - -
simple-distributed[$OF\ Y$] - *simple-distributed-joint*[$OF\ XY$]])
have *finite* $((\lambda x. (X\ x, Y\ x)) \text{ 'space } M)$
using XY **unfolding** *simple-distributed-def* **by** *auto*
from *finite-imageI*[$OF\ this, of\ fst$]
have [*simp*]: *finite* $(X \text{ 'space } M)$
by (*simp add: image-comp comp-def*)
note Y [*THEN simple-distributed-finite, simp*]
show *sigma-finite-measure* $(count\ space\ (X \text{ 'space } M))$
by (*simp add: sigma-finite-measure-count-space-finite*)
show *sigma-finite-measure* $(count\ space\ (Y \text{ 'space } M))$
by (*simp add: sigma-finite-measure-count-space-finite*)
let $?f = (\lambda x. \text{if } x \in (\lambda x. (X\ x, Y\ x)) \text{ 'space } M \text{ then } Pxy\ x \text{ else } 0)$
have $count\ space\ (X \text{ 'space } M) \otimes_M count\ space\ (Y \text{ 'space } M) = count\ space\ (X \text{ 'space } M \times Y \text{ 'space } M)$
(is $?P = ?C$)
using Y **by** (*simp add: simple-distributed-finite-pair-measure-count-space*)
have $eq: (\lambda(x, y). ?f\ (x, y) * \log b\ (?f\ (x, y) / Py\ y)) =$
 $(\lambda x. \text{if } x \in (\lambda x. (X\ x, Y\ x)) \text{ 'space } M \text{ then } Pxy\ x * \log b\ (Pxy\ x / Py\ (snd\ x))$
else $0)$
by *auto*
from Y **show** $- (\int (x, y). ?f\ (x, y) * \log b\ (?f\ (x, y) / Py\ y) \partial ?P) =$
 $- (\sum (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ 'space } M. Pxy\ (x, y) * \log b\ (Pxy\ (x, y) / Py\ y))$
by (*auto intro!*: *sum.cong simp add: (?P = ?C) lebesgue-integral-count-space-finite*
simple-distributed-finite eq sum.If-cases split-beta)
qed (*insert* $Y\ XY, auto$)

lemma (*in information-space*) *conditional-mutual-information-eq-conditional-entropy*:

assumes X : *simple-function* $M\ X$ **and** Y : *simple-function* $M\ Y$

shows $\mathcal{I}(X ; X \mid Y) = \mathcal{H}(X \mid Y)$

proof –

define Py **where** $Py\ x = (\text{if } x \in Y \text{ 'space } M \text{ then } measure\ M\ (Y - \{x\} \cap space\ M) \text{ else } 0)$ **for** x

define Pxy **where** $Pxy\ x =$

$(\text{if } x \in (\lambda x. (X\ x, Y\ x)) \text{ 'space } M \text{ then } measure\ M\ ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap space\ M) \text{ else } 0)$

for x

define $Pxxy$ **where** $Pxxy\ x =$

$(\text{if } x \in (\lambda x. (X\ x, X\ x, Y\ x)) \text{ 'space } M \text{ then } measure\ M\ ((\lambda x. (X\ x, X\ x, Y\ x)) - \{x\} \cap space\ M) \text{ else } 0)$

for x

let $?M = X \text{ 'space } M \times X \text{ 'space } M \times Y \text{ 'space } M$

note $XY = \text{simple-function-Pair}[OF\ X\ Y]$

```

note  $XXY = \text{simple-function-Pair}[OF X XY]$ 
have  $Py: \text{simple-distributed } M Y Py$ 
  using  $Y$  by  $(\text{rule simple-distributedI}) (\text{auto simp: } Py\text{-def measure-nonneg})$ 
have  $Pxy: \text{simple-distributed } M (\lambda x. (X x, Y x)) Pxy$ 
  using  $XY$  by  $(\text{rule simple-distributedI}) (\text{auto simp: } Pxy\text{-def measure-nonneg})$ 
have  $Pxxy: \text{simple-distributed } M (\lambda x. (X x, X x, Y x)) Pxxy$ 
  using  $XXY$  by  $(\text{rule simple-distributedI}) (\text{auto simp: } Pxxy\text{-def measure-nonneg})$ 
have  $eq: (\lambda x. (X x, X x, Y x)) \text{ 'space } M = (\lambda(x, y). (x, x, y)) \text{ ' } (\lambda x. (X x, Y x)) \text{ 'space } M$ 
  by  $\text{auto}$ 
have  $\text{inj: } \bigwedge A. \text{inj-on } (\lambda(x, y). (x, x, y)) A$ 
  by  $(\text{auto simp: inj-on-def})$ 
have  $Pxxy\text{-eq: } \bigwedge x y. Pxxy (x, x, y) = Pxy (x, y)$ 
  by  $(\text{auto simp: } Pxxy\text{-def } Pxy\text{-def intro!: arg-cong[where f=prob])$ 
have  $AE x \text{ in count-space } ((\lambda x. (X x, Y x)) \text{ 'space } M). Py (\text{snd } x) = 0 \longrightarrow Pxy x = 0$ 
  using  $Py Pxy$ 
  by  $(\text{intro subdensity-real[of snd, OF - } Pxy[\text{THEN simple-distributed}]] Py[\text{THEN simple-distributed}]))$ 
   $(\text{auto intro: measurable-Pair simp: AE-count-space})$ 
then show  $?thesis$ 
  apply  $(\text{subst conditional-mutual-information-eq}[OF Py Pxy Pxy Pxxy])$ 
  apply  $(\text{subst conditional-entropy-eq}[OF Py Pxy])$ 
  apply  $(\text{auto intro!: sum.cong simp: } Pxxy\text{-eq sum-negf[symmetric] eq sum.reindex}[OF inj])$ 
   $\text{log-simps zero-less-mult-iff zero-le-mult-iff field-simps mult-less-0-iff AE-count-space})$ 
  using  $Py[\text{THEN simple-distributed}] Pxy[\text{THEN simple-distributed}]$ 
  apply  $(\text{auto simp add: not-le AE-count-space less-le antisym simple-distributed-nonneg}[OF Py] \text{simple-distributed-nonneg}[OF Pxy])$ 
done
qed

```

lemma (in *information-space*) *conditional-entropy-nonneg*:

assumes $X: \text{simple-function } M X$ **and** $Y: \text{simple-function } M Y$ **shows** $0 \leq \mathcal{H}(X | Y)$

using *conditional-mutual-information-eq-conditional-entropy*[OF $X Y$] *conditional-mutual-information-nonneg*[OF $X X Y$]

by *simp*

12.8 Equalities

lemma (in *information-space*) *mutual-information-eq-entropy-conditional-entropy-distr*:

fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$

assumes $S: \text{sigma-finite-measure } S$ **and** $T: \text{sigma-finite-measure } T$

assumes $Px[\text{measurable}]: \text{distributed } M S X Px$

and $Px\text{-nn}[simp]: \bigwedge x. x \in \text{space } S \implies 0 \leq Px x$

and $Py[\text{measurable}]: \text{distributed } M T Y Py$

and $Py\text{-nn}[simp]: \bigwedge x. x \in \text{space } T \implies 0 \leq Py x$


```

and Pxy[measurable]: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
and Pxy-nn[simp]: ∧x y. x ∈ space S ⇒ y ∈ space T ⇒ 0 ≤ Pxy (x, y)
assumes Ix: integrable(S ⊗M T) (λx. Pxy x * log b (Px (fst x)))
assumes Iy: integrable(S ⊗M T) (λx. Pxy x * log b (Py (snd x)))
assumes Ixy: integrable(S ⊗M T) (λx. Pxy x * log b (Pxy x))
shows mutual-information b S T X Y = entropy b S X + entropy b T Y -
entropy b (S ⊗M T) (λx. (X x, Y x))
proof -
  have [measurable]: Px ∈ S →M borel
    using Px Px-nn by (intro distributed-real-measurable)
  have [measurable]: Py ∈ T →M borel
    using Py Py-nn by (intro distributed-real-measurable)
  have measurable-Pxy[measurable]: Pxy ∈ (S ⊗M T) →M borel
    using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

  have X: entropy b S X = - (∫ x. Pxy x * log b (Px (fst x)) ∂(S ⊗M T))
    using b-gt-1
    apply (subst entropy-distr[OF Px Px-nn], simp)
    apply (subst distributed-transform-integral[OF Pxy - Px, where T=fst])
    apply (auto intro!: integral-cong simp: space-pair-measure)
    done

  have Y: entropy b T Y = - (∫ x. Pxy x * log b (Py (snd x)) ∂(S ⊗M T))
    using b-gt-1
    apply (subst entropy-distr[OF Py Py-nn], simp)
    apply (subst distributed-transform-integral[OF Pxy - Py, where T=snd])
    apply (auto intro!: integral-cong simp: space-pair-measure)
    done

  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  have ST: sigma-finite-measure (S ⊗M T) ..

  have XY: entropy b (S ⊗M T) (λx. (X x, Y x)) = - (∫ x. Pxy x * log b (Pxy
x) ∂(S ⊗M T))
    by (subst entropy-distr[OF Pxy]) (auto intro!: integral-cong simp: space-pair-measure)

  have AE x in S ⊗M T. Px (fst x) = 0 → Pxy x = 0
    by (intro subdensity-real[of fst, OF - Pxy Px]) (auto intro: measurable-Pair
simp: space-pair-measure)
  moreover have AE x in S ⊗M T. Py (snd x) = 0 → Pxy x = 0
    by (intro subdensity-real[of snd, OF - Pxy Py]) (auto intro: measurable-Pair
simp: space-pair-measure)
  moreover have AE x in S ⊗M T. 0 ≤ Px (fst x)
    using Px by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measur-
able-fst')
  moreover have AE x in S ⊗M T. 0 ≤ Py (snd x)
    using Py by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measur-

```

able-snd'')

ultimately have $AE\ x\ in\ S \otimes_M T. Pxy\ x * \log\ b\ (Pxy\ x) - Pxy\ x * \log\ b\ (Px\ (fst\ x)) - Pxy\ x * \log\ b\ (Py\ (snd\ x)) =$
 $Pxy\ x * \log\ b\ (Pxy\ x / (Px\ (fst\ x) * Py\ (snd\ x)))$
(is $AE\ x\ in\ \cdot. ?f\ x = ?g\ x)$
using *AE-space*
proof *eventually-elim*
case *(elim x)*
show *?case*
proof *cases*
assume $Pxy\ x \neq 0$
with *elim* **have** $0 < Px\ (fst\ x)\ 0 < Py\ (snd\ x)\ 0 < Pxy\ x$
by *(auto simp: space-pair-measure less-le)*
then show *?thesis*
using *b-gt-1* **by** *(simp add: log-simps less-imp-le field-simps)*
qed *simp*
qed

have $entropy\ b\ S\ X + entropy\ b\ T\ Y - entropy\ b\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))$
 $=\ integral^L\ (S \otimes_M T)\ ?f$
unfolding *X Y XY*
apply *(subst Bochner-Integration.integral-diff)*
apply *(intro Bochner-Integration.integrable-diff Ixy Ix Iy)+*
apply *(subst Bochner-Integration.integral-diff)*
apply *(intro Ixy Ix Iy)+*
apply *(simp add: field-simps)*
done
also have $\dots =\ integral^L\ (S \otimes_M T)\ ?g$
using $\langle AE\ x\ in\ \cdot. ?f\ x = ?g\ x \rangle$ **by** *(intro integral-cong-AE) auto*
also have $\dots =\ mutual-information\ b\ S\ T\ X\ Y$
by *(rule mutual-information-distr[OF S T Px - Py - Pxy - , symmetric])*
(auto simp: space-pair-measure)
finally show *?thesis ..*
qed

lemma *(in information-space) mutual-information-eq-entropy-conditional-entropy'*:

fixes $Px :: 'b \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$ **and** $Pxy :: ('b \times 'c) \Rightarrow real$
assumes S : *sigma-finite-measure S* **and** T : *sigma-finite-measure T*
assumes Px : *distributed M S X Px* $\bigwedge x. x \in space\ S \implies 0 \leq Px\ x$
and Py : *distributed M T Y Py* $\bigwedge x. x \in space\ T \implies 0 \leq Py\ x$
assumes Pxy : *distributed M (S \otimes_M T) (\lambda x. (X\ x, Y\ x)) Pxy*
 $\bigwedge x. x \in space\ (S \otimes_M T) \implies 0 \leq Pxy\ x$
assumes Ix : *integrable(S \otimes_M T) (\lambda x. Pxy\ x * \log\ b\ (Px\ (fst\ x)))*
assumes Iy : *integrable(S \otimes_M T) (\lambda x. Pxy\ x * \log\ b\ (Py\ (snd\ x)))*
assumes Ixy : *integrable(S \otimes_M T) (\lambda x. Pxy\ x * \log\ b\ (Pxy\ x))*
shows $mutual-information\ b\ S\ T\ X\ Y = entropy\ b\ S\ X - conditional-entropy\ b\ S\ T\ X\ Y$
using
mutual-information-eq-entropy-conditional-entropy-distr[OF S T Px Py Pxy Ix

Iy Ixy]
conditional-entropy-eq-entropy[OF S T Py Pxy Ixy Iy]
by (*simp add: space-pair-measure*)

lemma (*in information-space*) *mutual-information-eq-entropy-conditional-entropy:*
assumes *sf-X: simple-function M X and sf-Y: simple-function M Y*
shows $\mathcal{I}(X ; Y) = \mathcal{H}(X) - \mathcal{H}(X | Y)$

proof –

have *X: simple-distributed M X* ($\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M)$)
using *sf-X by* (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
have *Y: simple-distributed M Y* ($\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M)$)
using *sf-Y by* (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
have *sf-XY: simple-function M* ($\lambda x. (X x, Y x)$)
using *sf-X sf-Y by* (*rule simple-function-Pair*)
then have *XY: simple-distributed M* ($\lambda x. (X x, Y x)$) ($\lambda x. \text{measure } M ((\lambda x. (X x, Y x)) - \{x\} \cap \text{space } M)$)
by (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
from *simple-distributed-joint-finite[OF this, simp]*
have *eq: count-space (X 'space M) \otimes_M count-space (Y 'space M) = count-space (X 'space M \times Y 'space M)*
by (*simp add: pair-measure-count-space*)

have $\mathcal{I}(X ; Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \text{entropy } b (\text{count-space } (X \text{'space } M) \otimes_M \text{count-space } (Y \text{'space } M)) (\lambda x. (X x, Y x))$
using *sigma-finite-measure-count-space-finite*
sigma-finite-measure-count-space-finite
simple-distributed[OF X] measure-nonneg simple-distributed[OF Y] measure-nonneg simple-distributed-joint[OF XY]
by (*rule mutual-information-eq-entropy-conditional-entropy-distr*)
(auto simp: eq-integrable-count-space measure-nonneg)
then show *?thesis*
unfolding *conditional-entropy-eq-entropy-simple[OF sf-X sf-Y]* **by** *simp*
qed

lemma (*in information-space*) *mutual-information-nonneg-simple:*
assumes *sf-X: simple-function M X and sf-Y: simple-function M Y*
shows $0 \leq \mathcal{I}(X ; Y)$

proof –

have *X: simple-distributed M X* ($\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M)$)
using *sf-X by* (*rule simple-distributedI*) (*auto simp: measure-nonneg*)
have *Y: simple-distributed M Y* ($\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M)$)
using *sf-Y by* (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

have *sf-XY: simple-function M* ($\lambda x. (X x, Y x)$)
using *sf-X sf-Y by* (*rule simple-function-Pair*)
then have *XY: simple-distributed M* ($\lambda x. (X x, Y x)$) ($\lambda x. \text{measure } M ((\lambda x. (X x, Y x)) - \{x\} \cap \text{space } M)$)
by (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

from *simple-distributed-joint-finite*[*OF this, simp*]
have *eq: count-space* (X ‘ *space* M) \otimes_M *count-space* (Y ‘ *space* M) = *count-space* (X ‘ *space* $M \times Y$ ‘ *space* M)
by (*simp add: pair-measure-count-space*)

show *?thesis*
by (*rule mutual-information-nonneg*[*OF - - simple-distributed*[*OF X*] - *simple-distributed*[*OF Y*] - *simple-distributed-joint*[*OF XY*]])
(*simp-all add: eq integrable-count-space sigma-finite-measure-count-space-finite measure-nonneg*)
qed

lemma (*in information-space*) *conditional-entropy-less-eq-entropy*:

assumes X : *simple-function* $M X$ **and** Z : *simple-function* $M Z$
shows $\mathcal{H}(X \mid Z) \leq \mathcal{H}(X)$
proof –
have $0 \leq \mathcal{I}(X ; Z)$ **using** $X Z$ **by** (*rule mutual-information-nonneg-simple*)
also have $\mathcal{I}(X ; Z) = \mathcal{H}(X) - \mathcal{H}(X \mid Z)$ **using** *mutual-information-eq-entropy-conditional-entropy*[*OF assms*].
finally show *?thesis* **by** *auto*
qed

lemma (*in information-space*)

fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes Px : *finite-entropy* $S X Px$ **and** PY : *finite-entropy* $T Y Py$
assumes Pxy : *finite-entropy* ($S \otimes_M T$) ($\lambda x. (X x, Y x)$) Pxy
shows *conditional-entropy* $b S T X Y \leq$ *entropy* $b S X$

proof –

have $0 \leq$ *mutual-information* $b S T X Y$
by (*rule mutual-information-nonneg'*) *fact+*
also have $\dots =$ *entropy* $b S X -$ *conditional-entropy* $b S T X Y$
apply (*rule mutual-information-eq-entropy-conditional-entropy'*)
using *assms*
by (*auto intro!*: *finite-entropy-integrable finite-entropy-distributed finite-entropy-integrable-transform*[*OF Px*]
finite-entropy-integrable-transform[*OF Py*]
intro: finite-entropy-nn)

finally show *?thesis* **by** *auto*

qed

lemma (*in information-space*) *entropy-chain-rule*:

assumes X : *simple-function* $M X$ **and** Y : *simple-function* $M Y$
shows $\mathcal{H}(\lambda x. (X x, Y x)) = \mathcal{H}(X) + \mathcal{H}(Y \mid X)$
proof –
note $XY =$ *simple-distributedI*[*OF simple-function-Pair*[*OF X Y*] *measure-nonneg refl*]
note $YX =$ *simple-distributedI*[*OF simple-function-Pair*[*OF Y X*] *measure-nonneg*]

refl]

note *simple-distributed-joint-finite*[OF this, simp]

let $?f = \lambda x. \text{prob } ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap \text{space } M)$

let $?g = \lambda x. \text{prob } ((\lambda x. (Y\ x, X\ x)) - \{x\} \cap \text{space } M)$

let $?h = \lambda x. \text{if } x \in (\lambda x. (Y\ x, X\ x)) \text{ ' space } M \text{ then prob } ((\lambda x. (Y\ x, X\ x)) - \{x\} \cap \text{space } M) \text{ else } 0$

have $\mathcal{H}(\lambda x. (X\ x, Y\ x)) = - (\sum_{x \in (\lambda x. (X\ x, Y\ x)) \text{ ' space } M}. ?f\ x * \log b\ (?f\ x))$

using *XY* **by** (*rule entropy-simple-distributed*)

also have $\dots = - (\sum_{x \in (\lambda(x, y). (y, x)) \text{ ' } (\lambda x. (X\ x, Y\ x)) \text{ ' space } M}. ?g\ x * \log b\ (?g\ x))$

by (*subst* (?)) *sum.reindex* (*auto simp: inj-on-def intro!: sum.cong arg-cong*[**where** $f = \lambda A. \text{prob } A * \log b\ (\text{prob } A)$])

also have $\dots = - (\sum_{x \in (\lambda x. (Y\ x, X\ x)) \text{ ' space } M}. ?h\ x * \log b\ (?h\ x))$

by (*auto intro!: sum.cong*)

also have $\dots = \text{entropy } b\ (\text{count-space } (Y \text{ ' space } M) \otimes_M \text{count-space } (X \text{ ' space } M)) (\lambda x. (Y\ x, X\ x))$

by (*subst entropy-distr*[OF *simple-distributed-joint*[OF *YX*]])

(*auto simp: pair-measure-count-space sigma-finite-measure-count-space-finite lebesgue-integral-count-space-finite*

cong del: sum.cong-simp intro!: sum.mono-neutral-left measure-nonneg)

finally have $\mathcal{H}(\lambda x. (X\ x, Y\ x)) = \text{entropy } b\ (\text{count-space } (Y \text{ ' space } M) \otimes_M \text{count-space } (X \text{ ' space } M)) (\lambda x. (Y\ x, X\ x)) .$

then show *?thesis*

unfolding *conditional-entropy-eq-entropy-simple*[OF *Y X*] **by** *simp*

qed

lemma (*in information-space*) *entropy-partition*:

assumes *X: simple-function M X*

shows $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$

proof –

note $fX = \text{simple-function-compose}$ [OF *X*, *of f*]

have $eq: (\lambda x. ((f \circ X)\ x, X\ x)) \text{ ' space } M = (\lambda x. (f\ x, x)) \text{ ' } X \text{ ' space } M$ **by** *auto*

have $inj: \bigwedge A. \text{inj-on } (\lambda x. (f\ x, x))\ A$

by (*auto simp: inj-on-def*)

show *?thesis*

apply (*subst entropy-chain-rule*[*symmetric*, OF *fX X*])

apply (*subst entropy-simple-distributed*[OF *simple-distributedI*[OF *simple-function-Pair*[OF *fX X*] *measure-nonneg refl*]])

apply (*subst entropy-simple-distributed*[OF *simple-distributedI*[OF *X* *measure-nonneg refl*]])

unfolding *eq*

apply (*subst sum.reindex*[OF *inj*])

apply (*auto intro!: sum.cong arg-cong*[**where** $f = \lambda A. \text{prob } A * \log b\ (\text{prob } A)$])

done

qed

corollary (*in information-space*) *entropy-data-processing*:

assumes *X: simple-function M X* **shows** $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$

```

proof –
  note  $fX = \text{simple-function-compose}[OF\ X, of\ f]$ 
  from  $X$  have  $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$  by (rule entropy-partition)
  then show  $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$ 
    by (simp only: conditional-entropy-nonneg [OF X fX] le-add-same-cancel1)
qed

corollary (in information-space) entropy-of-inj:
  assumes  $X$ : simple-function  $M\ X$  and inj: inj-on  $f$  ( $X$ 'space  $M$ )
  shows  $\mathcal{H}(f \circ X) = \mathcal{H}(X)$ 
proof (rule antisym)
  show  $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$  using entropy-data-processing[ $OF\ X$ ] .
next
  have sf: simple-function  $M$  ( $f \circ X$ )
    using  $X$  by auto
  have  $\mathcal{H}(X) = \mathcal{H}(\text{the-inv-into } (X\text{'space } M)\ f \circ (f \circ X))$ 
    unfolding o-assoc
  apply (subst entropy-simple-distributed[ $OF\ \text{simple-distributedI}$ [ $OF\ X\ \text{measure-nonneg}\ \text{refl}$ ]])
  apply (subst entropy-simple-distributed[ $OF\ \text{simple-distributedI}$ [ $OF\ \text{simple-function-compose}$ [ $OF\ X$ ]]], where  $f = \lambda x. \text{prob } (X - \{x\} \cap \text{space } M)$ )
  apply (auto intro!: sum.cong arg-cong[where  $f = \text{prob}$ ] image-eqI simp: the-inv-into-f-f[ $OF\ \text{inj}$ ] comp-def measure-nonneg)
  done
  also have  $\dots \leq \mathcal{H}(f \circ X)$ 
    using entropy-data-processing[ $OF\ sf$ ] .
  finally show  $\mathcal{H}(X) \leq \mathcal{H}(f \circ X)$  .
qed

end

```

13 Properties of Various Distributions

theory *Distributions*

imports *Convolution Information*

begin

lemma (*in prob-space*) *distributed-affine*:

fixes $f :: \text{real} \Rightarrow \text{ennreal}$

assumes f : *distributed* $M\ \text{lborel } X\ f$

assumes c : $c \neq 0$

shows *distributed* $M\ \text{lborel } (\lambda x. t + c * X\ x)\ (\lambda x. f\ ((x - t) / c) / |c|)$

unfolding *distributed-def*

proof *safe*

have [*measurable*]: $f \in \text{borel-measurable } \text{borel}$

using f **by** (*simp add: distributed-def*)

have [*measurable*]: $X \in \text{borel-measurable } M$

using f **by** (*simp add: distributed-def*)

```

show ( $\lambda x. f ((x - t) / c) / |c|$ )  $\in$  borel-measurable lborel
  by simp
show random-variable lborel ( $\lambda x. t + c * X x$ )
  by simp

have eq: ennreal |c| * (f x / ennreal |c|) = f x for  $x$ 
  using  $c$ 
  by (cases f x)
    (auto simp: divide-ennreal ennreal-mult[symmetric] ennreal-top-divide en-
nreal-mult-top)

have density lborel f = distr M lborel X
  using  $f$  by (simp add: distributed-def)
with  $c$  show distr M lborel ( $\lambda x. t + c * X x$ ) = density lborel ( $\lambda x. f ((x - t) /$ 
 $c) / ennreal |c|$ )
  by (subst (2) lborel-real-affine[where c=c and t=t])
    (simp-all add: density-density-eq density-distr distr-distr field-simps eq cong:
distr-cong)
qed

lemma (in prob-space) distributed-affineI:
  fixes  $f :: real \Rightarrow ennreal$  and  $c :: real$ 
  assumes  $f$ : distributed M lborel ( $\lambda x. (X x - t) / c$ ) ( $\lambda x. |c| * f (x * c + t)$ )
  assumes  $c$ :  $c \neq 0$ 
  shows distributed M lborel X f
proof –
  have eq: f x * ennreal |c| / ennreal |c| = f x for  $x$ 
    using  $c$  by (simp add: ennreal-times-divide[symmetric])

  show ?thesis
    using distributed-affine[OF f c, where t=t] c
    by (simp add: field-simps eq)
qed

lemma (in prob-space) distributed-AE2:
  assumes [measurable]: distributed M N X f Measurable.pred N P
  shows (AE x in M. P (X x))  $\longleftrightarrow$  (AE x in N. 0 < f x  $\longrightarrow$  P x)
proof –
  have (AE x in M. P (X x))  $\longleftrightarrow$  (AE x in distr M N X. P x)
    by (simp add: AE-distr-iff)
  also have  $\dots \longleftrightarrow$  (AE x in density N f. P x)
    unfolding distributed-distr-eq-density[OF assms(1)] ..
  also have  $\dots \longleftrightarrow$  (AE x in N. 0 < f x  $\longrightarrow$  P x)
    by (rule AE-density) simp
  finally show ?thesis .
qed

```

13.1 Erlang

lemma *nn-integral-power-times-exp-Icc*:

assumes *[arith]*: $0 \leq a$

shows $(\int^+ x. \text{ennreal } (x^k * \text{exp } (-x)) * \text{indicator } \{0 .. a\} x \ \partial \text{lborel}) =$
 $(1 - (\sum_{n \leq k}. (a^n * \text{exp } (-a)) / \text{fact } n)) * \text{fact } k$ (**is** *?I = -*)

proof –

let *?f* = $\lambda k x. x^k * \text{exp } (-x) / \text{fact } k$

let *?F* = $\lambda k x. - (\sum_{n \leq k}. (x^n * \text{exp } (-x)) / \text{fact } n)$

have *?I* * (*inverse* (*real-of-nat* (*fact k*))) =

$(\int^+ x. \text{ennreal } (x^k * \text{exp } (-x)) * \text{indicator } \{0 .. a\} x * (\text{inverse } (\text{real-of-nat}$
 $(\text{fact } k))) \ \partial \text{lborel})$

by (*intro nn-integral-mult[symmetric]*) *auto*

also have $\dots = (\int^+ x. \text{ennreal } (?f k x) * \text{indicator } \{0 .. a\} x \ \partial \text{lborel})$

by (*intro nn-integral-cong*)

(*simp add: field-simps ennreal-mult'[symmetric] indicator-mult-ennreal*)

also have $\dots = \text{ennreal } (?F k a - ?F k 0)$

proof (*rule nn-integral-FTC-Icc*)

fix *x* **assume** $x \in \{0..a\}$

show *DERIV* (*?F k*) *x* $>$ *?f k x*

proof(*induction k*)

case *0* **show** *?case* **by** (*auto intro!: derivative-eq-intros*)

next

case (*Suc k*)

have *DERIV* ($\lambda x. ?F k x - (x^{\text{Suc } k} * \text{exp } (-x)) / \text{fact } (\text{Suc } k)$) *x* $>$

*?f k x - ((real (Suc k) - x) * x^k * \text{exp } (-x)) / (\text{fact } (\text{Suc } k))*

by (*intro DERIV-diff Suc*)

(*auto intro!: derivative-eq-intros simp del: fact-Suc power-Suc*

simp add: field-simps power-Suc[symmetric])

also have ($\lambda x. ?F k x - (x^{\text{Suc } k} * \text{exp } (-x)) / \text{fact } (\text{Suc } k)$) = *?F* (*Suc k*)

by *simp*

also have *?f k x - ((real (Suc k) - x) * x^k * \text{exp } (-x)) / (\text{fact } (\text{Suc } k))*

= *?f* (*Suc k*) *x*

by (*auto simp: field-simps simp del: fact-Suc*)

(*simp-all add: of-nat-Suc field-simps*)

finally show *?case* .

qed

qed *auto*

also have $\dots = \text{ennreal } (1 - (\sum_{n \leq k}. (a^n * \text{exp } (-a)) / \text{fact } n))$

by (*auto simp: power-0-left if-distrib[where f= $\lambda x. x / a$ for a] sum.If-cases*)

also have $\dots = \text{ennreal } ((1 - (\sum_{n \leq k}. (a^n * \text{exp } (-a)) / \text{fact } n)) * \text{fact } k) * \text{ennreal } (\text{inverse } (\text{fact } k))$

by (*subst ennreal-mult'[symmetric]*) (*auto intro!: arg-cong[where f=ennreal]*)

finally show *?thesis*

by (*auto simp add: mult-right-ennreal-cancel le-less*)

qed

lemma *nn-integral-power-times-exp-Ici*:

shows $(\int^+ x. \text{ennreal } (x^k * \text{exp } (-x)) * \text{indicator } \{0 ..\} x \ \partial \text{lborel}) = \text{real-of-nat}$
 $(\text{fact } k)$

proof (rule LIMSEQ-unique)

let $?X = \lambda n. \int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots \text{real } n\} x \partial \text{lborel}$
show $?X \longrightarrow (\int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots\} x \partial \text{lborel})$
apply (intro nn-integral-LIMSEQ)
apply (auto simp: incseq-def le-fun-def eventually-sequentially
split: split-indicator intro!: tendsto-eventually)
apply (metis nat-ceiling-le-eq)
done

have $((\lambda x::\text{real}. (1 - (\sum_{n \leq k}. (x^n / \exp x) / (\text{fact } n))) * \text{fact } k) \longrightarrow$
 $(1 - (\sum_{n \leq k}. 0 / (\text{fact } n))) * \text{fact } k) \text{ at-top}$
by (intro tendsto-intros tendsto-power-div-exp-0) simp
then show $?X \longrightarrow \text{real-of-nat } (\text{fact } k)$
by (subst nn-integral-power-times-exp-Icc)
(auto simp: exp-minus field-simps intro!: filterlim-compose[OF - filterlim-real-sequentially])

qed

definition *erlang-density* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

erlang-density $k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } (l^{(\text{Suc } k)} * x^k * \exp(-l * x)) / \text{fact } k)$

definition *erlang-CDF* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

erlang-CDF $k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } 1 - (\sum_{n \leq k}. ((l * x)^n * \exp(-l * x)) / \text{fact } n))$

lemma *erlang-density-nonneg[simp]*: $0 \leq l \implies 0 \leq \text{erlang-density } k \ l \ x$

by (simp add: erlang-density-def)

lemma *borel-measurable-erlang-density[measurable]*: $\text{erlang-density } k \ l \in \text{borel-measurable borel}$

by (auto simp add: erlang-density-def[abs-def])

lemma *erlang-CDF-transform*: $0 < l \implies \text{erlang-CDF } k \ l \ a = \text{erlang-CDF } k \ 1 \ (l * a)$

by (auto simp add: erlang-CDF-def mult-less-0-iff)

lemma *erlang-CDF-nonneg[simp]*: **assumes** $0 < l$ **shows** $0 \leq \text{erlang-CDF } k \ l \ x$

unfolding *erlang-CDF-def*

proof (clarsimp simp: not-less)

assume $0 \leq x$

have $(\sum_{n \leq k}. (l * x)^n * \exp(-l * x)) / \text{fact } n =$
 $\exp(-l * x) * (\sum_{n \leq k}. (l * x)^n / \text{fact } n)$

unfolding *sum-distrib-left* **by** (intro sum.cong) (auto simp: field-simps)

also have $\dots = (\sum_{n \leq k}. (l * x)^n / \text{fact } n) / \exp(l * x)$

by (simp add: exp-minus field-simps)

also have $\dots \leq 1$

proof (subst divide-le-eq-1-pos)

show $(\sum_{n \leq k}. (l * x)^n / \text{fact } n) \leq \exp(l * x)$

using $\langle 0 < l \rangle \langle 0 \leq x \rangle$ *summable-exp-generic*[of $l * x$]

by (auto simp: exp-def divide-inverse ac-simps intro!: sum-le-suminf)
 qed simp
 finally show $(\sum n \leq k. (l * x) ^ n * \exp(- (l * x)) / \text{fact } n) \leq 1$.
 qed

lemma nn-integral-erlang-density:

assumes [arith]: $0 < l$
 shows $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) = \text{erlang-CDF } k \ l \ a$
proof (cases $0 \leq a$)
 case [arith]: True
 have eq: $\bigwedge x. \text{indicator } \{0..a\} (x / l) = \text{indicator } \{0..a * l\} x$
 by (simp add: field-simps split: split-indicator)
 have $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) =$
 $(\int^+ x. (l / \text{fact } k) * (\text{ennreal } ((l * x) ^ k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x) \ \partial \text{lborel})$
 by (intro nn-integral-cong)
 (auto simp: erlang-density-def power-mult-distrib ennreal-mult[symmetric]
 split: split-indicator)
 also have $\dots = (l / \text{fact } k) * (\int^+ x. \text{ennreal } ((l * x) ^ k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x \ \partial \text{lborel})$
 by (intro nn-integral-cmult) auto
 also have $\dots = \text{ennreal } (l / \text{fact } k) * ((1 / l) * (\int^+ x. \text{ennreal } (x ^ k * \exp(- x)) * \text{indicator } \{0 .. l * a\} \ x \ \partial \text{lborel}))$
 by (subst nn-integral-real-affine[where c=1 / l and t=0]) (auto simp: field-simps eq)
 also have $\dots = (1 - (\sum n \leq k. ((l * a) ^ n * \exp(- (l * a))) / \text{fact } n))$
 by (subst nn-integral-power-times-exp-Icc) (auto simp: ennreal-mult[symmetric])
 also have $\dots = \text{erlang-CDF } k \ l \ a$
 by (auto simp: erlang-CDF-def)
 finally show ?thesis .

next

case False
 then have $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) =$
 $(\int^+ x. 0 \ \partial (\text{lborel}::\text{real measure}))$
 by (intro nn-integral-cong) (auto simp: erlang-density-def)
 with False show ?thesis
 by (simp add: erlang-CDF-def)
 qed

lemma emeasure-erlang-density:

$0 < l \implies \text{emeasure } (\text{density } \text{lborel } (\text{erlang-density } k \ l)) \ \{.. \ a\} = \text{erlang-CDF } k \ l \ a$
 by (simp add: emeasure-density nn-integral-erlang-density)

lemma nn-integral-erlang-ith-moment:

fixes $k \ i :: \text{nat}$ and $l :: \text{real}$
 assumes [arith]: $0 < l$
 shows $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x * x ^ i) \ \partial \text{lborel}) = \text{fact } (k + i) /$

```

(fact k * l ^ i)
proof -
  have eq:  $\bigwedge x. \text{indicator } \{0..\} (x / l) = \text{indicator } \{0..\} x$ 
    by (simp add: field-simps split: split-indicator)
  have  $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x * x^i) \ \partial \text{lborel}) =$ 
 $(\int^+ x. (l / (\text{fact } k * l^i)) * (\text{ennreal } ((l*x)^(k+i) * \text{exp } (- (l*x))) * \text{indicator } \{0$ 
 $..\} x) \ \partial \text{lborel})$ 
    by (intro nn-integral-cong)
      (auto simp: erlang-density-def power-mult-distrib power-add ennreal-mult[symmetric]
split: split-indicator)
  also have  $\dots = (l / (\text{fact } k * l^i)) * (\int^+ x. \text{ennreal } ((l*x)^(k+i) * \text{exp } (- (l*x)))$ 
 $* \text{indicator } \{0 ..\} x \ \partial \text{lborel})$ 
    by (intro nn-integral-cmult) auto
  also have  $\dots = \text{ennreal } (l / (\text{fact } k * l^i)) * ((1/l) * (\int^+ x. \text{ennreal } (x^(k+i) * \text{exp } (- x)) * \text{indicator } \{0 ..\} x \ \partial \text{lborel}))$ 
    by (subst nn-integral-real-affine[where c=1 / l and t=0]) (auto simp: field-simps eq)
  also have  $\dots = \text{fact } (k + i) / (\text{fact } k * l^i)$ 
    by (subst nn-integral-power-times-exp-Ici) (auto simp: ennreal-mult[symmetric])
  finally show ?thesis .
qed

```

lemma *prob-space-erlang-density:*

```

  assumes l[arith]:  $0 < l$ 
  shows prob-space (density lborel (erlang-density k l)) (is prob-space ?D)
proof
  show emeasure ?D (space ?D) = 1
    using nn-integral-erlang-ith-moment[OF l, where k=k and i=0] by (simp
add: emeasure-density)
qed

```

lemma (in *prob-space*) *erlang-distributed-le:*

```

  assumes D: distributed M lborel X (erlang-density k l)
  assumes [simp, arith]:  $0 < l \ 0 \leq a$ 
  shows  $\mathcal{P}(x \text{ in } M. X \ x \leq a) = \text{erlang-CDF } k \ l \ a$ 
proof -
  have emeasure M {x ∈ space M. X x ≤ a} = emeasure (distr M lborel X) {.. a}
    using distributed-measurable[OF D]
  by (subst emeasure-distr) (auto intro!: arg-cong2[where f=emeasure])
  also have  $\dots = \text{emeasure } (\text{density } \text{lborel } (\text{erlang-density } k \ l)) \ \{.. \ a\}$ 
    unfolding distributed-distr-eq-density[OF D] ..
  also have  $\dots = \text{erlang-CDF } k \ l \ a$ 
    by (auto intro!: emeasure-erlang-density)
  finally show ?thesis
    by (auto simp: emeasure-eq-measure measure-nonneg)
qed

```

lemma (in *prob-space*) *erlang-distributed-gt:*

```

assumes  $D[simp]$ : distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )
assumes  $[arith]$ :  $0 < l$   $0 \leq a$ 
shows  $\mathcal{P}(x \text{ in } M. a < X x) = 1 - (\text{erlang-CDF } k \ l \ a)$ 
proof –
  have  $1 - (\text{erlang-CDF } k \ l \ a) = 1 - \mathcal{P}(x \text{ in } M. X x \leq a)$  by (subst erlang-distributed-le) auto
  also have  $\dots = \text{prob}(\text{space } M - \{x \in \text{space } M. X x \leq a\})$ 
    using distributed-measurable[OF  $D$ ] by (auto simp: prob-compl)
  also have  $\dots = \mathcal{P}(x \text{ in } M. a < X x)$  by (auto intro!: arg-cong[where  $f=\text{prob}$ ] simp: not-le)
  finally show ?thesis by simp
qed

```

```

lemma erlang-CDF-at0: erlang-CDF  $k$   $l$   $0 = 0$ 
  by (induction  $k$ ) (auto simp: erlang-CDF-def)

```

```

lemma erlang-distributedI:
  assumes  $X[\text{measurable}]$ :  $X \in \text{borel-measurable } M$  and  $[arith]$ :  $0 < l$ 
  and  $X\text{-distr}$ :  $\bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{erlang-CDF } k \ l \ a$ 
  shows distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )
proof (rule distributedI-borel-atMost)
  fix  $a :: \text{real}$ 
  { assume  $a \leq 0$ 
    with  $X$  have  $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} \leq \text{emeasure } M \{x \in \text{space } M. X x \leq 0\}$ 
    by (intro emeasure-mono) auto
    also have  $\dots = 0$  by (auto intro!: erlang-CDF-at0 simp:  $X\text{-distr}$ [of 0])
    finally have  $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} \leq 0$  by simp
    then have  $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = 0$  by simp
  }
  note eq-0 = this

```

```

show  $(\int^+ x. \text{erlang-density } k \ l \ x * \text{indicator } \{..a\} x \ \partial \text{lborel}) = \text{ennreal}(\text{erlang-CDF } k \ l \ a)$ 
  using nn-integral-erlang-density[of  $l$   $k$   $a$ ]
  by (simp add: ennreal-indicator ennreal-mult)

```

```

show  $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal}(\text{erlang-CDF } k \ l \ a)$ 
  using  $X\text{-distr}$ [of  $a$ ] eq-0 by (auto simp: one-ennreal-def erlang-CDF-def)
qed simp-all

```

```

lemma (in prob-space) erlang-distributed-iff:
  assumes  $[arith]$ :  $0 < l$ 
  shows distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )  $\longleftrightarrow$ 
    ( $X \in \text{borel-measurable } M \wedge 0 < l \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = \text{erlang-CDF } k \ l \ a)$ )
  using
    distributed-measurable[of  $M$  lborel  $X$  erlang-density  $k$   $l$ ]

```

$\text{emeasure-erlang-density}$ [of l]
 $\text{erlang-distributed-le}$ [of X k l]
by (*auto intro!*: $\text{erlang-distributedI}$ *simp*: one-ennreal-def $\text{emeasure-eq-measure}$)

lemma (*in prob-space*) $\text{erlang-distributed-mult-const}$:
assumes erlX : $\text{distributed } M \text{ lborel } X$ ($\text{erlang-density } k$ l)
assumes $a\text{-pos}$ [*arith*]: $0 < \alpha$ $0 < l$
shows $\text{distributed } M \text{ lborel } (\lambda x. \alpha * X x)$ ($\text{erlang-density } k$ (l / α))
proof (*subst erlang-distributed-iff*, *safe*)
have [*measurable*]: $\text{random-variable borel } X$ **and** [*arith*]: $0 < l$
and [*simp*]: $\bigwedge a. 0 \leq a \implies \text{prob } \{x \in \text{space } M. X x \leq a\} = \text{erlang-CDF } k$ l a
by(*insert erlX*, *auto simp: erlang-distributed-iff*)

show $\text{random-variable borel } (\lambda x. \alpha * X x)$ $0 < l / \alpha$ $0 < l / \alpha$
by (*auto simp: field-simps*)

fix a :: *real* **assume** [*arith*]: $0 \leq a$
obtain b :: *real* **where** [*simp*, *arith*]: $b = a / \alpha$ **by** *blast*

have [*arith*]: $0 \leq b$ **by** (*auto simp: divide-nonneg-pos*)

have $\text{prob } \{x \in \text{space } M. \alpha * X x \leq a\} = \text{prob } \{x \in \text{space } M. X x \leq b\}$
by (*rule arg-cong[where f= prob]*) (*auto simp: field-simps*)

moreover have $\text{prob } \{x \in \text{space } M. X x \leq b\} = \text{erlang-CDF } k$ l b **by** *auto*
moreover have $\text{erlang-CDF } k$ (l / α) $a = \text{erlang-CDF } k$ l b **unfolding** erlang-CDF-def **by** *auto*
ultimately show $\text{prob } \{x \in \text{space } M. \alpha * X x \leq a\} = \text{erlang-CDF } k$ (l / α) a
by *fastforce*
qed

lemma (*in prob-space*) $\text{has-bochner-integral-erlang-ith-moment}$:
fixes k i :: *nat* **and** l :: *real*
assumes [*arith*]: $0 < l$ **and** D : $\text{distributed } M \text{ lborel } X$ ($\text{erlang-density } k$ l)
shows $\text{has-bochner-integral } M$ ($\lambda x. X x \wedge i$) ($\text{fact } (k + i) / (\text{fact } k * l \wedge i)$)
proof (*rule has-bochner-integral-nn-integral*)
show $\text{AE } x \text{ in } M. 0 \leq X x \wedge i$
by (*subst distributed-AE2[OF D]*) (*auto simp: erlang-density-def*)
show $(\int^+ x. \text{ennreal } (X x \wedge i) \partial M) = \text{ennreal } (\text{fact } (k + i) / (\text{fact } k * l \wedge i))$
using $\text{nn-integral-erlang-ith-moment}$ [of l k i]
by (*subst distributed-nn-integral[symmetric, OF D]*) (*auto simp: ennreal-mult'*)
qed (*insert distributed-measurable[OF D]*, *auto*)

lemma (*in prob-space*) $\text{erlang-ith-moment-integrable}$:
 $0 < l \implies \text{distributed } M \text{ lborel } X$ ($\text{erlang-density } k$ l) $\implies \text{integrable } M$ ($\lambda x. X x \wedge i$)
by *rule* (*rule has-bochner-integral-erlang-ith-moment*)

lemma (*in prob-space*) erlang-ith-moment :

$0 < l \implies \text{distributed } M \text{ lborel } X \text{ (erlang-density } k \text{ } l) \implies$
 $\text{expectation } (\lambda x. X x \wedge i) = \text{fact } (k + i) / (\text{fact } k * l \wedge i)$
by (rule has-bochner-integral-integral-eq) (rule has-bochner-integral-erlang-ith-moment)

lemma (in prob-space) erlang-distributed-variance:

assumes [arith]: $0 < l$ **and** distributed M lborel X (erlang-density k l)

shows variance $X = (k + 1) / l^2$

proof (subst variance-eq)

show integrable M X integrable M $(\lambda x. (X x)^2)$

using erlang-ith-moment-integrable[OF assms, of 1] erlang-ith-moment-integrable[OF assms, of 2]

by auto

show expectation $(\lambda x. (X x)^2) - (\text{expectation } X)^2 = \text{real } (k + 1) / l^2$

using erlang-ith-moment[OF assms, of 1] erlang-ith-moment[OF assms, of 2]

by simp (auto simp: power2-eq-square field-simps of-nat-Suc)

qed

13.2 Exponential distribution

abbreviation exponential-density :: real \Rightarrow real \Rightarrow real **where**

exponential-density \equiv erlang-density 0

lemma exponential-density-def:

exponential-density l $x = (\text{if } x < 0 \text{ then } 0 \text{ else } l * \exp(-x * l))$

by (simp add: fun-eq-iff erlang-density-def)

lemma erlang-CDF-0: erlang-CDF 0 l $a = (\text{if } 0 \leq a \text{ then } 1 - \exp(-l * a) \text{ else } 0)$

by (simp add: erlang-CDF-def)

lemma prob-space-exponential-density: $0 < l \implies \text{prob-space (density lborel (exponential-density } l))$

by (rule prob-space-erlang-density)

lemma (in prob-space) exponential-distributedD-le:

assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l : $0 < l$

shows $\mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(-a * l)$

using erlang-distributed-le[OF D l a] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributedD-gt:

assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l : $0 < l$

shows $\mathcal{P}(x \text{ in } M. a < X x) = \exp(-a * l)$

using erlang-distributed-gt[OF D l a] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributed-memoryless:

assumes D : distributed M lborel X (exponential-density l) **and** a : $0 \leq a$ **and** l :

$0 < l$ and $t: 0 \leq t$
shows $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. t < X x)$
proof –
have $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. a + t < X x) / \mathcal{P}(x \text{ in } M. a < X x)$
using $\langle 0 \leq t \rangle$ **by** (*auto simp: cond-prob-def intro!: arg-cong[where f=prob] arg-cong2[where f=(/)]*)
also have $\dots = \exp(- (a + t) * l) / \exp(- a * l)$
using $a t$ **by** (*simp add: exponential-distributedD-gt[OF D - l]*)
also have $\dots = \exp(- t * l)$
using l **by** (*auto simp: field-simps exp-add[symmetric]*)
finally show *?thesis*
using t **by** (*simp add: exponential-distributedD-gt[OF D - l]*)
qed

lemma *exponential-distributedI*:
assumes $X[\text{measurable}]$: $X \in \text{borel-measurable } M$ **and** $[\text{arith}]$: $0 < l$
and $X\text{-distr}$: $\bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = 1 - \exp(- a * l)$
shows $\text{distributed } M \text{ lborel } X$ (*exponential-density l*)
proof (*rule erlang-distributedI*)
fix $a :: \text{real}$ **assume** $0 \leq a$ **then show** $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal}(\text{erlang-CDF } 0 l a)$
using $X\text{-distr}[of a]$ **by** (*simp add: erlang-CDF-def ennreal-minus ennreal-1[symmetric] del: ennreal-1*)
qed fact+

lemma (*in prob-space*) *exponential-distributed-iff*:
assumes $0 < l$
shows $\text{distributed } M \text{ lborel } X$ (*exponential-density l*) \longleftrightarrow
 $(X \in \text{borel-measurable } M \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(- a * l)))$
using *assms erlang-distributed-iff[of l X 0]* **by** (*auto simp: erlang-CDF-0*)

lemma (*in prob-space*) *exponential-distributed-expectation*:
 $0 < l \implies \text{distributed } M \text{ lborel } X$ (*exponential-density l*) $\implies \text{expectation } X = 1 / l$
using *erlang-ith-moment[of l X 0 1]* **by** *simp*

lemma *exponential-density-nonneg*: $0 < l \implies 0 \leq \text{exponential-density } l x$
by (*auto simp: exponential-density-def*)

lemma (*in prob-space*) *exponential-distributed-min*:
assumes $0 < l$ $0 < u$
assumes $\text{exp}X$: $\text{distributed } M \text{ lborel } X$ (*exponential-density l*)
assumes $\text{exp}Y$: $\text{distributed } M \text{ lborel } Y$ (*exponential-density u*)
assumes *ind*: *indep-var borel X borel Y*
shows $\text{distributed } M \text{ lborel } (\lambda x. \min(X x) (Y x))$ (*exponential-density (l + u)*)
proof (*subst exponential-distributed-iff, safe*)

```

have randX: random-variable borel X
  using expX ⟨0 < l⟩ by (simp add: exponential-distributed-iff)
moreover have randY: random-variable borel Y
  using expY ⟨0 < u⟩ by (simp add: exponential-distributed-iff)
ultimately show random-variable borel (λx. min (X x) (Y x)) by auto

show 0 < l + u
  using ⟨0 < l⟩ ⟨0 < u⟩ by auto

fix a::real assume a[arith]: 0 ≤ a
have gt1[simp]:  $\mathcal{P}(x \text{ in } M. a < X x) = \exp(-a * l)$ 
  by (rule exponential-distributedD-gt[OF expX a]) fact
have gt2[simp]:  $\mathcal{P}(x \text{ in } M. a < Y x) = \exp(-a * u)$ 
  by (rule exponential-distributedD-gt[OF expY a]) fact

have  $\mathcal{P}(x \text{ in } M. a < (\min (X x) (Y x))) = \mathcal{P}(x \text{ in } M. a < (X x) \wedge a < (Y x))$ 
by (auto intro!: arg-cong[where f=prob])

also have ... =  $\mathcal{P}(x \text{ in } M. a < (X x)) * \mathcal{P}(x \text{ in } M. a < (Y x))$ 
  using prob-indep-random-variable[OF ind, of {a <..} {a <..}] by simp
also have ... =  $\exp(-a * (l + u))$  by (auto simp: field-simps mult-exp-exp)
finally have indep-prob:  $\mathcal{P}(x \text{ in } M. a < (\min (X x) (Y x))) = \exp(-a * (l + u))$  .

have  $\{x \in \text{space } M. (\min (X x) (Y x)) \leq a\} = (\text{space } M - \{x \in \text{space } M. a < (\min (X x) (Y x))\})$ 
  by auto
then have  $1 - \text{prob } \{x \in \text{space } M. a < (\min (X x) (Y x))\} = \text{prob } \{x \in \text{space } M. (\min (X x) (Y x)) \leq a\}$ 
  using randX randY by (auto simp: prob-compl)
then show  $\text{prob } \{x \in \text{space } M. (\min (X x) (Y x)) \leq a\} = 1 - \exp(-a * (l + u))$ 
  using indep-prob by auto
qed

lemma (in prob-space) exponential-distributed-Min:
  assumes finI: finite I
  assumes A:  $I \neq \{\}$ 
  assumes l:  $\bigwedge i. i \in I \implies 0 < l i$ 
  assumes expX:  $\bigwedge i. i \in I \implies \text{distributed } M \text{ lborel } (X i) \text{ (exponential-density } (l i))$ 
  assumes ind: indep-vars (λi. borel) X I
  shows distributed M lborel (λx. Min ((λi. X i x)‘I)) (exponential-density ( $\sum_{i \in I. l i}$ ))
  using assms
  proof (induct rule: finite-ne-induct)
    case (singleton i) then show ?case by simp
  next
    case (insert i I)

```


then have *distributed M lborel* $(\lambda x. \min (X \ i \ x) (Min ((\lambda i. X \ i \ x) 'T)))$ (*exponential-density*
 $(l \ i + (\sum_{i \in I}. l \ i))$)
by (*intro exponential-distributed-min indep-vars-Min insert*)
(auto intro: indep-vars-subset sum-pos)
then show *?case*
using *insert by simp*
qed

lemma (*in prob-space*) *exponential-distributed-variance*:
 $0 < l \implies \text{distributed M lborel } X \text{ (exponential-density } l) \implies \text{variance } X = 1 / l^2$
using *erlang-distributed-variance[of l X 0]* **by** *simp*

lemma *nn-integral-zero'*: $AE \ x \ \text{in } M. f \ x = 0 \implies (\int^+ x. f \ x \ \partial M) = 0$
by (*simp cong: nn-integral-cong-AE*)

lemma *convolution-erlang-density*:
fixes $k_1 \ k_2 :: \text{nat}$
assumes [*simp, arith*]: $0 < l$
shows $(\lambda x. \int^+ y. \text{ennreal (erlang-density } k_1 \ l \ (x - y)) * \text{ennreal (erlang-density } k_2 \ l \ y) \ \partial \text{lborel}) =$
 $(\text{erlang-density (Suc } k_1 + \text{Suc } k_2 - 1) \ l)$
(is ?LHS = ?RHS)

proof

fix $x :: \text{real}$
have $x \leq 0 \vee 0 < x$
by *arith*
then show *?LHS x = ?RHS x*

proof

assume $x \leq 0$ **then show** *?thesis*
apply (*subst nn-integral-zero'*)
apply (*rule AE-I[where N={0}]*)
apply (*auto simp add: erlang-density-def not-less*)
done

next

note *zero-le-mult-iff[simp] zero-le-divide-iff[simp]*

have *I-eq1: integral^N lborel (erlang-density (Suc k₁ + Suc k₂ - 1) l) = 1*
using *nn-integral-erlang-ith-moment[of l Suc k₁ + Suc k₂ - 1 0]* **by** (*simp*
del: fact-Suc)

have *1: ($\int^+ x. \text{ennreal (erlang-density (Suc } k_1 + \text{Suc } k_2 - 1) \ l \ x * \text{indicator } \{0 < ..\} \ x) \ \partial \text{lborel}) = 1$*
apply (*subst I-eq1[symmetric]*)
unfolding *erlang-density-def*
by (*auto intro!: nn-integral-cong split:split-indicator*)

have *prob-space (density lborel ?LHS)*

by (*intro prob-space-convolution-density*)
(auto intro!: prob-space-erlang-density erlang-density-nonneg)

```

then have 2:  $\text{integral}^N \text{lborel } ?LHS = 1$ 
  by (auto dest!: prob-space.emeasure-space-1 simp: emeasure-density)

let ?I = ( $\text{integral}^N \text{lborel } (\lambda y. \text{ennreal } ((1 - y)^{k_1} * y^{k_2} * \text{indicator } \{0..1\} y))$ )
let ?C = ( $\text{fact } (\text{Suc } (k_1 + k_2))$ ) / ( $(\text{fact } k_1) * (\text{fact } k_2)$ )
let ?s =  $\text{Suc } k_1 + \text{Suc } k_2 - 1$ 
let ?L = ( $\lambda x. \int^+ y. \text{ennreal } (\text{erlang-density } k_1 \text{ l } (x - y) * \text{erlang-density } k_2 \text{ l } y * \text{indicator } \{0..x\} y) \partial \text{lborel}$ )

{ fix x :: real assume [arith]:  $0 < x$ 
  have *:  $\bigwedge x y n. (x - y * x :: \text{real})^n = x^n * (1 - y)^n$ 
    unfolding power-mult-distrib[symmetric] by (simp add: field-simps)

  have ?LHS x = ?L x
    unfolding erlang-density-def
    by (auto intro!: nn-integral-cong simp: ennreal-mult split:split-indicator)
  also have ... = ( $\lambda x. \text{ennreal } ?C * ?I * \text{erlang-density } ?s \text{ l } x$ ) x
    apply (subst nn-integral-real-affine[where c=x and t = 0])
  apply (simp-all add: nn-integral-cmult[symmetric] nn-integral-multc[symmetric]
del: fact-Suc)
    apply (intro nn-integral-cong)
    apply (auto simp add: erlang-density-def mult-less-0-iff exp-minus field-simps
exp-diff power-add *
      ennreal-mult[symmetric]
      simp del: fact-Suc split: split-indicator)
  done
  finally have ( $\int^+ y. \text{ennreal } (\text{erlang-density } k_1 \text{ l } (x - y) * \text{erlang-density } k_2 \text{ l } y) \partial \text{lborel}$ ) =
    ( $\lambda x. \text{ennreal } ?C * ?I * \text{erlang-density } ?s \text{ l } x$ ) x
    by (simp add: ennreal-mult) }
note * = this

assume [arith]:  $0 < x$ 
have 3:  $1 = \text{integral}^N \text{lborel } (\lambda xa. ?LHS xa * \text{indicator } \{0<..\} xa)$ 
  by (subst 2[symmetric])
    (auto intro!: nn-integral-cong-AE AE-I[where N={0}]
      simp: erlang-density-def nn-integral-multc[symmetric] indicator-def
split: if-split-asm)
  also have ... =  $\text{integral}^N \text{lborel } (\lambda x. (\text{ennreal } (?C) * ?I) * ((\text{erlang-density } ?s \text{ l } x) * \text{indicator } \{0<..\} x))$ 
  by (auto intro!: nn-integral-cong simp: ennreal-mult[symmetric] * split: split-indicator)
  also have ... =  $\text{ennreal } (?C) * ?I$ 
    using 1
    by (auto simp: nn-integral-cmult)
  finally have  $\text{ennreal } (?C) * ?I = 1$  by presburger

then show ?thesis
  using * by (simp add: ennreal-mult)

```

qed
qed

lemma (in *prob-space*) *sum-indep-erlang*:
assumes *indep*: *indep-var borel X borel Y*
assumes [*simp, arith*]: $0 < l$
assumes *erlX*: *distributed M lborel X (erlang-density k₁ l)*
assumes *erlY*: *distributed M lborel Y (erlang-density k₂ l)*
shows *distributed M lborel* $(\lambda x. X x + Y x)$ (*erlang-density (Suc k₁ + Suc k₂ - 1) l*)
using *assms*
apply (*subst convolution-erlang-density[symmetric, OF ‹0 < l›*)
apply (*intro distributed-convolution*)
apply *auto*
done

lemma (in *prob-space*) *erlang-distributed-sum*:
assumes *finI*: *finite I*
assumes *A*: $I \neq \{\}$
assumes [*simp, arith*]: $0 < l$
assumes *expX*: $\bigwedge i. i \in I \implies$ *distributed M lborel* $(X i)$ (*erlang-density (k i) l*)
assumes *ind*: *indep-vars* $(\lambda i. \text{borel}) X I$
shows *distributed M lborel* $(\lambda x. \sum_{i \in I}. X i x)$ (*erlang-density* $((\sum_{i \in I}. \text{Suc } (k i)) - 1) l$)
using *assms*
proof (*induct rule: finite-ne-induct*)
case (*singleton i*) **then show** *?case by auto*
next
case (*insert i I*)
then have *distributed M lborel* $(\lambda x. (X i x) + (\sum_{i \in I}. X i x))$ (*erlang-density* $(\text{Suc } (k i) + \text{Suc } ((\sum_{i \in I}. \text{Suc } (k i)) - 1) - 1) l$)
by (*intro sum-indep-erlang indep-vars-sum*) (*auto intro!: indep-vars-subset*)
also have $(\lambda x. (X i x) + (\sum_{i \in I}. X i x)) = (\lambda x. \sum_{i \in \text{insert } i I}. X i x)$
using *insert by auto*
also have $\text{Suc } (k i) + \text{Suc } ((\sum_{i \in I}. \text{Suc } (k i)) - 1) - 1 = (\sum_{i \in \text{insert } i I}. \text{Suc } (k i)) - 1$
using *insert by (auto intro!: Suc-pred simp: ac-simps)*
finally show *?case by fast*
qed

lemma (in *prob-space*) *exponential-distributed-sum*:
assumes *finI*: *finite I*
assumes *A*: $I \neq \{\}$
assumes *l*: $0 < l$
assumes *expX*: $\bigwedge i. i \in I \implies$ *distributed M lborel* $(X i)$ (*exponential-density l*)
assumes *ind*: *indep-vars* $(\lambda i. \text{borel}) X I$
shows *distributed M lborel* $(\lambda x. \sum_{i \in I}. X i x)$ (*erlang-density* $((\text{card } I) - 1) l$)
using *erlang-distributed-sum[OF assms]* **by** *simp*

lemma (in *information-space*) *entropy-exponential*:
assumes l [simp, arith]: $0 < l$
assumes D : distributed M lborel X (exponential-density l)
shows entropy b lborel $X = \log b$ (exp 1 / l)
proof –
have [simp]: integrable lborel (exponential-density l)
using distributed-integrable[OF D , of $\lambda\cdot. 1$] **by** simp

have [simp]: integral ^{L} lborel (exponential-density l) = 1
using distributed-integral[OF D , of $\lambda\cdot. 1$] **by** (simp add: prob-space)

have [simp]: integrable lborel ($\lambda x.$ exponential-density l $x * x$)
using erlang-ith-moment-integrable[OF l D , of 1] distributed-integrable[OF D , of $\lambda x. x$] **by** simp

have [simp]: integral ^{L} lborel ($\lambda x.$ exponential-density l $x * x$) = 1 / l
using erlang-ith-moment[OF l D , of 1] distributed-integral[OF D , of $\lambda x. x$] **by** simp

have entropy b lborel $X = - (\int x.$ exponential-density l $x * \log b$ (exponential-density l x) ∂ lborel)
using D **by** (rule entropy-distr) simp
also have $(\int x.$ exponential-density l $x * \log b$ (exponential-density l x) ∂ lborel)
= $(\int x.$ (ln $l * \text{exponential-density } l \ x - l * (\text{exponential-density } l \ x * x)) / \ln b$ ∂ lborel)
by (intro Bochner-Integration.integral-cong) (auto simp: log-def ln-mult exponential-density-def field-simps)
also have ... = (ln $l - 1$) / ln b
by simp
finally show ?thesis
by (simp add: log-def ln-div) (simp add: field-split-simps)
qed

13.3 Uniform distribution

lemma *uniform-distrI*:
assumes X : $X \in \text{measurable } M \ M'$
and A : $A \in \text{sets } M' \ \text{emeasure } M' \ A \neq \infty \ \text{emeasure } M' \ A \neq 0$
assumes *distr*: $\bigwedge B. B \in \text{sets } M' \implies \text{emeasure } M \ (X \text{ -' } B \cap \text{space } M) = \text{emeasure } M' \ (A \cap B) / \text{emeasure } M' \ A$
shows *distr* $M \ M' \ X = \text{uniform-measure } M' \ A$
unfolding *uniform-measure-def*
proof (intro *measure-eqI*)
let ? $f = \lambda x.$ indicator $A \ x / \text{emeasure } M' \ A$
fix B **assume** B : $B \in \text{sets } (distr \ M \ M' \ X)$
with X **have** $\text{emeasure } M \ (X \text{ -' } B \cap \text{space } M) = \text{emeasure } M' \ (A \cap B) / \text{emeasure } M' \ A$
by (simp add: *distr*[of B] *measurable-sets*)

also have $\dots = (1 / \text{emeasure } M' A) * \text{emeasure } M' (A \cap B)$
by (*simp add: divide-ennreal-def ac-simps*)
also have $\dots = (\int^+ x. (1 / \text{emeasure } M' A) * \text{indicator } (A \cap B) x \partial M')$
using $A B$
by (*intro nn-integral-cmult-indicator[symmetric]*) (*auto intro!*:)
also have $\dots = (\int^+ x. ?f x * \text{indicator } B x \partial M')$
by (*rule nn-integral-cong*) (*auto split: split-indicator*)
finally show $\text{emeasure } (\text{distr } M M' X) B = \text{emeasure } (\text{density } M' ?f) B$
using $A B X$ **by** (*auto simp add: emeasure-distr emeasure-density*)
qed simp

lemma *uniform-distrI-borel*:

fixes $A :: \text{real set}$
assumes $X[\text{measurable}]$: $X \in \text{borel-measurable } M$ **and** A : $\text{emeasure } \text{lborel } A = \text{ennreal } r$ $0 < r$
and $[\text{measurable}]$: $A \in \text{sets borel}$
assumes distr : $\bigwedge a. \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{emeasure } \text{lborel } (A \cap \{.. a\}) / r$
shows $\text{distributed } M \text{ lborel } X (\lambda x. \text{indicator } A x / \text{measure } \text{lborel } A)$
proof (*rule distributedI-borel-atMost*)
let $?f = \lambda x. 1 / r * \text{indicator } A x$
fix a
have $\text{emeasure } \text{lborel } (A \cap \{..a\}) \leq \text{emeasure } \text{lborel } A$
using A **by** (*intro emeasure-mono*) *auto*
also have $\dots < \infty$
using A **by** *simp*
finally have fin : $\text{emeasure } \text{lborel } (A \cap \{..a\}) \neq \text{top}$
by *simp*
from *emeasure-eq-ennreal-measure[OF this]*
have fin-eq : $\text{emeasure } \text{lborel } (A \cap \{..a\}) / r = \text{ennreal } (\text{measure } \text{lborel } (A \cap \{..a\}) / r)$
using A **by** (*simp add: divide-ennreal measure-nonneg*)
then show $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal } (\text{measure } \text{lborel } (A \cap \{..a\}) / r)$
using distr **by** *simp*

have $(\int^+ x. \text{ennreal } (\text{indicator } A x / \text{measure } \text{lborel } A * \text{indicator } \{..a\} x) \partial \text{lborel}) =$
 $(\int^+ x. \text{ennreal } (1 / \text{measure } \text{lborel } A) * \text{indicator } (A \cap \{..a\}) x \partial \text{lborel})$
by (*auto intro! nn-integral-cong split: split-indicator*)
also have $\dots = \text{ennreal } (1 / \text{measure } \text{lborel } A) * \text{emeasure } \text{lborel } (A \cap \{..a\})$
using $\langle A \in \text{sets borel} \rangle$
by (*intro nn-integral-cmult-indicator*) (*auto simp: measure-nonneg*)
also have $\dots = \text{ennreal } (\text{measure } \text{lborel } (A \cap \{..a\}) / r)$
unfolding *emeasure-eq-ennreal-measure[OF fin]* **using** A
by (*simp add: measure-def ennreal-mult[symmetric]*)
finally show $(\int^+ x. \text{ennreal } (\text{indicator } A x / \text{measure } \text{lborel } A * \text{indicator } \{..a\} x) \partial \text{lborel}) =$
 $\text{ennreal } (\text{measure } \text{lborel } (A \cap \{..a\}) / r) .$

qed (*auto simp: measure-nonneg*)

lemma (*in prob-space*) *uniform-distrI-borel-atLeastAtMost*:

fixes $a\ b :: \text{real}$

assumes $X: X \in \text{borel-measurable } M$ **and** $a < b$

assumes $\text{distr}: \bigwedge t. a \leq t \implies t \leq b \implies \mathcal{P}(x \text{ in } M. X\ x \leq t) = (t - a) / (b - a)$

shows $\text{distributed } M \text{ lborel } X (\lambda x. \text{indicator } \{a..b\} x / \text{measure lborel } \{a..b\})$

proof (*rule uniform-distrI-borel*)

fix t

have $t < a \vee (a \leq t \wedge t \leq b) \vee b < t$

by *auto*

then show $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = \text{emeasure lborel } (\{a..b\} \cap \{..t\}) / (b - a)$

proof (*elim disjE conjE*)

assume $t < a$

then have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} \leq \text{emeasure } M \{x \in \text{space } M. X\ x \leq a\}$

using X **by** (*auto intro!: emeasure-mono measurable-sets*)

also have $\dots = 0$

using $\text{distr}[of\ a] \langle a < b \rangle$ **by** (*simp add: emeasure-eq-measure*)

finally have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = 0$

by (*simp add: antisym measure-nonneg*)

with $\langle t < a \rangle$ **show** *?thesis* **by** *simp*

next

assume $\text{bnds}: a \leq t \leq b$

have $\{a..b\} \cap \{..t\} = \{a..t\}$

using bnds **by** *auto*

then show *?thesis* **using** $\langle a \leq t \rangle \langle a < b \rangle$

using $\text{distr}[OF\ \text{bnds}]$ **by** (*simp add: emeasure-eq-measure divide-ennreal*)

next

assume $b < t$

have $1 = \text{emeasure } M \{x \in \text{space } M. X\ x \leq b\}$

using $\text{distr}[of\ b] \langle a < b \rangle$ **by** (*simp add: one-ennreal-def emeasure-eq-measure*)

also have $\dots \leq \text{emeasure } M \{x \in \text{space } M. X\ x \leq t\}$

using $X \langle b < t \rangle$ **by** (*auto intro!: emeasure-mono measurable-sets*)

finally have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = 1$

by (*simp add: antisym emeasure-eq-measure*)

with $\langle b < t \rangle \langle a < b \rangle$ **show** *?thesis* **by** (*simp add: measure-def divide-ennreal*)

qed

qed (*insert X \langle a < b \rangle, auto*)

lemma (*in prob-space*) *uniform-distributed-measure*:

fixes $a\ b :: \text{real}$

assumes $D: \text{distributed } M \text{ lborel } X (\lambda x. \text{indicator } \{a..b\} x / \text{measure lborel } \{a..b\})$

assumes $t: a \leq t \leq b$

shows $\mathcal{P}(x \text{ in } M. X\ x \leq t) = (t - a) / (b - a)$

proof –

```

have emeasure  $M \{x \in \text{space } M. X x \leq t\} = \text{emeasure } (\text{distr } M \text{ lborel } X) \{.. t\}$ 
  using distributed-measurable[OF D]
  by (subst emeasure-distr) (auto intro!: arg-cong2[where  $f = \text{emeasure}$ ])
also have  $\dots = \int^+ x. \text{ennreal } (1 / (b - a)) * \text{indicator } \{a .. t\} x \partial \text{lborel}$ 
  using distributed-borel-measurable[OF D]  $\langle a \leq t \rangle \langle t \leq b \rangle$ 
  unfolding distributed-distr-eq-density[OF D]
  by (subst emeasure-density)
    (auto intro!: nn-integral-cong simp: measure-def split: split-indicator)
also have  $\dots = \text{ennreal } (1 / (b - a)) * (t - a)$ 
  using  $\langle a \leq t \rangle \langle t \leq b \rangle$ 
  by (subst nn-integral-cmult-indicator) auto
finally show ?thesis
  using  $t$  by (simp add: emeasure-eq-measure ennreal-mult''[symmetric] measure-nonneg)
qed

```

```

lemma (in prob-space) uniform-distributed-bounds:
  fixes  $a b :: \text{real}$ 
  assumes  $D$ : distributed  $M \text{ lborel } X (\lambda x. \text{indicator } \{a .. b\} x / \text{measure } \text{lborel } \{a .. b\})$ 
  shows  $a < b$ 
proof (rule ccontr)
  assume  $\neg a < b$ 
  then have  $\{a .. b\} = \{\} \vee \{a .. b\} = \{a .. a\}$  by simp
  with uniform-distributed-params[OF D] show False
  by (auto simp: measure-def)
qed

```

```

lemma (in prob-space) uniform-distributed-iff:
  fixes  $a b :: \text{real}$ 
  shows distributed  $M \text{ lborel } X (\lambda x. \text{indicator } \{a..b\} x / \text{measure } \text{lborel } \{a..b\}) \longleftrightarrow$ 
    ( $X \in \text{borel-measurable } M \wedge a < b \wedge (\forall t \in \{a .. b\}. \mathcal{P}(x \text{ in } M. X x \leq t) = (t - a) / (b - a))$ )
  using
    uniform-distributed-bounds[of X a b]
    uniform-distributed-measure[of X a b]
    distributed-measurable[of M lborel X]
  by (auto intro!: uniform-distrI-borel-atLeastAtMost simp del: content-real-if)

```

```

lemma (in prob-space) uniform-distributed-expectation:
  fixes  $a b :: \text{real}$ 
  assumes  $D$ : distributed  $M \text{ lborel } X (\lambda x. \text{indicator } \{a .. b\} x / \text{measure } \text{lborel } \{a .. b\})$ 
  shows expectation  $X = (a + b) / 2$ 
proof (subst distributed-integral[OF D, of  $\lambda x. x$ , symmetric])
  have  $a < b$ 
  using uniform-distributed-bounds[OF D] .

```

```

have  $(\int x. \text{indicator } \{a .. b\} x / \text{measure } \text{lborel } \{a .. b\} * x \partial \text{lborel}) =$ 

```

```

  (∫ x. (x / measure lborel {a .. b}) * indicator {a .. b} x ∂lborel)
  by (intro Bochner-Integration.integral-cong) auto
  also have (∫ x. (x / measure lborel {a .. b}) * indicator {a .. b} x ∂lborel) =
(a + b) / 2
  proof (subst integral-FTC-Icc-real)
  fix x
  show DERIV (λx. x2 / (2 * measure lborel {a..b})) x :> x / measure lborel
{a..b}
  using uniform-distributed-params[OF D]
  by (auto intro!: derivative-eq-intros simp del: content-real-if)
  show isCont (λx. x / Sigma-Algebra.measure lborel {a..b}) x
  using uniform-distributed-params[OF D]
  by (auto intro!: isCont-divide)
  have *: b2 / (2 * measure lborel {a..b}) - a2 / (2 * measure lborel {a..b}) =
(b*b - a * a) / (2 * (b - a))
  using ⟨a < b⟩
  by (auto simp: measure-def power2-eq-square diff-divide-distrib[symmetric])
  show b2 / (2 * measure lborel {a..b}) - a2 / (2 * measure lborel {a..b}) = (a
+ b) / 2
  using ⟨a < b⟩
  unfolding * square-diff-square-factored by (auto simp: field-simps)
  qed (insert ⟨a < b⟩, simp)
  finally show (∫ x. indicator {a .. b} x / measure lborel {a .. b} * x ∂lborel) =
(a + b) / 2 .
  qed (auto simp: measure-nonneg)

```

lemma (in prob-space) uniform-distributed-variance:

```

  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
.. b})
  shows variance X = (b - a)2 / 12
  proof (subst distributed-variance)
  have [arith]: a < b using uniform-distributed-bounds[OF D] .
  let ?μ = expectation X let ?D = λx. indicator {a..b} (x + ?μ) / measure lborel
{a..b}
  have (∫ x. x2 * (?D x) ∂lborel) = (∫ x. x2 * (indicator {a - ?μ .. b - ?μ} x) /
measure lborel {a .. b} ∂lborel)
  by (intro Bochner-Integration.integral-cong) (auto split: split-indicator)
  also have ... = (b - a)2 / 12
  by (simp add: integral-power uniform-distributed-expectation[OF D])
  (simp add: eval-nat-numeral field-simps )
  finally show (∫ x. x2 * ?D x ∂lborel) = (b - a)2 / 12 .
  qed (auto intro: D simp del: content-real-if)

```

13.4 Normal distribution

definition normal-density :: real ⇒ real ⇒ real ⇒ real **where**

normal-density μ σ x = 1 / sqrt (2 * pi * σ²) * exp (-(x - μ)² / (2 * σ²))

abbreviation *std-normal-density* :: *real* \Rightarrow *real* **where**
std-normal-density \equiv *normal-density* 0 1

lemma *std-normal-density-def*: *std-normal-density* $x = (1 / \text{sqrt } (2 * \text{pi})) * \text{exp}$
 $(- x^2 / 2)$
unfolding *normal-density-def* **by** *simp*

lemma *normal-density-nonneg*[*simp*]: $0 \leq \text{normal-density } \mu \sigma x$
by (*auto simp: normal-density-def*)

lemma *normal-density-pos*: $0 < \sigma \implies 0 < \text{normal-density } \mu \sigma x$
by (*auto simp: normal-density-def*)

lemma *borel-measurable-normal-density*[*measurable*]: *normal-density* $\mu \sigma \in$ *borel-measurable*
borel
by (*auto simp: normal-density-def*[*abs-def*])

lemma *gaussian-moment-0*:

has-bochner-integral lborel ($\lambda x. \text{indicator } \{0..\} x *_{\mathbb{R}} \text{exp } (- x^2)$) (*sqrt pi / 2*)

proof –

let *?pI* = $\lambda f. (\int^{+s}. f (s::\text{real}) * \text{indicator } \{0..\} s \partial \text{lborel})$

let *?gauss* = $\lambda x. \text{exp } (- x^2)$

let *?I* = *indicator* $\{0<..\} :: \text{real} \Rightarrow \text{real}$

let *?ff* = $\lambda x s. x * \text{exp } (- x^2 * (1 + s^2)) :: \text{real}$

have $*$: *?pI* *?gauss* = $(\int^{+x}. ?gauss x * ?I x \partial \text{lborel})$

by (*intro nn-integral-cong-AE AE-I*[**where** $N=\{0\}$]) (*auto split: split-indicator*)

have *?pI* *?gauss* $*$ *?pI* *?gauss* = $(\int^{+x}. \int^{+s}. ?gauss x * ?gauss s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel})$

by (*auto simp: nn-integral-cmult*[*symmetric*] *nn-integral-multc*[*symmetric*] $*$
ennreal-mult[*symmetric*])

intro!: *nn-integral-cong split: split-indicator*)

also have $\dots = (\int^{+x}. \int^{+s}. ?ff x s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel})$

proof (*rule nn-integral-cong, cases*)

fix $x :: \text{real}$ **assume** $x \neq 0$

then show $(\int^{+s}. ?gauss x * ?gauss s * ?I s * ?I x \partial \text{lborel}) = (\int^{+s}. ?ff x s * ?I s * ?I x \partial \text{lborel})$

by (*subst nn-integral-real-affine*[**where** $t=0$ **and** $c=x$])

(*auto simp: mult-exp-exp nn-integral-cmult*[*symmetric*] *field-simps zero-less-mult-iff*
ennreal-mult[*symmetric*])

intro!: *nn-integral-cong split: split-indicator*)

qed *simp*

also have $\dots = \int^{+s}. \int^{+x}. ?ff x s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel}$

by (*rule lborel-pair.Fubini'*[*symmetric*]) *auto*

also have $\dots = ?pI (\lambda s. ?pI (\lambda x. ?ff x s))$

by (*rule nn-integral-cong-AE*)

(*auto intro!*: *nn-integral-cong-AE AE-I*[**where** $N=\{0\}$] *split: split-indicator-asm*)

```

also have ... = ?pI (λs. ennreal (1 / (2 * (1 + s2))))
proof (intro nn-integral-cong ennreal-mult-right-cong)
  fix s :: real show ?pI (λx. ?ff x s) = ennreal (1 / (2 * (1 + s2)))
  proof (subst nn-integral-FTC-atLeast)
    have ((λa. - (exp (- (a2 * (1 + s2))) / (2 + 2 * s2))) → (- (0 / (2 +
2 * s2)))) at-top
    apply (intro tendsto-intros filterlim-compose[OF exp-at-bot] filterlim-compose[OF
filterlim-uminus-at-bot-at-top])
      apply (subst mult.commute)
      apply (auto intro!: filterlim-tendsto-pos-mult-at-top
        filterlim-at-top-mult-at-top[OF filterlim-ident filterlim-ident]
        simp: add-pos-nonneg power2-eq-square add-nonneg-eq-0-iff)
    done
  then show ((λa. - (exp (- a2 - s2 * a2) / (2 + 2 * s2))) → 0) at-top
  by (simp add: field-simps)
qed (auto intro!: derivative-eq-intros simp: field-simps add-nonneg-eq-0-iff)
qed
also have ... = ennreal (pi / 4)
proof (subst nn-integral-FTC-atLeast)
  show ((λa. arctan a / 2) → (pi / 2) / 2) at-top
  by (intro tendsto-intros) (simp-all add: tendsto-arctan-at-top)
qed (auto intro!: derivative-eq-intros simp: add-nonneg-eq-0-iff field-simps power2-eq-square)
finally have ?pI ?gauss2 = pi / 4
  by (simp add: power2-eq-square)
then have ?pI ?gauss = sqrt (pi / 4)
  using power-eq-iff-eq-base[of 2 enn2real (?pI ?gauss) sqrt (pi / 4)]
  by (cases ?pI ?gauss) (auto simp: power2-eq-square ennreal-mult[symmetric]
ennreal-top-mult)
also have ?pI ?gauss = (∫+x. indicator {0..} x *R exp (- x2) ∂lborel)
  by (intro nn-integral-cong) (simp split: split-indicator)
also have sqrt (pi / 4) = sqrt pi / 2
  by (simp add: real-sqrt-divide)
finally show ?thesis
  by (rule has-bochner-integral-nn-integral[rotated 3])
    auto
qed

```

lemma gaussian-moment-1:

*has-bochner-integral lborel (λx::real. indicator {0..} x *_R (exp (- x²) * x)) (1 / 2)*

proof –

have (∫⁺x. indicator {0..} x *_R (exp (- x²) * x) ∂lborel) =
(∫⁺x. ennreal (x * exp (- x²)) * indicator {0..} x ∂lborel)

by (intro nn-integral-cong)
(auto simp: ac-simps split: split-indicator)

also have ... = ennreal (0 - (- exp (- 0²) / 2))

proof (rule nn-integral-FTC-atLeast)

have ((λx::real. - exp (- x²) / 2) → - 0 / 2) at-top

by (intro tendsto-divide tendsto-minus filterlim-compose[OF exp-at-bot])

```

      filterlim-compose[OF filterlim-uminus-at-bot-at-top]
      filterlim-pow-at-top filterlim-ident)
    auto
  then show (( $\lambda a::real. - \exp(-a^2) / 2$ )  $\longrightarrow 0$ ) at-top
    by simp
  qed (auto intro!: derivative-eq-intros)
  also have ... = ennreal (1 / 2)
    by simp
  finally show ?thesis
    by (rule has-bochner-integral-nn-integral[rotated 3])
      (auto split: split-indicator)
qed

lemma
  fixes k :: nat
  shows gaussian-moment-even-pos:
    has-bochner-integral lborel ( $\lambda x::real. \text{indicator } \{0..\} x *_R (\exp(-x^2)*x^{2 * k})$ )
      (( $\sqrt{\pi} / 2$ ) * ( $\text{fact } (2 * k) / (2^{2 * k} * \text{fact } k)$ ))
      (is ?even)
  and gaussian-moment-odd-pos:
    has-bochner-integral lborel ( $\lambda x::real. \text{indicator } \{0..\} x *_R (\exp(-x^2)*x^{2 * k + 1})$ ) ( $\text{fact } k / 2$ )
      (is ?odd)
  proof -
    let ?M =  $\lambda k x. \exp(-x^2) * x^k :: real$ 

    { fix k I assume Mk: has-bochner-integral lborel ( $\lambda x. \text{indicator } \{0..\} x *_R ?M k$ 
    x) I
      have  $2 \neq 0 :: real$ 
        by linarith
      let ?f =  $\lambda b. \int x. \text{indicator } \{0..\} x *_R ?M (k + 2) x * \text{indicator } \{..b\} x \partial \text{lborel}$ 
      have (( $\lambda b. (k + 1) / 2 * (\int x. \text{indicator } \{..b\} x *_R (\text{indicator } \{0..\} x *_R ?M$ 
      k x)  $\partial \text{lborel}) - ?M (k + 1) b / 2$ )  $\longrightarrow$ 
      ( $(k + 1) / 2 * (\int x. \text{indicator } \{0..\} x *_R ?M k x \partial \text{lborel}) - 0 / 2$ ) at-top
      (is ?tendsto)
      proof (intro tendsto-intros <math>2 \neq 0</math> tendsto-integral-at-top sets-lborel Mk[THEN
      integrable.intros])
        show ( $?M (k + 1) \longrightarrow 0$ ) at-top
      proof cases
        assume even k
          have (( $\lambda x. ((x^2)^{k \text{ div } 2 + 1} / \exp(x^2)) * (1 / x) :: real$ )  $\longrightarrow 0 * 0$ )
            at-top
          by (intro tendsto-intros tendsto-divide-0[OF tendsto-const] filterlim-compose[OF
            tendsto-power-div-exp-0]
            filterlim-at-top-imp-at-infinity filterlim-ident filterlim-pow-at-top
            filterlim-ident)
            auto
          also have ( $\lambda x. ((x^2)^{k \text{ div } 2 + 1} / \exp(x^2)) * (1 / x) :: real$ ) = ?M (k

```

+ 1)

using $\langle \text{even } k \rangle$ **by** (*auto simp: fun-eq-iff exp-minus field-simps power2-eq-square power-mult elim: evenE*)

finally show *?thesis* **by** *simp*

next

assume *odd k*

have $((\lambda x. ((x^2) \wedge (k - 1) \text{ div } 2 + 1) / \exp(x^2)) :: \text{real}) \longrightarrow 0)$ *at-top*

by (*intro filterlim-compose[OF tendsto-power-div-exp-0] filterlim-at-top-imp-at-infinity filterlim-ident filterlim-pow-at-top*)

auto

also have $(\lambda x. ((x^2) \wedge (k - 1) \text{ div } 2 + 1) / \exp(x^2)) :: \text{real} = ?M(k + 1)$

using $\langle \text{odd } k \rangle$ **by** (*auto simp: fun-eq-iff exp-minus field-simps power2-eq-square power-mult elim: oddE*)

finally show *?thesis* **by** *simp*

qed

qed

also have $?tendsto \longleftrightarrow ((?f \longrightarrow (k + 1) / 2 * (\int x. \text{indicator } \{0.. \} x *_R ?M k x \partial \text{lborel}) - 0 / 2) \text{ at-top})$

proof (*intro filterlim-cong refl eventually-at-top-linorder[THEN iffD2] exI[of - 0] allI impI*)

fix $b :: \text{real}$ **assume** $b: 0 \leq b$

have $\text{Suc } k * (\int x. \text{indicator } \{0..b\} x *_R ?M k x \partial \text{lborel}) = (\int x. \text{indicator } \{0..b\} x *_R (\exp(-x^2) * ((\text{Suc } k) * x \wedge k)) \partial \text{lborel})$

unfolding *integral-mult-right-zero[symmetric]* **by** (*intro Bochner-Integration.integral-cong*)

auto

also have $\dots = \exp(-b^2) * b \wedge (\text{Suc } k) - \exp(-0^2) * 0 \wedge (\text{Suc } k) -$
 $(\int x. \text{indicator } \{0..b\} x *_R (-2 * x * \exp(-x^2) * x \wedge (\text{Suc } k)) \partial \text{lborel})$

by (*rule integral-by-parts'*)

(auto intro!: derivative-eq-intros b
simp: diff-Suc of-nat-Suc field-simps split: nat.split)

also have $\dots = \exp(-b^2) * b \wedge (\text{Suc } k) - (\int x. \text{indicator } \{0..b\} x *_R (-2 * (\exp(-x^2) * x \wedge (k + 2))) \partial \text{lborel})$

by (*auto simp: intro!: Bochner-Integration.integral-cong*)

also have $\dots = \exp(-b^2) * b \wedge (\text{Suc } k) + 2 * (\int x. \text{indicator } \{0..b\} x *_R ?M(k + 2) x \partial \text{lborel})$

unfolding *Bochner-Integration.integral-mult-right-zero [symmetric]*

by (*simp del: real-scaleR-def*)

finally have $\text{Suc } k * (\int x. \text{indicator } \{0..b\} x *_R ?M k x \partial \text{lborel}) =$
 $\exp(-b^2) * b \wedge (\text{Suc } k) + 2 * (\int x. \text{indicator } \{0..b\} x *_R ?M(k + 2) x \partial \text{lborel})$.

then show $(k + 1) / 2 * (\int x. \text{indicator } \{..b\} x *_R (\text{indicator } \{0.. \} x *_R ?M k x) \partial \text{lborel}) - \exp(-b^2) * b \wedge (k + 1) / 2 = ?f b$

by (*simp add: field-simps atLeastAtMost-def indicator-inter-arith*)

qed

finally have *int-M-at-top*: $((?f \longrightarrow (k + 1) / 2 * (\int x. \text{indicator } \{0.. \} x *_R ?M k x \partial \text{lborel})) \text{ at-top})$

by *simp*

have *has-bochner-integral lborel* $(\lambda x. \text{indicator } \{0.. \} x *_R ?M(k + 2) x) ((k$

```

+ 1) / 2 * I)
  proof (rule has-bochner-integral-monotone-convergence-at-top)
    fix y :: real
    have *: (λx. indicator {0..} x *R ?M (k + 2) x * indicator {..y} x :: real) =
      (λx. indicator {0..y} x *R ?M (k + 2) x)
    by rule (simp split: split-indicator)
    show integrable lborel (λx. indicator {0..} x *R (?M (k + 2) x) * indicator
{..y} x :: real)
      unfolding * by (rule borel-integrable-compact) (auto intro!: continu-
ous-intros)
    show ((?f → (k + 1) / 2 * I) at-top)
      using int-M-at-top has-bochner-integral-integral-eq[OF Mk] by simp
    qed (auto split: split-indicator) }
  note step = this

show ?even
proof (induct k)
  case (Suc k)
  note step[OF this]
  also have (real (2 * k + 1) / 2 * (sqrt pi / 2 * ((fact (2 * k)) / ((2 :: real) ^ (2 * k)
* fact k)))) =
    sqrt pi / 2 * ((fact (2 * Suc k)) / ((2 :: real) ^ (2 * Suc k) * fact (Suc k)))
  apply (simp add: field-simps del: fact-Suc)
  apply (simp add: of-nat-mult field-simps)
  done
  finally show ?case
    by simp
qed (insert gaussian-moment-0, simp)

show ?odd
proof (induct k)
  case (Suc k)
  note step[OF this]
  also have (real (2 * k + 1 + 1) / (2 :: real) * ((fact k) / 2)) = (fact (Suc k))
/ 2
  by (simp add: field-simps of-nat-Suc field-split-simps del: fact-Suc) (simp add:
field-simps)
  finally show ?case
    by simp
qed (insert gaussian-moment-1, simp)
qed

context
  fixes k :: nat and μ σ :: real assumes [arith]: 0 < σ
begin

lemma normal-moment-even:
  has-bochner-integral lborel (λx. normal-density μ σ x * (x - μ) ^ (2 * k)) (fact
(2 * k) / ((2 / σ2) ^ k * fact k))

```

proof –

have $eq: \bigwedge x::real. x^2 \frown k = (x \frown k)^2$
by (*simp add: power-mult[symmetric] ac-simps*)

have *has-bochner-integral lborel* $(\lambda x. \exp(-x^2) * x \frown (2 * k))$
 $(\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k)))$
using *has-bochner-integral-even-function[OF gaussian-moment-even-pos[where*
 $k=k]$ **by** *simp*

then have *has-bochner-integral lborel* $(\lambda x. (\exp(-x^2) * x \frown (2 * k)) * ((2 * \sigma^2) \frown k /$
 $\text{sqrt } (2 * \pi * \sigma^2)))$
 $((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k))) * ((2 * \sigma^2) \frown k / \text{sqrt } (2 * \pi$
 $* \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)

also have $(\lambda x. (\exp(-x^2) * x \frown (2 * k)) * ((2 * \sigma^2) \frown k / \text{sqrt } (2 * \pi * \sigma^2))) =$
 $(\lambda x. \exp(-((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2)) * ((\text{sqrt } 2 * \sigma) * x) \frown (2 * k) / \text{sqrt}$
 $(2 * \pi * \sigma^2)))$
by (*auto simp: fun-eq-iff field-simps real-sqrt-power[symmetric] real-sqrt-mult*
real-sqrt-divide power-mult eq)

also have $((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k))) * ((2 * \sigma^2) \frown k / \text{sqrt}$
 $(2 * \pi * \sigma^2))) =$
 $(\text{inverse } (\text{sqrt } 2 * \sigma) * ((\text{fact } (2 * k)) / ((2 / \sigma^2) \frown k * (\text{fact } k))))$
by (*auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric]*
real-sqrt-mult
power2-eq-square)

finally show *?thesis*
unfolding *normal-density-def*
by (*subst lborel-has-bochner-integral-real-affine-iff[where c=sqrt 2 * sigma and*
 $t=\mu]$ *simp-all*)
qed

lemma *normal-moment-abs-odd:*

has-bochner-integral lborel $(\lambda x. \text{normal-density } \mu \sigma x * |x - \mu| \frown (2 * k + 1)) (2 \frown k$
 $* \sigma \frown (2 * k + 1) * \text{fact } k * \text{sqrt } (2 / \pi))$

proof –

have *has-bochner-integral lborel* $(\lambda x::real. \text{indicator } \{0..\} x * \mathbb{R} (\exp(-x^2) * |x| \frown (2$
 $* k + 1))) (\text{fact } k / 2)$
by (*rule has-bochner-integral-cong[THEN iffD1, OF --- gaussian-moment-odd-pos[of*
 $k]$ *auto*

from *has-bochner-integral-even-function[OF this]*

have *has-bochner-integral lborel* $(\lambda x::real. \exp(-x^2) * |x| \frown (2 * k + 1)) (\text{fact } k)$
by *simp*

then have *has-bochner-integral lborel* $(\lambda x. (\exp(-x^2) * |x| \frown (2 * k + 1)) * (2 \frown k$
 $* \sigma \frown (2 * k + 1) / \text{sqrt } (\pi * \sigma^2)))$
 $(\text{fact } k * (2 \frown k * \sigma \frown (2 * k + 1) / \text{sqrt } (\pi * \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)

also have $(\lambda x. (\exp(-x^2) * |x| \frown (2 * k + 1)) * (2 \frown k * \sigma \frown (2 * k + 1) / \text{sqrt } (\pi$
 $* \sigma^2))) =$
 $(\lambda x. \exp(-(((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2))) * |\text{sqrt } 2 * \sigma * x| \frown (2 * k + 1)$
 $/ \text{sqrt } (2 * \pi * \sigma^2)))$

by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult real-sqrt-mult*)
also have ($\text{fact } k * (2^k * \sigma^{2 * k + 1}) / \text{sqrt } (\text{pi} * \sigma^2)$) =
 $(\text{inverse } (\text{sqrt } 2) * \text{inverse } \sigma * (2^k * (\sigma * \sigma^{2 * k})) * (\text{fact } k) * \text{sqrt } (2 / \text{pi}))$)
by (*auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric] real-sqrt-divide*
real-sqrt-mult)
finally show ?thesis
unfolding normal-density-def
by (*subst lborel-has-bochner-integral-real-affine-iff[where c=sqrt 2 * sigma and t=mu]*)
simp-all
qed

lemma normal-moment-odd:

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k + 1}$) 0
proof –
have *has-bochner-integral lborel* ($\lambda x. \text{exp } (-x^2) * x^{2 * k + 1} :: \text{real}$) 0
using gaussian-moment-odd-pos **by** (*rule has-bochner-integral-odd-function*)
simp
then have *has-bochner-integral lborel* ($\lambda x. (\text{exp } (-x^2) * x^{2 * k + 1}) * (2^k * \sigma^{2 * k}) / \text{sqrt } \text{pi}$)
 $(0 * (2^k * \sigma^{2 * k}) / \text{sqrt } \text{pi})$)
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\text{exp } (-x^2) * x^{2 * k + 1}) * (2^k * \sigma^{2 * k}) / \text{sqrt } \text{pi}$) =
 $(\lambda x. \text{exp } (-((\text{sqrt } 2 * \sigma * x)^2 / (2 * \sigma^2))) * (\text{sqrt } 2 * \sigma * x * (\text{sqrt } 2 * \sigma * x)^{2 * k}) / \text{sqrt } (2 * \text{pi} * \sigma^2))$
unfolding real-sqrt-mult
by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult fun-eq-iff*)
finally show ?thesis
unfolding normal-density-def
by (*subst lborel-has-bochner-integral-real-affine-iff[where c=sqrt 2 * sigma and t=mu]*) *simp-all*
qed

lemma integral-normal-moment-even:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k}$) = $\text{fact } (2 * k) / ((2 / \sigma^2)^k * \text{fact } k)$
using normal-moment-even **by** (*rule has-bochner-integral-integral-eq*)

lemma integral-normal-moment-abs-odd:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * |x - \mu|^{2 * k + 1}$) = $2^k * \sigma^{2 * k + 1} * \text{fact } k * \text{sqrt } (2 / \text{pi})$
using normal-moment-abs-odd **by** (*rule has-bochner-integral-integral-eq*)

lemma integral-normal-moment-odd:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k + 1}$) = 0
using normal-moment-odd **by** (*rule has-bochner-integral-integral-eq*)

end

context

fixes $\sigma :: \text{real}$

assumes $\sigma\text{-pos}[\text{arith}]: 0 < \sigma$

begin

lemma *normal-moment-nz-1: has-bochner-integral lborel* ($\lambda x.$ *normal-density* $\mu \sigma$ $x * x$) μ

proof –

note *normal-moment-even*[*OF* $\sigma\text{-pos}$, *of* μ 0]

note *normal-moment-odd*[*OF* $\sigma\text{-pos}$, *of* μ 0] *has-bochner-integral-mult-left*[*of* μ , *OF this*]

note *has-bochner-integral-add*[*OF this*]

then show *?thesis*

by (*simp add: power2-eq-square field-simps*)

qed

lemma *integral-normal-moment-nz-1:*

integral^L lborel ($\lambda x.$ *normal-density* $\mu \sigma$ $x * x$) = μ

using *normal-moment-nz-1* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-normal-moment-nz-1: integrable lborel* ($\lambda x.$ *normal-density* $\mu \sigma$ $x * x$)

using *normal-moment-nz-1* **by** *rule*

lemma *integrable-normal-moment: integrable lborel* ($\lambda x.$ *normal-density* $\mu \sigma$ $x * (x - \mu) \frown k$)

proof *cases*

assume *even* k **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-even*] **by** (*auto elim: evenE*)

next

assume *odd* k **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-odd*] **by** (*auto elim: oddE*)

qed

lemma *integrable-normal-moment-abs: integrable lborel* ($\lambda x.$ *normal-density* $\mu \sigma$ $x * |x - \mu| \frown k$)

proof *cases*

assume *even* k **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-even*] **by** (*auto simp add: power-even-abs elim: evenE*)

next

assume *odd* k **then show** *?thesis*

using *integrable.intros*[*OF normal-moment-abs-odd*] **by** (*auto elim: oddE*)

qed

lemma *integrable-normal-density*[simp, intro]: *integrable lborel (normal-density μ σ)*

using *integrable-normal-moment*[of μ 0] **by** *simp*

lemma *integral-normal-density*[simp]: $(\int x. \text{normal-density } \mu \ \sigma \ x \ \partial \text{lborel}) = 1$

using *integral-normal-moment-even*[of σ μ 0] **by** *simp*

lemma *prob-space-normal-density*:

prob-space (density lborel (normal-density μ σ))

proof qed (*simp add: emeasure-density nn-integral-eq-integral normal-density-nonneg*)

end

context

fixes $k :: \text{nat}$

begin

lemma *std-normal-moment-even*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k)}$) (fact (2 * k) / (2^k * fact k))*

using *normal-moment-even*[of 1 0 k] **by** *simp*

lemma *std-normal-moment-abs-odd*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)}$) (sqrt (2/pi) * 2^k * fact k)*

using *normal-moment-abs-odd*[of 1 0 k] **by** (*simp add: ac-simps*)

lemma *std-normal-moment-odd*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)}$) 0*

using *normal-moment-odd*[of 1 0 k] **by** *simp*

lemma *integral-std-normal-moment-even*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2*k)}$) = fact (2 * k) / (2^k * fact k)*

using *std-normal-moment-even* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-abs-odd*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)}$) = sqrt (2 / pi) * 2^k * fact k*

using *std-normal-moment-abs-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-odd*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)}$) = 0*

using *std-normal-moment-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-std-normal-moment-abs*: *integrable lborel ($\lambda x. \text{std-normal-density } x * |x|^k$)*

using *integrable-normal-moment-abs*[of 1 0 k] **by** *simp*

lemma *integrable-std-normal-moment*: *integrable lborel* ($\lambda x.$ *std-normal-density* $x * x \hat{\ }^k$)

using *integrable-normal-moment*[of 1 0 k] **by** *simp*

end

lemma (in *prob-space*) *normal-density-affine*:

assumes X : *distributed M lborel* X (*normal-density* $\mu \sigma$)

assumes [*simp, arith*]: $0 < \sigma$ $\alpha \neq 0$

shows *distributed M lborel* ($\lambda x.$ $\beta + \alpha * X x$) (*normal-density* $(\beta + \alpha * \mu)$ $(|\alpha| * \sigma)$)

proof –

have *eq*: $\bigwedge x.$ $|\alpha| * \text{normal-density } (\beta + \alpha * \mu) (|\alpha| * \sigma) (x * \alpha + \beta) =$
normal-density $\mu \sigma x$

by (*simp add: normal-density-def real-sqrt-mult field-simps*)

(*simp add: power2-eq-square field-simps*)

show *?thesis*

by (*rule distributed-affineI*[*OF* - $\langle \alpha \neq 0 \rangle$, **where** $t = \beta$])

(*simp-all add: eq X ennreal-mult'[symmetric]*)

qed

lemma (in *prob-space*) *normal-standard-normal-convert*:

assumes *pos-var*[*simp, arith*]: $0 < \sigma$

shows *distributed M lborel* X (*normal-density* $\mu \sigma$) = *distributed M lborel* ($\lambda x.$ $(X x - \mu) / \sigma$) *std-normal-density*

proof *auto*

assume *distributed M lborel* X ($\lambda x.$ *ennreal* (*normal-density* $\mu \sigma x$))

then have *distributed M lborel* ($\lambda x.$ $-\mu / \sigma + (1/\sigma) * X x$) ($\lambda x.$ *ennreal* (*normal-density* $(-\mu / \sigma + (1/\sigma) * \mu)$ $(|1/\sigma| * \sigma) x$))

by(*rule normal-density-affine*) *auto*

then show *distributed M lborel* ($\lambda x.$ $(X x - \mu) / \sigma$) ($\lambda x.$ *ennreal* (*std-normal-density* x))

by (*simp add: diff-divide-distrib[symmetric] field-simps*)

next

assume *: *distributed M lborel* ($\lambda x.$ $(X x - \mu) / \sigma$) ($\lambda x.$ *ennreal* (*std-normal-density* x))

have *distributed M lborel* ($\lambda x.$ $\mu + \sigma * ((X x - \mu) / \sigma)$) ($\lambda x.$ *ennreal* (*normal-density* $\mu |\sigma| x$))

using *normal-density-affine*[*OF* *, of $\sigma \mu$] **by** *simp*

then show *distributed M lborel* X ($\lambda x.$ *ennreal* (*normal-density* $\mu \sigma x$)) **by** *simp*

qed

lemma *conv-normal-density-zero-mean*:

assumes [*simp, arith*]: $0 < \sigma$ $0 < \tau$

shows ($\lambda x.$ $\int^+ y.$ *ennreal* (*normal-density* $0 \sigma (x - y) * \text{normal-density } 0 \tau y$) ∂lborel) =

$normal-density\ 0\ (sqrt\ (\sigma^2 + \tau^2))\ (is\ ?LHS = ?RHS)$
proof –
define $\sigma'\ \tau'$ **where** $\sigma' = \sigma^2$ **and** $\tau' = \tau^2$
then have $[simp, arith]: 0 < \sigma' < \tau'$
by *simp-all*
let $? \sigma = sqrt\ ((\sigma' * \tau') / (\sigma' + \tau'))$
have $sqrt: (sqrt\ (2 * pi * (\sigma' + \tau'))) * sqrt\ (2 * pi * (\sigma' * \tau') / (\sigma' + \tau')) =$
 $(sqrt\ (2 * pi * \sigma') * sqrt\ (2 * pi * \tau'))$
by $(subst\ power-eq-iff-eq-base[symmetric, where\ n=2])$
 $(simp-all\ add: real-sqrt-mult[symmetric]\ power2-eq-square)$
have $?LHS =$
 $(\lambda x. \int^+ y. ennreal((normal-density\ 0\ (sqrt\ (\sigma' + \tau'))\ x) * normal-density\ (\tau'$
 $*\ x / (\sigma' + \tau'))\ ?\sigma\ y)\ \partial lborel)$
apply $(intro\ ext\ nn-integral-cong)$
apply $(simp\ add: normal-density-def\ \sigma'-def[symmetric]\ \tau'-def[symmetric]\ sqrt$
 $mult-exp-exp)$
apply $(simp\ add: divide-simps\ power2-eq-square)$
apply $(simp\ add: algebra-simps)$
done

also have $... =$
 $(\lambda x. (normal-density\ 0\ (sqrt\ (\sigma^2 + \tau^2))\ x) * \int^+ y. ennreal(normal-density\ (\tau^2 * x / (\sigma^2 + \tau^2))\ ?\sigma\ y)\ \partial lborel)$
by $(subst\ nn-integral-cmult[symmetric])$
 $(auto\ simp: \sigma'-def\ \tau'-def\ normal-density-def\ ennreal-mult'[symmetric])$

also have $... = (\lambda x. (normal-density\ 0\ (sqrt\ (\sigma^2 + \tau^2))\ x))$
by $(subst\ nn-integral-eq-integral)\ (auto\ simp: normal-density-nonneg)$

finally show $?thesis$ **by** *fast*
qed

lemma *conv-std-normal-density:*

$(\lambda x. \int^+ y. ennreal\ (std-normal-density\ (x - y) * std-normal-density\ y)\ \partial lborel)$
 $=$
 $(normal-density\ 0\ (sqrt\ 2))$
by $(subst\ conv-normal-density-zero-mean)\ simp-all$

lemma $(in\ prob-space)$ *add-indep-normal:*

assumes *indep:* $indep-var\ borel\ X\ borel\ Y$
assumes *pos-var[arith]:* $0 < \sigma < \tau$
assumes $normalX[simp]: distributed\ M\ lborel\ X\ (normal-density\ \mu\ \sigma)$
assumes $normalY[simp]: distributed\ M\ lborel\ Y\ (normal-density\ \nu\ \tau)$
shows $distributed\ M\ lborel\ (\lambda x. X\ x + Y\ x)\ (normal-density\ (\mu + \nu)\ (sqrt\ (\sigma^2 + \tau^2)))$

proof –

have $ind[simp]: indep-var\ borel\ (\lambda x. -\mu + X\ x)\ borel\ (\lambda x. -\nu + Y\ x)$

proof –

have $indep-var\ borel\ ((\lambda x. -\mu + x) \circ X)\ borel\ ((\lambda x. -\nu + x) \circ Y)$

```

    by (auto intro!: indep-var-compose assms)
    then show ?thesis by (simp add: o-def)
qed

have distributed M lborel ( $\lambda x. -\mu + 1 * X x$ ) (normal-density ( $-\mu + 1 * \mu$ )
(|1| *  $\sigma$ ))
  by(rule normal-density-affine[OF normalX pos-var(1), of 1 - $\mu$ ]) simp
then have 1[simp]: distributed M lborel ( $\lambda x. -\mu + X x$ ) (normal-density 0  $\sigma$ )
by simp

have distributed M lborel ( $\lambda x. -\nu + 1 * Y x$ ) (normal-density ( $-\nu + 1 * \nu$ )
(|1| *  $\tau$ ))
  by(rule normal-density-affine[OF normalY pos-var(2), of 1 - $\nu$ ]) simp
then have 2[simp]: distributed M lborel ( $\lambda x. -\nu + Y x$ ) (normal-density 0  $\tau$ )
by simp

have *: distributed M lborel ( $\lambda x. (-\mu + X x) + (-\nu + Y x)$ ) ( $\lambda x. \text{ennreal}$ 
(normal-density 0 ( $\text{sqrt}(\sigma^2 + \tau^2)$ )  $x$ ))
  using distributed-convolution[OF ind 1 2] conv-normal-density-zero-mean[OF
pos-var]
  by (simp add: ennreal-mult[symmetric] normal-density-nonneg)

have distributed M lborel ( $\lambda x. \mu + \nu + 1 * (-\mu + X x + (-\nu + Y x))$ )
( $\lambda x. \text{ennreal}$  (normal-density ( $\mu + \nu + 1 * 0$ ) (|1| *  $\text{sqrt}(\sigma^2 + \tau^2)$ )  $x$ ))
  by (rule normal-density-affine[OF *, of 1  $\mu + \nu$ ]) (auto simp: add-pos-pos)

then show ?thesis by auto
qed

lemma (in prob-space) diff-indep-normal:
  assumes indep[simp]: indep-var borel X borel Y
  assumes [simp, arith]:  $0 < \sigma$   $0 < \tau$ 
  assumes normalX[simp]: distributed M lborel X (normal-density  $\mu$   $\sigma$ )
  assumes normalY[simp]: distributed M lborel Y (normal-density  $\nu$   $\tau$ )
  shows distributed M lborel ( $\lambda x. X x - Y x$ ) (normal-density ( $\mu - \nu$ ) ( $\text{sqrt}(\sigma^2 + \tau^2)$ ))
proof -
  have distributed M lborel ( $\lambda x. 0 + - 1 * Y x$ ) ( $\lambda x. \text{ennreal}$  (normal-density (0
+  $- 1 * \nu$ ) (|- 1| *  $\tau$ )  $x$ ))
    by(rule normal-density-affine, auto)
  then have [simp]:distributed M lborel ( $\lambda x. - Y x$ ) ( $\lambda x. \text{ennreal}$  (normal-density
( $-\nu$ )  $\tau$ )  $x$ )) by simp

  have distributed M lborel ( $\lambda x. X x + (- Y x)$ ) (normal-density ( $\mu + - \nu$ ) ( $\text{sqrt}$ 
( $\sigma^2 + \tau^2$ )))
    apply (rule add-indep-normal)
    apply (rule indep-var-compose[unfolded comp-def, of borel - borel -  $\lambda x. x - \lambda x.$ 
-  $x$ ])
    apply simp-all

```

done
 then show *?thesis* by *simp*
 qed

lemma (in *prob-space*) *sum-indep-normal*:

assumes *finite* I $I \neq \{\}$ *indep-vars* $(\lambda i. \text{borel } X \ I)$

assumes $\bigwedge i. i \in I \implies 0 < \sigma \ i$

assumes *normal*: $\bigwedge i. i \in I \implies \text{distributed } M \ \text{lborel } (X \ i) \ (\text{normal-density } (\mu \ i) \ (\sigma \ i))$

shows *distributed* $M \ \text{lborel } (\lambda x. \sum_{i \in I}. X \ i \ x) \ (\text{normal-density } (\sum_{i \in I}. \mu \ i) \ (\text{sqrt } (\sum_{i \in I}. (\sigma \ i)^2)))$

using *assms*

proof (*induct* I *rule*: *finite-ne-induct*)

case (*singleton* i) then show *?case* by (*simp add* : *abs-of-pos*)

next

case (*insert* $i \ I$)

then have 1: *distributed* $M \ \text{lborel } (\lambda x. (X \ i \ x) + (\sum_{i \in I}. X \ i \ x))$

$(\text{normal-density } (\mu \ i + \text{sum } \mu \ I) \ (\text{sqrt } ((\sigma \ i)^2 + (\text{sqrt } (\sum_{i \in I}. (\sigma \ i)^2))^2)))$

apply (*intro add-indep-normal indep-vars-sum insert real-sqrt-gt-zero*)

apply (*auto intro: indep-vars-subset intro!: sum-pos*)

apply *fastforce*

done

have 2: $(\lambda x. (X \ i \ x) + (\sum_{i \in I}. X \ i \ x)) = (\lambda x. (\sum_{j \in \text{insert } i \ I}. X \ j \ x))$
 $\mu \ i + \text{sum } \mu \ I = \text{sum } \mu \ (\text{insert } i \ I)$

using *insert by auto*

have 3: $(\text{sqrt } ((\sigma \ i)^2 + (\text{sqrt } (\sum_{i \in I}. (\sigma \ i)^2))^2)) = (\text{sqrt } (\sum_{i \in \text{insert } i \ I}. (\sigma \ i)^2))$

using *insert by (simp add: sum-nonneg)*

show *?case* using 1 2 3 by *simp*

qed

lemma (in *prob-space*) *standard-normal-distributed-expectation*:

assumes D : *distributed* $M \ \text{lborel } X \ \text{std-normal-density}$

shows *expectation* $X = 0$

using *integral-std-normal-moment-odd*[*of* 0]

distributed-integral[*OF* D , *of* $\lambda x. x$, *symmetric*]

by *auto*

lemma (in *prob-space*) *normal-distributed-expectation*:

assumes σ [*arith*]: $0 < \sigma$

assumes D : *distributed* $M \ \text{lborel } X \ (\text{normal-density } \mu \ \sigma)$

shows *expectation* $X = \mu$

using *integral-normal-moment-nz-1*[*OF* σ , *of* μ] *distributed-integral*[*OF* D , *of* $\lambda x. x$, *symmetric*]

by (*auto simp: field-simps*)

lemma (in *prob-space*) *normal-distributed-variance*:
fixes $a\ b :: \text{real}$
assumes [*simp, arith*]: $0 < \sigma$
assumes D : *distributed M lborel X (normal-density $\mu\ \sigma$)*
shows *variance $X = \sigma^2$*
using *integral-normal-moment-even[$\text{of } \sigma\ \mu\ 1$]*
by (*subst distributed-integral[OF D , symmetric]*)
(simp-all add: eval-nat-numeral normal-distributed-expectation[OF assms])

lemma (in *prob-space*) *standard-normal-distributed-variance*:
distributed M lborel X std-normal-density \implies variance $X = 1$
using *normal-distributed-variance[$\text{of } 1\ X\ 0$]* **by** *simp*

lemma (in *information-space*) *entropy-normal-density*:
assumes [*arith*]: $0 < \sigma$
assumes D : *distributed M lborel X (normal-density $\mu\ \sigma$)*
shows *entropy b lborel $X = \log b (2 * \text{pi} * \exp 1 * \sigma^2) / 2$*
proof –
have *entropy b lborel $X = - (\int x. \text{normal-density } \mu\ \sigma\ x * \log b (\text{normal-density } \mu\ \sigma\ x) \partial \text{lborel})$*
using D **by** (*rule entropy-distr*) *simp*
also have $\dots = - (\int x. \text{normal-density } \mu\ \sigma\ x * (- \ln (2 * \text{pi} * \sigma^2) - (x - \mu)^2 / \sigma^2) / (2 * \ln b) \partial \text{lborel})$
by (*intro arg-cong[where $f = \text{uminus}$] Bochner-Integration.integral-cong*)
(auto simp: normal-density-def field-simps ln-mult log-def ln-div ln-sqrt)
also have $\dots = - (\int x. - (\text{normal-density } \mu\ \sigma\ x * (\ln (2 * \text{pi} * \sigma^2)) + (\text{normal-density } \mu\ \sigma\ x * (x - \mu)^2) / \sigma^2) / (2 * \ln b) \partial \text{lborel})$
by (*intro arg-cong[where $f = \text{uminus}$] Bochner-Integration.integral-cong*) *(auto simp: field-split-simps field-simps)*
also have $\dots = (\int x. \text{normal-density } \mu\ \sigma\ x * (\ln (2 * \text{pi} * \sigma^2)) + (\text{normal-density } \mu\ \sigma\ x * (x - \mu)^2) / \sigma^2 \partial \text{lborel}) / (2 * \ln b)$
by (*simp del: minus-add-distrib*)
also have $\dots = (\ln (2 * \text{pi} * \sigma^2) + 1) / (2 * \ln b)$
using *integral-normal-moment-even[$\text{of } \sigma\ \mu\ 1$]* **by** (*simp add: integrable-normal-moment fact-numeral*)
also have $\dots = \log b (2 * \text{pi} * \exp 1 * \sigma^2) / 2$
by (*simp add: log-def field-simps ln-mult*)
finally show *?thesis .*
qed

end

14 Characteristic Functions

theory *Characteristic-Functions*

imports *Weak-Convergence Independent-Family Distributions*

begin

lemma *mult-min-right: $a \geq 0 \implies (a :: \text{real}) * \min b\ c = \min (a * b)\ (a * c)$*

by (*metis min.absorb-iff2 min-def mult-left-mono*)

lemma *sequentially-even-odd*:

assumes *E*: eventually $(\lambda n. P (2 * n))$ *sequentially* **and** *O*: eventually $(\lambda n. P (2 * n + 1))$ *sequentially*

shows eventually *P* *sequentially*

proof –

from *E* **obtain** *n-e* **where** [*intro*]: $\bigwedge n. n \geq n-e \implies P (2 * n)$

by (*auto simp: eventually-sequentially*)

moreover

from *O* **obtain** *n-o* **where** [*intro*]: $\bigwedge n. n \geq n-o \implies P (Suc (2 * n))$

by (*auto simp: eventually-sequentially*)

show ?*thesis*

unfolding *eventually-sequentially*

proof (*intro exI allI impI*)

fix *n* **assume** $\max (2 * n-e) (2 * n-o + 1) \leq n$ **then show** *P n*

by (*cases even n*) (*auto elim!: evenE oddE*)

qed

qed

lemma *limseq-even-odd*:

assumes $(\lambda n. f (2 * n)) \longrightarrow (l :: 'a :: \text{topological-space})$

and $(\lambda n. f (2 * n + 1)) \longrightarrow l$

shows $f \longrightarrow l$

using *assms* **by** (*auto simp: filterlim-iff intro: sequentially-even-odd*)

14.1 Application of the FTC: integrating $e^i x$

abbreviation *iexp* :: *real* \Rightarrow *complex* **where**

iexp $\equiv (\lambda x. \text{exp } (i * \text{complex-of-real } x))$

lemma *isCont-iexp* [*simp*]: *isCont* *iexp* *x*

by (*intro continuous-intros*)

lemma *has-vector-derivative-iexp*[*derivative-intros*]:

(*iexp* *has-vector-derivative* $i * \text{iexp } x$) (at *x* within *s*)

by (*auto intro!: derivative-eq-intros simp: Re-exp Im-exp has-vector-derivative-complex-iff*)

lemma *interval-integral-iexp*:

fixes *a b* :: *real*

shows $(CLBINT x=a..b. \text{iexp } x) = i * \text{iexp } a - i * \text{iexp } b$

by (*subst interval-integral-FTC-finite* [**where** $F = \lambda x. -i * \text{iexp } x$])

(*auto intro!: derivative-eq-intros continuous-intros*)

14.2 The Characteristic Function of a Real Measure.

definition

char :: *real measure* \Rightarrow *real* \Rightarrow *complex*

where

char *M* *t* = $CLINT x|M. \text{iexp } (t * x)$

lemma (in *real-distribution*) *char-zero*: $\text{char } M \ 0 = 1$
unfolding *char-def* **by** (*simp del: space-eq-univ add: prob-space*)

lemma (in *prob-space*) *integrable-iexp*:
assumes $f: f \in \text{borel-measurable } M \wedge x. \text{Im } (f \ x) = 0$
shows *integrable* $M \ (\lambda x. \text{exp } (i * (f \ x)))$
proof (*intro integrable-const-bound [of - 1]*)
from f **have** $\wedge x. \text{of-real } (\text{Re } (f \ x)) = f \ x$
by (*simp add: complex-eq-iff*)
then show $\text{AE } x \text{ in } M. \text{cmod } (\text{exp } (i * f \ x)) \leq 1$
using *norm-exp-i-times[of Re (f x) for x]* **by** *simp*
qed (*insert f, simp*)

lemma (in *real-distribution*) *cmod-char-le-1*: $\text{norm } (\text{char } M \ t) \leq 1$
proof –
have $\text{norm } (\text{char } M \ t) \leq (\int x. \text{norm } (i \text{exp } (t * x)) \ \partial M)$
unfolding *char-def* **by** (*intro integral-norm-bound*)
also have $\dots \leq 1$
by (*simp del: of-real-mult*)
finally show *?thesis* .
qed

lemma (in *real-distribution*) *isCont-char*: $\text{isCont } (\text{char } M) \ t$
unfolding *continuous-at-sequentially*
proof *safe*
fix X **assume** $X: X \longrightarrow t$
show $(\text{char } M \circ X) \longrightarrow \text{char } M \ t$
unfolding *comp-def char-def*
by (*rule integral-dominated-convergence[where w= λ -. 1]*) (*auto intro!: tendsto-intros X*)
qed

lemma (in *real-distribution*) *char-measurable [measurable]*: $\text{char } M \in \text{borel-measurable borel}$
by (*auto intro!: borel-measurable-continuous-onI continuous-at-imp-continuous-on isCont-char*)

14.3 Independence

lemma (in *prob-space*) *char-distr-add*:
fixes $X1 \ X2 :: 'a \Rightarrow \text{real}$ **and** $t :: \text{real}$
assumes *indep-var borel X1 borel X2*
shows $\text{char } (\text{distr } M \ \text{borel } (\lambda \omega. X1 \ \omega + X2 \ \omega)) \ t =$
 $\text{char } (\text{distr } M \ \text{borel } X1) \ t * \text{char } (\text{distr } M \ \text{borel } X2) \ t$
proof –
from *assms* **have** [*measurable*]: *random-variable borel X1* **by** (*elim indep-var-rv1*)
from *assms* **have** [*measurable*]: *random-variable borel X2* **by** (*elim indep-var-rv2*)


```

have char (distr M borel ( $\lambda\omega. X1 \ \omega + X2 \ \omega$ )) t = (CLINT x|M. iexp (t * (X1
x + X2 x)))
  by (simp add: char-def integral-distr)
also have ... = (CLINT x|M. iexp (t * (X1 x)) * iexp (t * (X2 x)))
  by (simp add: field-simps exp-add)
also have ... = (CLINT x|M. iexp (t * (X1 x))) * (CLINT x|M. iexp (t * (X2
x)))
  by (intro indep-var-lebesgue-integral indep-var-compose[unfolded comp-def, OF
assms])
    (auto intro!: integrable-iexp)
also have ... = char (distr M borel X1) t * char (distr M borel X2) t
  by (simp add: char-def integral-distr)
finally show ?thesis .
qed

```

```

lemma (in prob-space) char-distr-sum:
  indep-vars ( $\lambda i. \text{borel}$ ) X A  $\implies$ 
  char (distr M borel ( $\lambda\omega. \sum i \in A. X \ i \ \omega$ )) t = ( $\prod i \in A. \text{char} \ (\text{distr} \ M \ \text{borel} \ (X \ i)) \ t$ )
proof (induct A rule: infinite-finite-induct)
  case (insert x F) with indep-vars-subset[of  $\lambda-. \text{borel} \ X \ \text{insert} \ x \ F \ F$ ] show ?case
  by (auto simp add: char-distr-add indep-vars-sum)
qed (simp-all add: char-def integral-distr prob-space del: distr-const)

```

14.4 Approximations to e^{ix}

Proofs from Billingsley, page 343.

```

lemma CLBINT-I0c-power-mirror-iexp:
  fixes x :: real and n :: nat
  defines f s m  $\equiv$  complex-of-real ((x - s) ^ m)
  shows (CLBINT s=0..x. f s n * iexp s) =
    x ^ Suc n / Suc n + (i / Suc n) * (CLBINT s=0..x. f s (Suc n) * iexp s)
proof -
  have 1:
    (( $\lambda s. \text{complex-of-real}(-((x - s) ^ (\text{Suc } n) / (\text{Suc } n))) * iexp \ s$ )
      has-vector-derivative complex-of-real((x - s) ^ n) * iexp s + (i * iexp s) *
      complex-of-real(-((x - s) ^ (\text{Suc } n) / (\text{Suc } n))))
    (at s within A) for s A
  by (intro derivative-eq-intros) auto

  let ?F =  $\lambda s. \text{complex-of-real}(-((x - s) ^ (\text{Suc } n) / (\text{Suc } n))) * iexp \ s$ 
  have x ^ (\text{Suc } n) / (\text{Suc } n) = (CLBINT s=0..x. (f s n * iexp s + (i * iexp s) *
- (f s (\text{Suc } n) / (\text{Suc } n)))) (is ?LHS = ?RHS)
proof -
  have ?RHS = (CLBINT s=0..x. (f s n * iexp s + (i * iexp s) *
  complex-of-real(-((x - s) ^ (\text{Suc } n) / (\text{Suc } n))))
  by (cases 0  $\leq$  x) (auto intro!: simp: f-def[abs-def])
  also have ... = ?F x - ?F 0
  unfolding zero-ereal-def using 1

```

```

    by (intro interval-integral-FTC-finite)
      (auto simp: f-def add-nonneg-eq-0-iff complex-eq-iff
        intro!: continuous-at-imp-continuous-on continuous-intros)
  finally show ?thesis
    by auto
qed
show ?thesis
  unfolding ⟨?LHS = ?RHS⟩ f-def interval-lebesgue-integral-mult-right [symmetric]
    by (subst interval-lebesgue-integral-add(2) [symmetric])
      (auto intro!: interval-integrable-isCont continuous-intros simp: zero-ereal-def
        complex-eq-iff)
qed

```

```

lemma iexp-eq1:
  fixes x :: real
  defines f s m ≡ complex-of-real ((x - s) ^ m)
  shows iexp x =
    (∑ k ≤ n. (i * x) ^ k / (fact k)) + ((i ^ (Suc n)) / (fact n)) * (CLBINT s=0..x.
    (f s n) * (iexp s)) (is ?P n)
  proof (induction n)
    show ?P 0
      by (auto simp add: field-simps interval-integral-iexp f-def zero-ereal-def)
  next
    fix n assume ih: ?P n
    have **: ∧ a b :: real. a = -b ⟷ a + b = 0
      by linarith
    have *: of-nat n * of-nat (fact n) ≠ - (of-nat (fact n)::complex)
      unfolding of-nat-mult[symmetric]
      by (simp add: complex-eq-iff ** of-nat-add[symmetric] del: of-nat-mult of-nat-fact
        of-nat-add)
    show ?P (Suc n)
      unfolding sum.atMost-Suc ih f-def CLBINT-I0c-power-mirror-iexp[of - n]
      by (simp add: divide-simps add-eq-0-iff *) (simp add: field-simps)
  qed

```

```

lemma iexp-eq2:
  fixes x :: real
  defines f s m ≡ complex-of-real ((x - s) ^ m)
  shows iexp x = (∑ k ≤ Suc n. (i * x) ^ k / fact k) + i ^ Suc n / fact n * (CLBINT
  s=0..x. f s n * (iexp s - 1))
  proof -
    have isCont-f: isCont (λs. f s n) x for n x
      by (auto simp: f-def)
    let ?F = λs. complex-of-real (-((x - s) ^ (Suc n) / real (Suc n)))
    have calc1: (CLBINT s=0..x. f s n * (iexp s - 1)) =
      (CLBINT s=0..x. f s n * iexp s) - (CLBINT s=0..x. f s n)
    unfolding zero-ereal-def
    by (subst interval-lebesgue-integral-diff(2) [symmetric])
      (simp-all add: interval-integrable-isCont isCont-f field-simps)
  qed

```

have *calc2*: $(\text{CLBINT } s=0..x. f s n) = x^{\wedge} \text{Suc } n / \text{Suc } n$
unfolding *zero-ereal-def*
proof (*subst interval-integral-FTC-finite* [**where** $a = 0$ **and** $b = x$ **and** $f = \lambda s. f s n$ **and** $F = ?F$])
show (*?F has-vector-derivative* $f y n$) (at y within $\{\min 0 x.. \max 0 x\}$) **for** y
unfolding *f-def*
by (*intro has-vector-derivative-of-real*)
(auto intro!: derivative-eq-intros simp del: power-Suc simp add: add-nonneg-eq-0-iff)
qed (*auto intro: continuous-at-imp-continuous-on isCont-f*)

have *calc3*: $i^{\wedge} (\text{Suc } (\text{Suc } n)) / (\text{fact } (\text{Suc } n)) = (i^{\wedge} (\text{Suc } n) / (\text{fact } n)) * (i / (\text{Suc } n))$
by (*simp add: field-simps*)

show *?thesis*
unfolding *iexp-eq1* [**where** $n = \text{Suc } n$ **and** $x=x$] *calc1 calc2 calc3* **unfolding** *f-def*
by (*subst CLBINT-I0c-power-mirror-iexp* [**where** $n = n$]) *auto*
qed

lemma *abs-LBINT-I0c-abs-power-diff*:
 $|\text{LBINT } s=0..x. |(x - s)^{\wedge} n| = |x^{\wedge} (\text{Suc } n) / (\text{Suc } n)|$
proof –
have $|\text{LBINT } s=0..x. |(x - s)^{\wedge} n| = |\text{LBINT } s=0..x. (x - s)^{\wedge} n|$
proof *cases*
assume $0 \leq x \vee \text{even } n$
then have $(\text{LBINT } s=0..x. |(x - s)^{\wedge} n|) = \text{LBINT } s=0..x. (x - s)^{\wedge} n$
by (*auto simp add: zero-ereal-def power-even-abs power-abs min-absorb1 max-absorb2*)
intro!: interval-integral-cong
then show *?thesis* **by** *simp*
next
assume $\neg (0 \leq x \vee \text{even } n)$
then have $(\text{LBINT } s=0..x. |(x - s)^{\wedge} n|) = \text{LBINT } s=0..x. -((x - s)^{\wedge} n)$
by (*auto simp add: zero-ereal-def power-abs min-absorb1 max-absorb2*)
ereal-min[symmetric] eréal-max[symmetric] power-minus-odd[symmetric]
simp del: eréal-min eréal-max intro!: interval-integral-cong
also have $\dots = -(\text{LBINT } s=0..x. (x - s)^{\wedge} n)$
by (*subst interval-lebesgue-integral-uminus, rule refl*)
finally show *?thesis* **by** *simp*
qed
also have $\text{LBINT } s=0..x. (x - s)^{\wedge} n = x^{\wedge} \text{Suc } n / \text{Suc } n$
proof –
let $?F = \lambda t. -((x - t)^{\wedge} (\text{Suc } n) / \text{Suc } n)$
have $\text{LBINT } s=0..x. (x - s)^{\wedge} n = ?F x - ?F 0$
unfolding *zero-ereal-def*
by (*intro interval-integral-FTC-finite continuous-at-imp-continuous-on has-real-derivative-iff-has-vector-derivative[THEN iffD1]*)

(*auto simp del: power-Suc intro!: derivative-eq-intros simp add: add-nonneg-eq-0-iff*)
also have $\dots = x \wedge (\text{Suc } n) / (\text{Suc } n)$ **by** *simp*
finally show *?thesis* .
qed
finally show *?thesis* .
qed

lemma *iexp-approx1*: $\text{cmod } (iexp\ x - (\sum k \leq n. (i * x) \wedge k / \text{fact } k)) \leq |x| \wedge (\text{Suc } n) / \text{fact } (\text{Suc } n)$

proof –

have $iexp\ x - (\sum k \leq n. (i * x) \wedge k / \text{fact } k) =$
 $((i \wedge (\text{Suc } n)) / (\text{fact } n)) * (\text{CLBINT } s=0..x. (x - s) \wedge n * (iexp\ s))$ (**is** *?t1 = ?t2*)

by (*subst iexp-eq1 [of - n], simp add: field-simps*)

then have $\text{cmod } (?t1) = \text{cmod } (?t2)$

by *simp*

also have $\dots = (1 / \text{of-nat } (\text{fact } n)) * \text{cmod } (\text{CLBINT } s=0..x. (x - s) \wedge n * (iexp\ s))$

by (*simp add: norm-mult norm-divide norm-power*)

also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |\text{LBINT } s=0..x. \text{cmod } ((x - s) \wedge n * (iexp\ s))|$

by (*intro mult-left-mono interval-integral-norm2*)

(*auto simp: zero-ereal-def intro: interval-integrable-isCont*)

also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |\text{LBINT } s=0..x. |(x - s) \wedge n||$

by (*simp add: norm-mult del: of-real-diff of-real-power*)

also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |x \wedge (\text{Suc } n) / (\text{Suc } n)|$

by (*simp add: abs-LBINT-I0c-abs-power-diff*)

also have $1 / \text{real-of-nat } (\text{fact } n :: \text{nat}) * |x \wedge \text{Suc } n / \text{real } (\text{Suc } n)| =$
 $|x| \wedge \text{Suc } n / \text{fact } (\text{Suc } n)$

by (*simp add: abs-mult power-abs*)

finally show *?thesis* .

qed

lemma *iexp-approx2*: $\text{cmod } (iexp\ x - (\sum k \leq n. (i * x) \wedge k / \text{fact } k)) \leq 2 * |x| \wedge n / \text{fact } n$

proof (*induction n*) — really cases

case (*Suc n*)

have *: $\bigwedge a\ b. \text{interval-lebesgue-integrable } \text{lborel } a\ b\ f \implies \text{interval-lebesgue-integrable } \text{lborel } a\ b\ g \implies$

$|\text{LBINT } s=a..b. f\ s| \leq |\text{LBINT } s=a..b. g\ s|$

if $f: \bigwedge s. 0 \leq f\ s$ **and** $g: \bigwedge s. f\ s \leq g\ s$ **for** $f\ g :: - \Rightarrow \text{real}$

using *order-trans[OF f g] f g*

unfolding *interval-lebesgue-integral-def interval-lebesgue-integrable-def set-lebesgue-integral-def set-integrable-def*

by (*auto simp: integral-nonneg-AE[OF AE-I2] intro!: integral-mono mult-mono*)

have $iexp\ x - (\sum k \leq \text{Suc } n. (i * x) \wedge k / \text{fact } k) =$

$((i \wedge (\text{Suc } n)) / (\text{fact } n)) * (\text{CLBINT } s=0..x. (x - s) \wedge n * (iexp\ s - 1))$ (**is** *?t1 = ?t2*)

unfolding $iexp\text{-}eq2$ [of $x\ n$] **by** ($simp\ add: field\text{-}simps$)
then have $cmod\ (?t1) = cmod\ (?t2)$
by $simp$
also have $\dots = (1 / (fact\ n)) * cmod\ (CLBINT\ s=0..x.\ (x - s)^{\wedge}n * (iexp\ s - 1))$
by ($simp\ add: norm\text{-}mult\ norm\text{-}divide\ norm\text{-}power$)
also have $\dots \leq (1 / (fact\ n)) * |LBINT\ s=0..x.\ cmod\ ((x - s)^{\wedge}n * (iexp\ s - 1))|$
by ($intro\ mult\text{-}left\text{-}mono\ interval\text{-}integral\text{-}norm2$)
($auto\ intro!: interval\text{-}integrable\text{-}isCont\ simp: zero\text{-}ereal\text{-}def$)
also have $\dots = (1 / (fact\ n)) * |LBINT\ s=0..x.\ abs\ ((x - s)^{\wedge}n) * cmod\ ((iexp\ s - 1))|$
by ($simp\ add: norm\text{-}mult\ del: of\text{-}real\text{-}diff\ of\text{-}real\text{-}power$)
also have $\dots \leq (1 / (fact\ n)) * |LBINT\ s=0..x.\ abs\ ((x - s)^{\wedge}n) * 2|$
by ($intro\ mult\text{-}left\text{-}mono * order\text{-}trans [OF\ norm\text{-}triangle\text{-}ineq4]$)
($auto\ simp: mult\text{-}ac\ zero\text{-}ereal\text{-}def\ intro!: interval\text{-}integrable\text{-}isCont$)
also have $\dots = (2 / (fact\ n)) * |x^{\wedge}(Suc\ n) / (Suc\ n)|$
by ($simp\ add: abs\text{-}LBINT\text{-}I0c\text{-}abs\text{-}power\text{-}diff\ abs\text{-}mult$)
also have $2 / fact\ n * |x^{\wedge}Suc\ n / real\ (Suc\ n)| = 2 * |x|^{\wedge}Suc\ n / (fact\ (Suc\ n))$
by ($simp\ add: abs\text{-}mult\ power\text{-}abs$)
finally show $?case$.
qed ($insert\ norm\text{-}triangle\text{-}ineq4$ [of $iexp\ x\ 1$], $simp$)

lemma (in $real\text{-}distribution$) $char\text{-}approx1$:

assumes $integrable\text{-}moments: \bigwedge k. k \leq n \implies integrable\ M\ (\lambda x. x^{\wedge}k)$
shows $cmod\ (char\ M\ t - (\sum k \leq n. (i * t)^{\wedge}k / fact\ k) * expectation\ (\lambda x. x^{\wedge}k))$
 \leq
 $(2 * |t|^{\wedge}n / fact\ n) * expectation\ (\lambda x. |x|^{\wedge}n)$ (**is** $cmod\ (char\ M\ t - ?t1) \leq -$)

proof –

have $integ\text{-}iexp: integrable\ M\ (\lambda x. iexp\ (t * x))$
by ($intro\ integrable\text{-}const\text{-}bound$) $auto$

define c **where** [$abs\text{-}def$]: $c\ k\ x = (i * t)^{\wedge}k / fact\ k * complex\text{-}of\text{-}real\ (x^{\wedge}k)$ **for**
 $k\ x$

have $integ\text{-}c: \bigwedge k. k \leq n \implies integrable\ M\ (\lambda x. c\ k\ x)$
unfolding $c\text{-}def$ **by** ($intro\ integrable\text{-}mult\text{-}right\ integrable\text{-}of\text{-}real\ integrable\text{-}moments$)

have $k \leq n \implies expectation\ (c\ k) = (i * t)^{\wedge}k * (expectation\ (\lambda x. x^{\wedge}k)) / fact\ k$ **for** k

unfolding $c\text{-}def$ $integral\text{-}mult\text{-}right\text{-}zero\ integral\text{-}complex\text{-}of\text{-}real$ **by** $simp$

then have $norm\ (char\ M\ t - ?t1) = norm\ (char\ M\ t - (CLINT\ x | M. (\sum k \leq n. c\ k\ x)))$

by ($simp\ add: integ\text{-}c$)

also have $\dots = norm\ ((CLINT\ x | M. iexp\ (t * x) - (\sum k \leq n. c\ k\ x)))$

unfolding $char\text{-}def$ **by** ($subst\ Bochner\text{-}Integration.integral\text{-}diff [OF\ integ\text{-}iexp]$)
($auto\ intro!: integ\text{-}c$)

also have $\dots \leq expectation\ (\lambda x. cmod\ (iexp\ (t * x) - (\sum k \leq n. c\ k\ x)))$

by ($intro\ integral\text{-}norm\text{-}bound$)

also have $\dots \leq \text{expectation } (\lambda x. 2 * |t| ^ n / \text{fact } n * |x| ^ n)$
proof (rule integral-mono)
show integrable M $(\lambda x. \text{cmod } (\text{iexp } (t * x) - (\sum k \leq n. c k x)))$
by (intro integrable-norm Bochner-Integration.integrable-diff integ-iexp Bochner-Integration.integrable-sum integ-c) simp
show integrable M $(\lambda x. 2 * |t| ^ n / \text{fact } n * |x| ^ n)$
unfolding power-abs[symmetric]
by (intro integrable-mult-right integrable-abs integrable-moments) simp
show $\text{cmod } (\text{iexp } (t * x) - (\sum k \leq n. c k x)) \leq 2 * |t| ^ n / \text{fact } n * |x| ^ n$
for x
using iexp-approx2[of $t * x$ n] **by** (auto simp add: abs-mult field-simps c-def)
qed
finally show ?thesis
unfolding integral-mult-right-zero .
qed

lemma (in real-distribution) char-approx2:

assumes integrable-moments: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. x ^ k)$
shows $\text{cmod } (\text{char } M t - (\sum k \leq n. ((i * t) ^ k / \text{fact } k) * \text{expectation } (\lambda x. x ^ k)))$
 \leq
 $(|t| ^ n / \text{fact } (\text{Suc } n)) * \text{expectation } (\lambda x. \min (2 * |x| ^ n * \text{Suc } n) (|t| * |x| ^ \text{Suc } n))$
(is $\text{cmod } (\text{char } M t - ?t1) \leq -$)

proof –

have integ-iexp: integrable M $(\lambda x. \text{iexp } (t * x))$
by (intro integrable-const-bound) auto

define c **where** [abs-def]: $c k x = (i * t) ^ k / \text{fact } k * \text{complex-of-real } (x ^ k)$ **for** $k x$

have integ-c: $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. c k x)$
unfolding c-def **by** (intro integrable-mult-right integrable-of-real integrable-moments)

have *: $\min (2 * |t * x| ^ n / \text{fact } n) (|t * x| ^ \text{Suc } n / \text{fact } (\text{Suc } n)) =$
 $|t| ^ n / \text{fact } (\text{Suc } n) * \min (2 * |x| ^ n * \text{real } (\text{Suc } n)) (|t| * |x| ^ \text{Suc } n)$ **for** x
apply (subst mult-min-right)
apply simp
apply (rule arg-cong2[where $f = \min$])
apply (simp-all add: field-simps abs-mult del: fact-Suc)
apply (simp-all add: field-simps)
done

have $k \leq n \implies \text{expectation } (c k) = (i * t) ^ k * (\text{expectation } (\lambda x. x ^ k)) / \text{fact } k$ **for** k

unfolding c-def integral-mult-right-zero integral-complex-of-real **by** simp

then have norm $(\text{char } M t - ?t1) = \text{norm } (\text{char } M t - (\text{CLINT } x | M. (\sum k \leq n. c k x)))$

by (simp add: integ-c)

also have $\dots = \text{norm } ((\text{CLINT } x | M. \text{iexp } (t * x) - (\sum k \leq n. c k x)))$

unfolding char-def **by** (subst Bochner-Integration.integral-diff[OF integ-iexp])

```

(auto intro!: integ-c)
also have ... ≤ expectation (λx. cmod (iexp (t * x) - (∑ k ≤ n. c k x)))
  by (rule integral-norm-bound)
also have ... ≤ expectation (λx. min (2 * |t * x|^n / fact n) (|t * x|^(Suc n)
/ fact (Suc n)))
  (is - ≤ expectation ?f)
proof (rule integral-mono)
  show integrable M (λx. cmod (iexp (t * x) - (∑ k ≤ n. c k x)))
  by (intro integrable-norm Bochner-Integration.integrable-diff integ-iexp Bochner-Integration.integrable-sum
integ-c) simp
  show integrable M ?f
  by (rule Bochner-Integration.integrable-bound[where f=λx. 2 * |t * x|^n /
fact n])
  (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
  show cmod (iexp (t * x) - (∑ k ≤ n. c k x)) ≤ ?f x for x
  using iexp-approx1[of t * x n] iexp-approx2[of t * x n]
  by (auto simp add: abs-mult field-simps c-def intro!: min.boundedI)
qed
also have ... = (|t|^n / fact (Suc n)) * expectation (λx. min (2 * |x|^n * Suc
n) (|t| * |x|^(Suc n)))
  unfolding *
proof (rule Bochner-Integration.integral-mult-right)
  show integrable M (λx. min (2 * |x|^n * real (Suc n)) (|t| * |x|^(Suc n)))
  by (rule Bochner-Integration.integrable-bound[where f=λx. 2 * |x|^n * real
(Suc n)])
  (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
qed
finally show ?thesis
  unfolding integral-mult-right-zero .
qed

```

lemma (in real-distribution) char-approx3:

```

fixes t
assumes
  integrable-1: integrable M (λx. x) and
  integral-1: expectation (λx. x) = 0 and
  integrable-2: integrable M (λx. x^2) and
  integral-2: variance (λx. x) = σ2
shows cmod (char M t - (1 - t^2 * σ2 / 2)) ≤
  (t^2 / 6) * expectation (λx. min (6 * x^2) (abs t * (abs x)^3))
proof -
note real-distribution.char-approx2 [of M 2 t, simplified]
have [simp]: prob UNIV = 1 by (metis prob-space space-eq-univ)
from integral-2 have [simp]: expectation (λx. x * x) = σ2
  by (simp add: integral-1 numeral-eq-Suc)
have 1: k ≤ 2 ⇒ integrable M (λx. x^k) for k
  using assms by (auto simp: eval-nat-numeral le-Suc-eq)
note char-approx1
note 2 = char-approx1 [of 2 t, OF 1, simplified]

```

```

have cmod (char M t - (∑ k≤2. (i * t) ^ k * (expectation (λx. x ^ k)) / (fact k))) ≤
  t2 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) / fact (3::nat)
using char-approx2 [of 2 t, OF 1] by simp
also have (∑ k≤2. (i * t) ^ k * expectation (λx. x ^ k)) / (fact k) = 1 - t2
  * σ2 / 2
by (simp add: complex-eq-iff numeral-eq-Suc integral-1 Re-divide Im-divide)
also have fact 3 = 6 by (simp add: eval-nat-numeral)
also have t2 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) / 6 =
  t2 / 6 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) by (simp add: field-simps)
finally show ?thesis .
qed

```

This is a more familiar textbook formulation in terms of random variables, but we will use the previous version for the CLT.

```

lemma (in prob-space) char-approx3':
fixes μ :: real measure and X
assumes rv-X [simp]: random-variable borel X
and [simp]: integrable M X integrable M (λx. (X x) ^ 2) expectation X = 0
and var-X: variance X = σ2
and μ-def: μ = distr M borel X
shows cmod (char μ t - (1 - t2 * σ2 / 2)) ≤
  (t2 / 6) * expectation (λx. min (6 * (X x) ^ 2) (|t| * |X x| ^ 3))
using var-X unfolding μ-def
apply (subst integral-distr [symmetric, OF rv-X], simp)
apply (intro real-distribution.char-approx3)
apply (auto simp add: integrable-distr-eq integral-distr)
done

```

this is the formulation in the book – in terms of a random variable *with* the distribution, rather the distribution itself. I don't know which is more useful, though in principal we can go back and forth between them.

```

lemma (in prob-space) char-approx1':
fixes μ :: real measure and X
assumes integrable-moments : ∧k. k ≤ n ⇒ integrable M (λx. X x ^ k)
and rv-X[measurable]: random-variable borel X
and μ-distr : distr M borel X = μ
shows cmod (char μ t - (∑ k ≤ n. (i * t) ^ k / fact k) * expectation (λx. (X x) ^ k)) ≤
  (2 * |t| ^ n / fact n) * expectation (λx. |X x| ^ n)
unfolding μ-distr[symmetric]
apply (subst (1 2) integral-distr [symmetric, OF rv-X], simp, simp)
apply (intro real-distribution.char-approx1 [of distr M borel X n t] real-distribution-distr rv-X)
apply (auto simp: integrable-distr-eq integrable-moments)
done

```


14.5 Calculation of the Characteristic Function of the Standard Distribution

abbreviation

std-normal-distribution \equiv *density lborel std-normal-density*

lemma *real-dist-normal-dist: real-distribution std-normal-distribution*

using *prob-space-normal-density* **by** (*auto simp: real-distribution-def real-distribution-axioms-def*)

lemma *std-normal-distribution-even-moments:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. x^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

and *integrable std-normal-distribution* ($\lambda x. x^{(2 * k)}$)

using *integral-std-normal-moment-even*[*of* k]

by (*subst integral-density*)

(*auto simp: normal-density-nonneg integrable-density*

intro: integrable.intros std-normal-moment-even)

lemma *integrable-std-normal-distribution-moment: integrable std-normal-distribution*
($\lambda x. x^k$)

by (*auto simp: normal-density-nonneg integrable-std-normal-moment integrable-density*)

lemma *integral-std-normal-distribution-moment-odd:*

odd $k \implies \text{integral}^L \text{std-normal-distribution } (\lambda x. x^k) = 0$

using *integral-std-normal-moment-odd*[*of* $(k - 1) \text{ div } 2$]

by (*auto simp: integral-density normal-density-nonneg elim: oddE*)

lemma *std-normal-distribution-even-moments-abs:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. |x|^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

using *integral-std-normal-moment-even*[*of* k]

by (*subst integral-density*) (*auto simp: normal-density-nonneg power-even-abs*)

lemma *std-normal-distribution-odd-moments-abs:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. |x|^{(2 * k + 1)} = \text{sqrt } (2 / \text{pi}) * 2^k * \text{fact } k$)

using *integral-std-normal-moment-abs-odd*[*of* k]

by (*subst integral-density*) (*auto simp: normal-density-nonneg*)

theorem *char-std-normal-distribution:*

char std-normal-distribution = ($\lambda t. \text{complex-of-real } (\text{exp } (- (t^2) / 2))$)

proof (*intro ext LIMSEQ-unique*)

fix $t :: \text{real}$

let $?f' = \lambda k. (i * t)^k / \text{fact } k * (\text{LINT } x \mid \text{std-normal-distribution}. x^k)$

let $?f = \lambda n. (\sum k \leq n. ?f' k)$

show $?f \longrightarrow \text{exp } (- (t^2) / 2)$

```

proof (rule limseq-even-odd)
  have (i * complex-of-real t) ^ (2 * a) / (2 ^ a * fact a) = (- ((complex-of-real
t)^2 / 2)) ^ a / fact a for a
    by (subst power-mult) (simp add: field-simps uminus-power-if power-mult)
  then have *: ?f (2 * n) = complex-of-real (∑ k < Suc n. (1 / fact k) * (-
(t^2) / 2) ^ k) for n :: nat
    unfolding of-real-sum
    by (intro sum.reindex-bij-witness-not-neutral[symmetric, where
i=λn. n div 2 and j=λn. 2 * n and T'={i. i ≤ 2 * n ∧ odd i} and
S'={}])
    (auto simp: integral-std-normal-distribution-moment-odd std-normal-distribution-even-moments)
  show (λn. ?f (2 * n)) → exp (-(t^2) / 2)
    unfolding * using exp-converges[where 'a=real]
    by (intro tendsto-of-real LIMSEQ-Suc) (auto simp: inverse-eq-divide sums-def
[symmetric])
  have **: ?f (2 * n + 1) = ?f (2 * n) for n
    proof -
      have ?f (2 * n + 1) = ?f (2 * n) + ?f' (2 * n + 1)
        by simp
      also have ?f' (2 * n + 1) = 0
        by (subst integral-std-normal-distribution-moment-odd) simp-all
      finally show ?f (2 * n + 1) = ?f (2 * n)
        by simp
    qed
  show (λn. ?f (2 * n + 1)) → exp (-(t^2) / 2)
    unfolding ** by fact
  qed

have **: (λn. x ^ n / fact n) → 0 for x :: real
  using summable-LIMSEQ-zero [OF summable-exp] by (auto simp add: in-
verse-eq-divide)

let ?F = λn. 2 * |t| ^ n / fact n * (LINT x|std-normal-distribution. |x| ^ n)

show ?f → char std-normal-distribution t
proof (rule metric-tendsto-imp-tendsto[OF limseq-even-odd])
  show (λn. ?F (2 * n)) → 0
    proof (rule Lim-transform-eventually)
      show ∀F n in sequentially. 2 * ((t^2) / 2) ^ n / fact n = ?F (2 * n)
        unfolding std-normal-distribution-even-moments-abs by (simp add: power-mult
power-divide)
    qed (intro tendsto-mult-right-zero **)

  have *: ?F (2 * n + 1) = (2 * |t| * sqrt (2 / pi)) * ((2 * t^2) ^ n * fact n /
fact (2 * n + 1)) for n
    unfolding std-normal-distribution-odd-moments-abs
    by (simp add: field-simps power-mult[symmetric] power-even-abs)
  have norm ((2 * t^2) ^ n * fact n / fact (2 * n + 1)) ≤ (2 * t^2) ^ n / fact n
for n

```

```

using mult-mono[OF - square-fact-le-2-fact, of 1 1 + 2 * real n n]
by (auto simp add: divide-simps intro!: mult-left-mono)
then show ( $\lambda n. ?F (2 * n + 1)$ )  $\longrightarrow 0$ 
unfolding * by (intro tendsto-mult-right-zero Lim-null-comparison [OF - **
[of 2 * t2]]) auto

show  $\forall_F n$  in sequentially. dist (?f n) (char std-normal-distribution t)  $\leq$  dist
(?F n) 0
using real-distribution.char-approx1[OF real-dist-normal-dist integrable-std-normal-distribution-moment]
by (auto simp: dist-norm integral-nonneg-AE norm-minus-commute)
qed
qed

end

```

15 Helly’s selection theorem

The set of bounded, monotone, right continuous functions is sequentially compact

theory *Helly-Selection*

imports *HOL-Library.Diagonal-Subsequence Weak-Convergence*

begin

lemma *minus-one-less*: $x - 1 < (x::real)$

by *simp*

theorem *Helly-selection*:

fixes $f :: nat \Rightarrow real \Rightarrow real$

assumes *rcont*: $\bigwedge n x. continuous (at-right x) (f n)$

assumes *mono*: $\bigwedge n. mono (f n)$

assumes *bdd*: $\bigwedge n x. |f n x| \leq M$

shows $\exists s. strict-mono (s::nat \Rightarrow nat) \wedge (\exists F. (\forall x. continuous (at-right x) F) \wedge$
 $mono F \wedge (\forall x. |F x| \leq M) \wedge$

$(\forall x. continuous (at x) F \longrightarrow (\lambda n. f (s n) x) \longrightarrow F x))$

proof –

obtain $m :: real \Rightarrow nat$ **where** *bij-betw m* \mathbb{Q} *UNIV*

using *countable-rat Rats-infinite* **by** (*erule countableE-infinite*)

then obtain $r :: nat \Rightarrow real$ **where** *bij*: *bij-betw r* \mathbb{Q} *UNIV* \mathbb{Q}

using *bij-betw-inv* **by** *blast*

have *dense-r*: $\bigwedge x y. x < y \implies \exists n. x < r n \wedge r n < y$

by (*metis Rats-dense-in-real bij f-the-inv-into-f bij-betw-def*)

let $?P = \lambda n. \lambda s. convergent (\lambda k. f (s k) (r n))$

interpret *nat*: *subseqs* $?P$

proof (*unfold convergent-def, unfold subseqs-def, auto*)

fix $n :: nat$ **and** $s :: nat \Rightarrow nat$ **assume** *s*: *strict-mono s*

have *bounded* $\{-M..M\}$

```

    using bounded-closed-interval by auto
  moreover have  $\bigwedge k. f (s k) (r n) \in \{-M..M\}$ 
    using bdd by (simp add: abs-le-iff minus-le-iff)
  ultimately have  $\exists l s'. \text{strict-mono } s' \wedge ((\lambda k. f (s k) (r n)) \circ s') \longrightarrow l$ 
    using compact-Icc compact-imp-seq-compact seq-compactE by metis
  thus  $\exists s'. \text{strict-mono } (s'::\text{nat} \Rightarrow \text{nat}) \wedge (\exists l. (\lambda k. f (s (s' k)) (r n)) \longrightarrow l)$ 
    by (auto simp: comp-def)
qed
define d where  $d = \text{nat.diagseq}$ 
have subseq: strict-mono d
  unfolding d-def using nat.subseq-diagseq by auto
have rat-cnvt:  $?P n d$  for n
proof -
  have Pn-seqseq:  $?P n (\text{nat.seqseq } (Suc n))$ 
    by (rule nat.seqseq-holds)
  have 1:  $(\lambda k. f ((\text{nat.seqseq } (Suc n) \circ (\lambda k. \text{nat.fold-reduce } (Suc n) k (Suc n + k))) k) (r n)) = (\lambda k. f (\text{nat.seqseq } (Suc n) k) (r n)) \circ (\lambda k. \text{nat.fold-reduce } (Suc n) k (Suc n + k))$ 
    by auto
  have 2:  $?P n (d \circ ((+) (Suc n)))$ 
    unfolding d-def nat.diagseq-seqseq 1
    by (intro convergent-subseq-convergent Pn-seqseq nat.subseq-diagonal-rest)
  then obtain L where  $\exists: (\lambda na. f (d (na + Suc n)) (r n)) \longrightarrow L$ 
    by (auto simp: add.commute dest: convergentD)
  then have  $(\lambda k. f (d k) (r n)) \longrightarrow L$ 
    by (rule LIMSEQ-offset)
  then show ?thesis
    by (auto simp: convergent-def)
qed
let ?f =  $\lambda n. \lambda k. f (d k) (r n)$ 
have lim-f:  $?f n \longrightarrow \text{lim } (?f n)$  for n
  using rat-cnvt convergent-LIMSEQ-iff by auto
have lim-bdd:  $\text{lim } (?f n) \in \{-M..M\}$  for n
proof -
  have closed  $\{-M..M\}$  using closed-real-atLeastAtMost by auto
  hence  $(\forall i. ?f n i \in \{-M..M\}) \wedge ?f n \longrightarrow \text{lim } (?f n) \longrightarrow \text{lim } (?f n) \in \{-M..M\}$ 
    unfolding closed-sequential-limits by (drule-tac  $x = \lambda k. f (d k) (r n)$  in spec)
blast
  moreover have  $\forall i. ?f n i \in \{-M..M\}$ 
    using bdd by (simp add: abs-le-iff minus-le-iff)
  ultimately show  $\text{lim } (?f n) \in \{-M..M\}$ 
    using lim-f by auto
qed
then have limset-bdd:  $\bigwedge x. \{\text{lim } (?f n) \mid n. x < r n\} \subseteq \{-M..M\}$ 
  by auto
then have bdd-below: bdd-below  $\{\text{lim } (?f n) \mid n. x < r n\}$  for x
  by (metis (mono-tags) bdd-below-Icc bdd-below-mono)
have r-unbdd:  $\exists n. x < r n$  for x

```

```

using dense-r[OF less-add-one, of x] by auto
then have nonempty: {lim (?f n) | n. x < r n} ≠ {} for x
by auto

define F where F x = Inf {lim (?f n) | n. x < r n} for x
have F-eq: ereal (F x) = (INF n∈{n. x < r n}. ereal (lim (?f n))) for x
unfolding F-def by (subst ereal-Inf'[OF bdd-below nonempty]) (simp add:
setcompr-eq-image image-comp)
have mono-F: mono F
using nonempty by (auto intro!: cInf-superset-mono simp: F-def bdd-below
mono-def)
moreover have  $\bigwedge x$ . continuous (at-right x) F
unfolding continuous-within order-tendsto-iff eventually-at-right[OF less-add-one]
proof safe
show F x < u  $\implies \exists b > x. \forall y > x. y < b \longrightarrow F y < u$  for x u
unfolding F-def cInf-less-iff[OF nonempty bdd-below] by auto
next
show  $\exists b > x. \forall y > x. y < b \longrightarrow l < F y$  if l: l < F x for x l
using less-le-trans[OF l mono-F[THEN monoD, of x]] by (auto intro: less-add-one)
qed
moreover
{ fix x
have F x ∈ {−M..M}
unfolding F-def using limset-bdd bdd-below r-unbdd by (intro closed-subset-contains-Inf)
auto
then have |F x| ≤ M by auto }
moreover have  $(\lambda n. f (d n) x) \longrightarrow F x$  if cts: continuous (at x) F for x
proof (rule limsup-le-liminf-real)
show limsup  $(\lambda n. f (d n) x) \leq F x$ 
proof (rule tendsto-lowerbound)
show  $(F \longrightarrow \text{ereal } (F x))$  (at-right x)
using cts unfolding continuous-at-split by (auto simp: continuous-within)
show  $\forall_F i$  in at-right x. limsup  $(\lambda n. f (d n) x) \leq F i$ 
unfolding eventually-at-right[OF less-add-one]
proof (rule, rule, rule less-add-one, safe)
fix y assume y: x < y
with dense-r obtain N where x < r N r N < y by auto
have *: y < r n'  $\implies \text{lim } (?f N) \leq \text{lim } (?f n')$  for n'
using  $\langle r N < y \rangle$  by (intro LIMSEQ-le[OF lim-f lim-f]) (auto intro!:
mono[THEN monoD])
have limsup  $(\lambda n. f (d n) x) \leq \text{limsup } (?f N)$ 
using  $\langle x < r N \rangle$  by (auto intro!: Limsup-mono always-eventually
mono[THEN monoD])
also have ... = lim  $(\lambda n. \text{ereal } (?f N n))$ 
using rat-cnv[of N] by (force intro!: convergent-limsup-cl simp: conver-
gent-def)
also have ... ≤ F y
by (auto intro!: INF-greatest * simp: convergent-real-imp-convergent-ereal
rat-cnv F-eq)

```

```

    finally show  $\limsup (\lambda n. f (d n) x) \leq F y$  .
  qed
qed simp
show  $F x \leq \liminf (\lambda n. f (d n) x)$ 
proof (rule tendsto-upperbound)
  show  $(F \longrightarrow \text{ereal } (F x))$  (at-left x)
    using cts unfolding continuous-at-split by (auto simp: continuous-within)
  show  $\forall_F i$  in at-left x.  $F i \leq \liminf (\lambda n. f (d n) x)$ 
    unfolding eventually-at-left[OF minus-one-less]
  proof (rule, rule, rule minus-one-less, safe)
    fix y assume y:  $y < x$ 
    with dense-r obtain N where  $y < r N r N < x$  by auto
    have  $F y \leq \liminf (?f N)$ 
      using  $\langle y < r N \rangle$  by (auto simp: F-eq convergent-real-imp-convergent-ereal
        rat-cnv convergent-liminf-cl intro!: INF-lower2)
    also have  $\dots \leq \liminf (\lambda n. f (d n) x)$ 
      using  $\langle r N < x \rangle$  by (auto intro!: Liminf-mono monoD[OF mono] al-
        ways-eventually)
    finally show  $F y \leq \liminf (\lambda n. f (d n) x)$  .
  qed
qed simp
qed
ultimately show ?thesis using subseq by auto
qed

```

definition

```

tight :: (nat  $\Rightarrow$  real measure)  $\Rightarrow$  bool
where
  tight  $\mu \equiv (\forall n. \text{real-distribution } (\mu n)) \wedge (\forall (\varepsilon::\text{real})>0. \exists a b::\text{real}. a < b \wedge (\forall n.
    \text{measure } (\mu n) \{a<..b\} > 1 - \varepsilon))$ 

```

theorem tight-imp-convergent-subsubsequence:

```

assumes  $\mu$ : tight  $\mu$  strict-mono s
shows  $\exists r M. \text{strict-mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge \text{real-distribution } M \wedge \text{weak-conv-m}$ 
 $(\mu \circ s \circ r) M$ 
proof -
  define f where  $f k = \text{cdf } (\mu (s k))$  for k
  interpret  $\mu$ : real-distribution  $\mu k$  for k
    using  $\mu$  unfolding tight-def by auto

  have rcont:  $\bigwedge x. \text{continuous } (\text{at-right } x) (f k)$ 
    and mono: mono (f k)
    and top:  $(f k \longrightarrow 1)$  at-top
    and bot:  $(f k \longrightarrow 0)$  at-bot for k
  unfolding f-def mono-def
  using  $\mu$ .cdf-nondecreasing  $\mu$ .cdf-is-right-cont  $\mu$ .cdf-lim-at-top-prob  $\mu$ .cdf-lim-at-bot

```

by *auto*
have *bdd*: $|f\ k\ x| \leq 1$ **for** $k\ x$
by (*auto simp add: abs-le-iff minus-le-iff f-def μ .cdf-nonneg μ .cdf-bounded-prob*)

from *Helly-selection*[*OF rcont mono bdd, of $\lambda x. x$*] **obtain** $r\ F$
where F : *strict-mono* $r \wedge x. \text{continuous (at-right } x) F \text{ mono } F \wedge x. |F\ x| \leq 1$
and *lim-F*: $\wedge x. \text{continuous (at } x) F \implies (\lambda n. f\ (r\ n)\ x) \longrightarrow F\ x$
by *blast*

have $0 \leq f\ n\ x$ **for** $n\ x$
unfolding *f-def* **by** (*rule μ .cdf-nonneg*)
have *F-nonneg*: $0 \leq F\ x$ **for** x
proof –
obtain y **where** $y < x$ *isCont* $F\ y$
using *open-minus-countable*[*OF mono-ctble-discont*[*OF \langle mono $F\rangle$*], *of $\{..<$*
x}] **by** *auto*
then **have** $0 \leq F\ y$
by (*intro LIMSEQ-le-const*[*OF lim-F*]) (*auto simp: f-def μ .cdf-nonneg*)
also **have** $\dots \leq F\ x$
using $\langle y < x \rangle$ **by** (*auto intro!: monoD*[*OF \langle mono $F\rangle$*])
finally **show** $0 \leq F\ x$.
qed

have *Fab*: $\exists a\ b. (\forall x \geq b. F\ x \geq 1 - \varepsilon) \wedge (\forall x \leq a. F\ x \leq \varepsilon)$ **if** $\varepsilon: 0 < \varepsilon$ **for** ε
proof *auto*
obtain $a'\ b'$ **where** $a' < b' \wedge k. \text{measure } (\mu\ k) \{a' <.. b'\} > 1 - \varepsilon$
using $\varepsilon\ \mu$ **by** (*auto simp: tight-def*)
obtain a **where** $a: a < a'$ *isCont* $F\ a$
using *open-minus-countable*[*OF mono-ctble-discont*[*OF \langle mono $F\rangle$*], *of $\{..<$*
a'\}] **by** *auto*
obtain b **where** $b: b' < b$ *isCont* $F\ b$
using *open-minus-countable*[*OF mono-ctble-discont*[*OF \langle mono $F\rangle$*], *of $\{b' <..$*
<..\}] **by** *auto*
have $a < b$
using $a\ b\ a'\ b'$ **by** *simp*

let $? \mu = \lambda k. \text{measure } (\mu\ (s\ (r\ k)))$
have *ab*: $? \mu\ k \{a <.. b\} > 1 - \varepsilon$ **for** k
proof –
have $? \mu\ k \{a' <.. b'\} \leq ? \mu\ k \{a <.. b\}$
using $a\ b$ **by** (*intro μ .finite-measure-mono*) *auto*
then **show** *?thesis*
using $a' b' (2)$ **by** (*metis less-eq-real-def less-trans*)
qed

have $(\lambda k. ? \mu\ k \{.. b\}) \longrightarrow F\ b$
using $b(2)$ *lim-F* **unfolding** *f-def cdf-def o-def* **by** *auto*
then **have** $1 - \varepsilon \leq F\ b$
proof (*rule tendsto-lowerbound, intro always-eventually allI*)

```

fix  $k$ 
have  $1 - \varepsilon < ?\mu k \{a < ..b\}$ 
  using  $ab$  by  $auto$ 
also have  $\dots \leq ?\mu k \{..b\}$ 
  by  $(auto\ intro!: \mu.\text{finite-measure-mono})$ 
finally show  $1 - \varepsilon \leq ?\mu k \{..b\}$ 
  by  $(rule\ less-imp-le)$ 
qed  $(rule\ sequentially-bot)$ 
then show  $\exists b. \forall x \geq b. 1 - \varepsilon \leq F x$ 
  using  $F$  unfolding  $mono-def$  by  $(metis\ order.trans)$ 

have  $(\lambda k. ?\mu k \{..a\}) \longrightarrow F a$ 
  using  $a(2)\ lim-F$  unfolding  $f-def\ cdf-def\ o-def$  by  $auto$ 
then have  $Fa: F a \leq \varepsilon$ 
proof  $(rule\ tendsto-upperbound, intro\ always-eventually\ allI)$ 
  fix  $k$ 
  have  $?\mu k \{..a\} + ?\mu k \{a < ..b\} \leq 1$ 
    by  $(subst\ \mu.\text{finite-measure-Union[symmetric]})\ auto$ 
  then show  $?\mu k \{..a\} \leq \varepsilon$ 
    using  $ab[of\ k]$  by  $simp$ 
  qed  $(rule\ sequentially-bot)$ 
then show  $\exists a. \forall x \leq a. F x \leq \varepsilon$ 
  using  $F$  unfolding  $mono-def$  by  $(metis\ order.trans)$ 
qed

have  $(F \longrightarrow 1)$   $at-top$ 
proof  $(rule\ order-tendstoI)$ 
  show  $1 < y \implies \forall_F x\ in\ at-top. F x < y$  for  $y$ 
    using  $\langle \bigwedge x. |F x| \leq 1 \rangle \langle \bigwedge x. 0 \leq F x \rangle$  by  $(auto\ intro: le-less-trans\ always-eventually)$ 
  fix  $y :: real$  assume  $y < 1$ 
  then obtain  $z$  where  $y < z < 1$ 
    using  $dense[of\ y\ 1]$  by  $auto$ 
  with  $Fab[of\ 1 - z]$  show  $\forall_F x\ in\ at-top. y < F x$ 
    by  $(auto\ simp: eventually-at-top-linorder\ intro: less-le-trans)$ 
qed
moreover
have  $(F \longrightarrow 0)$   $at-bot$ 
proof  $(rule\ order-tendstoI)$ 
  show  $y < 0 \implies \forall_F x\ in\ at-bot. y < F x$  for  $y$ 
    using  $\langle \bigwedge x. 0 \leq F x \rangle$  by  $(auto\ intro: less-le-trans\ always-eventually)$ 
  fix  $y :: real$  assume  $0 < y$ 
  then obtain  $z$  where  $0 < z < y$ 
    using  $dense[of\ 0\ y]$  by  $auto$ 
  with  $Fab[of\ z]$  show  $\forall_F x\ in\ at-bot. F x < y$ 
    by  $(auto\ simp: eventually-at-bot-linorder\ intro: le-less-trans)$ 
qed
ultimately have  $M: real-distribution\ (interval-measure\ F)\ cdf\ (interval-measure\ F) = F$ 

```



```

using  $F$  by (auto intro!: real-distribution-interval-measure cdf-interval-measure
simp: mono-def)
with  $\text{lim-}F$   $\text{LIMSEQ-subseq-LIMSEQ}$   $M$  have  $\text{weak-conv-m}$   $(\mu \circ s \circ r)$  (interval-measure
 $F$ )
by (auto simp: weak-conv-def weak-conv-m-def f-def comp-def)
then show  $\exists r M. \text{strict-mono}$   $(r :: \text{nat} \Rightarrow \text{nat}) \wedge (\text{real-distribution } M \wedge \text{weak-conv-m}$ 
 $(\mu \circ s \circ r) M)$ 
using  $F M$  by auto
qed

```

corollary *tight-subseq-weak-converge:*

```

fixes  $\mu :: \text{nat} \Rightarrow \text{real measure}$  and  $M :: \text{real measure}$ 
assumes  $\bigwedge n. \text{real-distribution}$   $(\mu n)$   $\text{real-distribution } M$  and tight:  $\text{tight } \mu$  and
subseq:  $\bigwedge s \nu. \text{strict-mono } s \implies \text{real-distribution } \nu \implies \text{weak-conv-m}$   $(\mu \circ s) \nu$ 
 $\implies \text{weak-conv-m}$   $(\mu \circ s) M$ 
shows  $\text{weak-conv-m}$   $\mu M$ 
proof (rule ccontr)
define  $f$  where  $f n = \text{cdf}$   $(\mu n)$  for  $n$ 
define  $F$  where  $F = \text{cdf}$   $M$ 

assume  $\neg \text{weak-conv-m}$   $\mu M$ 
then obtain  $x$  where  $x: \text{isCont } F x \neg (\lambda n. f n x) \longrightarrow F x$ 
by (auto simp: weak-conv-m-def weak-conv-def f-def  $F$ -def)
then obtain  $\varepsilon$  where  $\varepsilon > 0$  and  $\text{infinite}$   $\{n. \neg \text{dist}$   $(f n x) (F x) < \varepsilon\}$ 
by (auto simp: tendsto-iff not-eventually INFM-iff-infinite cofinite-eq-sequentially[symmetric])
then obtain  $s :: \text{nat} \Rightarrow \text{nat}$  where  $s: \bigwedge n. \neg \text{dist}$   $(f (s n) x) (F x) < \varepsilon$  and
strict-mono  $s$ 
using enumerate-in-set enumerate-mono by (fastforce simp: strict-mono-def)
then obtain  $r \nu$  where  $r: \text{strict-mono}$   $r$   $\text{real-distribution } \nu$   $\text{weak-conv-m}$   $(\mu \circ s$ 
 $\circ r) \nu$ 
using tight-imp-convergent-subsubsequence[OF tight] by blast
then have  $\text{weak-conv-m}$   $(\mu \circ (s \circ r)) M$ 
using  $\langle \text{strict-mono } s \rangle r$  by (intro subseq strict-mono-o) (auto simp: comp-assoc)
then have  $(\lambda n. f (s (r n)) x) \longrightarrow F x$ 
using  $x$  by (auto simp: weak-conv-m-def weak-conv-def F-def f-def)
then show False
using  $s \langle \varepsilon > 0 \rangle$  by (auto dest: tendstoD)
qed

end

```

16 Integral of sinc

```

theory Sinc-Integral
imports Distributions
begin

```

16.1 Various preparatory integrals

Naming convention The theorem name consists of the following parts:

- Kind of integral: *has-bochner-integral / integrable / LBINT*
- Interval: Interval (0 / infinity / open / closed) (infinity / open / closed)
- Name of the occurring constants: power, exp, m (for minus), scale, sin, ...

lemma *has-bochner-integral-I0i-power-exp-m'*:

has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 \dots\} x :: \text{real}$) (fact *k*)

using *nn-integral-power-times-exp-Ici*[of *k*]

by (*intro has-bochner-integral-nn-integral*)

(*auto simp: nn-integral-set-ennreal split: split-indicator*)

lemma *has-bochner-integral-I0i-power-exp-m*:

has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 < \dots\} x :: \text{real}$) (fact *k*)

using *AE-lborel-singleton*[of 0]

by (*intro has-bochner-integral-cong-AE[THEN iffD1, OF - - - has-bochner-integral-I0i-power-exp-m']*)

(*auto split: split-indicator*)

lemma *integrable-I0i-exp-mscale*: $0 < (u :: \text{real}) \implies \text{set-integrable lborel } \{0 < \dots\} (\lambda x. \exp(-(x * u)))$

using *lborel-integrable-real-affine-iff*[of *u* $\lambda x. \text{indicator } \{0 < \dots\} x *_R \exp(-x)$ 0]

has-bochner-integral-I0i-power-exp-m[of 0]

by (*simp add: indicator-def zero-less-mult-iff mult-ac integrable.intros set-integrable-def*)

lemma *LBINT-I0i-exp-mscale*: $0 < (u :: \text{real}) \implies \text{LBINT } x=0..\infty. \exp(-(x * u)) = 1 / u$

using *lborel-integral-real-affine*[of *u* $\lambda x. \text{indicator } \{0 < \dots\} x *_R \exp(-x)$ 0]

has-bochner-integral-I0i-power-exp-m[of 0]

by (*auto simp: indicator-def zero-less-mult-iff interval-lebesgue-integral-0-infnty set-lebesgue-integral-def field-simps*)

dest!: has-bochner-integral-integral-eq)

lemma *LBINT-I0c-exp-mscale-sin*:

$\text{LBINT } x=0..t. \exp(-(u * x)) * \sin x =$

$(1 / (1 + u^2)) * (1 - \exp(-(u * t)) * (u * \sin t + \cos t))$ (**is** $= ?F t$)

unfolding *zero-ereal-def*

proof (*subst interval-integral-FTC-finite*)

show ($?F \text{ has-vector-derivative } \exp(-(u * x)) * \sin x$ (at x within $\{\text{min } 0 \dots \text{max } 0 \dots t\}$) **for** x)

by (*auto intro!: derivative-eq-intros*)

simp: has-real-derivative-iff-has-vector-derivative[symmetric] power2-eq-square)

(*simp-all add: field-simps add-nonneg-eq-0-iff*)

qed (*auto intro: continuous-at-imp-continuous-on*)

lemma *LBINT-I0i-exp-mscale-sin:*

assumes $0 < x$
shows $LBINT\ u=0..\infty. |exp\ (-u * x) * sin\ x| = |sin\ x| / x$
proof (*subst interval-integral-FTC-nonneg*)
let $?F = \lambda u. 1 / x * (1 - exp\ (-u * x)) * |sin\ x|$
show $\bigwedge t. (?F\ has-real-derivative\ |exp\ (-t * x) * sin\ x|)\ (at\ t)$
using $\langle 0 < x \rangle$ **by** (*auto intro!: derivative-eq-intros simp: abs-mult*)
show $((?F\ \circ\ real-of-ereal) \longrightarrow 0)\ (at-right\ 0)$
using $\langle 0 < x \rangle$ **by** (*auto simp: zero-ereal-def ereal-tendsto-simps intro!: tendsto-eq-intros*)
have $*$: $((\lambda t. exp\ (-t * x)) \longrightarrow 0)\ at-top$
using $\langle 0 < x \rangle$
by (*auto intro!: exp-at-bot[THEN filterlim-compose] filterlim-tendsto-pos-mult-at-top filterlim-ident*
simp: filterlim-uminus-at-bot mult.commute[of - x])
show $((?F\ \circ\ real-of-ereal) \longrightarrow |sin\ x| / x)\ (at-left\ \infty)$
using $\langle 0 < x \rangle$ **unfolding** *ereal-tendsto-simps*
by (*intro filterlim-compose[OF - *]) (auto intro!: tendsto-eq-intros filterlim-ident)*
qed *auto*

lemma

shows *integrable-inverse-1-plus-square:*
set-integrable lborel (einterval $(-\infty)\ \infty)$ $(\lambda x. inverse\ (1 + x^2))$
and *LBINT-inverse-1-plus-square:*
 $LBINT\ x=-\infty..\infty. inverse\ (1 + x^2) = pi$
proof –
have 1 : $-(pi / 2) < x \implies x * 2 < pi \implies (tan\ has-real-derivative\ 1 + (tan\ x)^2)\ (at\ x)$ **for** x
using *cos-gt-zero-pi[of x]* **by** (*subst tan-sec*) (*auto intro!: DERIV-tan simp: power-inverse*)
have 2 : $-(pi / 2) < x \implies x * 2 < pi \implies isCont\ (\lambda x. 1 + (tan\ x)^2)\ x$ **for** x
using *cos-gt-zero-pi[of x]* **by** *auto*
have 3 : $\llbracket -(pi / 2) < x; x * 2 < pi \rrbracket \implies isCont\ (\lambda x. inverse\ (1 + x^2))\ (tan\ x)$ **for** x
by (*rule continuous-intros | simp add: add-nonneg-eq-0-iff*) +
show $LBINT\ x=-\infty..\infty. inverse\ (1 + x^2) = pi$
by (*subst interval-integral-substitution-nonneg[of $-pi/2\ pi/2\ tan\ \lambda x. 1 + (tan\ x)^2$]*)
(auto intro: derivative-eq-intros 1 2 3 filterlim-tan-at-right
simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff
set-integrable-def)
show *set-integrable lborel (einterval $(-\infty)\ \infty)$ $(\lambda x. inverse\ (1 + x^2))$*
by (*subst interval-integral-substitution-nonneg[of $-pi/2\ pi/2\ tan\ \lambda x. 1 + (tan\ x)^2$]*)
(auto intro: derivative-eq-intros 1 2 3 filterlim-tan-at-right
simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff
set-integrable-def)

qed

lemma

shows *integrable-I0i-1-div-plus-square*:

interval-lebesgue-integrable lborel $0 \infty (\lambda x. 1 / (1 + x^2))$

and *LBINT-I0i-1-div-plus-square*:

LBINT $x=0..\infty. 1 / (1 + x^2) = \text{pi} / 2$

proof –

have 1: $0 < x \implies x * 2 < \text{pi} \implies (\text{tan has-real-derivative } 1 + (\text{tan } x)^2) \text{ (at } x)$

for x

using *cos-gt-zero-pi*[of x] by (*subst tan-sec*) (*auto intro!*: *DERIV-tan simp: power-inverse*)

have 2: $0 < x \implies x * 2 < \text{pi} \implies \text{isCont } (\lambda x. 1 + (\text{tan } x)^2) \text{ } x \text{ for } x$

using *cos-gt-zero-pi*[of x] by *auto*

show *LBINT* $x=0..\infty. 1 / (1 + x^2) = \text{pi} / 2$

by (*subst interval-integral-substitution-nonneg*[of $0 \text{ pi}/2 \text{ tan } \lambda x. 1 + (\text{tan } x)^2$])

(*auto intro: derivative-eq-intros 1 2 tendsto-eq-intros*

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff

set-integrable-def)

show *interval-lebesgue-integrable lborel* $0 \infty (\lambda x. 1 / (1 + x^2))$

unfolding *interval-lebesgue-integrable-def*

by (*subst interval-integral-substitution-nonneg*[of $0 \text{ pi}/2 \text{ tan } \lambda x. 1 + (\text{tan } x)^2$])

(*auto intro: derivative-eq-intros 1 2 tendsto-eq-intros*

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff

set-integrable-def)

qed

17 The sinc function, and the sine integral (Si)

abbreviation *sinc* :: *real* \Rightarrow *real* where

sinc $\equiv (\lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } \text{sin } x / x)$

lemma *sinc-at-0*: $((\lambda x. \text{sin } x / x :: \text{real}) \longrightarrow 1) \text{ (at } 0)$

using *DERIV-sin* [of 0] by (*auto simp add: has-field-derivative-def field-has-derivative-at*)

lemma *isCont-sinc*: *isCont* *sinc* x

proof cases

assume $x = 0$ then show ?thesis

using *LIM-equal* [where $g = \lambda x. \text{sin } x / x$ and $a=0$ and $f=\text{sinc}$ and $l=1$]

by (*auto simp: isCont-def sinc-at-0*)

next

assume $x \neq 0$ show ?thesis

by (*rule continuous-transform-within* [where $d = \text{abs } x$ and $f = \lambda x. \text{sin } x / x$])

(*auto simp add: dist-real-def ‹x ≠ 0›*)

qed

lemma *continuous-on-sinc*[*continuous-intros*]:

continuous-on $S f \implies \text{continuous-on } S (\lambda x. \text{sinc } (f x))$

using *continuous-on-compose*[of $S f \text{ sinc}$, *OF - continuous-at-imp-continuous-on*]
by (*auto simp: isCont-sinc*)

lemma *borel-measurable-sinc*[*measurable*]: $\text{sinc} \in \text{borel-measurable borel}$
by (*intro borel-measurable-continuous-onI continuous-at-imp-continuous-on ballI isCont-sinc*)

lemma *sinc-AE*: $\text{AE } x \text{ in } \text{lborel. } \sin x / x = \text{sinc } x$
by (*rule AE-I [where $N = \{0\}$], auto*)

definition $\text{Si} :: \text{real} \Rightarrow \text{real}$ **where** $\text{Si } t \equiv \text{LBINT } x=0..t. \sin x / x$

lemma *sinc-neg* [*simp*]: $\text{sinc } (-x) = \text{sinc } x$
by *auto*

lemma *Si-alt-def* : $\text{Si } t = \text{LBINT } x=0..t. \text{sinc } x$

proof *cases*

assume $0 \leq t$ **then show** *?thesis*

using *AE-lborel-singleton*[of 0]

by (*auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE*)

next

assume $\neg 0 \leq t$ **then show** *?thesis*

unfolding *Si-def* **using** *AE-lborel-singleton*[of 0]

by (*subst (1 2) interval-integral-endpoints-reverse*)

(*auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE*)

qed

lemma *Si-neg*:

assumes $T \geq 0$ **shows** $\text{Si } (-T) = - \text{Si } T$

proof $-$

have $\text{LBINT } x=\text{ereal } 0..T. -1 *_R \text{sinc } (-x) = \text{LBINT } y=\text{ereal } (-0).. \text{ereal } (-T). \text{sinc } y$

by (*rule interval-integral-substitution-finite [OF assms]*)

(*auto intro: derivative-intros continuous-at-imp-continuous-on isCont-sinc*)

also have $(\text{LBINT } x=\text{ereal } 0..T. -1 *_R \text{sinc } (-x)) = -(\text{LBINT } x=\text{ereal } 0..T. \text{sinc } x)$

by (*subst sinc-neg*) (*simp-all add: interval-lebesgue-integral-uminus*)

finally have $*$: $-(\text{LBINT } x=\text{ereal } 0..T. \text{sinc } x) = \text{LBINT } y=\text{ereal } 0.. \text{ereal } (-T). \text{sinc } y$

by *simp*

show *?thesis*

using *assms* **unfolding** *Si-alt-def*

by (*subst zero-ereal-def*) $+$ (*auto simp add: * [symmetric]*)

qed

lemma *integrable-sinc'*:

*interval-lebesgue-integrable lborel (ereal 0) (ereal T) ($\lambda t. \sin (t * \vartheta) / t$)*

proof $-$

have *: *interval-lebesgue-integrable lborel (ereal 0) (ereal T) ($\lambda t. \vartheta * \text{sinc } (t * \vartheta)$)*
by (*intro interval-lebesgue-integrable-mult-right interval-integrable-isCont continuous-within-compose3 [OF isCont-sinc]*)
auto
show ?thesis
by (*rule interval-lebesgue-integrable-cong-AE[THEN iffD1, OF - - - *]*)
(insert AE-lborel-singleton[of 0], auto)
qed

lemma *DERIV-Si: (Si has-real-derivative sinc x) (at x)*

proof –

have (*at x within {min 0 (x - 1)..max 0 (x + 1)}*) = *at x*
by (*intro at-within-interior*) *auto*
moreover **have** *min 0 (x - 1) ≤ x x ≤ max 0 (x + 1)*
by *auto*
ultimately show ?thesis
using *interval-integral-FTC2[of min 0 (x - 1) 0 max 0 (x + 1) sinc x]*
by (*auto simp: continuous-at-imp-continuous-on isCont-sinc Si-alt-def[abs-def]*)
zero-ereal-def
has-real-derivative-iff-has-vector-derivative
split del: if-split
qed

lemma *isCont-Si: isCont Si x*

using *DERIV-Si by (rule DERIV-isCont)*

lemma *borel-measurable-Si[measurable]: Si ∈ borel-measurable borel*

by (*auto intro: isCont-Si continuous-at-imp-continuous-on borel-measurable-continuous-onI*)

lemma *Si-at-top-LBINT:*

*(($\lambda t. (LBINT x=0..∞. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))$) → 0) at-top*

proof –

let ?F = *$\lambda x t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2) :: real$*
have *int: set-integrable lborel {0<..} ($\lambda x. \exp(-x) * (x + 1) :: real$)*
unfolding *distrib-left*
using *has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]*
by (*intro set-integral-add*) (*auto dest!: integrable.intros simp: ac-simps set-integrable-def*)

have (*$(\lambda t::real. LBINT x:\{0<..\}. ?F x t) → LBINT x::real:\{0<..\}. 0$*) *at-top*

unfolding *set-lebesgue-integral-def*

proof (*rule integral-dominated-convergence-at-top[OF - - int [unfolded set-integrable-def]],*

simp-all del: abs-divide split: split-indicator)

have *: *$0 < x \implies |x * \sin t + \cos t| / (1 + x^2) \leq (x * 1 + 1) / 1$* **for** *x t :: real*

```

    by (intro frac-le abs-triangle-ineq[THEN order-trans] add-mono)
      (auto simp add: abs-mult simp del: mult-1-right intro!: mult-mono)
  then have  $1 \leq t \implies 0 < x \implies |\int F x t| \leq \exp(-x) * (x + 1)$  for  $x t :: real$ 
    by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
      intro!: mult-mono)
  then show  $\forall_F i$  in at-top.  $AE x$  in lborel.  $0 < x \implies |\int F x i| \leq \exp(-x) * (x + 1)$ 
    by (auto intro: eventually-mono eventually-ge-at-top[of 1::real])
  show  $AE x$  in lborel.  $0 < x \implies (\int F x \longrightarrow 0)$  at-top
  proof (intro always-eventually impI allI)
    fix  $x :: real$  assume  $0 < x$ 
    show  $((\lambda t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2)) \longrightarrow 0)$  at-top
    proof (rule Lim-null-comparison)
      show  $\forall_F t$  in at-top.  $norm (\int F x t) \leq \exp(-(x * t)) * (x + 1)$ 
        using eventually-ge-at-top[of 1::real] *  $\langle 0 < x \rangle$ 
      by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
        intro!: mult-mono elim: eventually-mono)
    show  $((\lambda t. \exp(-(x * t)) * (x + 1)) \longrightarrow 0)$  at-top
      by (auto simp: filterlim-uminus-at-top [symmetric]
        intro!: filterlim-tendsto-pos-mult-at-top[OF tendsto-const]  $\langle 0 < x \rangle$ )
    filterlim-ident
      tendsto-mult-left-zero filterlim-compose[OF exp-at-bot])
  qed
  qed
  qed
  then show  $((\lambda t. (LBINT x=0..\infty. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))) \longrightarrow 0)$  at-top
    by (simp add: interval-lebesgue-integral-0-infnty set-lebesgue-integral-def)
  qed

lemma Si-at-top-integrable:
  assumes  $t \geq 0$ 
  shows interval-lebesgue-integrable lborel  $0 \infty (\lambda x. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))$ 
  using  $\langle 0 \leq t \rangle$  unfolding le-less
  proof
    assume  $0 = t$  then show ?thesis
      using integrable-I0i-1-div-plus-square by simp
  next
    assume [arith]:  $0 < t$ 
    have *:  $0 \leq a \implies 0 < x \implies a / (1 + x^2) \leq a$  for  $a x :: real$ 
      using zero-le-power2[of x, arith] using divide-left-mono[of 1  $1 + x^2$  a] by auto
    have set-integrable lborel  $\{0 < ..\}$   $(\lambda x. (\exp(-x) * x) * (\sin t / t) + \exp(-x) * \cos t)$ 
      using has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]
    by (intro set-integral-add set-integrable-mult-left)

```

```

    (auto dest!: integrable.intros simp: ac-simps set-integrable-def)
  from lborel-integrable-real-affine[OF this [unfolded set-integrable-def], of t 0]
  show ?thesis
    unfolding interval-lebesgue-integral-0-infty set-integrable-def
    by (rule Bochner-Integration.integrable-bound) (auto simp: field-simps * split:
split-indicator)
qed

```

lemma *Si-at-top*: $(Si \longrightarrow pi / 2)$ at-top

proof –

```

  have †:  $\forall_F t$  in at-top.  $pi / 2 - (LBINT u=0..\infty. exp \ (-(u * t)) * (u * sin t + cos t) / (1 + u^2)) = Si t$ 

```

```

  using eventually-ge-at-top[of 0]

```

proof eventually-elim

```

  fix t :: real assume t ≥ 0

```

```

  have Si t = LBINT x=0..t. sin x * (LBINT u=0..\infty. exp \ -(u * x))

```

```

  unfolding Si-def using ‹0 ≤ t›

```

```

  by (intro interval-integral-discrete-difference[where X={0}]) (auto simp:
LBINT-I0i-exp-mscale)

```

```

  also have ... = LBINT x. (LBINT u=ereal 0..\infty. indicator {0 <..< t} x *R
sin x * exp \ -(u * x))

```

```

  using ‹0 ≤ t› by (simp add: zero-ereal-def interval-lebesgue-integral-le-eq
mult-ac set-lebesgue-integral-def)

```

```

  also have ... = LBINT x. (LBINT u. indicator ({0 <..} × {0 <..< t}) (u, x)
*_R (sin x * exp \ -(u * x)))

```

```

  apply (subst interval-integral-I0i)

```

```

  unfolding set-lebesgue-integral-def by (simp-all add: indicator-times ac-simps
)

```

```

  also have ... = LBINT u. (LBINT x. indicator ({0 <..} × {0 <..< t}) (u, x)
*_R (sin x * exp \ -(u * x)))

```

```

  proof (intro lborel-pair.Fubini-integral[symmetric] lborel-pair.Fubini-integrable)

```

```

  show  $(\lambda(x, y). indicator \ ({0 <..} \times \ {0 <..< t}) (y, x) *_R (sin x * exp \ -(y
* x)))$ 

```

```

  ∈ borel-measurable (lborel  $\otimes_M$  lborel) (is ?f ∈ borel-measurable -)

```

```

  by measurable

```

```

  have AE x in lborel. indicator {0..t} x *_R abs (sinc x) = LBINT y. norm (?f
(x, y))

```

```

  using AE-lborel-singleton[of 0] AE-lborel-singleton[of t]

```

proof eventually-elim

```

  fix x :: real assume x: x ≠ 0 x ≠ t

```

```

  have LBINT y. |indicator \ ({0 <..} × \ {0 <..< t}) (y, x) *_R (sin x * exp \ -(y
* x))| =

```

```

  LBINT y. |sin x| * exp \ -(y * x) * indicator {0 <..} y * indicator
\ {0 <..< t} x

```

```

  by (intro Bochner-Integration.integral-cong) (auto split: split-indicator
simp: abs-mult)

```

```

  also have ... = |sin x| * indicator {0 <..< t} x * (LBINT y=0..\infty. exp \ -(y
* x))

```



```

    by (cases x > 0)
      (auto simp: field-simps interval-lebesgue-integral-0-infty set-lebesgue-integral-def
split: split-indicator)
    also have ... = |sin x| * indicator {0 <..<t} x * (1 / x)
      by (cases x > 0) (auto simp add: LBINT-I0i-exp-mscale)
    also have ... = indicator {0..t} x *R |sinc x|
      using x by (simp add: field-simps split: split-indicator)
    finally show indicator {0..t} x *R abs (sinc x) = LBINT y. norm (?f (x,
y))
      by simp
  qed
  moreover have integrable lborel (λx. indicat-real {0..t} x *R |sinc x|)
  by (auto intro!: borel-integrable-compact continuous-intros simp del: real-scaleR-def)
  ultimately show integrable lborel (λx. LBINT y. norm (?f (x, y)))
    by (rule integrable-cong-AE-imp[rotated 2]) simp

  have 0 < x ⇒ set-integrable lborel {0<..} (λy. sin x * exp (-(y * x))) for
x :: real
    by (intro set-integrable-mult-right integrable-I0i-exp-mscale)
  then show AE x in lborel. integrable lborel (λy. ?f (x, y))
  by (intro AE-I2) (auto simp: indicator-times set-integrable-def split: split-indicator)
  qed
  also have ... = LBINT u=0..∞. (LBINT x=0..t. exp (-(u * x)) * sin x)
    using <0≤t>
  by (auto simp: interval-lebesgue-integral-def set-lebesgue-integral-def zero-ereal-def
ac-simps
      split: split-indicator intro!: Bochner-Integration.integral-cong)
  also have ... = LBINT u=0..∞. 1 / (1 + u2) - 1 / (1 + u2) * (exp (-(u
* t)) * (u * sin t + cos t))
  by (auto simp: divide-simps LBINT-I0c-exp-mscale-sin intro!: interval-integral-cong)
  also have ... = pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos
t) / (1 + u2))
  using Si-at-top-integrable[OF <0≤t>] by (simp add: integrable-I0i-1-div-plus-square
LBINT-I0i-1-div-plus-square)
  finally show pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos t)
/ (1 + u2)) = Si t ..
  qed
  show ?thesis
  by (rule Lim-transform-eventually [OF - †])
    (auto intro!: tendsto-eq-intros Si-at-top-LBINT)
  qed

```

17.1 The final theorems: boundedness and scalability

lemma *bounded-Si*: $\exists B. \forall T. |Si T| \leq B$

proof –

```

  have *: 0 ≤ y ⇒ dist x y < z ⇒ abs x ≤ y + z for x y z :: real
  by (simp add: dist-real-def)

```

have eventually $(\lambda T. \text{dist } (Si\ T) \ (pi / 2) < 1)$ *at-top*
using *Si-at-top* **by** *(elim tendstoD) simp*
then have eventually $(\lambda T. 0 \leq T \wedge |Si\ T| \leq pi / 2 + 1)$ *at-top*
using *eventually-ge-at-top[of 0]* **by** *eventually-elim (insert *[of pi/2 Si - 1], auto)*
then have $\exists T. 0 \leq T \wedge (\forall t \geq T. |Si\ t| \leq pi / 2 + 1)$
by *(auto simp add: eventually-at-top-linorder)*
then obtain T **where** $T: 0 \leq T \wedge t \geq T \implies |Si\ t| \leq pi / 2 + 1$
by *auto*
moreover have $t \leq -T \implies |Si\ t| \leq pi / 2 + 1$ **for** t
using $T(1)\ T(2)[of -t]\ Si\ neg[of -t]$ **by** *simp*
moreover have $\exists M. \forall t. -T \leq t \wedge t \leq T \longrightarrow |Si\ t| \leq M$
by *(rule isCont-bounded) (auto intro!: isCont-Si continuous-intros <0 ≤ T>)*
then obtain M **where** $M: \bigwedge t. -T \leq t \wedge t \leq T \implies |Si\ t| \leq M$
by *auto*
ultimately show *?thesis*
by *(intro exI[of - max M (pi/2 + 1)]) (meson le-max-iff-disj linorder-not-le order-le-less)*
qed

lemma *LBINT-I0c-sin-scale-divide:*

assumes $T \geq 0$
shows $LBINT\ t=0..T. \sin (t * \vartheta) / t = \text{sgn } \vartheta * Si (T * |\vartheta|)$
proof –
{ assume $0 < \vartheta$
have $(LBINT\ t=ereal\ 0..T. \sin (t * \vartheta) / t) = (LBINT\ t=ereal\ 0..T. \vartheta *_{\mathbb{R}} \text{sinc } (t * \vartheta))$
by *(rule interval-integral-discrete-difference[of {0}]) auto*
also have $\dots = (LBINT\ t=ereal\ (0 * \vartheta)..T * \vartheta. \text{sinc } t)$
apply *(rule interval-integral-substitution-finite [OF assms])*
apply *(subst mult.commute, rule DERIV-subset)*
by *(auto intro!: derivative-intros continuous-at-imp-continuous-on isCont-sinc)*
also have $\dots = (LBINT\ t=ereal\ (0 * \vartheta)..T * \vartheta. \sin\ t / t)$
by *(rule interval-integral-discrete-difference[of {0}]) auto*
finally have $(LBINT\ t=ereal\ 0..T. \sin (t * \vartheta) / t) = (LBINT\ t=ereal\ 0..T * \vartheta. \sin\ t / t)$
by *simp*
hence $LBINT\ x. \text{indicator } \{0 <.. < T\} x * \sin (x * \vartheta) / x =$
 $LBINT\ x. \text{indicator } \{0 <.. < T * \vartheta\} x * \sin\ x / x$
using *assms <0 < \vartheta>* **unfolding** *interval-lebesgue-integral-def einterval-eq zero-ereal-def*
by *(auto simp: ac-simps set-lebesgue-integral-def)*
} note *aux1 = this*
{ assume $0 > \vartheta$
have *[simp]: integrable lborel* $(\lambda x. \sin (x * \vartheta) * \text{indicator } \{0 <.. < T\} x / x)$
using *integrable-sinc' [of T \vartheta] assms*
by *(simp add: interval-lebesgue-integrable-def set-integrable-def ac-simps)*
have $(LBINT\ t=ereal\ 0..T. \sin (t * -\vartheta) / t) = (LBINT\ t=ereal\ 0..T. -\vartheta *_{\mathbb{R}} \text{sinc } (t * -\vartheta))$

```

    by (rule interval-integral-discrete-difference[of {0}]) auto
  also have ... = (LBINT t=ereal (0 * -∅)..T * -∅. sinc t)
    apply (rule interval-integral-substitution-finite [OF assms])
    apply (subst mult.commute, rule DERIV-subset)
  by (auto intro!: derivative-intros continuous-at-imp-continuous-on isCont-sinc)
  also have ... = (LBINT t=ereal (0 * -∅)..T * -∅. sin t / t)
    by (rule interval-integral-discrete-difference[of {0}]) auto
  finally have (LBINT t=ereal 0..T. sin (t * -∅) / t) = (LBINT t=ereal 0..T
* -∅. sin t / t)
    by simp
  hence LBINT x. indicator {0<..

```

18 The Levy inversion theorem, and the Levy continuity theorem.

```

theory Levy
  imports Characteristic-Functions Helly-Selection Sinc-Integral
begin

```

18.1 The Levy inversion theorem

```

lemma Levy-Inversion-aux1:

```

```

  fixes a b :: real
  assumes a ≤ b
  shows ((λt. (iexp (-(t * a)) - iexp (-(t * b))) / (i * t)) ⟶ b - a) (at 0)
    (is (?F ⟶ -) (at -))
proof -
  have 1: cmod (?F t - (b - a)) ≤ a2 / 2 * abs t + b2 / 2 * abs t if t ≠ 0
for t
proof -
  have cmod (?F t - (b - a)) = cmod (
    (iexp (-(t * a)) - (1 + i * -(t * a))) / (i * t) -
    (iexp (-(t * b)) - (1 + i * -(t * b))) / (i * t))
    (is - = cmod (?one / (i * t) - ?two / (i * t)))
  )
  using ⟨t ≠ 0⟩ by (intro arg-cong[where f=norm]) (simp add: field-simps)

```

```

also have ... ≤ cmod (?one / (i * t)) + cmod (?two / (i * t))
  by (rule norm-triangle-ineq4)
also have cmod (?one / (i * t)) = cmod ?one / abs t
  by (simp add: norm-divide norm-mult)
also have cmod (?two / (i * t)) = cmod ?two / abs t
  by (simp add: norm-divide norm-mult)
also have cmod ?one / abs t + cmod ?two / abs t ≤
  ((- (a * t))^2 / 2) / abs t + ((- (b * t))^2 / 2) / abs t
  apply (rule add-mono)
  apply (rule divide-right-mono)
using iexp-approx1 [of -(t * a) 1] apply (simp add: field-simps eval-nat-numeral)
  apply force
  apply (rule divide-right-mono)
using iexp-approx1 [of -(t * b) 1] apply (simp add: field-simps eval-nat-numeral)
  by force
also have ... = a^2 / 2 * abs t + b^2 / 2 * abs t
  using ‹t ≠ 0› apply (case-tac t ≥ 0, simp add: field-simps power2-eq-square)
  using ‹t ≠ 0› by (subst (1 2) abs-of-neg, auto simp add: field-simps
power2-eq-square)
  finally show cmod (?F t - (b - a)) ≤ a^2 / 2 * abs t + b^2 / 2 * abs t .
qed
show ?thesis
  apply (rule LIM-zero-cancel)
  apply (rule tendsto-norm-zero-cancel)
  apply (rule real-LIM-sandwich-zero [OF - - 1])
  apply (auto intro!: tendsto-eq-intros)
done
qed

```

lemma *Levy-Inversion-aux2*:

```

fixes a b t :: real
assumes a ≤ b and t ≠ 0
shows cmod ((iexp (t * b) - iexp (t * a)) / (i * t)) ≤ b - a (is ?F ≤ -)
proof -
  have ?F = cmod (iexp (t * a) * (iexp (t * (b - a)) - 1) / (i * t))
    using ‹t ≠ 0› by (intro arg-cong[where f=norm]) (simp add: field-simps
exp-diff exp-minus)
  also have ... = cmod (iexp (t * (b - a)) - 1) / abs t
    unfolding norm-divide norm-mult norm-exp-i-times using ‹t ≠ 0›
    by (simp add: complex-eq-iff norm-mult)
  also have ... ≤ abs (t * (b - a)) / abs t
    using iexp-approx1 [of t * (b - a) 0]
    by (intro divide-right-mono) (auto simp add: field-simps eval-nat-numeral)
  also have ... = b - a
    using assms by (auto simp add: abs-mult)
  finally show ?thesis .
qed

```

theorem (in *real-distribution*) *Levy-Inversion*:

```

fixes  $a\ b :: \text{real}$ 
assumes  $a \leq b$ 
defines  $\mu \equiv \text{measure } M$  and  $\varphi \equiv \text{char } M$ 
assumes  $\mu \{a\} = 0$  and  $\mu \{b\} = 0$ 
shows  $(\lambda T. 1 / (2 * \text{pi}) * (\text{CLBINT } t=-T..T. (\text{iexp } (-(t * a)) - \text{iexp } (-(t * b)))) / (\text{i} * t) * \varphi t)$ 
   $\longrightarrow \mu \{a<..b\}$ 
  (is  $(\lambda T. 1 / (2 * \text{pi}) * (\text{CLBINT } t=-T..T. ?F t * \varphi t)) \longrightarrow \text{of-real } (\mu \{a<..b\})$ )
proof -
interpret  $P$ : pair-sigma-finite lborel  $M ..$ 
from bounded-Si obtain  $B$  where  $B\text{prop}$ :  $\bigwedge T. \text{abs } (Si\ T) \leq B$  by auto
from  $B\text{prop}$  [of 0] have [simp]:  $B \geq 0$  by auto
let  $?f = \lambda t\ x :: \text{real}. (\text{iexp } (t * (x - a)) - \text{iexp } (t * (x - b))) / (\text{i} * t)$ 
{ fix  $T :: \text{real}$ 
  assume  $T \geq 0$ 
  let  $?f' = \lambda(t, x). \text{indicator } \{-T<..<T\} t *_R ?f t x$ 
  { fix  $x$ 
    have 1: complex-interval-lebesgue-integrable lborel  $u\ v$   $(\lambda t. ?f t x)$  for  $u\ v :: \text{real}$ 
    using Levy-Inversion-aux2 [of  $x - b\ x - a$ ]
    apply (simp add: interval-lebesgue-integrable-def set-integrable-def del: times-divide-eq-left)
    apply (intro integrableI-bounded-set-indicator [where  $B=b - a$ ] conjI impI)
    apply (auto intro!: AE-I [of - - {0}] simp: assms)
    done
    have  $(\text{CLBINT } t. ?f' (t, x)) = (\text{CLBINT } t=-T..T. ?f t x)$ 
    using  $\langle T \geq 0 \rangle$  by (simp add: interval-lebesgue-integral-def set-lebesgue-integral-def)
    also have  $\dots = (\text{CLBINT } t=-T..(0 :: \text{real}). ?f t x) + (\text{CLBINT } t=(0 :: \text{real})..T. ?f t x)$ 
    (is  $- = - + ?t$ )
    using 1 by (intro interval-integral-sum [symmetric]) (simp add: min-absorb1 max-absorb2  $\langle T \geq 0 \rangle$ )
    also have  $(\text{CLBINT } t=-T..(0 :: \text{real}). ?f t x) = (\text{CLBINT } t=(0::\text{real})..T. ?f (-t) x)$ 
    by (subst interval-integral-reflect) auto
    also have  $\dots + ?t = (\text{CLBINT } t=(0::\text{real})..T. ?f (-t) x + ?f t x)$ 
    using 1
    by (intro interval-lebesgue-integral-add (2) [symmetric] interval-integrable-mirror [THEN iffD2]) simp-all
    also have  $\dots = (\text{CLBINT } t=(0::\text{real})..T. ((\text{iexp } (t * (x - a)) - \text{iexp } (-(t * (x - a)))) - (\text{iexp } (t * (x - b)) - \text{iexp } (-(t * (x - b))))) / (\text{i} * t)$ 
    using  $\langle T \geq 0 \rangle$  by (intro interval-integral-cong) (auto simp add: field-split-simps)
    also have  $\dots = (\text{CLBINT } t=(0::\text{real})..T. \text{complex-of-real} (2 * (\text{sin } (t * (x - a)) / t) - 2 * (\text{sin } (t * (x - b)) / t)))$ 
    using  $\langle T \geq 0 \rangle$ 
    apply (intro interval-integral-cong)

```

```

apply (simp add: field-simps cis.ctr Im-divide Re-divide Im-exp Re-exp
complex-eq-iff)
unfolding minus-diff-eq[symmetric, of y * x y * a for y a] sin-minus
cos-minus
apply (simp add: field-simps power2-eq-square)
done
also have ... = complex-of-real (LBINT t=(0::real)..T.
  2 * (sin (t * (x - a)) / t) - 2 * (sin (t * (x - b)) / t))
by (rule interval-lebesgue-integral-of-real)
also have ... = complex-of-real (2 * (sgn (x - a) * Si (T * abs (x - a)) -
  sgn (x - b) * Si (T * abs (x - b))))
apply (subst interval-lebesgue-integral-diff)
apply (rule interval-lebesgue-integrable-mult-right, rule integrable-sinc)+
apply (subst interval-lebesgue-integral-mult-right)+
apply (simp add: zero-ereal-def[symmetric] LBINT-I0c-sin-scale-divide[OF
<T ≥ 0>])
done
finally have (CLBINT t. ?f' (t, x)) =
  2 * (sgn (x - a) * Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x
- b))) .
} note main-eq = this
have (CLBINT t=-T..T. ?F t * φ t) =
  (CLBINT t. (CLINT x | M. ?F t * iexp (t * x) * indicator {-T<..using <T ≥ 0> unfolding φ-def char-def interval-lebesgue-integral-def set-lebesgue-integral-def
by (auto split: split-indicator intro!: Bochner-Integration.integral-cong)
also have ... = (CLBINT t. (CLINT x | M. ?f' (t, x)))
by (auto intro!: Bochner-Integration.integral-cong simp: field-simps exp-diff
exp-minus split: split-indicator)
also have ... = (CLINT x | M. (CLBINT t. ?f' (t, x)))
proof (intro P.Fubini-integral [symmetric] integrableI-bounded-set [where B=b
- a])
show emeasure (lborel ⊗M M) ({- T<..using <T ≥ 0>
by (subst emeasure-pair-measure-Times)
  (auto simp: ennreal-mult-less-top not-less top-unique)
show AE x∈{- T<..M M. cmod (case x of (t,
x) ⇒ ?f' (t, x)) ≤ b - a
using Levy-Inversion-aux2[of x - b x - a for x] <a ≤ b>
by (intro AE-I [of - - {0} × UNIV]) (force simp: emeasure-pair-measure-Times)+
qed (auto split: split-indicator split-indicator-asm)
also have ... = (CLINT x | M. (complex-of-real (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b))))))
using main-eq by (intro Bochner-Integration.integral-cong, auto)
also have ... = complex-of-real (LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b))))))
by (rule integral-complex-of-real)
finally have (CLBINT t=-T..T. ?F t * φ t) =
  complex-of-real (LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))))) .

```

```

} note main-eq2 = this

have (λT :: nat. LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))) →
  (LINT x | M. 2 * pi * indicator {a<..b} x)
proof (rule integral-dominated-convergence [where w=λx. 4 * B])
  show integrable M (λx. 4 * B)
  by (rule integrable-const-bound [of - 4 * B]) auto
next
let ?S = λn::nat. λx. sgn (x - a) * Si (n * |x - a|) - sgn (x - b) * Si (n *
|x - b|)
{ fix n x
  have norm (?S n x) ≤ norm (sgn (x - a) * Si (n * |x - a|)) + norm (sgn
(x - b) * Si (n * |x - b|))
  by (rule norm-triangle-ineq4)
  also have ... ≤ B + B
  using Bprop by (intro add-mono) (auto simp: abs-mult abs-sgn-eq)
  finally have norm (2 * ?S n x) ≤ 4 * B
  by simp }
then show ∧n. AE x in M. norm (2 * ?S n x) ≤ 4 * B
  by auto
have AE x in M. x ≠ a AE x in M. x ≠ b
  using prob-eq-0[of {a}] prob-eq-0[of {b}] ⟨μ {a} = 0⟩ ⟨μ {b} = 0⟩ by (auto
simp: μ-def)
then show AE x in M. (λn. 2 * ?S n x) → 2 * pi * indicator {a<..b} x
proof eventually-elim
  fix x assume x: x ≠ a x ≠ b
  then have (λn. 2 * (sgn (x - a) * Si (|x - a| * n) - sgn (x - b) * Si (|x
- b| * n)))
    → 2 * (sgn (x - a) * (pi / 2) - sgn (x - b) * (pi / 2))
  by (intro tendsto-intros filterlim-compose[OF Si-at-top]
  filterlim-tendsto-pos-mult-at-top[OF tendsto-const] filterlim-real-sequentially)
  auto
  also have (λn. 2 * (sgn (x - a) * Si (|x - a| * n) - sgn (x - b) * Si (|x
b| * n))) = (λn. 2 * ?S n x)
  by (auto simp: ac-simps)
  also have 2 * (sgn (x - a) * (pi / 2) - sgn (x - b) * (pi / 2)) = 2 * pi *
indicator {a<..b} x
  using x ⟨a ≤ b⟩ by (auto split: split-indicator)
  finally show (λn. 2 * ?S n x) → 2 * pi * indicator {a<..b} x .
qed
qed simp-all
also have (LINT x | M. 2 * pi * indicator {a<..b} x) = 2 * pi * μ {a<..b}
  by (simp add: μ-def)
finally have (λT. LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))) →
  2 * pi * μ {a<..b} .
with main-eq2 show ?thesis
  by (auto intro!: tendsto-eq-intros)

```

qed

theorem *Levy-uniqueness:*

fixes $M1\ M2 :: \text{real measure}$

assumes *real-distribution* $M1$ *real-distribution* $M2$ **and**

$\text{char } M1 = \text{char } M2$

shows $M1 = M2$

proof –

interpret $M1$: *real-distribution* $M1$ **by** (*rule assms*)

interpret $M2$: *real-distribution* $M2$ **by** (*rule assms*)

have *countable* ($\{x. \text{measure } M1 \{x\} \neq 0\} \cup \{x. \text{measure } M2 \{x\} \neq 0\}$)

by (*intro countable-Un* $M2.\text{countable-support } M1.\text{countable-support}$)

then have *count*: *countable* $\{x. \text{measure } M1 \{x\} \neq 0 \vee \text{measure } M2 \{x\} \neq 0\}$

by (*simp add: Un-def*)

have $\text{cdf } M1 = \text{cdf } M2$

proof (*rule ext*)

fix x

let $?D = \lambda x. |\text{cdf } M1\ x - \text{cdf } M2\ x|$

{ **fix** $\varepsilon :: \text{real}$

assume $\varepsilon > 0$

have ($?D \longrightarrow 0$) *at-bot*

using $M1.\text{cdf-lim-at-bot } M2.\text{cdf-lim-at-bot}$ **by** (*intro tendsto-eq-intros*) *auto*

then have *eventually* ($\lambda y. ?D\ y < \varepsilon / 2 \wedge y \leq x$) *at-bot*

using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def diff-0-right eventually-conj eventually-le-at-bot abs-idempotent*)

then obtain M **where** $\bigwedge y. y \leq M \implies ?D\ y < \varepsilon / 2 \wedge M \leq x$

unfolding *eventually-at-bot-linorder* **by** *auto*

with *open-minus-countable*[*OF count*, of $\{.. < M\}$] **obtain** a **where**

$\text{measure } M1\ \{a\} = 0 \text{ measure } M2\ \{a\} = 0 \wedge a < M \wedge a \leq x$ **and** $1: ?D\ a < \varepsilon$

$/ 2$

by *auto*

have ($?D \longrightarrow ?D\ x$) *(at-right x)*

using $M1.\text{cdf-is-right-cont } [of\ x] \ M2.\text{cdf-is-right-cont } [of\ x]$

by (*intro tendsto-intros*) (*auto simp add: continuous-within*)

then have *eventually* ($\lambda y. |?D\ y - ?D\ x| < \varepsilon / 2$) *(at-right x)*

using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def eventually-conj eventually-at-right-less*)

then obtain N **where** $N > x \wedge \lambda y. x < y \implies y < N \implies |?D\ y - ?D\ x| < \varepsilon / 2$

by (*auto simp add: eventually-at-right*[*OF less-add-one*])

with *open-minus-countable*[*OF count*, of $\{x <.. < N\}$] **obtain** b **where** $x < b < N$

$\text{measure } M1\ \{b\} = 0 \text{ measure } M2\ \{b\} = 0$ **and** $2: |?D\ b - ?D\ x| < \varepsilon / 2$

by (*auto simp: abs-minus-commute*)

from $\langle a \leq x \rangle \langle x < b \rangle$ **have** $a < b \wedge a \leq b$ **by** *auto*


```

from ⟨char M1 = char M2⟩
  M1.Levy-Inversion [OF ⟨a ≤ b⟩ ⟨measure M1 {a} = 0⟩ ⟨measure M1 {b}
= 0⟩]
  M2.Levy-Inversion [OF ⟨a ≤ b⟩ ⟨measure M2 {a} = 0⟩ ⟨measure M2 {b}
= 0⟩]
have complex-of-real (measure M1 {a<..b}) = complex-of-real (measure M2
{a<..b})
  by (intro LIMSEQ-unique) auto
then have ?D a = ?D b
unfolding of-real-eq-iff M1.cdf-diff-eq [OF ⟨a < b⟩, symmetric] M2.cdf-diff-eq
[OF ⟨a < b⟩, symmetric] by simp
then have ?D x = |( ?D b - ?D x) - ?D a|
  by simp
also have ... ≤ |?D b - ?D x| + |?D a|
  by (rule abs-triangle-ineq4)
also have ... ≤ ε / 2 + ε / 2
  using 1 2 by (intro add-mono) auto
finally have ?D x ≤ ε by simp }
then show cdf M1 x = cdf M2 x
  by (metis abs-le-zero-iff dense-ge eq-iff-diff-eq-0)
qed
thus ?thesis
  by (rule cdf-unique [OF ⟨real-distribution M1⟩ ⟨real-distribution M2⟩])
qed

```

18.2 The Levy continuity theorem

theorem *levy-continuity1:*

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes ∧n. real-distribution (M n) real-distribution M' weak-conv-m M M'
shows (λn. char (M n) t) ⟶ char M' t
unfolding char-def using assms by (rule weak-conv-imp-integral-bdd-continuous-conv)
auto

```

theorem *levy-continuity:*

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes real-distr-M : ∧n. real-distribution (M n)
  and real-distr-M': real-distribution M'
  and char-conv: ∧t. (λn. char (M n) t) ⟶ char M' t
shows weak-conv-m M M'

```

proof –

```

interpret Mn: real-distribution M n for n by fact
interpret M': real-distribution M' by fact

```

```

have *: ∧u x. u > 0 ⟹ x ≠ 0 ⟹ (CLBINT t:{-u..u}. 1 - iexp (t * x)) =
  2 * (u - sin (u * x) / x)

```

proof –

```

fix u :: real and x :: real
assume u > 0 and x ≠ 0

```

```

    hence (CLBINT t:{-u..u}. 1 - iexp (t * x)) = (CLBINT t=-u..u. 1 - iexp
(t * x))
      by (subst interval-integral-Icc, auto)
    also have ... = (CLBINT t=-u..0. 1 - iexp (t * x)) + (CLBINT t=0..u. 1
- iexp (t * x))
      using ⟨u > 0⟩
      apply (subst interval-integral-sum)
      apply (simp add: min-absorb1 min-absorb2 max-absorb1 max-absorb2)
      apply (rule interval-integrable-isCont)
      apply auto
      done
    also have ... = (CLBINT t=ereal 0..u. 1 - iexp (t * -x)) + (CLBINT t=ereal
0..u. 1 - iexp (t * x))
      apply (subgoal-tac 0 = ereal 0, erule ssubst)
      by (subst interval-integral-reflect, auto)
    also have ... = (LBINT t=ereal 0..u. 2 - 2 * cos (t * x))
      apply (subst interval-lebesgue-integral-add (2) [symmetric])
      apply ((rule interval-integrable-isCont, auto)+) [2]
      unfolding exp-Euler cos-of-real
      apply (simp add: of-real-mult interval-lebesgue-integral-of-real[symmetric])
      done
    also have ... = 2 * u - 2 * sin (u * x) / x
      by (subst interval-lebesgue-integral-diff)
        (auto intro!: interval-integrable-isCont
          simp: interval-lebesgue-integral-of-real integral-cos [OF ⟨x ≠ 0⟩]
            mult.commute[of - x])
    finally show (CLBINT t:{-u..u}. 1 - iexp (t * x)) = 2 * (u - sin (u * x)
/ x)
      by (simp add: field-simps)
  qed
  have main-bound: ∧u n. u > 0 ⇒ Re (CLBINT t:{-u..u}. 1 - char (M n) t)
≥
  u * measure (M n) {x. abs x ≥ 2 / u}
  proof -
    fix u :: real and n
    assume u > 0
    interpret P: pair-sigma-finite M n lborel ..

    have Mn1 [simp]: measure (M n) UNIV = 1 by (metis Mn.prob-space Mn.space-eq-univ)

    have Mn2 [simp]: ∧x. complex-integrable (M n) (λt. exp (i * complex-of-real
(x * t)))
      by (rule Mn.integrable-const-bound [where B = 1], auto)
    have Mn3: set-integrable (M n ⊗M lborel) (UNIV × {- u..u}) (λa. 1 - exp
(i * complex-of-real (snd a * fst a)))
      using ⟨0 < u⟩
      unfolding set-integrable-def
      by (intro integrableI-bounded-set-indicator [where B=2])
        (auto simp: lborel.emmeasure-pair-measure-Times ennreal-mult-less-top not-less

```

top-unique

split: split-indicator
intro!: order-trans [OF norm-triangle-ineq4])
have (CLBINT $t:\{-u..u\}$. $1 - \text{char } (M \ n) \ t =$
 (CLBINT $t:\{-u..u\}$. (CLINT $x \mid M \ n$. $1 - \text{iexp } (t * x)$)))
unfolding char-def by (rule *set-lebesgue-integral-cong*, *auto simp del: of-real-mult*)
also have ... = (CLBINT t . (CLINT $x \mid M \ n$. *indicator* $\{-u..u\} \ t *_{\mathbb{R}} (1 -$
iexp $(t * x)$)))
unfolding set-lebesgue-integral-def
by (rule *Bochner-Integration.integral-cong*) (*auto split: split-indicator*)
also have ... = (CLINT $x \mid M \ n$. (CLBINT $t:\{-u..u\}$. $1 - \text{iexp } (t * x)$))
using Mn3 by (*subst P.Fubini-integral*) (*auto simp: indicator-times split-beta'*
set-integrable-def set-lebesgue-integral-def)
also have ... = (CLINT $x \mid M \ n$. (*if* $x = 0$ *then* 0 *else* $2 * (u - \sin (u * x)$
 $/ x)$))
using $\langle u > 0 \rangle$ **by** (*intro Bochner-Integration.integral-cong*, *auto simp add: **
simp del: of-real-mult)
also have ... = (LINT $x \mid M \ n$. (*if* $x = 0$ *then* 0 *else* $2 * (u - \sin (u * x) /$
 $x)$))
by (rule *integral-complex-of-real*)
finally have *Re* (CLBINT $t:\{-u..u\}$. $1 - \text{char } (M \ n) \ t =$
 (LINT $x \mid M \ n$. (*if* $x = 0$ *then* 0 *else* $2 * (u - \sin (u * x) / x)$)) **by** *simp*
also have ... \geq (LINT $x : \{x. \text{abs } x \geq 2 / u\} \mid M \ n$. u)
proof -
have *complex-integrable* $(M \ n) (\lambda x. \text{CLBINT } t:\{-u..u\}. 1 - \text{iexp } (\text{snd } (x, t)$
 $* \text{fst } (x, t))$)
using Mn3 unfolding set-integrable-def set-lebesgue-integral-def
by (*intro P.integrable-fst*) (*simp add: indicator-times split-beta'*)
hence *complex-integrable* $(M \ n) (\lambda x. \text{if } x = 0 \text{ then } 0 \text{ else } 2 * (u - \sin (u * x)$
 $/ x)$)
using $\langle u > 0 \rangle$
unfolding set-integrable-def
by (*subst Bochner-Integration.integrable-cong*) (*auto simp add: * simp del:*
of-real-mult)
hence ****: *integrable* $(M \ n) (\lambda x. \text{if } x = 0 \text{ then } 0 \text{ else } 2 * (u - \sin (u * x) /$
 $x)$)
unfolding complex-of-real-integrable-eq .
have $2 * \sin x \leq x$ **if** $2 \leq x$ **for** $x :: \text{real}$
by (rule *order-trans[OF - $\langle 2 \leq x \rangle$]*) *auto*
moreover have $x \leq 2 * \sin x$ **if** $x \leq -2$ **for** $x :: \text{real}$
by (rule *order-trans[OF $\langle x \leq -2 \rangle$]*) *auto*
moreover have $x < 0 \implies x \leq \sin x$ **for** $x :: \text{real}$
using *sin-x-le-x[of -x]* **by** *simp*
ultimately show *?thesis*
using $\langle u > 0 \rangle$ **unfolding set-lebesgue-integral-def**
by (*intro integral-mono [OF - **]*)
 (*auto simp: divide-simps sin-x-le-x mult.commute[of u] mult-neg-pos*
top-unique less-top[symmetric]
split: split-indicator)

```

qed
also (xtrans) have (LINT  $x : \{x. \text{abs } x \geq 2 / u\} \mid M \ n. u = u * \text{measure } (M \ n) \{x. \text{abs } x \geq 2 / u\}$ )
  unfolding set-lebesgue-integral-def
  by (simp add: Mn.emmeasure-eq-measure)
  finally show Re (CLBINT  $t:\{-u..u\}. 1 - \text{char } (M \ n) \ t) \geq u * \text{measure } (M \ n) \{x. \text{abs } x \geq 2 / u\} .$ 
qed

have tight-aux:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \exists a \ b. a < b \wedge (\forall n. 1 - \varepsilon < \text{measure } (M \ n) \{a..b\})$ 
proof -
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  note M'.isCont-char [of 0]
  hence  $\exists d > 0. \forall t. \text{abs } t < d \implies \text{cmod } (\text{char } M' \ t - 1) < \varepsilon / 4$ 
  apply (subst (asm) continuous-at-eps-delta)
  apply (drule-tac  $x = \varepsilon / 4$  in spec)
  using  $\langle \varepsilon > 0 \rangle$  by (auto simp add: dist-real-def dist-complex-def M'.char-zero)
  then obtain  $d$  where  $d > 0 \wedge (\forall t. (\text{abs } t < d \implies \text{cmod } (\text{char } M' \ t - 1) < \varepsilon / 4)) ..$ 
  hence  $d0: d > 0$  and  $d1: \bigwedge t. \text{abs } t < d \implies \text{cmod } (\text{char } M' \ t - 1) < \varepsilon / 4$ 
by auto
  have  $1: \bigwedge x. \text{cmod } (1 - \text{char } M' \ x) \leq 2$ 
  by (rule order-trans [OF norm-triangle-ineq4], auto simp add: M'.cmod-char-le-1)
  then have  $2: \bigwedge u \ v. \text{complex-set-integrable lborel } \{u..v\} (\lambda x. 1 - \text{char } M' \ x)$ 
  unfolding set-integrable-def
  by (intro integrableI-bounded-set-indicator[where  $B=2$ ]) (auto simp: emeasure-lborel-Icc-eq)
  have  $3: \bigwedge u \ v. \text{integrable lborel } (\lambda x. \text{indicat-real } \{u..v\} \ x *_{\mathbb{R}} \text{cmod } (1 - \text{char } M' \ x))$ 
  by (intro borel-integrable-compact[OF compact-Icc] continuous-at-imp-continuous-on continuous-intros ballI M'.isCont-char continuous-intros)
  have  $\text{cmod } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t) \leq \text{LBINT } t:\{-d/2..d/2\}. \text{cmod } (1 - \text{char } M' \ t)$ 
  unfolding set-lebesgue-integral-def
  using integral-norm-bound[of -  $\lambda x. \text{indicator } \{u..v\} \ x *_{\mathbb{R}} (1 - \text{char } M' \ x)$ ]
for  $u \ v$ ] by simp
  also have  $4: \dots \leq \text{LBINT } t:\{-d/2..d/2\}. \varepsilon / 4$ 
  unfolding set-lebesgue-integral-def
  apply (rule integral-mono [OF 3])
  apply (simp add: emeasure-lborel-Icc-eq)
  apply (case-tac  $x \in \{-d/2..d/2\}$ )
  apply auto
  apply (subst norm-minus-commute)
  apply (rule less-imp-le)
  apply (rule d1 [simplified])
  using  $d0$  apply auto
done

```

also from $d0\ 4$ **have** $\dots = d * \varepsilon / 4$
unfolding *set-lebesgue-integral-def* **by** *simp*
finally have $\text{cmod } (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t) \leq d * \varepsilon / 4$.
have $\text{cmod } (1 - \text{char } (M\ n)\ x) \leq 2$ **for** $n\ x$
by (*rule order-trans [OF norm-triangle-ineq4], auto simp add: Mn.cmod-char-le-1*)
then have $(\lambda n. CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) \longrightarrow (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)$
unfolding *set-lebesgue-integral-def*
apply (*intro integral-dominated-convergence[where w= $\lambda x. \text{indicator } \{-d/2..d/2\}$]*
 $x *_{\mathbb{R}} 2$)
apply (*auto intro!: char-conv tendsto-intros*
simp: emeasure-lborel-Icc-eq
split: split-indicator)
done
hence eventually $(\lambda n. \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4)$ *sequentially*
using $d0\ \langle \varepsilon > 0$ **apply** (*subst (asm) tendsto-iff*)
by (*subst (asm) dist-complex-def, drule spec, erule mp, auto*)
hence $\exists N. \forall n \geq N. \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4$ **by** (*simp add: eventually-sequentially*)
then obtain N
where $\forall n \geq N. \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4$..
hence $N: \bigwedge n. n \geq N \implies \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) < d * \varepsilon / 4$ **by** *auto*
{ fix n
assume $n \geq N$
have $\text{cmod } (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) = \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) + (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)$ **by** *simp*
also have $\dots \leq \text{cmod } ((CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) - (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)) + \text{cmod } (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } M' t)$
by (*rule norm-triangle-ineq*)
also have $\dots < d * \varepsilon / 4 + d * \varepsilon / 4$
by (*rule add-less-le-mono [OF N [OF $\langle n \geq N \rangle$] bound]*)
also have $\dots = d * \varepsilon / 2$ **by** *auto*
finally have $\text{cmod } (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t) < d * \varepsilon / 2$.
hence $d * \varepsilon / 2 > \text{Re } (CLBINT\ t:\{-d/2..d/2\}. 1 - \text{char } (M\ n)\ t)$
by (*rule order-le-less-trans [OF complex-Re-le-cmod]*)
hence $d * \varepsilon / 2 > \text{Re } (CLBINT\ t:\{-(d/2)..d/2\}. 1 - \text{char } (M\ n)\ t)$ (*is -*
 $> ?lhs$) **by** *simp*
also have $?lhs \geq (d / 2) * \text{measure } (M\ n)\ \{x. \text{abs } x \geq 2 / (d / 2)\}$
using $d0$ **by** (*intro main-bound, simp*)

```

finally (xtrans) have  $d * \varepsilon / 2 > (d / 2) * \text{measure } (M n) \{x. \text{abs } x \geq 2 / (d / 2)\}$  .
with  $d0 \langle \varepsilon > 0 \rangle$  have  $\varepsilon > \text{measure } (M n) \{x. \text{abs } x \geq 2 / (d / 2)\}$  by (simp add: field-simps)
hence  $\varepsilon > 1 - \text{measure } (M n) (UNIV - \{x. \text{abs } x \geq 2 / (d / 2)\})$ 
apply (subst Mn.borel-UNIV [symmetric])
by (subst Mn.prob-compl, auto)
also have  $UNIV - \{x. \text{abs } x \geq 2 / (d / 2)\} = \{x. -(4 / d) < x \wedge x < (4 / d)\}$ 
using  $d0$  apply (auto simp add: field-simps)

apply (case-tac x ≥ 0, auto simp add: field-simps)
apply (subgoal-tac 0 ≤ x * d, arith, rule mult-nonneg-nonneg, auto)
apply (case-tac x ≥ 0, auto simp add: field-simps)
apply (subgoal-tac x * d ≤ 0, arith)
apply (rule mult-nonpos-nonneg, auto)
by (case-tac x ≥ 0, auto simp add: field-simps)
finally have  $\text{measure } (M n) \{x. -(4 / d) < x \wedge x < (4 / d)\} > 1 - \varepsilon$ 
by auto
} note 6 = this
{ fix  $n :: \text{nat}$ 
have *: ( $UN (k :: \text{nat}). \{- \text{real } k < .. \text{real } k\} = UNIV$ )
by (auto, metis leI le-less-trans less-imp-le minus-less-iff reals-Archimedean2)
have ( $\lambda k. \text{measure } (M n) \{- \text{real } k < .. \text{real } k\} \longrightarrow \text{measure } (M n) (UN (k :: \text{nat}). \{- \text{real } k < .. \text{real } k\})$ )
by (rule Mn.finite-Lim-measure-incseq, auto simp add: incseq-def)
hence ( $\lambda k. \text{measure } (M n) \{- \text{real } k < .. \text{real } k\} \longrightarrow 1$ )
using Mn.prob-space unfolding * Mn.borel-UNIV by simp
hence eventually ( $\lambda k. \text{measure } (M n) \{- \text{real } k < .. \text{real } k\} > 1 - \varepsilon$ ) sequentially
apply (elim order-tendstoD (1))
using  $\langle \varepsilon > 0 \rangle$  by auto
} note 7 = this
{ fix  $n :: \text{nat}$ 
have eventually ( $\lambda k. \forall m < n. \text{measure } (M m) \{- \text{real } k < .. \text{real } k\} > 1 - \varepsilon$ )
sequentially
(is ?P n)
proof (induct n)
case (Suc n) with 7[of n] show ?case
by eventually-elim (auto simp add: less-Suc-eq)
qed simp
} note 8 = this
from 8 [of N] have  $\exists K :: \text{nat}. \forall k \geq K. \forall m < N. 1 - \varepsilon < \text{Sigma-Algebra.measure } (M m) \{- \text{real } k < .. \text{real } k\}$ 
by (auto simp add: eventually-sequentially)
hence  $\exists K :: \text{nat}. \forall m < N. 1 - \varepsilon < \text{Sigma-Algebra.measure } (M m) \{- \text{real } K < .. \text{real } K\}$  by auto
then obtain  $K :: \text{nat}$  where
 $\forall m < N. 1 - \varepsilon < \text{Sigma-Algebra.measure } (M m) \{- \text{real } K < .. \text{real } K\}$  ..
hence  $K: \bigwedge m. m < N \implies 1 - \varepsilon < \text{Sigma-Algebra.measure } (M m) \{- \text{real } K < .. \text{real } K\}$ 

```

```

K <.. real K}
  by auto
  let ?K' = max K (4 / d)
  have -?K' < ?K' ∧ (∀ n. 1 - ε < measure (M n) {-?K' <.. ?K'})
    using d0 apply auto
  apply (rule max.strict-coboundedI2, auto)
  proof -
    fix n
    show 1 - ε < measure (M n) {- max (real K) (4 / d) <.. max (real K) (4
/ d)}
      apply (case-tac n < N)
      apply (rule order-less-le-trans)
      apply (erule K)
      apply (rule Mn.finite-measure-mono, auto)
      apply (rule order-less-le-trans)
      apply (rule 6, erule leI)
      by (rule Mn.finite-measure-mono, auto)
    qed
  thus ∃ a b. a < b ∧ (∀ n. 1 - ε < measure (M n) {a <.. b}) by (intro exI)
  qed
  have tight: tight M
    by (auto simp: tight-def intro: assms tight-aux)
  show ?thesis
  proof (rule tight-subseq-weak-converge [OF real-distr-M real-distr-M' tight])
    fix s ν
    assume s: strict-mono (s :: nat ⇒ nat)
    assume nu: weak-conv-m (M ∘ s) ν
    assume *: real-distribution ν
    have 2: ∧ n. real-distribution ((M ∘ s) n) unfolding comp-def by (rule assms)
    have 3: ∧ t. (λ n. char ((M ∘ s) n) t) ⟶ char ν t by (intro levy-continuity1
[OF 2 * nu])
    have 4: ∧ t. (λ n. char ((M ∘ s) n) t) = ((λ n. char (M n) t) ∘ s) by (rule ext,
simp)
    have 5: ∧ t. (λ n. char ((M ∘ s) n) t) ⟶ char M' t
      by (subst 4, rule LIMSEQ-subseq-LIMSEQ [OF - s], rule assms)
    hence char ν = char M' by (intro ext, intro LIMSEQ-unique [OF 3 5])
    hence ν = M' by (rule Levy-uniqueness [OF * ⟨real-distribution M'⟩])
    thus weak-conv-m (M ∘ s) M'
      by (elim subst) (rule nu)
    qed
  qed
end

```

19 The Central Limit Theorem

```

theory Central-Limit-Theorem
  imports Levy
begin

```

theorem (in *prob-space*) *central-limit-theorem-zero-mean*:

fixes $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

and $\mu :: \text{real measure}$

and $\sigma :: \text{real}$

and $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $X\text{-indep}$: *indep-vars* ($\lambda i. \text{borel}$) X *UNIV*

and $X\text{-mean-0}$: $\bigwedge n. \text{expectation } (X\ n) = 0$

and $\sigma\text{-pos}$: $\sigma > 0$

and $X\text{-square-integrable}$: $\bigwedge n. \text{integrable } M\ (\lambda x. (X\ n\ x)^2)$

and $X\text{-variance}$: $\bigwedge n. \text{variance } (X\ n) = \sigma^2$

and $X\text{-distrib}$: $\bigwedge n. \text{distr } M\ \text{borel } (X\ n) = \mu$

defines $S\ n \equiv \lambda x. \sum_{i < n}. X\ i\ x$

shows *weak-conv-m* ($\lambda n. \text{distr } M\ \text{borel } (\lambda x. S\ n\ x / \text{sqrt } (n * \sigma^2))$) *std-normal-distribution*

proof –

let $?S' = \lambda n\ x. S\ n\ x / \text{sqrt } (\text{real } n * \sigma^2)$

define φ **where** $\varphi\ n = \text{char } (\text{distr } M\ \text{borel } (?S'\ n))$ **for** n

define ψ **where** $\psi\ n\ t = \text{char } \mu\ (t / \text{sqrt } (\sigma^2 * n))$ **for** $n\ t$

have $X\text{-rv}$ [*simp*, *measurable*]: *random-variable borel* ($X\ n$) **for** n

using $X\text{-indep}$ **unfolding** *indep-vars-def2* **by** *simp*

have $X\text{-integrable}$ [*simp*, *intro*]: *integrable* $M\ (X\ n)$ **for** n

by (*rule square-integrable-imp-integrable*[*OF* - $X\text{-square-integrable}$]) *simp-all*

interpret μ : *real-distribution* μ

by (*subst* $X\text{-distrib}$ [*symmetric*, *of 0*], *rule real-distribution-distr*, *simp*)

have $\mu\text{-integrable}$ [*simp*]: *integrable* $\mu\ (\lambda x. x)$

and $\mu\text{-mean-integrable}$ [*simp*]: $\mu.\text{expectation } (\lambda x. x) = 0$

and $\mu\text{-square-integrable}$ [*simp*]: *integrable* $\mu\ (\lambda x. x^2)$

and $\mu\text{-variance}$ [*simp*]: $\mu.\text{expectation } (\lambda x. x^2) = \sigma^2$

using *assms* **by** (*simp-all add*: $X\text{-distrib}$ [*symmetric*, *of 0*] *integrable-distr-eq* *integral-distr*)

have *main*: $\forall_F n$ *in sequentially*.

$cmod\ (\varphi\ n\ t - (1 + (-t^2) / 2) / n) \wedge n \leq$

$t^2 / (6 * \sigma^2) * (LINT\ x|\mu. \min\ (6 * x^2) (|t / \text{sqrt } (\sigma^2 * n)| * |x| \wedge 3))$ **for** t

proof (*rule eventually-sequentiallyI*)

fix $n :: \text{nat}$

assume $n \geq \text{nat } (\text{ceiling } (t^2 / 4))$

hence $n: n \geq t^2 / 4$ **by** (*subst* *nat-ceiling-le-eq* [*symmetric*])

let $?t = t / \text{sqrt } (\sigma^2 * n)$

define ψ' **where** $\psi'\ n\ i = \text{char } (\text{distr } M\ \text{borel } (\lambda x. X\ i\ x / \text{sqrt } (\sigma^2 * n)))$ **for** $n\ i$

have $*$: $\bigwedge n\ i\ t. \psi'\ n\ i\ t = \psi\ n\ t$

unfolding $\psi\text{-def}$ $\psi'\text{-def}$ *char-def*

by (*subst* $X\text{-distrib}$ [*symmetric*]) (*auto* *simp*: *integral-distr*)


```

have  $\varphi n t = \text{char } (\text{distr } M \text{ borel } (\lambda x. \sum_{i < n}. X i x / \text{sqrt } (\sigma^2 * \text{real } n))) t$ 
  by (auto simp:  $\varphi$ -def S-def sum-divide-distrib ac-simps)
also have  $\dots = (\prod_{i < n}. \psi' n i t)$ 
  unfolding  $\psi'$ -def
  apply (rule char-distr-sum)
  apply (rule indep-vars-compose2[where X=X])
  apply (rule indep-vars-subset)
  apply (rule X-indep)
  apply auto
  done
also have  $\dots = (\psi n t)^{\wedge} n$ 
  by (auto simp add: * prod-constant)
finally have  $\varphi$ -eq:  $\varphi n t = (\psi n t)^{\wedge} n$  .

have norm  $(\psi n t - (1 - ?t^{\wedge} 2 * \sigma^2 / 2)) \leq ?t^2 / 6 * (\text{LINT } x | \mu. \text{min } (6 * x^2) (|?t| * |x|^{\wedge} 3))$ 
  unfolding  $\psi$ -def by (rule  $\mu$ .char-approx3, auto)
also have  $?t^{\wedge} 2 * \sigma^2 = t^{\wedge} 2 / n$ 
  using  $\sigma$ -pos by (simp add: power-divide)
also have  $t^{\wedge} 2 / n / 2 = (t^{\wedge} 2 / 2) / n$ 
  by simp
finally have **: norm  $(\psi n t - (1 + (-(t^{\wedge} 2) / 2) / n)) \leq$ 
   $?t^2 / 6 * (\text{LINT } x | \mu. \text{min } (6 * x^2) (|?t| * |x|^{\wedge} 3))$  by simp

have norm  $(\varphi n t - (\text{complex-of-real } (1 + (-(t^{\wedge} 2) / 2) / n))^{\wedge} n) \leq$ 
   $n * \text{norm } (\psi n t - (\text{complex-of-real } (1 + (-(t^{\wedge} 2) / 2) / n)))$ 
  using n
  by (auto intro!: norm-power-diff  $\mu$ .cmod-char-le-1 abs-leI
    simp del: of-real-diff simp: of-real-diff[symmetric] divide-le-eq  $\varphi$ -eq
 $\psi$ -def)
also have  $\dots \leq n * (?t^2 / 6 * (\text{LINT } x | \mu. \text{min } (6 * x^2) (|?t| * |x|^{\wedge} 3)))$ 
  by (rule mult-left-mono [OF **], simp)
also have  $\dots = (t^2 / (6 * \sigma^2)) * (\text{LINT } x | \mu. \text{min } (6 * x^2) (|?t| * |x|^{\wedge} 3))$ 
  using  $\sigma$ -pos by (simp add: field-simps min-absorb2)
finally show norm  $(\varphi n t - (1 + (-(t^{\wedge} 2) / 2) / n))^{\wedge} n \leq$ 
   $(t^2 / (6 * \sigma^2)) * (\text{LINT } x | \mu. \text{min } (6 * x^2) (|?t| * |x|^{\wedge} 3))$ 
  by simp
qed

show ?thesis
proof (rule levy-continuity)
  fix t
  let ?t =  $\lambda n. t / \text{sqrt } (\sigma^2 * n)$ 
  have  $\bigwedge x. (\lambda n. \text{min } (6 * x^2) (|t| * |x|^{\wedge} 3 / \text{sqrt } (\sigma^2 * \text{real } n))) \longrightarrow 0$ 
    using  $\sigma$ -pos
    by (auto simp: real-sqrt-mult min-absorb2
      intro!: tendsto-min[THEN tendsto-eq-rhs] sqrt-at-top[THEN filter-
lim-compose]
      filterlim-tendsto-pos-mult-at-top filterlim-at-top-imp-at-infinity

```

tendsto-divide-0 filterlim-real-sequentially)

then have $(\lambda n. LINT x|\mu. \min (6 * x^2) (|?t n| * |x| ^ 3)) \longrightarrow (LINT x|\mu. 0)$

by (*intro integral-dominated-convergence [where $w = \lambda x. 6 * x^2$] auto*)

then have $*$: $(\lambda n. t^2 / (6 * \sigma^2) * (LINT x|\mu. \min (6 * x^2) (|?t n| * |x| ^ 3))) \longrightarrow 0$

by (*simp only: integral-zero tendsto-mult-right-zero*)

have $(\lambda n. \text{complex-of-real } ((1 + (-(t^2) / 2) / n) ^ n)) \longrightarrow \text{complex-of-real } (\exp (-(t^2) / 2))$

by (*rule isCont-tendsto-compose [OF - tendsto-exp-limit-sequentially] auto*)

then have $(\lambda n. \varphi n t) \longrightarrow \text{complex-of-real } (\exp (-(t^2) / 2))$

by (*rule Lim-transform*) (*rule Lim-null-comparison [OF main *]*)

then show $(\lambda n. \text{char } (\text{distr } M \text{ borel } (?S' n)) t) \longrightarrow \text{char std-normal-distribution } t$

by (*simp add: φ -def char-std-normal-distribution*)

qed (*auto intro!: real-dist-normal-dist simp: S-def*)

qed

theorem (*in prob-space*) *central-limit-theorem*:

fixes $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

and $\mu :: \text{real measure}$

and $\sigma :: \text{real}$

and $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $X\text{-indep}$: *indep-vars* $(\lambda i. \text{borel}) X \text{ UNIV}$

and $X\text{-mean}$: $\bigwedge n. \text{expectation } (X n) = m$

and $\sigma\text{-pos}$: $\sigma > 0$

and $X\text{-square-integrable}$: $\bigwedge n. \text{integrable } M (\lambda x. (X n x)^2)$

and $X\text{-variance}$: $\bigwedge n. \text{variance } (X n) = \sigma^2$

and $X\text{-distrib}$: $\bigwedge n. \text{distr } M \text{ borel } (X n) = \mu$

defines $X' i x \equiv X i x - m$

shows *weak-conv-m* $(\lambda n. \text{distr } M \text{ borel } (\lambda x. (\sum i < n. X' i x) / \text{sqrt } (n * \sigma^2)))$

std-normal-distribution

proof (*intro central-limit-theorem-zero-mean*)

show *indep-vars* $(\lambda i. \text{borel}) X' \text{ UNIV}$

unfolding $X'\text{-def}$ [*abs-def*] **using** $X\text{-indep}$ **by** (*rule indep-vars-compose2*) *auto*

have $X\text{-rv}$ [*simp, measurable*]: *random-variable* $\text{borel } (X n)$ **for** n

using $X\text{-indep}$ **unfolding** *indep-vars-def2* **by** *simp*

have $X\text{-integrable}$ [*simp, intro*]: *integrable* $M (X n)$ **for** n

by (*rule square-integrable-imp-integrable*[*OF - X-square-integrable*]) *simp-all*

show *expectation* $(X' n) = 0$ **for** n

using $X\text{-mean}$ **by** (*auto simp: X'-def*[*abs-def*] *prob-space*)

show $\sigma > 0$ *integrable* $M (\lambda x. (X' n x)^2)$ *variance* $(X' n) = \sigma^2$ **for** n

using $\langle 0 < \sigma \rangle$ $X\text{-integrable}$ $X\text{-mean}$ $X\text{-square-integrable}$ $X\text{-variance}$ **unfolding** $X'\text{-def}$

by (*auto simp: prob-space power2-diff*)

show *distr* $M \text{ borel } (X' n) = \text{distr } \mu \text{ borel } (\lambda x. x - m)$ **for** n

unfolding $X\text{-distrib}$ [*of n, symmetric*] **using** $X\text{-integrable}$

by (*subst distr-distr*) (*auto simp: X'-def*[*abs-def*] *comp-def*)

qed**end**

theory *Discrete-Topology*
imports *HOL-Analysis.Analysis*
begin

Copy of discrete types with discrete topology. This space is polish.

typedef *'a discrete = UNIV::'a set*
morphisms *of-discrete discrete*
 ..

instantiation *discrete :: (type) metric-space*
begin

definition *dist-discrete :: 'a discrete \Rightarrow 'a discrete \Rightarrow real*
where *dist-discrete n m = (if n = m then 0 else 1)*

definition *uniformity-discrete :: ('a discrete \times 'a discrete) filter* **where**
(uniformity::('a discrete \times 'a discrete) filter) = (INF e \in {0 <..}. principal {(x, y). dist x y < e})

definition *open-discrete :: 'a discrete set \Rightarrow bool* **where**
(open::'a discrete set \Rightarrow bool) U \longleftrightarrow ($\forall x \in U$. eventually ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) uniformity)

instance proof qed *(auto simp: uniformity-discrete-def open-discrete-def dist-discrete-def*
intro: exI[where x=1])
end

lemma *open-discrete: open (S :: 'a discrete set)*
unfolding *open-dist dist-discrete-def* **by** *(auto intro!: exI[of - 1 / 2])*

instance *discrete :: (type) complete-space*
proof

fix *X::nat \Rightarrow 'a discrete*
assume *Cauchy X*
then obtain *n* **where** $\forall m \geq n$. *X n = X m*
by *(force simp: dist-discrete-def Cauchy-def split: if-split-asm dest:spec[where x=1])*
thus *convergent X*
by *(intro convergentI[where L=X n] tendstoI eventually-sequentiallyI[of n])*
(simp add: dist-discrete-def)

qed

instance *discrete :: (countable) countable*
proof

```

have inj ( $\lambda c::'a$  discrete. to-nat (of-discrete c))
  by (simp add: inj-on-def of-discrete-inject)
thus  $\exists f::'a$  discrete  $\Rightarrow$  nat. inj f by blast
qed

```

```

instance discrete :: (countable) second-countable-topology
proof

```

```

  let ?B = range ( $\lambda n::'a$  discrete. {n})
  have  $\bigwedge S$ . generate-topology ?B ( $\bigcup x \in S$ . {x})
    by (intro generate-topology-Union) (auto intro: generate-topology.intros)
  then have open = generate-topology ?B
    by (auto intro!: ext simp: open-discrete)
  moreover have countable ?B by simp
  ultimately show  $\exists B::'a$  discrete set set. countable B  $\wedge$  open = generate-topology
  B by blast
qed

```

```

instance discrete :: (countable) polish-space ..

```

```

end

```

20 Probability mass function

```

theory Probability-Mass-Function

```

```

imports

```

```

  Giry-Monad

```

```

  HOL-Library.Multiset

```

```

begin

```

```

  Conflicting notation from HOL-Analysis.Infinite-Sum

```

```

no-notation Infinite-Sum.abs-summable-on (infixr abs'-summable'-on 46)

```

```

lemma AE-emeasure-singleton:

```

```

  assumes  $x$ : emeasure M {x}  $\neq$  0 and  $ae$ : AE  $x$  in M. P  $x$  shows P  $x$ 

```

```

proof –

```

```

  from  $x$  have  $x$ -M: {x}  $\in$  sets M

```

```

    by (auto intro: emeasure-notin-sets)

```

```

  from  $ae$  obtain N where N: { $x \in$ space M.  $\neg$  P  $x$ }  $\subseteq$  N emeasure M N = 0 N
   $\in$  sets M

```

```

    by (auto elim: AE-E)

```

```

  { assume  $\neg$  P  $x$ 

```

```

    with  $x$ -M[THEN sets.sets-into-space] N have emeasure M {x}  $\leq$  emeasure M
  N

```

```

    by (intro emeasure-mono) auto

```

```

    with  $x$  N have False

```

```

    by (auto simp:) }

```

```

  then show P  $x$  by auto

```

```

qed

```

lemma *AE-measure-singleton*: $\text{measure } M \{x\} \neq 0 \implies \text{AE } x \text{ in } M. P x \implies P x$
by (*metis AE-emeasure-singleton measure-def emeasure-empty measure-empty*)

lemma (*in finite-measure*) *AE-support-countable*:

assumes [*simp*]: *sets* $M = \text{UNIV}$

shows ($\text{AE } x \text{ in } M. \text{measure } M \{x\} \neq 0$) $\longleftrightarrow (\exists S. \text{countable } S \wedge (\text{AE } x \text{ in } M. x \in S))$

proof

assume $\exists S. \text{countable } S \wedge (\text{AE } x \text{ in } M. x \in S)$

then obtain S **where** S [*intro*]: *countable* S **and** *ae*: $\text{AE } x \text{ in } M. x \in S$

by *auto*

then have $\text{emeasure } M (\bigcup_{x \in S} \text{emeasure } M \{x\} \neq 0). \{x\} =$

$(\int^+ x. \text{emeasure } M \{x\} * \text{indicator } \{x \in S. \text{emeasure } M \{x\} \neq 0\} x \partial \text{count-space UNIV})$

by (*subst emeasure-UN-countable*)

(*auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] restrict-count-space*)

also have $\dots = (\int^+ x. \text{emeasure } M \{x\} * \text{indicator } S x \partial \text{count-space UNIV})$

by (*auto intro!: nn-integral-cong split: split-indicator*)

also have $\dots = \text{emeasure } M (\bigcup_{x \in S} \{x\})$

by (*subst emeasure-UN-countable*)

(*auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] restrict-count-space*)

also have $\dots = \text{emeasure } M (\text{space } M)$

using *ae* **by** (*intro emeasure-eq-AE*) *auto*

finally have $\text{emeasure } M \{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\} = \text{emeasure } M (\text{space } M)$

by (*simp add: emeasure-single-in-space cong: rev-conj-cong*)

with *finite-measure-compl*[*of* $\{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\}$]

have $\text{AE } x \text{ in } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0$

by (*intro AE-I[OF order-refl]*) (*auto simp: emeasure-eq-measure measure-nonneg set-diff-eq cong: conj-cong*)

then show $\text{AE } x \text{ in } M. \text{measure } M \{x\} \neq 0$

by (*auto simp: emeasure-eq-measure*)

qed (*auto intro!: exI[of - $\{x. \text{measure } M \{x\} \neq 0\}$] countable-support*)

20.1 PMF as measure

typedef *'a pmf* = $\{M :: 'a \text{ measure. prob-space } M \wedge \text{sets } M = \text{UNIV} \wedge (\text{AE } x \text{ in } M. \text{measure } M \{x\} \neq 0)\}$

morphisms *measure-pmf Abs-pmf*

by (*intro exI[of - uniform-measure (count-space UNIV) {undefined}]*)

(*auto intro!: prob-space-uniform-measure AE-uniform-measureI*)

declare [[*coercion measure-pmf*]]

lemma *prob-space-measure-pmf*: *prob-space* (*measure-pmf* p)

using *pmf.measure-pmf*[*of* p] **by** *auto*

interpretation *measure-pmf*: *prob-space measure-pmf M for M*
by (*rule prob-space-measure-pmf*)

interpretation *measure-pmf*: *subprob-space measure-pmf M for M*
by (*rule prob-space-imp-subprob-space*) *unfold-locales*

lemma *subprob-space-measure-pmf*: *subprob-space (measure-pmf x)*
by *unfold-locales*

locale *pmf-as-measure*
begin

setup-lifting *type-definition-pmf*

end

context
begin

interpretation *pmf-as-measure* .

lemma *sets-measure-pmf[simp]*: *sets (measure-pmf p) = UNIV*
by *transfer blast*

lemma *sets-measure-pmf-count-space[measurable-cong]*:
sets (measure-pmf M) = sets (count-space UNIV)
by *simp*

lemma *space-measure-pmf[simp]*: *space (measure-pmf p) = UNIV*
using *sets-eq-imp-space-eq[of measure-pmf p count-space UNIV]* **by** *simp*

lemma *measure-pmf-UNIV [simp]*: *measure (measure-pmf p) UNIV = 1*
using *measure-pmf.prob-space[of p]* **by** *simp*

lemma *measure-pmf-in-subprob-algebra[measurable (raw)]*: *measure-pmf x ∈ space*
(subprob-algebra (count-space UNIV))
by (*simp add: space-subprob-algebra subprob-space-measure-pmf*)

lemma *measurable-pmf-measure1 [simp]*: *measurable (M :: 'a pmf) N = UNIV →*
space N
by (*auto simp: measurable-def*)

lemma *measurable-pmf-measure2 [simp]*: *measurable N (M :: 'a pmf) = measurable*
N (count-space UNIV)
by (*intro measurable-cong-sets*) *simp-all*

lemma *measurable-pair-restrict-pmf2*:
assumes *countable A*
assumes [*measurable*]: $\bigwedge y. y \in A \implies (\lambda x. f(x, y)) \in \text{measurable } M L$

shows $f \in \text{measurable } (M \otimes_M \text{ restrict-space } (\text{measure-pmf } N) A) L$ (**is** $f \in \text{measurable } ?M$ -)

proof –

have [*measurable-cong*]: $\text{sets } (\text{restrict-space } (\text{count-space } UNIV) A) = \text{sets } (\text{count-space } A)$

by (*simp add: restrict-count-space*)

show *?thesis*

by (*intro measurable-compose-countable'*[**where** $f = \lambda a b. f (\text{fst } b, a)$ **and** $g = \text{snd}$ **and** $I = A$,
unfolded prod.collapse] *assms*)

measurable

qed

lemma *measurable-pair-restrict-pmf1*:

assumes *countable A*

assumes [*measurable*]: $\bigwedge x. x \in A \implies (\lambda y. f (x, y)) \in \text{measurable } N L$

shows $f \in \text{measurable } (\text{restrict-space } (\text{measure-pmf } M) A \otimes_M N) L$

proof –

have [*measurable-cong*]: $\text{sets } (\text{restrict-space } (\text{count-space } UNIV) A) = \text{sets } (\text{count-space } A)$

by (*simp add: restrict-count-space*)

show *?thesis*

by (*intro measurable-compose-countable'*[**where** $f = \lambda a b. f (a, \text{snd } b)$ **and** $g = \text{fst}$ **and** $I = A$,
unfolded prod.collapse] *assms*)

measurable

qed

lift-definition *pmf* :: $'a \text{ pmf} \Rightarrow 'a \Rightarrow \text{real}$ **is** $\lambda M x. \text{measure } M \{x\}$.

lift-definition *set-pmf* :: $'a \text{ pmf} \Rightarrow 'a \text{ set}$ **is** $\lambda M. \{x. \text{measure } M \{x\} \neq 0\}$.

declare [[*coercion set-pmf*]]

lemma *AE-measure-pmf*: $AE x \text{ in } (M :: 'a \text{ pmf}). x \in M$

by *transfer simp*

lemma *emeasure-pmf-single-eq-zero-iff*:

fixes $M :: 'a \text{ pmf}$

shows $\text{emeasure } M \{y\} = 0 \iff y \notin M$

unfolding *set-pmf.rep-eq* **by** (*simp add: measure-pmf.emeasure-eq-measure*)

lemma *AE-measure-pmf-iff*: $(AE x \text{ in } \text{measure-pmf } M. P x) \iff (\forall y \in M. P y)$

using *AE-measure-singleton[of M]* *AE-measure-pmf[of M]*

by (*auto simp: set-pmf.rep-eq*)

lemma *AE-pmfI*: $(\bigwedge y. y \in \text{set-pmf } M \implies P y) \implies \text{almost-everywhere } (\text{measure-pmf } M) P$

by(*simp add: AE-measure-pmf-iff*)

lemma *countable-set-pmf* [*simp*]: *countable (set-pmf p)*
by *transfer (metis prob-space.finite-measure finite-measure.countable-support)*

lemma *pmf-positive*: $x \in \text{set-pmf } p \implies 0 < \text{pmf } p \ x$
by *transfer (simp add: less-le)*

lemma *pmf-nonneg*[*simp*]: $0 \leq \text{pmf } p \ x$
by *transfer simp*

lemma *pmf-not-neg* [*simp*]: $\neg \text{pmf } p \ x < 0$
by (*simp add: not-less pmf-nonneg*)

lemma *pmf-pos* [*simp*]: $\text{pmf } p \ x \neq 0 \implies \text{pmf } p \ x > 0$
using *pmf-nonneg[of p x]* **by** *linarith*

lemma *pmf-le-1*: $\text{pmf } p \ x \leq 1$
by (*simp add: pmf.rep-eq*)

lemma *set-pmf-not-empty*: $\text{set-pmf } M \neq \{\}$
using *AE-measure-pmf[of M]* **by** (*intro notI simp*)

lemma *set-pmf-iff*: $x \in \text{set-pmf } M \longleftrightarrow \text{pmf } M \ x \neq 0$
by *transfer simp*

lemma *pmf-positive-iff*: $0 < \text{pmf } p \ x \longleftrightarrow x \in \text{set-pmf } p$
unfolding *less-le* **by** (*simp add: set-pmf-iff*)

lemma *set-pmf-eq*: $\text{set-pmf } M = \{x. \text{pmf } M \ x \neq 0\}$
by (*auto simp: set-pmf-iff*)

lemma *set-pmf-eq'*: $\text{set-pmf } p = \{x. \text{pmf } p \ x > 0\}$
proof *safe*
fix *x* **assume** $x \in \text{set-pmf } p$
hence $\text{pmf } p \ x \neq 0$ **by** (*auto simp: set-pmf-eq*)
with *pmf-nonneg[of p x]* **show** $\text{pmf } p \ x > 0$ **by** *simp*
qed (*auto simp: set-pmf-eq*)

lemma *emeasure-pmf-single*:
fixes $M :: 'a \text{ pmf}$
shows $\text{emeasure } M \ \{x\} = \text{pmf } M \ x$
by *transfer (simp add: finite-measure.emeasure-eq-measure[OF prob-space.finite-measure])*

lemma *measure-pmf-single*: $\text{measure (measure-pmf } M) \ \{x\} = \text{pmf } M \ x$
using *emeasure-pmf-single[of M x]* **by**(*simp add: measure-pmf.emeasure-eq-measure pmf-nonneg measure-nonneg*)

lemma *emeasure-measure-pmf-finite*: $\text{finite } S \implies \text{emeasure (measure-pmf } M) \ S$

= $(\sum_{s \in S}. \text{pmf } M \ s)$
by (*subst emeasure-eq-sum-singleton*) (*auto simp: emeasure-pmf-single pmf-nonneg*)

lemma *measure-measure-pmf-finite*: *finite S* \implies *measure (measure-pmf M) S = sum (pmf M) S*

using *emeasure-measure-pmf-finite*[of *S M*]
by (*simp add: measure-pmf.emeasure-eq-measure measure-nonneg sum-nonneg pmf-nonneg*)

lemma *sum-pmf-eq-1*:

assumes *finite A set-pmf p* $\subseteq A$
shows $(\sum_{x \in A}. \text{pmf } p \ x) = 1$
proof –
have $(\sum_{x \in A}. \text{pmf } p \ x) = \text{measure-pmf.prob } p \ A$
by (*simp add: measure-measure-pmf-finite assms*)
also from *assms* **have** $\dots = 1$
by (*subst measure-pmf.prob-eq-1*) (*auto simp: AE-measure-pmf-iff*)
finally show *?thesis* .
qed

lemma *nn-integral-measure-pmf-support*:

fixes *f* :: *'a* \Rightarrow *ennreal*
assumes *f*: *finite A* **and** *nn*: $\bigwedge x. x \in A \implies 0 \leq f \ x \ \bigwedge x. x \in \text{set-pmf } M \implies x \notin A \implies f \ x = 0$
shows $(\int^+ x. f \ x \ \partial \text{measure-pmf } M) = (\sum_{x \in A}. f \ x * \text{pmf } M \ x)$
proof –
have $(\int^+ x. f \ x \ \partial M) = (\int^+ x. f \ x * \text{indicator } A \ x \ \partial M)$
using *nn* **by** (*intro nn-integral-cong-AE*) (*auto simp: AE-measure-pmf-iff split: split-indicator*)
also have $\dots = (\sum_{x \in A}. f \ x * \text{emeasure } M \ \{x\})$
using *assms* **by** (*intro nn-integral-indicator-finite*) *auto*
finally show *?thesis*
by (*simp add: emeasure-measure-pmf-finite*)
qed

lemma *nn-integral-measure-pmf-finite*:

fixes *f* :: *'a* \Rightarrow *ennreal*
assumes *f*: *finite (set-pmf M)* **and** *nn*: $\bigwedge x. x \in \text{set-pmf } M \implies 0 \leq f \ x$
shows $(\int^+ x. f \ x \ \partial \text{measure-pmf } M) = (\sum_{x \in \text{set-pmf } M}. f \ x * \text{pmf } M \ x)$
using *assms* **by** (*intro nn-integral-measure-pmf-support*) *auto*

lemma *integrable-measure-pmf-finite*:

fixes *f* :: *'a* \Rightarrow *'b*::*{banach, second-countable-topology}*
shows *finite (set-pmf M)* \implies *integrable M f*
by (*auto intro!: integrableI-bounded simp: nn-integral-measure-pmf-finite ennreal-mult-less-top*)

lemma *integral-measure-pmf-real*:

assumes [*simp*]: *finite A* **and** $\bigwedge a. a \in \text{set-pmf } M \implies f \ a \neq 0 \implies a \in A$
shows $(\int x. f \ x \ \partial \text{measure-pmf } M) = (\sum_{a \in A}. f \ a * \text{pmf } M \ a)$

proof –

have $(\int x. f x \partial \text{measure-pmf } M) = (\int x. f x * \text{indicator } A x \partial \text{measure-pmf } M)$
using *assms(2)* **by** (*intro integral-cong-AE*) (*auto split: split-indicator simp: AE-measure-pmf-iff*)
also have $\dots = (\sum a \in A. f a * \text{pmf } M a)$
by (*subst integral-indicator-finite-real*)
(auto simp: measure-def emeasure-measure-pmf-finite pmf-nonneg)
finally show *?thesis* .
qed

lemma *integrable-pmf: integrable (count-space X) (pmf M)*

proof –

have $(\int^+ x. \text{pmf } M x \partial \text{count-space } X) = (\int^+ x. \text{pmf } M x \partial \text{count-space } (M \cap X))$
by (*auto simp add: nn-integral-count-space-indicator set-pmf-iff intro!: nn-integral-cong split: split-indicator*)
then have *integrable (count-space X) (pmf M) = integrable (count-space (M ∩ X)) (pmf M)*
by (*simp add: integrable-iff-bounded pmf-nonneg*)
then show *?thesis*
by (*simp add: pmf.rep-eq measure-pmf.integrable-measure disjoint-family-on-def*)
qed

lemma *integral-pmf: $(\int x. \text{pmf } M x \partial \text{count-space } X) = \text{measure } M X$*

proof –

have $(\int x. \text{pmf } M x \partial \text{count-space } X) = (\int^+ x. \text{pmf } M x \partial \text{count-space } X)$
by (*simp add: pmf-nonneg integrable-pmf nn-integral-eq-integral*)
also have $\dots = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count-space } (X \cap M))$
by (*auto intro!: nn-integral-cong-AE split: split-indicator simp: pmf.rep-eq measure-pmf.emeasure-eq-measure nn-integral-count-space-indicator AE-count-space set-pmf-iff*)
also have $\dots = \text{emeasure } M (X \cap M)$
by (*rule emeasure-countable-singleton[symmetric]*) (*auto intro: countable-set-pmf*)
also have $\dots = \text{emeasure } M X$
by (*auto intro!: emeasure-eq-AE simp: AE-measure-pmf-iff*)
finally show *?thesis*
by (*simp add: measure-pmf.emeasure-eq-measure measure-nonneg integral-nonneg pmf-nonneg*)
qed

lemma *integral-pmf-restrict:*

$(f::'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}) \in \text{borel-measurable (count-space UNIV)} \implies$

$(\int x. f x \partial \text{measure-pmf } M) = (\int x. f x \partial \text{restrict-space } M M)$

by (*auto intro!: integral-cong-AE simp add: integral-restrict-space AE-measure-pmf-iff*)

lemma *emeasure-pmf: $\text{emeasure } (M::'a \text{ pmf}) M = 1$*

proof –

have $\text{emeasure } (M::'a \text{ pmf}) M = \text{emeasure } (M::'a \text{ pmf}) (\text{space } M)$

by (intro emeasure-eq-AE) (simp-all add: AE-measure-pmf)
 then show ?thesis
 using measure-pmf.emeasure-space-1 by simp
 qed

lemma emeasure-pmf-UNIV [simp]: emeasure (measure-pmf M) UNIV = 1
 using measure-pmf.emeasure-space-1[of M] by simp

lemma in-null-sets-measure-pmfI:
 $A \cap \text{set-pmf } p = \{\} \implies A \in \text{null-sets (measure-pmf } p)$
 using emeasure-eq-0-AE[where ?P= $\lambda x. x \in A$ and $M = \text{measure-pmf } p$]
 by(auto simp add: null-sets-def AE-measure-pmf-iff)

lemma measure-subprob: measure-pmf M \in space (subprob-algebra (count-space UNIV))
 by (simp add: space-subprob-algebra subprob-space-measure-pmf)

20.2 Monad Interpretation

lemma measurable-measure-pmf[measurable]:
 $(\lambda x. \text{measure-pmf (M } x)) \in \text{measurable (count-space UNIV) (subprob-algebra (count-space UNIV))}$
 by (auto simp: space-subprob-algebra intro!: prob-space-imp-subprob-space) unfold-locales

lemma bind-measure-pmf-cong:
 assumes $\bigwedge x. A x \in \text{space (subprob-algebra } N)$ $\bigwedge x. B x \in \text{space (subprob-algebra } N)$
 assumes $\bigwedge i. i \in \text{set-pmf } x \implies A i = B i$
 shows bind (measure-pmf x) A = bind (measure-pmf x) B
proof (rule measure-eqI)
 show sets (measure-pmf x \ggg A) = sets (measure-pmf x \ggg B)
 using assms by (subst (1 2) sets-bind) (auto simp: space-subprob-algebra)
next
 fix X assume X \in sets (measure-pmf x \ggg A)
 then have X: X \in sets N
 using assms by (subst (asm) sets-bind) (auto simp: space-subprob-algebra)
 show emeasure (measure-pmf x \ggg A) X = emeasure (measure-pmf x \ggg B) X
 using assms
 by (subst (1 2) emeasure-bind[where N=N, OF - - X])
 (auto intro!: nn-integral-cong-AE simp: AE-measure-pmf-iff)
 qed

lift-definition bind-pmf :: 'a pmf \Rightarrow ('a \Rightarrow 'b pmf) \Rightarrow 'b pmf is bind

proof (clarify, intro conjI)
 fix f :: 'a measure and g :: 'a \Rightarrow 'b measure
 assume prob-space f
 then interpret f: prob-space f .
 assume sets f = UNIV and ae-f: AE x in f. measure f {x} \neq 0

then have $s\text{-}f[simp]: \text{sets } f = \text{sets } (\text{count-space } UNIV)$
by $simp$
assume $g: \bigwedge x. \text{prob-space } (g \ x) \wedge \text{sets } (g \ x) = UNIV \wedge (AE \ y \text{ in } g \ x. \text{measure } (g \ x) \ \{y\} \neq 0)$
then have $g: \bigwedge x. \text{prob-space } (g \ x)$ **and** $s\text{-}g[simp]: \bigwedge x. \text{sets } (g \ x) = \text{sets } (\text{count-space } UNIV)$
and $ae\text{-}g: \bigwedge x. AE \ y \text{ in } g \ x. \text{measure } (g \ x) \ \{y\} \neq 0$
by $auto$

have $[measurable]: g \in \text{measurable } f \ (\text{subprob-algebra } (\text{count-space } UNIV))$
by $(\text{auto } simp: \text{measurable-def } \text{space-subprob-algebra } \text{prob-space-imp-subprob-space } g)$

show $\text{prob-space } (f \ggg g)$
using g **by** $(\text{intro } f.\text{prob-space-bind}[\text{where } S=\text{count-space } UNIV]) \text{ auto}$
then interpret $fg: \text{prob-space } f \ggg g$.
show $[simp]: \text{sets } (f \ggg g) = UNIV$
using $\text{sets-eq-imp-space-eq}[OF \ s\text{-}f]$
by $(\text{subst } \text{sets-bind}[\text{where } N=\text{count-space } UNIV]) \text{ auto}$
show $AE \ x \text{ in } f \ggg g. \text{measure } (f \ggg g) \ \{x\} \neq 0$
apply $(\text{simp } \text{add: } fg.\text{prob-eq-0 } AE\text{-bind}[\text{where } B=\text{count-space } UNIV])$
using $ae\text{-}f$
apply eventually-elim
using $ae\text{-}g$
apply eventually-elim
apply $(\text{auto } \text{dest: } AE\text{-measure-singleton})$
done

qed

ad hoc-overloading $Monad\text{-Syntax.bind } bind\text{-pmf}$

lemma $ennreal\text{-pmf-bind}: \text{pmf } (\text{bind-pmf } N \ f) \ i = (\int^+ x. \text{pmf } (f \ x) \ i \ \partial \text{measure-pmf } N)$

unfolding $\text{pmf.rep-eq } \text{bind-pmf.rep-eq}$
by $(\text{auto } simp: \text{measure-pmf.measure-bind}[\text{where } N=\text{count-space } UNIV] \ \text{measure-subprob } \text{measure-nonneg}$
 $\text{intro!}: \text{nn-integral-eq-integral}[\text{symmetric}] \ \text{measure-pmf.integrable-const-bound}[\text{where } B=1])$

lemma $\text{pmf-bind}: \text{pmf } (\text{bind-pmf } N \ f) \ i = (\int x. \text{pmf } (f \ x) \ i \ \partial \text{measure-pmf } N)$

using $ennreal\text{-pmf-bind}[\text{of } N \ f \ i]$
by $(\text{subst } (\text{asm}) \ \text{nn-integral-eq-integral})$
 $(\text{auto } simp: \text{pmf-nonneg } \text{pmf-le-1 } \text{pmf-nonneg } \text{integral-nonneg}$
 $\text{intro!}: \text{nn-integral-eq-integral}[\text{symmetric}] \ \text{measure-pmf.integrable-const-bound}[\text{where } B=1])$

lemma $\text{bind-pmf-const}[simp]: \text{bind-pmf } M \ (\lambda x. \ c) = c$

by $\text{transfer } (\text{simp } \text{add: } \text{bind-const}' \ \text{prob-space-imp-subprob-space})$

lemma *set-bind-pmf[simp]*: $set\text{-pmf} (bind\text{-pmf} M N) = (\bigcup M \in set\text{-pmf} M. set\text{-pmf} (N M))$

proof –

have $set\text{-pmf} (bind\text{-pmf} M N) = \{x. ennreal (pmf (bind\text{-pmf} M N) x) \neq 0\}$

by (*simp add: set-pmf-eq pmf-nonneg*)

also have $\dots = (\bigcup M \in set\text{-pmf} M. set\text{-pmf} (N M))$

unfolding *ennreal-pmf-bind*

by (*subst nn-integral-0-iff-AE*) (*auto simp: AE-measure-pmf-iff pmf-nonneg set-pmf-eq*)

finally show *?thesis* .

qed

lemma *bind-pmf-cong [fundef-cong]*:

assumes $p = q$

shows $(\bigwedge x. x \in set\text{-pmf} q \implies f x = g x) \implies bind\text{-pmf} p f = bind\text{-pmf} q g$

unfolding $\langle p = q \rangle$ [*symmetric*] *measure-pmf-inject[symmetric]* *bind-pmf.rep-eq*

by (*auto simp: AE-measure-pmf-iff Pi-iff space-subprob-algebra subprob-space-measure-pmf sets-bind[where N=count-space UNIV]*) *emeasure-bind[where*

N=count-space UNIV]

intro!: nn-integral-cong-AE measure-eqI)

lemma *bind-pmf-cong-simp*:

$p = q \implies (\bigwedge x. x \in set\text{-pmf} q = simp \implies f x = g x) \implies bind\text{-pmf} p f = bind\text{-pmf} q g$

by (*simp add: simp-implies-def cong: bind-pmf-cong*)

lemma *measure-pmf-bind*: $measure\text{-pmf} (bind\text{-pmf} M f) = (measure\text{-pmf} M \gg (\lambda x. measure\text{-pmf} (f x)))$

by *transfer simp*

lemma *nn-integral-bind-pmf[simp]*: $(\int^+ x. f x \partial bind\text{-pmf} M N) = (\int^+ x. \int^+ y. f y \partial N x \partial M)$

using *measurable-measure-pmf[of N]*

unfolding *measure-pmf-bind*

apply (*intro nn-integral-bind[where B=count-space UNIV]*)

apply *auto*

done

lemma *emeasure-bind-pmf[simp]*: $emeasure (bind\text{-pmf} M N) X = (\int^+ x. emeasure (N x) X \partial M)$

using *measurable-measure-pmf[of N]*

unfolding *measure-pmf-bind*

by (*subst emeasure-bind[where N=count-space UNIV]*) *auto*

lift-definition *return-pmf* :: $'a \Rightarrow 'a$ pmf **is** *return (count-space UNIV)*

by (*auto intro!: prob-space-return simp: AE-return measure-return*)

lemma *bind-return-pmf*: $bind\text{-pmf} (return\text{-pmf} x) f = f x$

by *transfer*

(*auto intro!*: *prob-space-imp-subprob-space* *bind-return*[**where** $N = \text{count-space UNIV}$]

simp: *space-subprob-algebra*)

lemma *set-return-pmf*[*simp*]: *set-pmf* (*return-pmf* x) = $\{x\}$
by *transfer* (*auto simp add*: *measure-return split*: *split-indicator*)

lemma *bind-return-pmf'*: *bind-pmf* N *return-pmf* = N

proof (*transfer*, *clarify*)

fix $N :: 'a \text{ measure}$ **assume** *sets* $N = \text{UNIV}$ **then show** $N \gg= \text{return (count-space UNIV) = } N$

by (*subst return-sets-cong*[**where** $N = N$]) (*simp-all add*: *bind-return'*)

qed

lemma *bind-assoc-pmf*: *bind-pmf* (*bind-pmf* A B) C = *bind-pmf* A ($\lambda x.$ *bind-pmf* (B x) C)

by *transfer*

(*auto intro!*: *bind-assoc*[**where** $N = \text{count-space UNIV}$ **and** $R = \text{count-space UNIV}$]

simp: *measurable-def space-subprob-algebra prob-space-imp-subprob-space*)

definition *map-pmf* f M = *bind-pmf* M ($\lambda x.$ *return-pmf* (f x))

lemma *map-bind-pmf*: *map-pmf* f (*bind-pmf* M g) = *bind-pmf* M ($\lambda x.$ *map-pmf* f (g x))

by (*simp add*: *map-pmf-def bind-assoc-pmf*)

lemma *bind-map-pmf*: *bind-pmf* (*map-pmf* f M) g = *bind-pmf* M ($\lambda x.$ g (f x))

by (*simp add*: *map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *map-pmf-transfer*[*transfer-rule*]:

rel-fun (=) (*rel-fun cr-pmf cr-pmf*) (λf $M.$ *distr* M (*count-space UNIV*) f)
map-pmf

proof –

have *rel-fun* (=) (*rel-fun pmf-as-measure.cr-pmf pmf-as-measure.cr-pmf*)

(λf $M.$ $M \gg= (\text{return (count-space UNIV) o } f)$) *map-pmf*

unfolding *map-pmf-def*[*abs-def*] *comp-def* **by** *transfer-prover*

then show *?thesis*

by (*force simp*: *rel-fun-def cr-pmf-def bind-return-distr*)

qed

lemma *map-pmf-rep-eq*:

measure-pmf (*map-pmf* f M) = *distr* (*measure-pmf* M) (*count-space UNIV*) f

unfolding *map-pmf-def bind-pmf.rep-eq comp-def return-pmf.rep-eq*

using *bind-return-distr*[*of* M f *count-space UNIV*] **by** (*simp add*: *comp-def*)

lemma *map-pmf-id*[*simp*]: *map-pmf* *id* = *id*

by (*rule*, *transfer*) (*auto simp*: *emeasure-distr measurable-def intro!*: *measure-eqI*)

lemma *map-pmf-ident*[simp]: $\text{map-pmf } (\lambda x. x) = (\lambda x. x)$
using *map-pmf-id* **unfolding** *id-def* .

lemma *map-pmf-compose*: $\text{map-pmf } (f \circ g) = \text{map-pmf } f \circ \text{map-pmf } g$
by (*rule, transfer*) (*simp add: distr-distr[symmetric, where N=count-space UNIV]*
measurable-def)

lemma *map-pmf-comp*: $\text{map-pmf } f (\text{map-pmf } g M) = \text{map-pmf } (\lambda x. f (g x)) M$
using *map-pmf-compose[of f g]* **by** (*simp add: comp-def*)

lemma *map-pmf-cong*: $p = q \implies (\bigwedge x. x \in \text{set-pmf } q \implies f x = g x) \implies \text{map-pmf } f p = \text{map-pmf } f q$
unfolding *map-pmf-def* **by** (*rule bind-pmf-cong*) *auto*

lemma *pmf-set-map*: $\text{set-pmf } \circ \text{map-pmf } f = (\cdot) f \circ \text{set-pmf}$
by (*auto simp add: comp-def fun-eq-iff map-pmf-def*)

lemma *set-map-pmf*[simp]: $\text{set-pmf } (\text{map-pmf } f M) = f' \text{set-pmf } M$
using *pmf-set-map[of f]* **by** (*auto simp: comp-def fun-eq-iff*)

lemma *emeasure-map-pmf*[simp]: $\text{emeasure } (\text{map-pmf } f M) X = \text{emeasure } M (f -' X)$
unfolding *map-pmf-rep-eq* **by** (*subst emeasure-distr*) *auto*

lemma *measure-map-pmf*[simp]: $\text{measure } (\text{map-pmf } f M) X = \text{measure } M (f -' X)$
using *emeasure-map-pmf[of f M X]* **by** (*simp add: measure-pmf.emeasure-eq-measure*
measure-nonneg)

lemma *nn-integral-map-pmf*[simp]: $(\int^+ x. f x \partial \text{map-pmf } g M) = (\int^+ x. f (g x) \partial M)$
unfolding *map-pmf-rep-eq* **by** (*intro nn-integral-distr*) *auto*

lemma *ennreal-pmf-map*: $\text{pmf } (\text{map-pmf } f p) x = (\int^+ y. \text{indicator } (f -' \{x\}) y \partial \text{measure-pmf } p)$
proof (*transfer fixing: f x*)
fix *p* :: 'b *measure*
presume *prob-space p*
then interpret *prob-space p* .
presume *sets p = UNIV*
then show *ennreal (measure (distr p (count-space UNIV) f) {x}) = integral^N p (indicator (f -' {x}))*
by (*simp add: measure-distr measurable-def emeasure-eq-measure*)
qed *simp-all*

lemma *pmf-map*: $\text{pmf } (\text{map-pmf } f p) x = \text{measure } p (f -' \{x\})$
proof (*transfer fixing: f x*)
fix *p* :: 'b *measure*
presume *prob-space p*

then interpret *prob-space p* .
presume *sets p = UNIV*
then show $\text{measure } (\text{distr } p \text{ (count-space UNIV) } f) \{x\} = \text{measure } p (f - \{x\})$
by(*simp add: measure-distr measurable-def emeasure-eq-measure*)
qed *simp-all*

lemma *nn-integral-pmf*: $(\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } A) = \text{emeasure } (\text{measure-pmf } p) \ A$

proof –

have $(\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } A) = (\int^+ x. \text{pmf } p \ x \ \partial \text{count-space } (A \cap \text{set-pmf } p))$

by(*auto simp add: nn-integral-count-space-indicator indicator-def set-pmf-iff intro: nn-integral-cong*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) (\bigcup x \in A \cap \text{set-pmf } p. \{x\})$

by(*subst emeasure-UN-countable*)(*auto simp add: emeasure-pmf-single disjoint-family-on-def*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) ((\bigcup x \in A \cap \text{set-pmf } p. \{x\}) \cup \{x. x \in A \wedge x \notin \text{set-pmf } p\})$

by(*rule emeasure-Un-null-set[symmetric]*)(*auto intro: in-null-sets-measure-pmfI*)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) \ A$

by(*auto intro: arg-cong2[where f=emeasure]*)

finally show *?thesis* .

qed

lemma *integral-map-pmf*[*simp*]:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

shows $\text{integral}^L (\text{map-pmf } g \ p) \ f = \text{integral}^L \ p (\lambda x. f (g \ x))$

by (*simp add: integral-distr map-pmf-rep-eq*)

lemma *integrable-map-pmf-eq* [*simp*]:

fixes $g :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

shows $\text{integrable } (\text{map-pmf } f \ p) \ g \longleftrightarrow \text{integrable } (\text{measure-pmf } p) (\lambda x. g (f \ x))$

by (*subst map-pmf-rep-eq, subst integrable-distr-eq auto*)

lemma *integrable-map-pmf* [*intro*]:

fixes $g :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$

shows $\text{integrable } (\text{measure-pmf } p) (\lambda x. g (f \ x)) \Longrightarrow \text{integrable } (\text{map-pmf } f \ p) \ g$

by (*subst integrable-map-pmf-eq*)

lemma *pmf-abs-summable* [*intro*]: *pmf p abs-summable-on A*

by (*rule abs-summable-on-subset[OF - subset-UNIV]*)

(*auto simp: abs-summable-on-def integrable-iff-bounded nn-integral-pmf*)

lemma *measure-pmf-conv-infsetsum*: $\text{measure } (\text{measure-pmf } p) \ A = \text{infsetsum } (\text{pmf } p) \ A$

unfolding *infsetsum-def* **by** (*simp add: integral-eq-nn-integral nn-integral-pmf measure-def*)

lemma *infsetsum-pmf-eq-1*:

assumes $\text{set-pmf } p \subseteq A$

shows $\text{infsetsum } (\text{pmf } p) A = 1$
proof –
have $\text{infsetsum } (\text{pmf } p) A = \text{lebesgue-integral } (\text{count-space } UNIV) (\text{pmf } p)$
using **assms** **unfolding** $\text{infsetsum-altdef set-lebesgue-integral-def}$
by $(\text{intro Bochner-Integration.integral-cong}) (\text{auto simp: indicator-def set-pmf-eq})$
also have $\dots = 1$
by $(\text{subst integral-eq-nn-integral}) (\text{auto simp: nn-integral-pmf})$
finally show $?thesis$.
qed

lemma $\text{map-return-pmf } [simp]: \text{map-pmf } f (\text{return-pmf } x) = \text{return-pmf } (f x)$
by $\text{transfer } (\text{simp add: distr-return})$

lemma $\text{map-pmf-const}[simp]: \text{map-pmf } (\lambda-. c) M = \text{return-pmf } c$
by $\text{transfer } (\text{auto simp: prob-space.distr-const})$

lemma $\text{pmf-return } [simp]: \text{pmf } (\text{return-pmf } x) y = \text{indicator } \{y\} x$
by $\text{transfer } (\text{simp add: measure-return})$

lemma $\text{nn-integral-return-pmf}[simp]: 0 \leq f x \implies (\int^+ x. f x \partial \text{return-pmf } x) = f x$
unfolding return-pmf.rep-eq **by** $(\text{intro nn-integral-return}) \text{ auto}$

lemma $\text{emeasure-return-pmf}[simp]: \text{emeasure } (\text{return-pmf } x) X = \text{indicator } X x$
unfolding return-pmf.rep-eq **by** $(\text{intro emeasure-return}) \text{ auto}$

lemma $\text{measure-return-pmf } [simp]: \text{measure-pmf.prob } (\text{return-pmf } x) A = \text{indicator } A x$

proof –
have $\text{ennreal } (\text{measure-pmf.prob } (\text{return-pmf } x) A) =$
 $\text{emeasure } (\text{measure-pmf } (\text{return-pmf } x)) A$
by $(\text{simp add: measure-pmf.emeasure-eq-measure})$
also have $\dots = \text{ennreal } (\text{indicator } A x)$ **by** $(\text{simp add: ennreal-indicator})$
finally show $?thesis$ **by** simp

qed

lemma $\text{return-pmf-inj}[simp]: \text{return-pmf } x = \text{return-pmf } y \longleftrightarrow x = y$
by $(\text{metis insertI1 set-return-pmf singletonD})$

lemma $\text{map-pmf-eq-return-pmf-iff}:$
 $\text{map-pmf } f p = \text{return-pmf } x \longleftrightarrow (\forall y \in \text{set-pmf } p. f y = x)$

proof
assume $\text{map-pmf } f p = \text{return-pmf } x$
then have $\text{set-pmf } (\text{map-pmf } f p) = \text{set-pmf } (\text{return-pmf } x)$ **by** simp
then show $\forall y \in \text{set-pmf } p. f y = x$ **by** auto

next

assume $\forall y \in \text{set-pmf } p. f y = x$
then show $\text{map-pmf } f p = \text{return-pmf } x$
unfolding $\text{map-pmf-const}[symmetric, of - p]$ **by** $(\text{intro map-pmf-cong}) \text{ auto}$

qed

definition $pair\text{-}pmf\ A\ B = bind\text{-}pmf\ A\ (\lambda x. bind\text{-}pmf\ B\ (\lambda y. return\text{-}pmf\ (x, y)))$

lemma $pmf\text{-}pair$: $pmf\ (pair\text{-}pmf\ M\ N)\ (a, b) = pmf\ M\ a * pmf\ N\ b$
unfolding $pair\text{-}pmf\text{-}def\ pmf\text{-}bind\ pmf\text{-}return$
apply ($subst\ integral\text{-}measure\text{-}pmf\text{-}real[\mathbf{where}\ A=\{b\}]$)
apply ($auto\ simp: indicator\text{-}eq\text{-}0\text{-}iff$)
apply ($subst\ integral\text{-}measure\text{-}pmf\text{-}real[\mathbf{where}\ A=\{a\}]$)
apply ($auto\ simp: indicator\text{-}eq\text{-}0\text{-}iff\ sum\text{-}nonneg\text{-}eq\text{-}0\text{-}iff\ pmf\text{-}nonneg$)
done

lemma $set\text{-}pair\text{-}pmf[simp]$: $set\text{-}pmf\ (pair\text{-}pmf\ A\ B) = set\text{-}pmf\ A \times set\text{-}pmf\ B$
unfolding $pair\text{-}pmf\text{-}def\ set\text{-}bind\text{-}pmf\ set\text{-}return\text{-}pmf$ **by** $auto$

lemma $measure\text{-}pmf\text{-}in\text{-}subprob\text{-}space[measurable\ (raw)]$:
 $measure\text{-}pmf\ M \in space\ (subprob\text{-}algebra\ (count\text{-}space\ UNIV))$
by ($simp\ add: space\text{-}subprob\text{-}algebra\ intro\text{-}locales$)

lemma $nn\text{-}integral\text{-}pair\text{-}pmf'$: $(\int^{+x}. f\ x\ \partial pair\text{-}pmf\ A\ B) = (\int^{+a}. \int^{+b}. f\ (a, b)\ \partial B\ \partial A)$

proof –

have $(\int^{+x}. f\ x\ \partial pair\text{-}pmf\ A\ B) = (\int^{+x}. f\ x * indicator\ (A \times B)\ x\ \partial pair\text{-}pmf\ A\ B)$
by ($auto\ simp: AE\text{-}measure\text{-}pmf\text{-}iff\ intro!: nn\text{-}integral\text{-}cong\text{-}AE$)
also have $\dots = (\int^{+a}. \int^{+b}. f\ (a, b) * indicator\ (A \times B)\ (a, b)\ \partial B\ \partial A)$
by ($simp\ add: pair\text{-}pmf\text{-}def$)
also have $\dots = (\int^{+a}. \int^{+b}. f\ (a, b)\ \partial B\ \partial A)$
by ($auto\ intro!: nn\text{-}integral\text{-}cong\text{-}AE\ simp: AE\text{-}measure\text{-}pmf\text{-}iff$)
finally show $?thesis$.

qed

lemma $bind\text{-}pair\text{-}pmf$:

assumes $M[measurable]$: $M \in measurable\ (count\text{-}space\ UNIV \otimes_M count\text{-}space\ UNIV)\ (subprob\text{-}algebra\ N)$

shows $measure\text{-}pmf\ (pair\text{-}pmf\ A\ B) \ggg M = (measure\text{-}pmf\ A \ggg (\lambda x. measure\text{-}pmf\ B \ggg (\lambda y. M\ (x, y))))$

(**is** $?L = ?R$)

proof ($rule\ measure\text{-}eqI$)

have $M'[measurable]$: $M \in measurable\ (pair\text{-}pmf\ A\ B)\ (subprob\text{-}algebra\ N)$

using $M[THEN\ measurable\text{-}space]$ **by** ($simp\text{-}all\ add: space\text{-}pair\text{-}measure$)

note $measurable\text{-}bind[\mathbf{where}\ N=count\text{-}space\ UNIV, measurable]$

note $measure\text{-}pmf\text{-}in\text{-}subprob\text{-}space[simp]$

have $sets\text{-}eq\text{-}N$: $sets\ ?L = N$

by ($subst\ sets\text{-}bind[OF\ sets\text{-}kernel[OF\ M']]$) $auto$

show $sets\ ?L = sets\ ?R$

using $measurable\text{-}space[OF\ M]$

by ($simp\ add: sets\text{-}eq\text{-}N\ space\text{-}pair\text{-}measure\ space\text{-}subprob\text{-}algebra$)

```

fix X assume X ∈ sets ?L
then have X[measurable]: X ∈ sets N
  unfolding sets-eq-N .
then show emeasure ?L X = emeasure ?R X
  apply (simp add: emeasure-bind[OF - M' X])
  apply (simp add: nn-integral-bind[where B=count-space UNIV] pair-pmf-def
measure-pmf-bind[of A]
      nn-integral-measure-pmf-finite)
  apply (subst emeasure-bind[OF - - X])
  apply measurable
  apply (subst emeasure-bind[OF - - X])
  apply measurable
  done
qed

```

```

lemma map-fst-pair-pmf: map-pmf fst (pair-pmf A B) = A
by (simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')

```

```

lemma map-snd-pair-pmf: map-pmf snd (pair-pmf A B) = B
by (simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')

```

```

lemma nn-integral-pmf':
  inj-on f A ⇒ (∫+ x. pmf p (f x) ∂count-space A) = emeasure p (f ' A)
by (subst nn-integral-bij-count-space[where g=f and B=f'A])
  (auto simp: bij-betw-def nn-integral-pmf)

```

```

lemma pmf-le-0-iff[simp]: pmf M p ≤ 0 ⇔ pmf M p = 0
using pmf-nonneg[of M p] by arith

```

```

lemma min-pmf-0[simp]: min (pmf M p) 0 = 0 min 0 (pmf M p) = 0
using pmf-nonneg[of M p] by arith+

```

```

lemma pmf-eq-0-set-pmf: pmf M p = 0 ⇔ p ∉ set-pmf M
unfolding set-pmf-iff by simp

```

```

lemma pmf-map-inj: inj-on f (set-pmf M) ⇒ x ∈ set-pmf M ⇒ pmf (map-pmf
f M) (f x) = pmf M x
by (auto simp: pmf.rep-eq map-pmf.rep-eq measure-distr AE-measure-pmf-iff
inj-onD
  intro!: measure-pmf.finite-measure-eq-AE)

```

```

lemma pair-return-pmf [simp]: pair-pmf (return-pmf x) (return-pmf y) = return-pmf
(x, y)
by (auto simp: pair-pmf-def bind-return-pmf)

```

```

lemma pmf-map-inj': inj f ⇒ pmf (map-pmf f M) (f x) = pmf M x
apply (cases x ∈ set-pmf M)
apply (simp add: pmf-map-inj[OF subset-inj-on])
apply (simp add: pmf-eq-0-set-pmf[symmetric])

```

apply(*auto simp add: pmf-eq-0-set-pmf dest: injD*)
done

lemma *expectation-pair-pmf-fst* [*simp*]:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{measure-pmf.expectation (pair-pmf } p \text{ } q) (\lambda x. f (\text{fst } x)) = \text{measure-pmf.expectation } p \text{ } f$
proof –
have $\text{measure-pmf.expectation (pair-pmf } p \text{ } q) (\lambda x. f (\text{fst } x)) =$
 $\text{measure-pmf.expectation (map-pmf fst (pair-pmf } p \text{ } q)) f$ **by** *simp*
also have $\text{map-pmf fst (pair-pmf } p \text{ } q) = p$
by (*simp add: map-fst-pair-pmf*)
finally show *?thesis* .
qed

lemma *expectation-pair-pmf-snd* [*simp*]:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{measure-pmf.expectation (pair-pmf } p \text{ } q) (\lambda x. f (\text{snd } x)) = \text{measure-pmf.expectation } q \text{ } f$
proof –
have $\text{measure-pmf.expectation (pair-pmf } p \text{ } q) (\lambda x. f (\text{snd } x)) =$
 $\text{measure-pmf.expectation (map-pmf snd (pair-pmf } p \text{ } q)) f$ **by** *simp*
also have $\text{map-pmf snd (pair-pmf } p \text{ } q) = q$
by (*simp add: map-snd-pair-pmf*)
finally show *?thesis* .
qed

lemma *pmf-map-outside*: $x \notin f \text{ ` set-pmf } M \implies \text{pmf (map-pmf } f \text{ } M) x = 0$
unfolding *pmf-eq-0-set-pmf* **by** *simp*

lemma *measurable-set-pmf*[*measurable*]: $\text{Measurable.pred (count-space UNIV)} (\lambda x. x \in \text{set-pmf } M)$
by *simp*

20.3 PMFs as function

context
fixes $f :: 'a \Rightarrow \text{real}$
assumes *nonneg*: $\bigwedge x. 0 \leq f x$
assumes *prob*: $(f \text{ } ^+ x. f x \text{ } \partial \text{count-space UNIV}) = 1$
begin

lift-definition *embed-pmf* :: $'a \text{ pmf is density (count-space UNIV) (ennreal } \circ f)$

proof (*intro conjI*)
have $*[simp]: \bigwedge x y. \text{ennreal (} f y) * \text{indicator } \{x\} y = \text{ennreal (} f x) * \text{indicator } \{x\} y$
by (*simp split: split-indicator*)
show $AE x \text{ in density (count-space UNIV) (ennreal } \circ f).$
 $\text{measure (density (count-space UNIV) (ennreal } \circ f)) \{x\} \neq 0$

by (simp add: AE-density nonneg measure-def emeasure-density max-def)
 show prob-space (density (count-space UNIV) (ennreal ∘ f))
 by standard (simp add: emeasure-density prob)
 qed simp

lemma pmf-embed-pmf: pmf embed-pmf x = f x

proof transfer

have *[simp]: $\bigwedge x y. \text{ennreal } (f y) * \text{indicator } \{x\} y = \text{ennreal } (f x) * \text{indicator } \{x\} y$

by (simp split: split-indicator)

fix x show measure (density (count-space UNIV) (ennreal ∘ f)) {x} = f x

by transfer (simp add: measure-def emeasure-density nonneg max-def)

qed

lemma set-embed-pmf: set-pmf embed-pmf = {x. f x ≠ 0}

by(auto simp add: set-pmf-eq pmf-embed-pmf)

end

lemma embed-pmf-transfer:

rel-fun (eq-onp (λf. (∀x. 0 ≤ f x) ∧ (∫⁺x. ennreal (f x) ∂count-space UNIV) = 1)) pmf-as-measure.cr-pmf (λf. density (count-space UNIV) (ennreal ∘ f)) embed-pmf

by (auto simp: rel-fun-def eq-onp-def embed-pmf.transfer)

lemma measure-pmf-eq-density: measure-pmf p = density (count-space UNIV) (pmf p)

proof (transfer, elim conjE)

fix M :: 'a measure assume [simp]: sets M = UNIV and ae: AE x in M. measure M {x} ≠ 0

assume prob-space M then interpret prob-space M .

show M = density (count-space UNIV) (λx. ennreal (measure M {x}))

proof (rule measure-eqI)

fix A :: 'a set

have (∫⁺x. ennreal (measure M {x}) * indicator A x ∂count-space UNIV) = (∫⁺x. emeasure M {x} * indicator (A ∩ {x. measure M {x} ≠ 0}) x ∂count-space UNIV)

by (auto intro!: nn-integral-cong simp: emeasure-eq-measure split: split-indicator)

also have ... = (∫⁺x. emeasure M {x} ∂count-space (A ∩ {x. measure M {x} ≠ 0}))

by (subst nn-integral-restrict-space[symmetric]) (auto simp: restrict-count-space)

also have ... = emeasure M (⋃x∈(A ∩ {x. measure M {x} ≠ 0}). {x})

by (intro emeasure-UN-countable[symmetric] countable-Int2 countable-support) (auto simp: disjoint-family-on-def)

also have ... = emeasure M A

using ae by (intro emeasure-eq-AE) auto

finally show emeasure M A = emeasure (density (count-space UNIV) (λx. ennreal (measure M {x}))) A

using emeasure-space-1 by (simp add: emeasure-density)

qed *simp*
qed

lemma *td-pmf-embed-pmf*:

type-definition pmf embed-pmf $\{f :: 'a \Rightarrow \text{real}. (\forall x. 0 \leq f x) \wedge (\int^+ x. \text{ennreal } (f x) \partial \text{count-space UNIV}) = 1\}$

unfolding *type-definition-def*

proof *safe*

fix $p :: 'a \text{ pmf}$

have $(\int^+ x. 1 \partial \text{measure-pmf } p) = 1$

using *measure-pmf.emeasure-space-1* [of p] **by** *simp*

then show $*$: $(\int^+ x. \text{ennreal } (pmf p x) \partial \text{count-space UNIV}) = 1$

by (*simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg del: nn-integral-const*)

show *embed-pmf* $(pmf p) = p$

by (*intro measure-pmf-inject* [THEN *iffD1*])

(*simp add: * embed-pmf.rep-eq pmf-nonneg measure-pmf-eq-density* [of p])

comp-def)

next

fix $f :: 'a \Rightarrow \text{real}$ **assume** $\forall x. 0 \leq f x$ $(\int^+ x. f x \partial \text{count-space UNIV}) = 1$

then show *pmf* $(\text{embed-pmf } f) = f$

by (*auto intro!: pmf-embed-pmf*)

qed (*rule pmf-nonneg*)

end

lemma *nn-integral-measure-pmf*: $(\int^+ x. f x \partial \text{measure-pmf } p) = \int^+ x. \text{ennreal } (pmf p x) * f x \partial \text{count-space UNIV}$

by (*simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg*)

lemma *integral-measure-pmf*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$

assumes A : *finite* A

shows $(\bigwedge a. a \in \text{set-pmf } M \implies f a \neq 0 \implies a \in A) \implies (\text{LINT } x | M. f x) = (\sum_{a \in A. pmf M a *_{\mathbb{R}} f a)$

unfolding *measure-pmf-eq-density*

apply (*simp add: integral-density*)

apply (*subst lebesgue-integral-count-space-finite-support*)

apply (*auto intro!: finite-subset* [OF - $\langle \text{finite } A \rangle$] *sum.mono-neutral-left simp: pmf-eq-0-set-pmf*)

done

lemma *expectation-return-pmf* [*simp*]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$

shows *measure-pmf.expectation* $(\text{return-pmf } x) f = f x$

by (*subst integral-measure-pmf* [of $\{x\}$]) *simp-all*

lemma *pmf-expectation-bind*:

fixes $p :: 'a \text{ pmf}$ **and** $f :: 'a \Rightarrow 'b \text{ pmf}$

and $h :: 'b \Rightarrow 'c :: \{\text{banach, second-countable-topology}\}$
assumes $\text{finite } A \wedge x. x \in A \implies \text{finite } (\text{set-pmf } (f x)) \text{ set-pmf } p \subseteq A$
shows $\text{measure-pmf.expectation } (p \gg f) h =$
 $(\sum a \in A. \text{pmf } p a *_{\mathbb{R}} \text{measure-pmf.expectation } (f a) h)$
proof –
have $\text{measure-pmf.expectation } (p \gg f) h = (\sum a \in (\bigcup x \in A. \text{set-pmf } (f x)). \text{pmf } (p \gg f) a *_{\mathbb{R}} h a)$
using *assms* **by** (*intro integral-measure-pmf*) *auto*
also have $\dots = (\sum x \in (\bigcup x \in A. \text{set-pmf } (f x)). (\sum a \in A. (\text{pmf } p a * \text{pmf } (f a) x) *_{\mathbb{R}} h x))$
proof (*intro sum.cong refl, goal-cases*)
case (1 *x*)
thus ?*case*
by (*subst pmf-bind, subst integral-measure-pmf[of A]*)
(insert assms, auto simp: scaleR-sum-left)
qed
also have $\dots = (\sum j \in A. \text{pmf } p j *_{\mathbb{R}} (\sum i \in (\bigcup x \in A. \text{set-pmf } (f x)). \text{pmf } (f j) i *_{\mathbb{R}} h i))$
by (*subst sum.swap*) (*simp add: scaleR-sum-right*)
also have $\dots = (\sum j \in A. \text{pmf } p j *_{\mathbb{R}} \text{measure-pmf.expectation } (f j) h)$
proof (*intro sum.cong refl, goal-cases*)
case (1 *x*)
thus ?*case*
by (*subst integral-measure-pmf[of ($\bigcup x \in A. \text{set-pmf } (f x)$)]*)
(insert assms, auto simp: scaleR-sum-left)
qed
finally show ?*thesis* .
qed

lemma *continuous-on-LINT-pmf*: — This is dominated convergence!?

fixes $f :: 'i \Rightarrow 'a :: \text{topological-space} \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
assumes $f: \bigwedge i. i \in \text{set-pmf } M \implies \text{continuous-on } A (f i)$
and $\text{bnd}: \bigwedge a. a \in A \implies i \in \text{set-pmf } M \implies \text{norm } (f i a) \leq B$
shows $\text{continuous-on } A (\lambda a. \text{LINT } i | M. f i a)$
proof *cases*
assume *finite M* **with** *f* **show** ?*thesis*
using *integral-measure-pmf[OF <finite M>]*
by (*subst integral-measure-pmf[OF <finite M>]*)
(auto intro!: continuous-on-sum continuous-on-scaleR continuous-on-const)
next
assume *infinite M*
let ?*f* = $\lambda i x. \text{pmf } (\text{map-pmf } (\text{to-nat-on } M) M) i *_{\mathbb{R}} f (\text{from-nat-into } M i) x$

show ?*thesis*
proof (*rule uniform-limit-theorem*)
show $\forall_F n \text{ in sequentially. continuous-on } A (\lambda a. \sum i < n. ?f i a)$
by (*intro always-eventually allI continuous-on-sum continuous-on-scaleR continuous-on-const f*)
from-nat-into set-pmf-not-empty)

```

show uniform-limit A ( $\lambda n a. \sum i < n. ?f i a$ ) ( $\lambda a. LINT i | M. f i a$ ) sequentially
proof (subst uniform-limit-cong [where  $g = \lambda n a. \sum i < n. ?f i a$ ])
  fix a assume  $a \in A$ 
  have 1: ( $LINT i | M. f i a$ ) = ( $LINT i | map-pmf (to-nat-on M) M. f (from-nat-into$ 
M i) a)
    by (auto intro!: integral-cong-AE AE-pmfI)
  have 2:  $\dots = (LINT i | count-space UNIV. pmf (map-pmf (to-nat-on M) M)$ 
i *R f (from-nat-into M i) a)
    by (simp add: measure-pmf-eq-density integral-density)
  have ( $\lambda n. ?f n a$ ) sums ( $LINT i | M. f i a$ )
    unfolding 1 2
  proof (intro sums-integral-count-space-nat)
    have A: integrable M ( $\lambda i. f i a$ )
      using  $\langle a \in A \rangle$  by (auto intro!: measure-pmf.integrable-const-bound AE-pmfI
bnd)
    have integrable ( $map-pmf (to-nat-on M) M$ ) ( $\lambda i. f (from-nat-into M i) a$ )
      by (auto simp add: map-pmf-rep-eq integrable-distr-eq intro!: AE-pmfI
integrable-cong-AE-imp[OF A])
    then show integrable (count-space UNIV) ( $\lambda n. ?f n a$ )
      by (simp add: measure-pmf-eq-density integrable-density)
    qed
  then show ( $LINT i | M. f i a$ ) = ( $\sum n. ?f n a$ )
    by (simp add: sums-unique)
  next
  show uniform-limit A ( $\lambda n a. \sum i < n. ?f i a$ ) ( $\lambda a. (\sum n. ?f n a)$ ) sequentially
  proof (rule Weierstrass-m-test)
    fix n a assume  $a \in A$ 
    then show norm ( $?f n a$ )  $\leq pmf (map-pmf (to-nat-on M) M) n * B$ 
      using bnd by (auto intro!: mult-mono simp: from-nat-into set-pmf-not-empty)
    next
    have integrable ( $map-pmf (to-nat-on M) M$ ) ( $\lambda n. B$ )
      by auto
    then show summable ( $\lambda n. pmf (map-pmf (to-nat-on (set-pmf M)) M) n * B$ )
      by (fastforce simp add: measure-pmf-eq-density integrable-density inte-
grable-count-space-nat-iff summable-mult2)
    qed
  qed simp
qed simp
qed

```

lemma *continuous-on-LBINT*:

```

fixes f :: real  $\Rightarrow$  real
assumes f:  $\bigwedge b. a \leq b \implies set-integrable lborel \{a..b\} f$ 
shows continuous-on UNIV ( $\lambda b. LBINT x:\{a..b\}. f x$ )
proof (subst set-borel-integral-eq-integral)
  { fix b :: real assume  $a \leq b$ 
    from f [OF this] have continuous-on  $\{a..b\}$  ( $\lambda b. integral \{a..b\} f$ )
    by (intro indefinite-integral-continuous-1 set-borel-integral-eq-integral) }

```



```

note * = this

have continuous-on ( $\bigcup b \in \{a.. \}. \{a <.. < b\}$ ) ( $\lambda b. \text{integral } \{a..b\} f$ )
proof (intro continuous-on-open-UN)
  show  $b \in \{a.. \} \implies \text{continuous-on } \{a <.. < b\}$  ( $\lambda b. \text{integral } \{a..b\} f$ ) for  $b$ 
    using *[of b] by (rule continuous-on-subset) auto
qed simp
also have ( $\bigcup b \in \{a.. \}. \{a <.. < b\}$ ) =  $\{a <.. \}$ 
  by (auto simp: lt-ex gt-ex less-imp-le) (simp add: Bex-def less-imp-le gt-ex cong: rev-conj-cong)
finally have continuous-on  $\{a+1 ..\}$  ( $\lambda b. \text{integral } \{a..b\} f$ )
  by (rule continuous-on-subset) auto
moreover have continuous-on  $\{a..a+1\}$  ( $\lambda b. \text{integral } \{a..b\} f$ )
  by (rule *) simp
moreover
have  $x \leq a \implies \{a..x\} = (\text{if } a = x \text{ then } \{a\} \text{ else } \{\})$  for  $x$ 
  by auto
then have continuous-on  $\{..a\}$  ( $\lambda b. \text{integral } \{a..b\} f$ )
  by (subst continuous-on-cong[OF refl, where g= $\lambda x. 0$ ]) (auto intro!: continuous-on-const)
ultimately have continuous-on ( $\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\}$ ) ( $\lambda b. \text{integral } \{a..b\} f$ )
  by (intro continuous-on-closed-UN) auto
also have  $\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\} = \text{UNIV}$ 
  by auto
finally show continuous-on UNIV ( $\lambda b. \text{integral } \{a..b\} f$ )
  by auto
next
show set-integrable lborel  $\{a..b\} f$  for  $b$ 
  using  $f$  by (cases a ≤ b) auto
qed

locale pmf-as-function
begin

setup-lifting td-pmf-embed-pmf

lemma set-pmf-transfer[transfer-rule]:
  assumes bi-total A
  shows rel-fun (pcr-pmf A) (rel-set A) ( $\lambda f. \{x. f x \neq 0\}$ ) set-pmf
  using  $\langle \text{bi-total } A \rangle$ 
  by (auto simp: pcr-pmf-def cr-pmf-def rel-fun-def rel-set-def bi-total-def Bex-def set-pmf-iff)
    metis+

end

context
begin

```

interpretation *pmf-as-function* .

lemma *pmf-eqI*: $(\bigwedge i. \text{pmf } M \ i = \text{pmf } N \ i) \implies M = N$
by *transfer auto*

lemma *pmf-eq-iff*: $M = N \longleftrightarrow (\forall i. \text{pmf } M \ i = \text{pmf } N \ i)$
by (*auto intro: pmf-eqI*)

lemma *pmf-neq-exists-less*:

assumes $M \neq N$

shows $\exists x. \text{pmf } M \ x < \text{pmf } N \ x$

proof (*rule ccontr*)

assume $\neg(\exists x. \text{pmf } M \ x < \text{pmf } N \ x)$

hence *ge*: $\text{pmf } M \ x \geq \text{pmf } N \ x$ **for** x **by** (*auto simp: not-less*)

from *assms* **obtain** x **where** $\text{pmf } M \ x \neq \text{pmf } N \ x$ **by** (*auto simp: pmf-eq-iff*)

with *ge[of x]* **have** *gt*: $\text{pmf } M \ x > \text{pmf } N \ x$ **by** *simp*

have $1 = \text{measure } (\text{measure-pmf } M) \ \text{UNIV}$ **by** *simp*

also have $\dots = \text{measure } (\text{measure-pmf } N) \ \{x\} + \text{measure } (\text{measure-pmf } N) \ (\text{UNIV} - \{x\})$

by (*subst measure-pmf.finite-measure-Union [symmetric] simp-all*)

also from *gt* **have** $\text{measure } (\text{measure-pmf } N) \ \{x\} < \text{measure } (\text{measure-pmf } M) \ \{x\}$

by (*simp add: measure-pmf-single*)

also have $\text{measure } (\text{measure-pmf } N) \ (\text{UNIV} - \{x\}) \leq \text{measure } (\text{measure-pmf } M) \ (\text{UNIV} - \{x\})$

by (*subst (1 2) integral-pmf [symmetric]*)

(*intro integral-mono integrable-pmf, simp-all add: ge*)

also have $\text{measure } (\text{measure-pmf } M) \ \{x\} + \dots = 1$

by (*subst measure-pmf.finite-measure-Union [symmetric] simp-all*)

finally show *False* **by** *simp-all*

qed

lemma *bind-commute-pmf*: $\text{bind-pmf } A \ (\lambda x. \text{bind-pmf } B \ (C \ x)) = \text{bind-pmf } B \ (\lambda y. \text{bind-pmf } A \ (\lambda x. C \ x \ y))$

unfolding *pmf-eq-iff pmf-bind*

proof

fix i

interpret B : *prob-space restrict-space B B*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)

(*auto simp: AE-measure-pmf-iff*)

interpret A : *prob-space restrict-space A A*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)

(*auto simp: AE-measure-pmf-iff*)

interpret AB : *pair-prob-space restrict-space A A restrict-space B B*

by *unfold-locales*

have $(\int x. \int y. \text{pmf } (C \ x \ y) \ i \ \partial B \ \partial A) = (\int x. (\int y. \text{pmf } (C \ x \ y) \ i \ \partial \text{restrict-space})$

$B B) \partial A)$
by (rule *Bochner-Integration.integral-cong*) (auto intro!: *integral-pmf-restrict*)
also have $\dots = (\int x. (\int y. \text{pmf } (C x y) i \partial \text{restrict-space } B B) \partial \text{restrict-space } A A)$
by (intro *integral-pmf-restrict B.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)
also have $\dots = (\int y. \int x. \text{pmf } (C x y) i \partial \text{restrict-space } A A \partial \text{restrict-space } B B)$
by (rule *AB.Fubini-integral[symmetric]*)
(auto intro!: *AB.integrable-const-bound[where B=1] measurable-pair-restrict-pmf2 simp: pmf-nonneg pmf-le-1 measurable-restrict-space1*)
also have $\dots = (\int y. \int x. \text{pmf } (C x y) i \partial \text{restrict-space } A A \partial B)$
by (intro *integral-pmf-restrict[symmetric] A.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)
also have $\dots = (\int y. \int x. \text{pmf } (C x y) i \partial A \partial B)$
by (rule *Bochner-Integration.integral-cong*) (auto intro!: *integral-pmf-restrict[symmetric]*)
finally show $(\int x. \int y. \text{pmf } (C x y) i \partial B \partial A) = (\int y. \int x. \text{pmf } (C x y) i \partial A \partial B)$.
qed

lemma *pair-map-pmf1*: $\text{pair-pmf } (\text{map-pmf } f A) B = \text{map-pmf } (\text{apfst } f) (\text{pair-pmf } A B)$

proof (safe intro!: *pmf-eqI*)

fix $a :: 'a$ and $b :: 'b$

have [*simp*]: $\bigwedge c d. \text{indicator } (\text{apfst } f - \{(a, b)\}) (c, d) = \text{indicator } (f - \{a\}) c * (\text{indicator } \{b\} d :: \text{ennreal})$

by (auto split: *split-indicator*)

have $\text{ennreal } (\text{pmf } (\text{pair-pmf } (\text{map-pmf } f A) B) (a, b)) = \text{ennreal } (\text{pmf } (\text{map-pmf } (\text{apfst } f) (\text{pair-pmf } A B)) (a, b))$

unfolding *pmf-pair ennreal-pmf-map*

by (*simp add: nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-mult pmf-nonneg*)

emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)

then show $\text{pmf } (\text{pair-pmf } (\text{map-pmf } f A) B) (a, b) = \text{pmf } (\text{map-pmf } (\text{apfst } f) (\text{pair-pmf } A B)) (a, b)$

by (*simp add: pmf-nonneg*)

qed

lemma *pair-map-pmf2*: $\text{pair-pmf } A (\text{map-pmf } f B) = \text{map-pmf } (\text{apsnd } f) (\text{pair-pmf } A B)$

proof (safe intro!: *pmf-eqI*)

fix $a :: 'a$ and $b :: 'b$

have [*simp*]: $\bigwedge c d. \text{indicator } (\text{apsnd } f - \{(a, b)\}) (c, d) = \text{indicator } \{a\} c * (\text{indicator } (f - \{b\}) d :: \text{ennreal})$

by (auto split: *split-indicator*)

have $\text{ennreal } (\text{pmf } (\text{pair-pmf } A (\text{map-pmf } f B)) (a, b)) =$

```

      ennreal (pmf (map-pmf (apsnd f) (pair-pmf A B)) (a, b))
    unfolding pmf-pair ennreal-pmf-map
    by (simp add: nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-cmult
nn-integral-multc pmf-nonneg
      emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)
    then show pmf (pair-pmf A (map-pmf f B)) (a, b) = pmf (map-pmf (apsnd f)
(pair-pmf A B)) (a, b)
    by (simp add: pmf-nonneg)
qed

```

```

lemma map-pair: map-pmf (λ(a, b). (f a, g b)) (pair-pmf A B) = pair-pmf (map-pmf
f A) (map-pmf g B)
  by (simp add: pair-map-pmf2 pair-map-pmf1 map-pmf-comp split-beta')

```

end

```

lemma pair-return-pmf1: pair-pmf (return-pmf x) y = map-pmf (Pair x) y
by(simp add: pair-pmf-def bind-return-pmf map-pmf-def)

```

```

lemma pair-return-pmf2: pair-pmf x (return-pmf y) = map-pmf (λx. (x, y)) x
by(simp add: pair-pmf-def bind-return-pmf map-pmf-def)

```

```

lemma pair-pair-pmf: pair-pmf (pair-pmf u v) w = map-pmf (λ(x, (y, z)). ((x,
y), z)) (pair-pmf u (pair-pmf v w))
by(simp add: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf)

```

```

lemma pair-commute-pmf: pair-pmf x y = map-pmf (λ(x, y). (y, x)) (pair-pmf y
x)
unfolding pair-pmf-def by(subst bind-commute-pmf)(simp add: map-pmf-def bind-assoc-pmf
bind-return-pmf)

```

```

lemma set-pmf-subset-singleton: set-pmf p ⊆ {x} ↔ p = return-pmf x

```

```

proof(intro iffI pmf-eqI)
  fix i
  assume x: set-pmf p ⊆ {x}
  hence *: set-pmf p = {x} using set-pmf-not-empty[of p] by auto
  have ennreal (pmf p x) = ∫+ i. indicator {x} i ∂p by(simp add: emeasure-pmf-single)
  also have ... = ∫+ i. 1 ∂p by(rule nn-integral-cong-AE)(simp add: AE-measure-pmf-iff
*)
  also have ... = 1 by simp
  finally show pmf p i = pmf (return-pmf x) i using x
  by(auto split: split-indicator simp add: pmf-eq-0-set-pmf)
qed auto

```

```

lemma bind-eq-return-pmf:

```

```

  bind-pmf p f = return-pmf x ↔ (∀ y ∈ set-pmf p. f y = return-pmf x)
  (is ?lhs ↔ ?rhs)
proof(intro iffI strip)
  fix y

```

```

assume  $y: y \in \text{set-pmf } p$ 
assume  $?lhs$ 
hence  $\text{set-pmf } (\text{bind-pmf } p \ f) = \{x\}$  by  $\text{simp}$ 
hence  $(\bigcup_{y \in \text{set-pmf } p}. \text{set-pmf } (f \ y)) = \{x\}$  by  $\text{simp}$ 
hence  $\text{set-pmf } (f \ y) \subseteq \{x\}$  using  $y$  by  $\text{auto}$ 
thus  $f \ y = \text{return-pmf } x$  by  $(\text{simp add: set-pmf-subset-singleton})$ 
next
assume  $?: ?rhs$ 
show  $?lhs$ 
proof  $(\text{rule pmf-eqI})$ 
  fix  $i$ 
  have  $\text{ennreal } (\text{pmf } (\text{bind-pmf } p \ f) \ i) = \int^+ y. \text{ennreal } (\text{pmf } (f \ y) \ i) \ \partial p$ 
    by  $(\text{simp add: ennreal-pmf-bind})$ 
  also have  $\dots = \int^+ y. \text{ennreal } (\text{pmf } (\text{return-pmf } x) \ i) \ \partial p$ 
    by  $(\text{rule nn-integral-cong-AE})(\text{simp add: AE-measure-pmf-iff } *)$ 
  also have  $\dots = \text{ennreal } (\text{pmf } (\text{return-pmf } x) \ i)$ 
    by  $\text{simp}$ 
  finally show  $\text{pmf } (\text{bind-pmf } p \ f) \ i = \text{pmf } (\text{return-pmf } x) \ i$ 
    by  $(\text{simp add: pmf-nonneg})$ 
qed
qed

```

lemma $\text{pmf-False-conv-True}: \text{pmf } p \ \text{False} = 1 - \text{pmf } p \ \text{True}$

proof –

```

have  $\text{pmf } p \ \text{False} + \text{pmf } p \ \text{True} = \text{measure } p \ \{\text{False}\} + \text{measure } p \ \{\text{True}\}$ 
  by  $(\text{simp add: measure-pmf-single})$ 
also have  $\dots = \text{measure } p \ (\{\text{False}\} \cup \{\text{True}\})$ 
  by  $(\text{subst measure-pmf.finite-measure-Union}) \ \text{simp-all}$ 
also have  $\{\text{False}\} \cup \{\text{True}\} = \text{space } p$  by  $\text{auto}$ 
finally show  $?thesis$  by  $\text{simp}$ 
qed

```

lemma $\text{pmf-True-conv-False}: \text{pmf } p \ \text{True} = 1 - \text{pmf } p \ \text{False}$

by $(\text{simp add: pmf-False-conv-True})$

20.4 Conditional Probabilities

lemma $\text{measure-pmf-zero-iff}: \text{measure } (\text{measure-pmf } p) \ s = 0 \iff \text{set-pmf } p \ \cap \ s = \{\}$

by $(\text{subst measure-pmf.prob-eq-0}) \ (\text{auto simp: AE-measure-pmf-iff})$

context

fixes $p :: 'a \ \text{pmf}$ **and** $s :: 'a \ \text{set}$

assumes $\text{not-empty}: \text{set-pmf } p \ \cap \ s \neq \{\}$

begin

interpretation pmf-as-measure .

lemma $\text{emeasure-measure-pmf-not-zero}: \text{emeasure } (\text{measure-pmf } p) \ s \neq 0$

proof

assume $\text{emeasure } (\text{measure-pmf } p) \ s = 0$
then have $AE \ x \text{ in } \text{measure-pmf } p. \ x \notin s$
by $(\text{rule } AE\text{-I}[\text{rotated}]) \ \text{auto}$
with not-empty show $False$
by $(\text{auto simp: } AE\text{-measure-pmf-iff})$

qed

lemma $\text{measure-measure-pmf-not-zero: } \text{measure } (\text{measure-pmf } p) \ s \neq 0$
using $\text{emeasure-measure-pmf-not-zero}$ **by** $(\text{simp add: } \text{measure-pmf.emeasure-eq-measure-measure-nonneg})$

lift-definition $\text{cond-pmf} :: 'a \ \text{pmf} \ \text{is}$
 $\text{uniform-measure } (\text{measure-pmf } p) \ s$

proof (intro conjI)

show $\text{prob-space } (\text{uniform-measure } (\text{measure-pmf } p) \ s)$
by $(\text{intro prob-space-uniform-measure}) \ (\text{auto simp: } \text{emeasure-measure-pmf-not-zero})$
show $AE \ x \text{ in } \text{uniform-measure } (\text{measure-pmf } p) \ s. \ \text{measure } (\text{uniform-measure } (\text{measure-pmf } p) \ s) \ \{x\} \neq 0$
by $(\text{simp add: } \text{emeasure-measure-pmf-not-zero } \text{measure-measure-pmf-not-zero } AE\text{-uniform-measure } AE\text{-measure-pmf-iff } \text{set-pmf.rep-eq less-top[symmetric]})$

qed simp

lemma $\text{pmf-cond: } \text{pmf } \text{cond-pmf } x = (\text{if } x \in s \text{ then } \text{pmf } p \ x / \text{measure } p \ s \ \text{else } 0)$
by $\text{transfer } (\text{simp add: } \text{emeasure-measure-pmf-not-zero } \text{pmf.rep-eq})$

lemma $\text{set-cond-pmf[simp]: } \text{set-pmf } \text{cond-pmf} = \text{set-pmf } p \ \cap \ s$

by $(\text{auto simp add: } \text{set-pmf-iff } \text{pmf-cond } \text{measure-measure-pmf-not-zero } \text{split: if-split-asm})$

end

lemma $\text{measure-pmf-posI: } x \in \text{set-pmf } p \implies x \in A \implies \text{measure-pmf.prob } p \ A > 0$

using $\text{measure-measure-pmf-not-zero[of } p \ A]$ **by** $(\text{subst zero-less-measure-iff}) \ \text{blast}$

lemma cond-map-pmf:

assumes $\text{set-pmf } p \ \cap \ f \text{ - ' } s \neq \{\}$

shows $\text{cond-pmf } (\text{map-pmf } f \ p) \ s = \text{map-pmf } f \ (\text{cond-pmf } p \ (f \text{ - ' } s))$

proof $-$

have $*$: $\text{set-pmf } (\text{map-pmf } f \ p) \ \cap \ s \neq \{\}$

using assms **by** auto

{ fix x

have $\text{ennreal } (\text{pmf } (\text{map-pmf } f \ (\text{cond-pmf } p \ (f \text{ - ' } s))) \ x) = \text{emeasure } p \ (f \text{ - ' } s \ \cap \ f \text{ - ' } \{x\}) / \text{emeasure } p \ (f \text{ - ' } s)$

unfolding $\text{ennreal-pmf-map cond-pmf.rep-eq[OF assms]}$ **by** $(\text{simp add: } \text{nn-integral-uniform-measure})$

also have $f \text{ - ' } s \ \cap \ f \text{ - ' } \{x\} = (\text{if } x \in s \text{ then } f \text{ - ' } \{x\} \ \text{else } \{\})$

by auto

also have $\text{emeasure } p \text{ (if } x \in s \text{ then } f - \{x\} \text{ else } \{\}) / \text{emeasure } p \text{ (} f - s \text{) =}$
 $\text{ennreal (pmf (cond-pmf (map-pmf } f \text{ } p) \text{ } s) \text{ } x)$
using $\text{measure-measure-pmf-not-zero}[OF *]$
by ($\text{simp add: pmf-cond}[OF *]$ $\text{ennreal-pmf-map measure-pmf.emeasure-eq-measure}$
 $\text{divide-ennreal pmf-nonneg measure-nonneg zero-less-measure-iff}$
 pmf-map)
finally have $\text{ennreal (pmf (cond-pmf (map-pmf } f \text{ } p) \text{ } s) \text{ } x) = \text{ennreal (pmf}$
 $\text{(map-pmf } f \text{ (cond-pmf } p \text{ (} f - s \text{))) } x)$
by simp }
then show $?thesis$
by (intro pmf-eqI) ($\text{simp add: pmf-nonneg}$)
qed

lemma $\text{bind-cond-pmf-cancel}$:

assumes [simp]: $\bigwedge x. x \in \text{set-pmf } p \implies \text{set-pmf } q \cap \{y. R \ x \ y\} \neq \{\}$
assumes [simp]: $\bigwedge y. y \in \text{set-pmf } q \implies \text{set-pmf } p \cap \{x. R \ x \ y\} \neq \{\}$
assumes [simp]: $\bigwedge x \ y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } q \implies R \ x \ y \implies \text{measure}$
 $q \ \{y. R \ x \ y\} = \text{measure } p \ \{x. R \ x \ y\}$
shows $\text{bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\}) = q$
proof (rule pmf-eqI)
fix i
have $\text{ennreal (pmf (bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\})) \ i) =$
 $(\int^{+x. \text{ennreal (pmf } q \ i / \text{measure } p \ \{x. R \ x \ i\}) * \text{ennreal (indicator } \{x. R \ x \ i\}}$
 $x) \ \partial p)$
by ($\text{auto simp add: ennreal-pmf-bind AE-measure-pmf-iff pmf-cond pmf-eq-0-set-pmf}$
 $\text{pmf-nonneg measure-nonneg}$
 $\text{intro!: nn-integral-cong-AE}$)
also have $\dots = (\text{pmf } q \ i * \text{measure } p \ \{x. R \ x \ i\}) / \text{measure } p \ \{x. R \ x \ i\}$
by ($\text{simp add: pmf-nonneg measure-nonneg zero-ennreal-def[symmetric] en-$
 nreal-indicator
 $\text{nn-integral-cmult measure-pmf.emeasure-eq-measure ennreal-mult[symmetric]}$)
also have $\dots = \text{pmf } q \ i$
by ($\text{cases pmf } q \ i = 0$)
 $(\text{simp-all add: pmf-eq-0-set-pmf measure-measure-pmf-not-zero pmf-nonneg})$
finally show $\text{pmf (bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\})) \ i = \text{pmf } q \ i$
by ($\text{simp add: pmf-nonneg}$)
qed

20.5 Relator

inductive $\text{rel-pmf} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \ \text{pmf} \Rightarrow 'b \ \text{pmf} \Rightarrow \text{bool}$

for $R \ p \ q$

where

$\llbracket \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y;$
 $\text{map-pmf fst } pq = p; \text{map-pmf snd } pq = q \rrbracket$
 $\implies \text{rel-pmf } R \ p \ q$

lemma rel-pmfI :

assumes $R: \text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q)$

assumes $eq: \bigwedge x y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } q \implies R x y \implies$
 $\text{measure } p \{x. R x y\} = \text{measure } q \{y. R x y\}$
shows $\text{rel-pmf } R p q$
proof
let $?pq = \text{bind-pmf } p (\lambda x. \text{bind-pmf } (\text{cond-pmf } q \{y. R x y\}) (\lambda y. \text{return-pmf } (x,$
 $y)))$
have $\bigwedge x. x \in \text{set-pmf } p \implies \text{set-pmf } q \cap \{y. R x y\} \neq \{\}$
using R **by** $(\text{auto simp: rel-set-def})$
then show $\bigwedge x y. (x, y) \in \text{set-pmf } ?pq \implies R x y$
by auto
show $\text{map-pmf fst } ?pq = p$
by $(\text{simp add: map-bind-pmf bind-return-pmf'})$

show $\text{map-pmf snd } ?pq = q$
using $R eq$
apply $(\text{simp add: bind-cond-pmf-cancel map-bind-pmf bind-return-pmf'})$
apply $(\text{rule bind-cond-pmf-cancel})$
apply $(\text{auto simp: rel-set-def})$
done
qed

lemma $\text{rel-pmf-imp-rel-set: rel-pmf } R p q \implies \text{rel-set } R (\text{set-pmf } p) (\text{set-pmf } q)$
by $(\text{force simp add: rel-pmf.simps rel-set-def})$

lemma rel-pmfD-measure:

assumes $\text{rel-R: rel-pmf } R p q$ **and** $R: \bigwedge a b. R a b \implies R a y \longleftrightarrow R x b$
assumes $x \in \text{set-pmf } p y \in \text{set-pmf } q$
shows $\text{measure } p \{x. R x y\} = \text{measure } q \{y. R x y\}$
proof –
from rel-R **obtain** pq **where** $pq: \bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$
and $eq: p = \text{map-pmf fst } pq \text{ } q = \text{map-pmf snd } pq$
by $(\text{auto elim: rel-pmf.cases})$
have $\text{measure } p \{x. R x y\} = \text{measure } pq \{x. R (\text{fst } x) y\}$
by $(\text{simp add: eq map-pmf-rep-eq measure-distr})$
also have $\dots = \text{measure } pq \{y. R x (\text{snd } y)\}$
by $(\text{intro measure-pmf.finite-measure-eq-AE})$
 $(\text{auto simp: AE-measure-pmf-iff } R \text{ dest!: } pq)$
also have $\dots = \text{measure } q \{y. R x y\}$
by $(\text{simp add: eq map-pmf-rep-eq measure-distr})$
finally show $\text{measure } p \{x. R x y\} = \text{measure } q \{y. R x y\} .$
qed

lemma rel-pmf-measureD:

assumes $\text{rel-pmf } R p q$
shows $\text{measure } (\text{measure-pmf } p) A \leq \text{measure } (\text{measure-pmf } q) \{y. \exists x \in A. R x y\}$ **(is ?lhs \leq ?rhs)**
using assms
proof cases
fix pq


```

assume  $R: \bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$ 
and  $p[\text{symmetric}]: \text{map-pmf fst } pq = p$ 
and  $q[\text{symmetric}]: \text{map-pmf snd } pq = q$ 
have  $?lhs = \text{measure } (\text{measure-pmf } pq) (\text{fst } - ' A) \text{ by } (\text{simp add: } p)$ 
also have  $\dots \leq \text{measure } (\text{measure-pmf } pq) \{y. \exists x \in A. R x (snd y)\}$ 
by  $(\text{rule } \text{measure-pmf.finite-measure-mono-AE})(\text{auto } 4 \ 3 \ \text{simp add: } \text{AE-measure-pmf-iff}$ 
 $\text{dest: } R)$ 
also have  $\dots = ?rhs \text{ by } (\text{simp add: } q)$ 
finally show  $?thesis .$ 
qed

```

lemma *rel-pmf-iff-measure:*

```

assumes  $\text{symp } R \ \text{transp } R$ 
shows  $\text{rel-pmf } R \ p \ q \longleftrightarrow$ 
 $\text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q) \wedge$ 
 $(\forall x \in \text{set-pmf } p. \forall y \in \text{set-pmf } q. R x y \longrightarrow \text{measure } p \{x. R x y\} = \text{measure } q$ 
 $\{y. R x y\})$ 
by  $(\text{safe intro!}: \text{rel-pmf-imp-rel-set } \text{rel-pmfI})$ 
 $(\text{auto intro!}: \text{rel-pmfD-measure } \text{dest: } \text{sympD}[OF \ \langle \text{symp } R \rangle] \ \text{transpD}[OF \ \langle \text{transp}$ 
 $R \rangle])$ 

```

lemma *quotient-rel-set-disjoint:*

```

 $\text{equivp } R \implies C \in \text{UNIV} // \{(x, y). R x y\} \implies \text{rel-set } R \ A \ B \implies A \cap C = \{\}$ 
 $\longleftrightarrow B \cap C = \{\}$ 
using  $\text{in-quotient-imp-closed}[of \ \text{UNIV} \ \{(x, y). R x y\} \ C]$ 
by  $(\text{auto } 0 \ 0 \ \text{simp: } \text{equivp-equiv } \text{rel-set-def } \text{set-eq-iff } \text{elim: } \text{equivpE})$ 
 $(\text{blast } \text{dest: } \text{equivp-symp})+$ 

```

lemma *quotientD:* $\text{equiv } X \ R \implies A \in X // R \implies x \in A \implies A = R \ \{x\}$

by $(\text{metis } \text{Image-singleton-iff } \text{equiv-class-eq-iff } \text{quotientE})$

lemma *rel-pmf-iff-equivp:*

```

assumes  $\text{equivp } R$ 
shows  $\text{rel-pmf } R \ p \ q \longleftrightarrow (\forall C \in \text{UNIV} // \{(x, y). R x y\}. \text{measure } p \ C = \text{measure}$ 
 $q \ C)$ 
 $(\text{is } - \longleftrightarrow (\forall C \in // ?R. -))$ 

```

proof $(\text{subst } \text{rel-pmf-iff-measure}, \text{ safe})$

show $\text{symp } R \ \text{transp } R$

using assms **by** $(\text{auto } \text{simp: } \text{equivp-reflp-symp-transp})$

next

fix C **assume** $C: C \in \text{UNIV} // ?R$ **and** $R: \text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q)$

assume $\text{eq: } \forall x \in \text{set-pmf } p. \forall y \in \text{set-pmf } q. R x y \longrightarrow \text{measure } p \ \{x. R x y\} =$
 $\text{measure } q \ \{y. R x y\}$

show $\text{measure } p \ C = \text{measure } q \ C$

proof $(\text{cases } p \cap C = \{\})$

case True

then have $q \cap C = \{\}$

using $\text{quotient-rel-set-disjoint}[OF \ \text{assms } C \ R]$ **by** simp

```

with True show ?thesis
  unfolding measure-pmf-zero-iff[symmetric] by simp
next
case False
then have  $q \cap C \neq \{\}$ 
  using quotient-rel-set-disjoint[OF assms C R] by simp
with False obtain  $x y$  where in-set:  $x \in \text{set-pmf } p$   $y \in \text{set-pmf } q$  and in-C:
 $x \in C$   $y \in C$ 
  by auto
then have  $R x y$ 
  using in-quotient-imp-in-rel[of UNIV ?R C x y] C assms
  by (simp add: equivp-equiv)
with in-set eq have  $\text{measure } p \{x. R x y\} = \text{measure } q \{y. R x y\}$ 
  by auto
moreover have  $\{y. R x y\} = C$ 
  using assms  $\langle x \in C \rangle$  C quotientD[of UNIV ?R C x] by (simp add: equivp-equiv)
moreover have  $\{x. R x y\} = C$ 
  using assms  $\langle y \in C \rangle$  C quotientD[of UNIV ?R C y] sympD[of R]
  by (auto simp add: equivp-equiv elim: equivpE)
ultimately show ?thesis
  by auto
qed
next
assume eq:  $\forall C \in \text{UNIV} // ?R. \text{measure } p C = \text{measure } q C$ 
show rel-set R (set-pmf p) (set-pmf q)
  unfolding rel-set-def
proof safe
fix x assume x:  $x \in \text{set-pmf } p$ 
have  $\{y. R x y\} \in \text{UNIV} // ?R$ 
  by (auto simp: quotient-def)
with eq have *:  $\text{measure } q \{y. R x y\} = \text{measure } p \{y. R x y\}$ 
  by auto
have  $\text{measure } q \{y. R x y\} \neq 0$ 
  using x assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
then show  $\exists y \in \text{set-pmf } q. R x y$ 
  unfolding measure-pmf-zero-iff by auto
next
fix y assume y:  $y \in \text{set-pmf } q$ 
have  $\{x. R x y\} \in \text{UNIV} // ?R$ 
  using assms by (auto simp: quotient-def dest: equivp-symp)
with eq have *:  $\text{measure } p \{x. R x y\} = \text{measure } q \{x. R x y\}$ 
  by auto
have  $\text{measure } p \{x. R x y\} \neq 0$ 
  using y assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
then show  $\exists x \in \text{set-pmf } p. R x y$ 
  unfolding measure-pmf-zero-iff by auto
qed

```

```

fix x y assume x ∈ set-pmf p y ∈ set-pmf q R x y
have {y. R x y} ∈ UNIV // ?R {x. R x y} = {y. R x y}
using assms ⟨R x y⟩ by (auto simp: quotient-def dest: equivp-symp equivp-transp)
with eq show measure p {x. R x y} = measure q {y. R x y}
by auto
qed

bnf pmf: 'a pmf map: map-pmf sets: set-pmf bd : card-suc natLeq rel: rel-pmf
proof –
show map-pmf id = id by (rule map-pmf-id)
show  $\bigwedge f g. \text{map-pmf } (f \circ g) = \text{map-pmf } f \circ \text{map-pmf } g$  by (rule map-pmf-compose)
show  $\bigwedge f g :: 'a \Rightarrow 'b. \bigwedge p. (\bigwedge x. x \in \text{set-pmf } p \Longrightarrow f x = g x) \Longrightarrow \text{map-pmf } f p =$ 
 $\text{map-pmf } g p$ 
by (intro map-pmf-cong refl)

show  $\bigwedge f :: 'a \Rightarrow 'b. \text{set-pmf } \circ \text{map-pmf } f = (\cdot) f \circ \text{set-pmf}$ 
by (rule pmf-set-map)

show card-order (card-suc natLeq) using natLeq-card-order by (rule card-order-card-suc)
show BNF-Cardinal-Arithmetic.cinfinite (card-suc natLeq)
using natLeq-Cinfinite natLeq-card-order Cinfinite-card-suc by blast
show regularCard (card-suc natLeq) using natLeq-card-order natLeq-Cinfinite
by (rule regularCard-card-suc)

show (card-of (set-pmf p), card-suc natLeq) ∈ ordLess for p :: 's pmf
proof –
have (card-of (set-pmf p), card-of (UNIV :: nat set)) ∈ ordLeq
by (rule card-of-ordLeqI[where f=to-nat-on (set-pmf p)])
(auto intro: countable-set-pmf)
also have (card-of (UNIV :: nat set), natLeq) ∈ ordLeq
by (metis Field-natLeq card-of-least natLeq-Well-order)
finally show ?thesis using card-suc-greater natLeq-card-order ordLeq-ordLess-trans
by blast
qed

show  $\bigwedge R. \text{rel-pmf } R = (\lambda x y. \exists z. \text{set-pmf } z \subseteq \{(x, y). R x y\} \wedge$ 
 $\text{map-pmf } \text{fst } z = x \wedge \text{map-pmf } \text{snd } z = y)$ 
by (auto simp add: fun-eq-iff rel-pmf.simps)

show rel-pmf R OO rel-pmf S ≤ rel-pmf (R OO S)
for R :: 'a ⇒ 'b ⇒ bool and S :: 'b ⇒ 'c ⇒ bool
proof –
{ fix p q r
assume pq: rel-pmf R p q
and qr: rel-pmf S q r
from pq obtain pq where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \Longrightarrow R x y$ 
and p: p = map-pmf fst pq and q: q = map-pmf snd pq by cases auto
from qr obtain qr where qr:  $\bigwedge y z. (y, z) \in \text{set-pmf } qr \Longrightarrow S y z$ 

```

and q' : $q = \text{map-pmf fst } qr$ **and** $r = \text{map-pmf snd } qr$ **by** *cases auto*

define pr **where** $pr =$
 $\text{bind-pmf } pq (\lambda xy. \text{bind-pmf } (\text{cond-pmf } qr \{yz. \text{fst } yz = \text{snd } xy\})$
 $(\lambda yz. \text{return-pmf } (\text{fst } xy, \text{snd } yz)))$

have $pr\text{-welldefined}$: $\bigwedge y. y \in q \implies qr \cap \{yz. \text{fst } yz = y\} \neq \{\}$
by (*force simp: q'*)

have $rel\text{-pmf } (R \text{ OO } S) p r$
proof (*rule rel-pmf.intros*)
fix $x z$ **assume** $(x, z) \in pr$
then have $\exists y. (x, y) \in pq \wedge (y, z) \in qr$
by (*auto simp: q pr-welldefined pr-def split-beta*)
with $pq qr$ **show** $(R \text{ OO } S) x z$
by *blast*

next
have $\text{map-pmf snd } pr = \text{map-pmf snd } (\text{bind-pmf } q (\lambda y. \text{cond-pmf } qr \{yz. \text{fst } yz = y\}))$
by (*simp add: pr-def q split-beta bind-map-pmf map-pmf-def[symmetric] map-bind-pmf map-pmf-comp*)
then show $\text{map-pmf snd } pr = r$
unfolding $r q'$ **bind-map-pmf** **by** (*subst (asm) bind-cond-pmf-cancel*) (*auto simp: eq-commute*)
qed (*simp add: pr-def map-bind-pmf split-beta map-pmf-def[symmetric] p map-pmf-comp*)

}
then show *?thesis*
by(*auto simp add: le-fun-def*)

qed
qed

lemma map-pmf-idI : $(\bigwedge x. x \in \text{set-pmf } p \implies f x = x) \implies \text{map-pmf } f p = p$
by(*simp cong: pmf.map-cong*)

lemma $rel\text{-pmf-conj}[simp]$:
 $rel\text{-pmf } (\lambda x y. P \wedge Q x y) x y \longleftrightarrow P \wedge rel\text{-pmf } Q x y$
 $rel\text{-pmf } (\lambda x y. Q x y \wedge P) x y \longleftrightarrow P \wedge rel\text{-pmf } Q x y$
using set-pmf-not-empty **by** (*fastforce simp: pmf.in-rel subset-eq*)+

lemma $rel\text{-pmf-top}[simp]$: $rel\text{-pmf top} = top$
by (*auto simp: pmf.in-rel[abs-def] fun-eq-iff map-fst-pair-pmf map-snd-pair-pmf intro: exI[of - pair-pmf x y for x y]*)

lemma $rel\text{-pmf-return-pmf1}$: $rel\text{-pmf } R (\text{return-pmf } x) M \longleftrightarrow (\forall a \in M. R x a)$
proof *safe*
fix a **assume** $a \in M$ $rel\text{-pmf } R (\text{return-pmf } x) M$
then obtain pq **where** $*$: $\bigwedge a b. (a, b) \in \text{set-pmf } pq \implies R a b$
and eq : $\text{return-pmf } x = \text{map-pmf fst } pq$ $M = \text{map-pmf snd } pq$
by (*force elim: rel-pmf.cases*)

```

moreover have set-pmf (return-pmf x) = {x}
  by simp
with ⟨a ∈ M⟩ have (x, a) ∈ pq
  by (force simp: eq)
with * show R x a
  by auto
qed (auto intro!: rel-pmf.intros[where pq=pair-pmf (return-pmf x) M]
  simp: map-fst-pair-pmf map-snd-pair-pmf)

lemma rel-pmf-return-pmf2: rel-pmf R M (return-pmf x) ↔ (∀ a∈M. R a x)
  by (subst pmf.rel-flip[symmetric]) (simp add: rel-pmf-return-pmf1)

lemma rel-return-pmf[simp]: rel-pmf R (return-pmf x1) (return-pmf x2) = R x1
x2
  unfolding rel-pmf-return-pmf2 set-return-pmf by simp

lemma rel-pmf-False[simp]: rel-pmf (λx y. False) x y = False
  unfolding pmf.in-rel fun-eq-iff using set-pmf-not-empty by fastforce

lemma rel-pmf-rel-prod:
  rel-pmf (rel-prod R S) (pair-pmf A A') (pair-pmf B B') ↔ rel-pmf R A B ∧
rel-pmf S A' B'
proof safe
  assume rel-pmf (rel-prod R S) (pair-pmf A A') (pair-pmf B B')
  then obtain pq where pq: ∧ a b c d. ((a, c), (b, d)) ∈ set-pmf pq ⇒ R a b ∧
S c d
  and eq: map-pmf fst pq = pair-pmf A A' map-pmf snd pq = pair-pmf B B'
  by (force elim: rel-pmf.cases)
show rel-pmf R A B
proof (rule rel-pmf.intros)
  let ?f = λ(a, b). (fst a, fst b)
  have [simp]: (λx. fst (?f x)) = fst o fst (λx. snd (?f x)) = fst o snd
  by auto

  show map-pmf fst (map-pmf ?f pq) = A
  by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf)
  show map-pmf snd (map-pmf ?f pq) = B
  by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf)

  fix a b assume (a, b) ∈ set-pmf (map-pmf ?f pq)
  then obtain c d where ((a, c), (b, d)) ∈ set-pmf pq
  by auto
  from pq[OF this] show R a b ..
qed
show rel-pmf S A' B'
proof (rule rel-pmf.intros)
  let ?f = λ(a, b). (snd a, snd b)
  have [simp]: (λx. fst (?f x)) = snd o fst (λx. snd (?f x)) = snd o snd
  by auto

```

```

show map-pmf fst (map-pmf ?f pq) = A'
by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf)
show map-pmf snd (map-pmf ?f pq) = B'
by (simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf)

fix c d assume (c, d) ∈ set-pmf (map-pmf ?f pq)
then obtain a b where ((a, c), (b, d)) ∈ set-pmf pq
  by auto
from pq[OF this] show S c d ..
qed
next
assume rel-pmf R A B rel-pmf S A' B'
then obtain Rpq Spq
  where Rpq:  $\bigwedge a b. (a, b) \in \text{set-pmf } Rpq \implies R a b$ 
    map-pmf fst Rpq = A map-pmf snd Rpq = B
  and Spq:  $\bigwedge a b. (a, b) \in \text{set-pmf } Spq \implies S a b$ 
    map-pmf fst Spq = A' map-pmf snd Spq = B'
  by (force elim: rel-pmf.cases)

let ?f =  $(\lambda((a, c), (b, d)). ((a, b), (c, d)))$ 
let ?pq = map-pmf ?f (pair-pmf Rpq Spq)
have [simp]:  $(\lambda x. \text{fst } (?f x)) = (\lambda(a, b). (\text{fst } a, \text{fst } b))$   $(\lambda x. \text{snd } (?f x)) = (\lambda(a, b). (\text{snd } a, \text{snd } b))$ 
  by auto

show rel-pmf (rel-prod R S) (pair-pmf A A') (pair-pmf B B')
by (rule rel-pmf.intros[where pq=?pq])
  (auto simp: map-snd-pair-pmf map-fst-pair-pmf map-pmf-comp Rpq Spq
    map-pair)
qed

lemma rel-pmf-refl:
assumes  $\bigwedge x. x \in \text{set-pmf } p \implies P x x$ 
shows rel-pmf P p p
by (rule rel-pmf.intros[where pq=map-pmf  $(\lambda x. (x, x))$  p])
  (auto simp add: pmf.map-comp o-def assms)

lemma rel-pmf-bij-betw:
assumes f: bij-betw f (set-pmf p) (set-pmf q)
and eq:  $\bigwedge x. x \in \text{set-pmf } p \implies \text{pmf } p x = \text{pmf } q (f x)$ 
shows rel-pmf  $(\lambda x y. f x = y)$  p q
proof(rule rel-pmf.intros)
let ?pq = map-pmf  $(\lambda x. (x, f x))$  p
show map-pmf fst ?pq = p by(simp add: pmf.map-comp o-def)

have map-pmf f p = q
proof(rule pmf-eqI)
  fix i

```

```

show pmf (map-pmf f p) i = pmf q i
proof(cases i ∈ set-pmf q)
  case True
  with f obtain j where i = f j j ∈ set-pmf p
    by(auto simp add: bij-betw-def image-iff)
  thus ?thesis using f by(simp add: bij-betw-def pmf-map-inj eq)
next
  case False thus ?thesis
    by(subst pmf-map-outside)(auto simp add: set-pmf-iff eq[symmetric])
qed
qed
then show map-pmf snd ?pq = q by(simp add: pmf.map-comp o-def)
qed auto

context
begin

interpretation pmf-as-measure .

definition join-pmf M = bind-pmf M (λx. x)

lemma bind-eq-join-pmf: bind-pmf M f = join-pmf (map-pmf f M)
  unfolding join-pmf-def bind-map-pmf ..

lemma join-eq-bind-pmf: join-pmf M = bind-pmf M id
  by (simp add: join-pmf-def id-def)

lemma pmf-join: pmf (join-pmf N) i = (∫ M. pmf M i ∂measure-pmf N)
  unfolding join-pmf-def pmf-bind ..

lemma ennreal-pmf-join: ennreal (pmf (join-pmf N) i) = (∫+ M. pmf M i ∂measure-pmf N)
  unfolding join-pmf-def ennreal-pmf-bind ..

lemma set-pmf-join-pmf[simp]: set-pmf (join-pmf f) = (∪ p∈set-pmf f. set-pmf p)
  by (simp add: join-pmf-def)

lemma join-return-pmf: join-pmf (return-pmf M) = M
  by (simp add: integral-return pmf-eq-iff pmf-join return-pmf.rep-eq)

lemma map-join-pmf: map-pmf f (join-pmf AA) = join-pmf (map-pmf (map-pmf f) AA)
  by (simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf)

lemma join-map-return-pmf: join-pmf (map-pmf return-pmf A) = A
  by (simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')

end

```

lemma *rel-pmf-joinI*:

assumes *rel-pmf* (*rel-pmf* *P*) *p* *q*

shows *rel-pmf* *P* (*join-pmf* *p*) (*join-pmf* *q*)

proof –

from *assms* **obtain** *pq* **where** *p*: *p* = *map-pmf fst pq*

and *q*: *q* = *map-pmf snd pq*

and *P*: $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{rel-pmf } P x y$

by *cases auto*

from *P* **obtain** *PQ*

where *PQ*: $\bigwedge x y a b. \llbracket (x, y) \in \text{set-pmf } pq; (a, b) \in \text{set-pmf } (PQ x y) \rrbracket \implies P a b$

and *x*: $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf fst } (PQ x y) = x$

and *y*: $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf snd } (PQ x y) = y$

by(*metis rel-pmf.simps*)

let *?r* = *bind-pmf pq* ($\lambda(x, y). PQ x y$)

have $\bigwedge a b. (a, b) \in \text{set-pmf } ?r \implies P a b$ **by** (*auto intro: PQ*)

moreover **have** *map-pmf fst ?r* = *join-pmf p* *map-pmf snd ?r* = *join-pmf q*

by (*simp-all add: p q x y join-pmf-def map-bind-pmf bind-map-pmf split-def cong: bind-pmf-cong*)

ultimately show *?thesis ..*

qed

lemma *rel-pmf-bindI*:

assumes *pq*: *rel-pmf* *R* *p* *q*

and *fg*: $\bigwedge x y. R x y \implies \text{rel-pmf } P (f x) (g y)$

shows *rel-pmf* *P* (*bind-pmf p f*) (*bind-pmf q g*)

unfolding *bind-eq-join-pmf*

by (*rule rel-pmf-joinI*)

(*auto simp add: pmf.rel-map intro: pmf.rel-mono[THEN le-funD, THEN le-funD, THEN le-boolD, THEN mp, OF - pq] fg*)

Proof that *rel-pmf* preserves orders. Antisymmetry proof follows Thm. 1 in N. Saheb-Djahromi, Cpo’s of measures for nondeterminism, Theoretical Computer Science 12(1):19–37, 1980, [https://doi.org/10.1016/0304-3975\(80\)90003-1](https://doi.org/10.1016/0304-3975(80)90003-1)

lemma

assumes ***: *rel-pmf* *R* *p* *q*

and *refl*: *reflp* *R* **and** *trans*: *transp* *R*

shows *measure-Ici*: *measure* *p* {*y*. *R* *x* *y*} \leq *measure* *q* {*y*. *R* *x* *y*} (**is** *?thesis1*)

and *measure-Ioi*: *measure* *p* {*y*. *R* *x* *y* \wedge \neg *R* *y* *x*} \leq *measure* *q* {*y*. *R* *x* *y* \wedge \neg *R* *y* *x*} (**is** *?thesis2*)

proof –

from *** **obtain** *pq*

where *pq*: $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$

and *p*: *p* = *map-pmf fst pq*

and *q*: *q* = *map-pmf snd pq*

by *cases auto*

show *?thesis1 ?thesis2 unfolding p q map-pmf-rep-eq using refl trans*
by(*auto 4 3 simp add: measure-distr reflpD AE-measure-pmf-iff intro!: measure-pmf.finite-measure-mono-AE dest!: pq elim: transpE*)
qed

lemma *rel-pmf-inf:*

fixes *p q :: 'a pmf*
assumes *1: rel-pmf R p q*
assumes *2: rel-pmf R q p*
and *refl: reflp R and trans: transp R*
shows *rel-pmf (inf R R⁻¹⁻¹) p q*
proof (*subst rel-pmf-iff-equivp, safe*)
show *equivp (inf R R⁻¹⁻¹)*
using *trans refl by (auto simp: equivp-reflp-symp-transp intro: sympI transpI reflpI dest: transpD reflpD)*

fix *C assume C ∈ UNIV // {(x, y). inf R R⁻¹⁻¹ x y}*
then obtain x where C: C = {y. R x y ∧ R y x}
by (*auto elim: quotientE*)

let *?R = λx y. R x y ∧ R y x*
let *?μR = λy. measure q {x. ?R x y}*
have *measure p {y. ?R x y} = measure p ({y. R x y} - {y. R x y ∧ ¬ R y x})*
by(*auto intro!: arg-cong[where f=measure p]*)
also have *... = measure p {y. R x y} - measure p {y. R x y ∧ ¬ R y x}*
by (*rule measure-pmf.finite-measure-Diff*) *auto*
also have *measure p {y. R x y ∧ ¬ R y x} = measure q {y. R x y ∧ ¬ R y x}*
using *1 2 refl trans by (auto intro!: Orderings.antisym measure-Ioi)*
also have *measure p {y. R x y} = measure q {y. R x y}*
using *1 2 refl trans by (auto intro!: Orderings.antisym measure-Ici)*
also have *measure q {y. R x y} - measure q {y. R x y ∧ ¬ R y x} =*
measure q ({y. R x y} - {y. R x y ∧ ¬ R y x})
by(*rule measure-pmf.finite-measure-Diff[symmetric]*) *auto*
also have *... = ?μR x*
by(*auto intro!: arg-cong[where f=measure q]*)
finally show *measure p C = measure q C*
by (*simp add: C conj-commute*)
qed

lemma *rel-pmf-antisym:*

fixes *p q :: 'a pmf*
assumes *1: rel-pmf R p q*
assumes *2: rel-pmf R q p*
and *refl: reflp R and trans: transp R and antisym: antisymp R*
shows *p = q*
proof –
from *1 2 refl trans have rel-pmf (inf R R⁻¹⁻¹) p q by (rule rel-pmf-inf)*
also have *inf R R⁻¹⁻¹ = (=)*
using *refl antisym by (auto intro!: ext simp add: reflpD dest: antisympD)*

finally show *?thesis unfolding pmf.rel-eq* .
qed

lemma *reflp-rel-pmf*: $\text{reflp } R \implies \text{reflp } (\text{rel-pmf } R)$
by (*fact pmf.rel-reflp*)

lemma *antisymp-rel-pmf*:
 $[\text{reflp } R; \text{transp } R; \text{antisymp } R]$
 $\implies \text{antisymp } (\text{rel-pmf } R)$
by(*rule antisympI*)(*blast intro: rel-pmf-antisym*)

lemma *transp-rel-pmf*:
assumes *transp R*
shows *transp (rel-pmf R)*
using *assms by (fact pmf.rel-transp)*

20.6 Distributions

context
begin

interpretation *pmf-as-function* .

20.6.1 Bernoulli Distribution

lift-definition *bernoulli-pmf* :: $\text{real} \implies \text{bool pmf}$ **is**
 $\lambda p b. ((\lambda p. \text{if } b \text{ then } p \text{ else } 1 - p) \circ \text{min } 1 \circ \text{max } 0) p$
by (*auto simp: nn-integral-count-space-finite*[**where** $A = \{\text{False}, \text{True}\}$] *UNIV-bool*
split: split-max split-min)

lemma *pmf-bernoulli-True*[*simp*]: $0 \leq p \implies p \leq 1 \implies \text{pmf } (\text{bernoulli-pmf } p)$
 $\text{True} = p$
by *transfer simp*

lemma *pmf-bernoulli-False*[*simp*]: $0 \leq p \implies p \leq 1 \implies \text{pmf } (\text{bernoulli-pmf } p)$
 $\text{False} = 1 - p$
by *transfer simp*

lemma *set-pmf-bernoulli*[*simp*]: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{bernoulli-pmf } p)$
 $= \text{UNIV}$
by (*auto simp add: set-pmf-iff UNIV-bool*)

lemma *nn-integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: $0 \leq p \leq 1 \wedge x. 0 \leq f x$
shows $(\int^+ x. f x \partial \text{bernoulli-pmf } p) = f \text{True} * p + f \text{False} * (1 - p)$
by (*subst nn-integral-measure-pmf-support*[*of UNIV*])
(auto simp: UNIV-bool field-simps)

lemma *integral-bernoulli-pmf*[*simp*]:
assumes [*simp*]: $0 \leq p \leq 1$

shows $(\int x. f x \partial \text{bernoulli-pmf } p) = f \text{ True} * p + f \text{ False} * (1 - p)$
by $(\text{subst integral-measure-pmf}[of UNIV]) (\text{auto simp: UNIV-bool})$

lemma pmf-bernoulli-half $[simp]: \text{pmf } (\text{bernoulli-pmf } (1 / 2)) x = 1 / 2$
by $(\text{cases } x) \text{ simp-all}$

lemma measure-pmf-bernoulli-half: $\text{measure-pmf } (\text{bernoulli-pmf } (1 / 2)) = \text{uniform-count-measure UNIV}$

by $(\text{rule measure-eqI})$
 $(\text{simp-all add: nn-integral-pmf}[symmetric] \text{ emeasure-uniform-count-measure}$
 $\text{ ennreal-divide-numeral}[symmetric]$
 $\text{ nn-integral-count-space-finite sets-uniform-count-measure di-}$
 $\text{vide-ennreal-def mult-ac}$
 $\text{ ennreal-of-nat-eq-real-of-nat})$

20.6.2 Geometric Distribution

context

fixes $p :: \text{real}$ **assumes** $p[\text{arith}]: 0 < p \leq 1$
begin

lift-definition geometric-pmf $:: \text{nat pmf is } \lambda n. (1 - p)^{\wedge n} * p$

proof

have $(\sum i. \text{ennreal } (p * (1 - p)^{\wedge i})) = \text{ennreal } (p * (1 / (1 - (1 - p))))$
by $(\text{intro suminf-ennreal-eq sums-mult geometric-sums}) \text{ auto}$
then show $(\int^+ x. \text{ennreal } ((1 - p)^{\wedge x} * p) \partial \text{count-space UNIV}) = 1$
by $(\text{simp add: nn-integral-count-space-nat field-simps})$

qed simp

lemma pmf-geometric $[simp]: \text{pmf } \text{geometric-pmf } n = (1 - p)^{\wedge n} * p$
by transfer rule

end

lemma geometric-pmf-1 $[simp]: \text{geometric-pmf } 1 = \text{return-pmf } 0$
by $(\text{intro pmf-eqI}) (\text{auto simp: indicator-def})$

lemma set-pmf-geometric: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{geometric-pmf } p) = \text{UNIV}$

by $(\text{auto simp: set-pmf-iff})$

lemma geometric-sums-times-n:

fixes $c :: 'a :: \{\text{banach, real-normed-field}\}$

assumes $\text{norm } c < 1$

shows $(\lambda n. c^{\wedge n} * \text{of-nat } n) \text{ sums } (c / (1 - c)^2)$

proof –

have $(\lambda n. c * z^{\wedge n}) \text{ sums } (c / (1 - z))$ **if** $\text{norm } z < 1$ **for** z

using $\text{geometric-sums sums-mult}$ **that** **by** fastforce

moreover have $((\lambda z. c / (1 - z)) \text{ has-field-derivative } (c / (1 - c)^2)) (\text{at } c)$

using *assms* **by** (*auto intro!*: *derivative-eq-intros simp add: semiring-normalization-rules*)
ultimately have $(\lambda n. \text{diffs } (\lambda n. c) n * c^{\wedge} n) \text{ sums } (c / (1 - c)^2)$
using *assms* **by** (*intro termdiffs-sums-strong*)
then have $(\lambda n. \text{of-nat } (\text{Suc } n) * c^{\wedge} (\text{Suc } n)) \text{ sums } (c / (1 - c)^2)$
unfolding *diffs-def* **by** (*simp add: power-eq-if mult.assoc*)
then show *?thesis*
by (*subst (asm) sums-Suc-iff*) (*auto simp add: mult.commute*)
qed

lemma *geometric-sums-times-norm*:
fixes *c::'a::{banach,real-normed-field}*
assumes *norm c < 1*
shows $(\lambda n. \text{norm } (c^{\wedge} n * \text{of-nat } n)) \text{ sums } (\text{norm } c / (1 - \text{norm } c)^2)$
proof –
have $\text{norm } (c^{\wedge} n * \text{of-nat } n) = (\text{norm } c)^{\wedge} n * \text{of-nat } n$ **for** *n::nat*
by (*simp add: norm-power norm-mult*)
then show *?thesis*
using *geometric-sums-times-n[of norm c] assms*
by force
qed

lemma *integrable-real-geometric-pmf*:
assumes *p ∈ {0<..1}*
shows *integrable (geometric-pmf p) real*
proof –
have *summable* $(\lambda x. p * ((1 - p)^{\wedge} x * \text{real } x))$
using *geometric-sums-times-norm[of 1 - p] assms*
by (*intro summable-mult*) (*auto simp: sums-iff*)
hence *summable* $(\lambda x. (1 - p)^{\wedge} x * \text{real } x)$
by (*rule summable-mult-D*) (*use assms in auto*)
thus *?thesis*
unfolding *measure-pmf-eq-density using assms*
by (*subst integrable-density*)
(auto simp: integrable-count-space-nat-iff mult-ac)
qed

lemma *expectation-geometric-pmf*:
assumes *p ∈ {0<..1}*
shows *measure-pmf.expectation (geometric-pmf p) real = (1 - p) / p*
proof –
have $(\lambda n. p * ((1 - p)^{\wedge} n * n)) \text{ sums } (p * ((1 - p) / p^{\wedge} 2))$
using *assms geometric-sums-times-n[of 1-p] by (intro sums-mult) auto*
moreover have $(\lambda n. p * ((1 - p)^{\wedge} n * n)) = (\lambda n. (1 - p)^{\wedge} n * p * \text{real } n)$
by auto
ultimately have $*$: $(\lambda n. (1 - p)^{\wedge} n * p * \text{real } n) \text{ sums } ((1 - p) / p)$
using *assms sums-subst by (auto simp add: power2-eq-square)*
have *measure-pmf.expectation (geometric-pmf p) real =*
*(∫ n. pmf (geometric-pmf p) n * real n ∂count-space UNIV)*
unfolding *measure-pmf-eq-density by (subst integral-density) auto*

also have *integrable (count-space UNIV) (λn. pmf (geometric-pmf p) n * real n)*
using * *assms unfolding integrable-count-space-nat-iff by (simp add: sums-iff)*
hence $(\int n. \text{pmf } (\text{geometric-pmf } p) n * \text{real } n \ \partial \text{count-space UNIV}) = (1 - p) / p$
using * *assms by (subst integral-count-space-nat) (simp-all add: sums-iff)*
finally show ?thesis **by auto**
qed

lemma *geometric-bind-pmf-unfold:*

assumes $p \in \{0 < .. 1\}$

shows *geometric-pmf p =*

do {b ← bernoulli-pmf p;

if b then return-pmf 0 else map-pmf Suc (geometric-pmf p)}

proof –

have *: $(\text{Suc } - ' \{i\}) = (\text{if } i = 0 \text{ then } \{ \} \text{ else } \{i - 1\})$ **for** i

by force

have $\text{pmf } (\text{geometric-pmf } p) i =$

$\text{pmf } (\text{bernoulli-pmf } p) \gg$

$(\lambda b. \text{if } b \text{ then return-pmf } 0 \text{ else map-pmf Suc } (\text{geometric-pmf } p))$

i **for** i

proof –

have $\text{pmf } (\text{geometric-pmf } p) i =$

$(\text{if } i = 0 \text{ then } p \text{ else } (1 - p) * \text{pmf } (\text{geometric-pmf } p) (i - 1))$

using *assms by (simp add: power-eq-if)*

also have ... = $(\text{if } i = 0 \text{ then } p \text{ else } (1 - p) * \text{pmf } (\text{map-pmf Suc } (\text{geometric-pmf } p)) i)$

by *(simp add: pmf-map indicator-def measure-pmf-single *)*

also have ... = *measure-pmf.expectation (bernoulli-pmf p)*

$(\lambda x. \text{pmf } (\text{if } x \text{ then return-pmf } 0 \text{ else map-pmf Suc } (\text{geometric-pmf } p)) i)$

using *assms by (auto simp add: pmf-map *)*

also have ... = $\text{pmf } (\text{bernoulli-pmf } p) \gg$

$(\lambda b. \text{if } b \text{ then return-pmf } 0 \text{ else map-pmf Suc } (\text{geometric-pmf } p))$

i

by *(auto simp add: pmf-bind)*

finally show ?thesis .

qed

then show ?thesis

using *pmf-eqI by blast*

qed

20.6.3 Uniform Multiset Distribution

context

fixes $M :: 'a \text{ multiset}$ **assumes** *M-not-empty: M ≠ {#}*

begin

lift-definition *pmf-of-multiset :: 'a pmf is λx. count M x / size M*

proof

```

show ( $\int^+ x. \text{ennreal} (\text{real} (\text{count } M \ x) / \text{real} (\text{size } M)) \ \partial \text{count-space } UNIV) =
1
using M-not-empty
by (simp add: zero-less-divide-iff nn-integral-count-space nonempty-has-size
sum-divide-distrib[symmetric])
(auto simp: size-multiset-overloaded-eq intro!: sum.cong)
qed simp$ 
```

```

lemma pmf-of-multiset[simp]: pmf pmf-of-multiset  $x = \text{count } M \ x / \text{size } M$ 
by transfer rule

```

```

lemma set-pmf-of-multiset[simp]: set-pmf pmf-of-multiset = set-mset  $M$ 
by (auto simp: set-pmf-iff)

```

end

20.6.4 Uniform Distribution

context

```

fixes  $S :: 'a \text{ set}$  assumes S-not-empty:  $S \neq \{\}$  and S-finite: finite  $S$ 
begin

```

```

lift-definition pmf-of-set :: 'a pmf is  $\lambda x. \text{indicator } S \ x / \text{card } S$ 

```

proof

```

show ( $\int^+ x. \text{ennreal} (\text{indicator } S \ x / \text{real} (\text{card } S)) \ \partial \text{count-space } UNIV) = 1
using S-not-empty S-finite
by (subst nn-integral-count-space[of  $S$ ])
(auto simp: ennreal-of-nat-eq-real-of-nat ennreal-mult[symmetric])
qed simp$ 
```

```

lemma pmf-of-set[simp]: pmf pmf-of-set  $x = \text{indicator } S \ x / \text{card } S$ 
by transfer rule

```

```

lemma set-pmf-of-set[simp]: set-pmf pmf-of-set =  $S$ 
using S-finite S-not-empty by (auto simp: set-pmf-iff)

```

```

lemma emeasure-pmf-of-set-space[simp]: emeasure pmf-of-set  $S = 1$ 
by (rule measure-pmf.emeasure-eq-1-AE) (auto simp: AE-measure-pmf-iff)

```

```

lemma nn-integral-pmf-of-set: nn-integral (measure-pmf pmf-of-set)  $f = \text{sum } f \ S$ 
/ card  $S$ 
by (subst nn-integral-measure-pmf-finite)
(simp-all add: sum-distrib-right[symmetric] card-gt-0-iff S-not-empty S-finite
divide-ennreal-def
divide-ennreal[symmetric] ennreal-of-nat-eq-real-of-nat[symmetric]
ennreal-times-divide)

```

```

lemma integral-pmf-of-set: integralL (measure-pmf pmf-of-set)  $f = \text{sum } f \ S / \text{card } S$ 

```

by (*subst integral-measure-pmf*[of S]) (*auto simp: S-finite sum-divide-distrib*)

lemma *emeasure-pmf-of-set*: *emeasure* (*measure-pmf pmf-of-set*) $A = \text{card } (S \cap A) / \text{card } S$

by (*subst nn-integral-indicator*[*symmetric*], *simp*)

(*simp add: S-finite S-not-empty card-gt-0-iff indicator-def sum.If-cases divide-ennreal*

ennreal-of-nat-eq-real-of-nat nn-integral-pmf-of-set)

lemma *measure-pmf-of-set*: *measure* (*measure-pmf pmf-of-set*) $A = \text{card } (S \cap A) / \text{card } S$

using *emeasure-pmf-of-set*[of A]

by (*simp add: measure-nonneg measure-pmf.emeasure-eq-measure*)

end

lemma *pmf-expectation-bind-pmf-of-set*:

fixes $A :: 'a \text{ set}$ **and** $f :: 'a \Rightarrow 'b \text{ pmf}$

and $h :: 'b \Rightarrow 'c :: \{\text{banach, second-countable-topology}\}$

assumes $A \neq \{\}$ *finite* $A \wedge x. x \in A \implies \text{finite } (\text{set-pmf } (f x))$

shows *measure-pmf.expectation* (*pmf-of-set* $A \gg f$) $h =$

$(\sum a \in A. \text{measure-pmf.expectation } (f a) h /_{\mathbb{R}} \text{real } (\text{card } A))$

using *assms* **by** (*subst pmf-expectation-bind*[of A]) (*auto simp: field-split-simps*)

lemma *map-pmf-of-set*:

assumes *finite* A $A \neq \{\}$

shows *map-pmf* f (*pmf-of-set* A) = *pmf-of-multiset* (*image-mset* f (*mset-set* A))

(*is ?lhs = ?rhs*)

proof (*intro pmf-eqI*)

fix x

from *assms* **have** *ennreal* (*pmf* $?lhs$ x) = *ennreal* (*pmf* $?rhs$ x)

by (*subst ennreal-pmf-map*)

(*simp-all add: emeasure-pmf-of-set mset-set-empty-iff count-image-mset*

Int-commute)

thus *pmf* $?lhs$ $x = \text{pmf } ?rhs$ x **by** *simp*

qed

lemma *pmf-bind-pmf-of-set*:

assumes $A \neq \{\}$ *finite* A

shows *pmf* (*bind-pmf* (*pmf-of-set* A) f) $x =$

$(\sum xa \in A. \text{pmf } (f xa) x) / \text{real-of-nat } (\text{card } A)$ (*is ?lhs = ?rhs*)

proof –

from *assms* **have** *card* $A > 0$ **by** *auto*

with *assms* **have** *ennreal* $?lhs = \text{ennreal } ?rhs$

by (*subst ennreal-pmf-bind*)

(*simp-all add: nn-integral-pmf-of-set max-def pmf-nonneg divide-ennreal*

[*symmetric*]

sum-nonneg ennreal-of-nat-eq-real-of-nat)

thus *?thesis* **by** (*subst (asm) ennreal-inj*) (*auto intro!: sum-nonneg divide-nonneg-nonneg*)
qed

lemma *pmf-of-set-singleton*: $\text{pmf-of-set } \{x\} = \text{return-pmf } x$
by(*rule pmf-eqI*)(*simp add: indicator-def*)

lemma *map-pmf-of-set-inj*:
assumes *f: inj-on f A*
and [*simp*]: $A \neq \{\}$ *finite A*
shows $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } (f \text{ ' } A)$ (**is** *?lhs = ?rhs*)
proof(*rule pmf-eqI*)
fix *i*
show $\text{pmf } ?lhs \ i = \text{pmf } ?rhs \ i$
proof(*cases i ∈ f ' A*)
case *True*
then obtain *i'* **where** $i = f \ i' \ i' \in A$ **by** *auto*
thus *?thesis* **using** *f* **by**(*simp add: card-image pmf-map-inj*)
next
case *False*
hence $\text{pmf } ?lhs \ i = 0$ **by**(*simp add: pmf-eq-0-set-pmf set-map-pmf*)
moreover **have** $\text{pmf } ?rhs \ i = 0$ **using** *False* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
qed

lemma *map-pmf-of-set-bij-betw*:
assumes *bij-betw f A B A ≠ {}* *finite A*
shows $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } B$
proof –
have $\text{map-pmf } f (\text{pmf-of-set } A) = \text{pmf-of-set } (f \text{ ' } A)$
by (*intro map-pmf-of-set-inj assms bij-betw-imp-inj-on[OF assms(1)]*)
also from *assms* **have** $f \text{ ' } A = B$ **by** (*simp add: bij-betw-def*)
finally show *?thesis* .
qed

Choosing an element uniformly at random from the union of a disjoint family of finite non-empty sets with the same size is the same as first choosing a set from the family uniformly at random and then choosing an element from the chosen set uniformly at random.

lemma *pmf-of-set-UN*:
assumes *finite* $(\bigcup (f \text{ ' } A))$ $A \neq \{\}$ $\bigwedge x. x \in A \implies f \ x \neq \{\}$
 $\bigwedge x. x \in A \implies \text{card } (f \ x) = n$ *disjoint-family-on f A*
shows $\text{pmf-of-set } (\bigcup (f \text{ ' } A)) = \text{do } \{x \leftarrow \text{pmf-of-set } A; \text{pmf-of-set } (f \ x)\}$
(is *?lhs = ?rhs*)
proof (*intro pmf-eqI*)
fix *x*
from *assms* **have** [*simp*]: *finite A*
using *infinite-disjoint-family-imp-infinite-UNION[of A f]* **by** *blast*
from *assms* **have** *ereal* $(\text{pmf } (\text{pmf-of-set } (\bigcup (f \text{ ' } A))) \ x) =$


```

    ereal (indicator (⋃ x∈A. f x) x / real (card (⋃ x∈A. f x)))
  by (subst pmf-of-set) auto
also from assms have card (⋃ x∈A. f x) = card A * n
  by (subst card-UN-disjoint) (auto simp: disjoint-family-on-def)
also from assms
  have indicator (⋃ x∈A. f x) x / real ... =
    indicator (⋃ x∈A. f x) x / (n * real (card A))
  by (simp add: sum-divide-distrib [symmetric] mult-ac)
also from assms have indicator (⋃ x∈A. f x) x = (∑ y∈A. indicator (f y) x)
  by (intro indicator-UN-disjoint) simp-all
also from assms have ereal ((∑ y∈A. indicator (f y) x) / (real n * real (card
A))) =
    ereal (pmf ?rhs x)
  by (subst pmf-bind-pmf-of-set) (simp-all add: sum-divide-distrib)
finally show pmf ?lhs x = pmf ?rhs x by simp
qed

```

```

lemma bernoulli-pmf-half-conv-pmf-of-set: bernoulli-pmf (1 / 2) = pmf-of-set UNIV
  by (rule pmf-eqI) simp-all

```

20.6.5 Poisson Distribution

context

fixes *rate* :: real **assumes** *rate-pos*: 0 < *rate*

begin

lift-definition *poisson-pmf* :: nat pmf **is** $\lambda k. \text{rate}^k / \text{fact } k * \exp(-\text{rate})$

proof

```

  have summable: summable ( $\lambda x::\text{nat}. \text{rate}^x / \text{fact } x$ ) using summable-exp
  by (simp add: field-simps divide-inverse [symmetric])
  have ( $\int^+(x::\text{nat}). \text{rate}^x / \text{fact } x * \exp(-\text{rate}) \partial \text{count-space UNIV}$ ) =
     $\exp(-\text{rate}) * (\int^+(x::\text{nat}). \text{rate}^x / \text{fact } x \partial \text{count-space UNIV})$ 
  by (simp add: field-simps nn-integral-cmult[symmetric] ennreal-mult'[symmetric])
  also from rate-pos have ( $\int^+(x::\text{nat}). \text{rate}^x / \text{fact } x \partial \text{count-space UNIV}$ ) =
    ( $\sum x. \text{rate}^x / \text{fact } x$ )
  by (simp-all add: nn-integral-count-space-nat suminf-ennreal summable en-
nreal-suminf-neq-top)
  also have ... =  $\exp \text{rate}$  unfolding exp-def
  by (simp add: field-simps divide-inverse [symmetric])
  also have ennreal ( $\exp(-\text{rate})$ ) * ennreal ( $\exp \text{rate}$ ) = 1
  by (simp add: mult-exp-exp ennreal-mult[symmetric])
  finally show ( $\int^+ x. \text{ennreal}(\text{rate}^x / (\text{fact } x) * \exp(-\text{rate})) \partial \text{count-space UNIV}$ ) = 1 .
qed (simp add: rate-pos[THEN less-imp-le])

```

```

lemma pmf-poisson[simp]: pmf poisson-pmf k =  $\text{rate}^k / \text{fact } k * \exp(-\text{rate})$ 
  by transfer rule

```

```

lemma set-pmf-poisson[simp]: set-pmf poisson-pmf = UNIV

```

using *rate-pos* by (*auto simp: set-pmf-iff*)

end

20.6.6 Binomial Distribution

context

fixes $n :: \text{nat}$ and $p :: \text{real}$ assumes *p-nonneg*: $0 \leq p$ and *p-le-1*: $p \leq 1$

begin

lift-definition *binomial-pmf* :: nat pmf is $\lambda k. (n \text{ choose } k) * p^k * (1 - p)^{(n - k)}$

proof

have $(\int^{+k}. \text{ennreal } (\text{real } (n \text{ choose } k) * p^k * (1 - p)^{(n - k)}) \partial \text{count-space UNIV}) =$

$\text{ennreal } (\sum_{k \leq n}. \text{real } (n \text{ choose } k) * p^k * (1 - p)^{(n - k)})$

using *p-le-1 p-nonneg* by (*subst nn-integral-count-space'*) *auto*

also have $(\sum_{k \leq n}. \text{real } (n \text{ choose } k) * p^k * (1 - p)^{(n - k)}) = (p + (1 - p))^n$

by (*subst binomial-ring*) (*simp add: atLeast0AtMost*)

finally show $(\int^{+x}. \text{ennreal } (\text{real } (n \text{ choose } x) * p^x * (1 - p)^{(n - x)}) \partial \text{count-space UNIV}) = 1$

by *simp*

qed (*insert p-nonneg p-le-1, simp*)

lemma *pmf-binomial[simp]*: $\text{pmf } \text{binomial-pmf } k = (n \text{ choose } k) * p^k * (1 - p)^{(n - k)}$

by *transfer rule*

lemma *set-pmf-binomial-eq*: $\text{set-pmf } \text{binomial-pmf} = (\text{if } p = 0 \text{ then } \{0\} \text{ else if } p = 1 \text{ then } \{n\} \text{ else } \{..n\})$

using *p-nonneg p-le-1 unfolding set-eq-iff set-pmf-iff pmf-binomial* by (*auto simp: set-pmf-iff*)

end

end

lemma *set-pmf-binomial-0[simp]*: $\text{set-pmf } (\text{binomial-pmf } n \ 0) = \{0\}$

by (*simp add: set-pmf-binomial-eq*)

lemma *set-pmf-binomial-1[simp]*: $\text{set-pmf } (\text{binomial-pmf } n \ 1) = \{n\}$

by (*simp add: set-pmf-binomial-eq*)

lemma *set-pmf-binomial[simp]*: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{binomial-pmf } n \ p) = \{..n\}$

by (*simp add: set-pmf-binomial-eq*)

lemma *finite-set-pmf-binomial-pmf [intro]*: $p \in \{0..1\} \implies \text{finite } (\text{set-pmf } (\text{binomial-pmf}$

$n\ p)$
by (*subst set-pmf-binomial-eq*) *auto*

lemma *expectation-binomial-pmf'*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes $p: p \in \{0..1\}$

shows $\text{measure-pmf.expectation } (\text{binomial-pmf } n\ p)\ f =$
 $(\sum k \leq n. (\text{real } (n\ \text{choose } k) * p^k * (1 - p)^{(n - k)}) *_{\mathbb{R}} f\ k)$

using p **by** (*subst integral-measure-pmf[where A = {..n}]*)
(auto simp: set-pmf-binomial-eq split: if-splits)

lemma *integrable-binomial-pmf [simp, intro]*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$

assumes $p: p \in \{0..1\}$

shows *integrable* (*binomial-pmf* $n\ p$) f

by (*rule integrable-measure-pmf-finite*) (*use assms in auto*)

context includes *lifting-syntax*

begin

lemma *bind-pmf-parametric [transfer-rule]*:

$(\text{rel-pmf } A \text{ ===> } (A \text{ ===> } \text{rel-pmf } B) \text{ ===> } \text{rel-pmf } B)$ *bind-pmf* *bind-pmf*
by(*blast intro: rel-pmf-bindI dest: rel-funD*)

lemma *return-pmf-parametric [transfer-rule]*: $(A \text{ ===> } \text{rel-pmf } A)$ *return-pmf*
return-pmf

by(*rule rel-funI*) *simp*

end

primrec *replicate-pmf* :: $\text{nat} \Rightarrow 'a\ \text{pmf} \Rightarrow 'a\ \text{list}\ \text{pmf}$ **where**

replicate-pmf 0 = *return-pmf* []

| *replicate-pmf* (Suc n) $p = \text{do } \{x \leftarrow p; xs \leftarrow \text{replicate-pmf } n\ p; \text{return-pmf } (x\#\!xs)\}$

lemma *replicate-pmf-1*: *replicate-pmf* 1 $p = \text{map-pmf } (\lambda x. [x])\ p$

by (*simp add: map-pmf-def bind-return-pmf*)

lemma *set-replicate-pmf*:

set-pmf (*replicate-pmf* $n\ p$) = $\{xs \in \text{lists } (\text{set-pmf } p). \text{length } xs = n\}$

by (*induction n*) (*auto simp: length-Suc-conv*)

lemma *replicate-pmf-distrib*:

replicate-pmf ($m + n$) $p =$

$\text{do } \{xs \leftarrow \text{replicate-pmf } m\ p; ys \leftarrow \text{replicate-pmf } n\ p; \text{return-pmf } (xs\ @\ ys)\}$

by (*induction m*) (*simp-all add: bind-return-pmf bind-return-pmf' bind-assoc-pmf*)

lemma *power-diff'*:

assumes $b \leq a$

```

shows  $x \wedge (a - b) = (\text{if } x = 0 \wedge a = b \text{ then } 1 \text{ else } x \wedge a / (x::'a::\text{field}) \wedge b)$ 
proof (cases  $x = 0$ )
  case True
    with assms show ?thesis by (cases  $a - b$ ) simp-all
qed (insert assms, simp-all add: power-diff)

```

lemma *binomial-pmf-Suc*:

```

assumes  $p \in \{0..1\}$ 
shows  $\text{binomial-pmf } (\text{Suc } n) p =$ 
   $\text{do } \{b \leftarrow \text{bernoulli-pmf } p;$ 
     $k \leftarrow \text{binomial-pmf } n p;$ 
     $\text{return-pmf } ((\text{if } b \text{ then } 1 \text{ else } 0) + k)\}$  (is  $- = ?rhs$ )
proof (intro pmf-eqI)
  fix  $k$ 
  have  $A: \text{indicator } \{\text{Suc } a\} (\text{Suc } b) = \text{indicator } \{a\} b$  for  $a b$ 
    by (simp add: indicator-def)
  show  $\text{pmf } (\text{binomial-pmf } (\text{Suc } n) p) k = \text{pmf } ?rhs k$ 
    by (cases  $k$ ; cases  $k > n$ )
    (insert assms, auto simp: pmf-bind measure-pmf-single A field-split-simps
algebra-simps
  not-less less-eq-Suc-le [symmetric] power-diff')
qed

```

lemma *binomial-pmf-0*: $p \in \{0..1\} \implies \text{binomial-pmf } 0 p = \text{return-pmf } 0$
by (*rule* *pmf-eqI*) (*simp-all* add: *indicator-def*)

lemma *binomial-pmf-altdef*:

```

assumes  $p \in \{0..1\}$ 
shows  $\text{binomial-pmf } n p = \text{map-pmf } (\text{length} \circ \text{filter } \text{id}) (\text{replicate-pmf } n$ 
   $(\text{bernoulli-pmf } p))$ 
by (induction  $n$ )
  (insert assms, auto simp: binomial-pmf-Suc map-pmf-def bind-return-pmf
bind-assoc-pmf
  bind-return-pmf' binomial-pmf-0 intro!: bind-pmf-cong)

```

20.7 Negative Binomial distribution

The negative binomial distribution counts the number of times a weighted coin comes up tails before having come up heads n times. In other words: how many failures do we see before seeing the n -th success?

An alternative view is that the negative binomial distribution is the sum of n i.i.d. geometric variables (this is the definition that we use).

Note that there are sometimes different conventions for this distributions in the literature; for instance, sometimes the number of *attempts* is counted instead of the number of failures. This only shifts the entire distribution by a constant number and is thus not a big difference. I think that the convention we use is the most natural one since the support of the distribution starts

at 0, whereas for the other convention it starts at n .

primrec *neg-binomial-pmf* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{nat pmf}$ **where**

```

  neg-binomial-pmf 0  $p$  = return-pmf 0
| neg-binomial-pmf (Suc  $n$ )  $p$  =
  map-pmf ( $\lambda(x,y). (x + y)$ ) (pair-pmf (geometric-pmf  $p$ ) (neg-binomial-pmf  $n$ 
   $p$ ))

```

lemma *neg-binomial-pmf-Suc-0* [*simp*]: *neg-binomial-pmf* (*Suc* 0) p = *geometric-pmf* p

by (*auto simp: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf bind-return-pmf'*)

lemmas *neg-binomial-pmf-Suc* [*simp del*] = *neg-binomial-pmf.simps*(2)

lemma *neg-binomial-prob-1* [*simp*]: *neg-binomial-pmf* n 1 = *return-pmf* 0

by (*induction* n) (*simp-all add: neg-binomial-pmf-Suc*)

We can now show the aforementioned intuition about counting the failures before the n -th success with the following recurrence:

lemma *neg-binomial-pmf-unfold*:

assumes $p: p \in \{0 < .. 1\}$

shows *neg-binomial-pmf* (*Suc* n) p =

do { $b \leftarrow$ *bernoulli-pmf* p ;

if b *then* *neg-binomial-pmf* n p *else* *map-pmf* *Suc* (*neg-binomial-pmf*

(*Suc* n) p)}

(**is** - = ?*rhs*)

unfolding *neg-binomial-pmf-Suc*

by (*subst geometric-bind-pmf-unfold[OF p]*)

(*auto simp: map-pmf-def pair-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'*)

intro!: *bind-pmf-cong*)

Next, we show an explicit formula for the probability mass function of the negative binomial distribution:

lemma *pmf-neg-binomial*:

assumes $p: p \in \{0 < .. 1\}$

shows *pmf* (*neg-binomial-pmf* n p) k = *real* (($k + n - 1$) *choose* k) * p $^ n$ * ($1 - p$) $^ k$

proof (*induction* n *arbitrary: k*)

case 0

thus ?*case* **using** *assms* **by** (*auto simp: indicator-def*)

next

case (*Suc* n)

show ?*case*

proof (*cases* $n = 0$)

case *True*

thus ?*thesis* **using** *assms* **by** *auto*

next

case *False*

let ?*f* = *pmf* (*neg-binomial-pmf* n p)

```

have pmf (neg-binomial-pmf (Suc n) p) k =
  pmf (geometric-pmf p ≫ (λx. map-pmf ((+) x) (neg-binomial-pmf n
p))) k
  by (auto simp: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf
neg-binomial-pmf-Suc)
  also have ... = measure-pmf.expectation (geometric-pmf p)
    (λx. measure-pmf.prob (neg-binomial-pmf n p) ((+) x - ‘{k}’))
  by (simp add: pmf-bind pmf-map)
  also have (λx. (+) x - ‘{k}’) = (λx. if x ≤ k then {k - x} else {})
  by (auto simp: fun-eq-iff)
  also have (λx. measure-pmf.prob (neg-binomial-pmf n p) (... x)) =
    (λx. if x ≤ k then ?f(k - x) else 0)
  by (auto simp: fun-eq-iff measure-pmf-single)
  also have measure-pmf.expectation (geometric-pmf p) ... =
    (∑ i ≤ k. pmf (neg-binomial-pmf n p) (k - i) * pmf (geometric-pmf
p) i)
  by (subst integral-measure-pmf-real[where A = {...k}]) (auto split: if-splits)
  also have ... = p^(n+1) * (1-p)^k * real (∑ i ≤ k. (k - i + n - 1) choose
(k - i))
  unfolding sum-distrib-left of-nat-sum
  proof (intro sum.cong refl, goal-cases)
  case (1 i)
  have pmf (neg-binomial-pmf n p) (k - i) * pmf (geometric-pmf p) i =
    real ((k - i + n - 1) choose (k - i)) * p^(n+1) * ((1-p)^(k-i) *
(1-p)^i)
  using assms Suc.IH by (simp add: mult-ac)
  also have (1-p)^(k-i) * (1-p)^i = (1-p)^k
  using 1 by (subst power-add [symmetric]) auto
  finally show ?case by simp
  qed
  also have (∑ i ≤ k. (k - i + n - 1) choose (k - i)) = (∑ i ≤ k. (n - 1 + i)
choose i)
  by (intro sum.reindex-bij-witness[of - λi. k - i λi. k - i])
    (use ⟨n ≠ 0⟩ in ⟨auto simp: algebra-simps⟩)
  also have ... = (n + k) choose k
  by (subst sum-choose-lower) (use ⟨n ≠ 0⟩ in auto)
  finally show ?thesis
  by (simp add: add-ac)
qed
qed

```

lemma gbinomial-0-left: 0 gchoose k = (if k = 0 then 1 else 0)
by (cases k) auto

The following alternative formula highlights why it is called ‘negative binomial distribution’:

lemma pmf-neg-binomial':
assumes p: p ∈ {0 <..1}

```

shows pmf (neg-binomial-pmf n p) k = (-1) ^ k * ((-real n) gchoose k) * p
^ n * (1 - p) ^ k
proof (cases n > 0)
  case n: True
    have pmf (neg-binomial-pmf n p) k = real ((k + n - 1) choose k) * p ^ n * (1
- p) ^ k
    by (rule pmf-neg-binomial) fact+
    also have real ((k + n - 1) choose k) = ((real k + real n - 1) gchoose k)
    using n by (subst binomial-gbinomial) (auto simp: of-nat-diff)
    also have ... = (-1) ^ k * ((-real n) gchoose k)
    by (subst gbinomial-negated-upper) auto
    finally show ?thesis by simp
qed (auto simp: indicator-def gbinomial-0-left)

```

The cumulative distribution function of the negative binomial distribution can be expressed in terms of that of the ‘normal’ binomial distribution.

lemma *prob-neg-binomial-pmf-atMost*:

```

assumes p: p ∈ {0<..1}
shows measure-pmf.prob (neg-binomial-pmf n p) {..k} =
  measure-pmf.prob (binomial-pmf (n + k) (1 - p)) {..k}
proof (cases n = 0)
  case [simp]: True
    have set-pmf (binomial-pmf (n + k) (1 - p)) ⊆ {..n+k}
    using p by (subst set-pmf-binomial-eq) auto
    hence measure-pmf.prob (binomial-pmf (n + k) (1 - p)) {..k} = 1
    by (subst measure-pmf.prob-eq-1) (auto intro!: AE-pmfI)
    thus ?thesis by simp
  next
    case False
    hence n: n > 0 by auto
    have measure-pmf.prob (binomial-pmf (n + k) (1 - p)) {..k} = (∑ i≤k. pmf
(binomial-pmf (n + k) (1 - p)) i)
    by (intro measure-measure-pmf-finite) auto
    also have ... = (∑ i≤k. real ((n + k) choose i) * p ^ (n + k - i) * (1 - p) ^
i)
    using p by (simp add: mult-ac)
    also have ... = p ^ n * (∑ i≤k. real ((n + k) choose i) * (1 - p) ^ i * p ^ (k
- i))
    unfolding sum-distrib-left by (intro sum.cong) (auto simp: algebra-simps simp
flip: power-add)
    also have (∑ i≤k. real ((n + k) choose i) * (1 - p) ^ i * p ^ (k - i)) =
  (∑ i≤k. ((n + i - 1) choose i) * (1 - p) ^ i)
    using gbinomial-partial-sum-poly-xpos[of k real n 1 - p p] n
    by (simp add: binomial-gbinomial add-ac of-nat-diff)
    also have p ^ n * ... = (∑ i≤k. pmf (neg-binomial-pmf n p) i)
    using p unfolding sum-distrib-left by (simp add: pmf-neg-binomial alge-
bra-simps)
    also have ... = measure-pmf.prob (neg-binomial-pmf n p) {..k}
    by (intro measure-measure-pmf-finite [symmetric]) auto

```

finally show *?thesis ..*
qed

lemma *prob-neg-binomial-pmf-lessThan*:
assumes $p: p \in \{0 < .. 1\}$
shows $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{.. < k\} =$
 $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k - 1) \ (1 - p)) \ \{.. < k\}$
proof (*cases* $k = 0$)
case *False*
hence $\{.. < k\} = \{.. k - 1\}$
by *auto*
thus *?thesis*
using *prob-neg-binomial-pmf-atMost[OF p, of n k - 1] False by simp*
qed *auto*

The expected value of the negative binomial distribution is $n(1 - p)/p$:

lemma *nn-integral-neg-binomial-pmf-real*:
assumes $p: p \in \{0 < .. 1\}$
shows $\text{nn-integral } (\text{measure-pmf } (\text{neg-binomial-pmf } n \ p)) \ \text{of-nat} = \text{ennreal } (n * (1 - p) / p)$
proof (*induction* n)
case 0
thus *?case by auto*
next
case (*Suc* n)
have $\text{nn-integral } (\text{measure-pmf } (\text{neg-binomial-pmf } (\text{Suc } n) \ p)) \ \text{of-nat} =$
 $\text{nn-integral } (\text{measure-pmf } (\text{geometric-pmf } p)) \ \text{of-nat} +$
 $\text{nn-integral } (\text{measure-pmf } (\text{neg-binomial-pmf } n \ p)) \ \text{of-nat}$
by (*simp add: neg-binomial-pmf-Suc case-prod-unfold nn-integral-add nn-integral-pair-pmf'*)
also have $\text{nn-integral } (\text{measure-pmf } (\text{geometric-pmf } p)) \ \text{of-nat} = \text{ennreal } ((1 - p) / p)$
unfolding *ennreal-of-nat-eq-real-of-nat*
using *expectation-geometric-pmf[OF p] integrable-real-geometric-pmf[OF p]*
by (*subst nn-integral-eq-integral*) *auto*
also have $\text{nn-integral } (\text{measure-pmf } (\text{neg-binomial-pmf } n \ p)) \ \text{of-nat} = n * (1 - p) / p$ **using** p
by (*subst Suc.IH*)
(auto simp: ennreal-of-nat-eq-real-of-nat ennreal-mult simp flip: divide-ennreal ennreal-minus)
also have $\text{ennreal } ((1 - p) / p) + \text{ennreal } (\text{real } n * (1 - p) / p) =$
 $\text{ennreal } ((1 - p) / p + \text{real } n * (1 - p) / p)$
by (*intro ennreal-plus [symmetric] divide-nonneg-pos mult-nonneg-nonneg*) (*use p in auto*)
also have $(1 - p) / p + \text{real } n * (1 - p) / p = \text{real } (\text{Suc } n) * (1 - p) / p$
using p **by** (*auto simp: field-simps*)
finally show *?case*
by (*simp add: ennreal-of-nat-eq-real-of-nat*)
qed

lemma *integrable-neg-binomial-pmf-real*:
assumes $p: p \in \{0 < .. 1\}$
shows *integrable* (*measure-pmf* (*neg-binomial-pmf* n p)) *real*
using *nn-integral-neg-binomial-pmf-real*[*OF* p , *of* n]
by (*subst integrable-iff-bounded*) (*auto simp flip: ennreal-of-nat-eq-real-of-nat*)

lemma *expectation-neg-binomial-pmf*:
assumes $p: p \in \{0 < .. 1\}$
shows *measure-pmf.expectation* (*neg-binomial-pmf* n p) *real* = $n * (1 - p) / p$
proof –
have *nn-integral* (*measure-pmf* (*neg-binomial-pmf* n p)) *of-nat* = *ennreal* ($n * (1 - p) / p$)
by (*intro nn-integral-neg-binomial-pmf-real* p)
also have *of-nat* = ($\lambda x.$ *ennreal* (*real* x))
by (*simp add: ennreal-of-nat-eq-real-of-nat fun-eq-iff*)
finally show *?thesis*
using p **by** (*subst (asm) nn-integral-eq-integrable*) *auto*
qed

20.8 PMFs from association lists

definition *pmf-of-list* :: $('a \times \text{real}) \text{ list} \Rightarrow 'a \text{ pmf}$ **where**
pmf-of-list xs = *embed-pmf* ($\lambda x.$ *sum-list* (*map snd* (*filter* ($\lambda z.$ *fst* $z = x$) xs)))

definition *pmf-of-list-wf* **where**
pmf-of-list-wf $xs \longleftrightarrow (\forall x \in \text{set} (\text{map snd } xs) . x \geq 0) \wedge \text{sum-list} (\text{map snd } xs) = 1$

lemma *pmf-of-list-wfI*:
 $(\bigwedge x. x \in \text{set} (\text{map snd } xs) \implies x \geq 0) \implies \text{sum-list} (\text{map snd } xs) = 1 \implies$
pmf-of-list-wf xs
unfolding *pmf-of-list-wf-def* **by** *simp*

context
begin

private lemma *pmf-of-list-aux*:
assumes $\bigwedge x. x \in \text{set} (\text{map snd } xs) \implies x \geq 0$
assumes *sum-list* (*map snd* xs) = 1
shows $(\int^+ x. \text{ennreal} (\text{sum-list} (\text{map snd } [z \leftarrow xs . \text{fst } z = x]))) \text{ } \partial \text{count-space UNIV} = 1$
proof –
have $(\int^+ x. \text{ennreal} (\text{sum-list} (\text{map snd} (\text{filter} (\lambda z. \text{fst } z = x) xs)))) \text{ } \partial \text{count-space UNIV} =$
 $(\int^+ x. \text{ennreal} (\text{sum-list} (\text{map} (\lambda(x',p). \text{indicator } \{x\} x * p) xs))) \text{ } \partial \text{count-space UNIV}$
apply (*intro nn-integral-cong ennreal-cong, subst sum-list-map-filter'*)
apply (*rule arg-cong[where f = sum-list]*)
apply (*auto cong: map-cong*)

done
also have $\dots = (\sum (x',p) \leftarrow xs. (\int^+ x. \text{ennreal} (\text{indicator } \{x'\} x * p) \partial \text{count-space UNIV}))$
using *assms(1)*
proof (*induction xs*)
case (*Cons x xs*)
from *Cons.prem*s **have** $\text{snd } x \geq 0$ **by** *simp*
moreover have $b \geq 0$ **if** $(a,b) \in \text{set } xs$ **for** $a b$
using *Cons.prem*s[*of b*] **that by force**
ultimately have $(\int^+ y. \text{ennreal} (\sum (x', p) \leftarrow x \# xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}) =$
 $(\int^+ y. \text{ennreal} (\text{indicator } \{\text{fst } x\} y * \text{snd } x) +$
 $\text{ennreal} (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV})$
by (*intro nn-integral-cong, subst ennreal-plus [symmetric]*)
(auto simp: case-prod-unfold indicator-def intro!: sum-list-nonneg)
also have $\dots = (\int^+ y. \text{ennreal} (\text{indicator } \{\text{fst } x\} y * \text{snd } x) \partial \text{count-space UNIV}) +$
 $(\int^+ y. \text{ennreal} (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV})$
by (*intro nn-integral-add*)
(force intro!: sum-list-nonneg AE-I2 intro: Cons simp: indicator-def)+
also have $(\int^+ y. \text{ennreal} (\sum (x', p) \leftarrow xs. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}) =$
 $(\sum (x', p) \leftarrow xs. (\int^+ y. \text{ennreal} (\text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}))$
using *Cons(1)* **by** (*intro Cons simp-all*)
finally show *?case* **by** (*simp add: case-prod-unfold*)
qed simp
also have $\dots = (\sum (x',p) \leftarrow xs. \text{ennreal } p * (\int^+ x. \text{indicator } \{x'\} x \partial \text{count-space UNIV}))$
using *assms(1)*
by (*simp cong: map-cong only: case-prod-unfold, subst nn-integral-cmult [symmetric]*)
(auto intro!: assms(1) simp: max-def times-ereal.simps [symmetric] mult-ac ereal-indicator
 $\text{simp del: times-ereal.simps})+$
also from *assms* **have** $\dots = \text{sum-list} (\text{map } \text{snd } xs)$ **by** (*simp add: case-prod-unfold sum-list-ennreal*)
also have $\dots = 1$ **using** *assms(2)* **by** *simp*
finally show *?thesis* .
qed

lemma *pmf-pmf-of-list*:
assumes *pmf-of-list-wf xs*
shows $\text{pmf} (\text{pmf-of-list } xs) x = \text{sum-list} (\text{map } \text{snd} (\text{filter } (\lambda z. \text{fst } z = x) xs))$
using *assms pmf-of-list-aux [of xs]* **unfolding** *pmf-of-list-def pmf-of-list-wf-def*
by (*subst pmf-embed-pmf (auto intro!: sum-list-nonneg)*)

end

lemma *set-pmf-of-list*:

assumes *pmf-of-list-wf xs*

shows $\text{set-pmf } (\text{pmf-of-list } xs) \subseteq \text{set } (\text{map } \text{fst } xs)$

proof *clarify*

fix *x* **assume** *A*: $x \in \text{set-pmf } (\text{pmf-of-list } xs)$

show $x \in \text{set } (\text{map } \text{fst } xs)$

proof (*rule ccontr*)

assume $x \notin \text{set } (\text{map } \text{fst } xs)$

hence $[z \leftarrow xs . \text{fst } z = x] = []$ **by** (*auto simp: filter-empty-conv*)

with *A* **assms** **show** *False* **by** (*simp add: pmf-pmf-of-list set-pmf-eq*)

qed

qed

lemma *finite-set-pmf-of-list*:

assumes *pmf-of-list-wf xs*

shows *finite* ($\text{set-pmf } (\text{pmf-of-list } xs)$)

using *assms* **by** (*rule finite-subset[OF set-pmf-of-list] simp-all*)

lemma *emeasure-Int-set-pmf*:

$\text{emeasure } (\text{measure-pmf } p) (A \cap \text{set-pmf } p) = \text{emeasure } (\text{measure-pmf } p) A$

by (*rule emeasure-eq-AE*) (*auto simp: AE-measure-pmf-iff*)

lemma *measure-Int-set-pmf*:

$\text{measure } (\text{measure-pmf } p) (A \cap \text{set-pmf } p) = \text{measure } (\text{measure-pmf } p) A$

using *emeasure-Int-set-pmf[of p A]* **by** (*simp add: Sigma-Algebra.measure-def*)

lemma *measure-prob-cong-0*:

assumes $\bigwedge x. x \in A - B \implies \text{pmf } p \ x = 0$

assumes $\bigwedge x. x \in B - A \implies \text{pmf } p \ x = 0$

shows $\text{measure } (\text{measure-pmf } p) A = \text{measure } (\text{measure-pmf } p) B$

proof –

have $\text{measure-pmf.prob } p \ A = \text{measure-pmf.prob } p \ (A \cap \text{set-pmf } p)$

by (*simp add: measure-Int-set-pmf*)

also have $A \cap \text{set-pmf } p = B \cap \text{set-pmf } p$

using *assms* **by** (*auto simp: set-pmf-eq*)

also have $\text{measure-pmf.prob } p \ \dots = \text{measure-pmf.prob } p \ B$

by (*simp add: measure-Int-set-pmf*)

finally show *?thesis* .

qed

lemma *emeasure-pmf-of-list*:

assumes *pmf-of-list-wf xs*

shows $\text{emeasure } (\text{pmf-of-list } xs) A = \text{ennreal } (\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) xs)))$

proof –

have $\text{emeasure } (\text{pmf-of-list } xs) A = \text{nn-integral } (\text{measure-pmf } (\text{pmf-of-list } xs))$
(*indicator A*)

by *simp*

also from *assms*

have ... = ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } [z \leftarrow xs . \text{fst } z = x]))$)
by (*subst nn-integral-measure-pmf-finite*) (*simp-all add: finite-set-pmf-of-list pmf-pmf-of-list Int-def*)
also from *assms*
have ... = $\text{ennreal } (\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{sum-list } (\text{map } \text{snd } [z \leftarrow xs . \text{fst } z = x]))$
by (*subst sum-ennreal*) (*auto simp: pmf-of-list-wf-def intro!: sum-list-nonneg*)
also have ... = $\text{ennreal } (\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{indicator } A \ x * \text{pmf } (\text{pmf-of-list } xs) \ x)$ (*is - = ennreal ?S*)
using *assms* **by** (*intro ennreal-cong sum.cong*) (*auto simp: pmf-pmf-of-list*)
also have ?S = ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs). \text{indicator } A \ x * \text{pmf } (\text{pmf-of-list } xs) \ x$)
using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list finite-set-pmf-of-list*) *auto*
also have ... = ($\sum x \in \text{set } (\text{map } \text{fst } xs). \text{indicator } A \ x * \text{pmf } (\text{pmf-of-list } xs) \ x$)
using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list*) (*auto simp: set-pmf-eq*)
also have ... = ($\sum x \in \text{set } (\text{map } \text{fst } xs). \text{indicator } A \ x * \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) \ xs)))$)
using *assms* **by** (*simp add: pmf-pmf-of-list*)
also have ... = ($\sum x \in \text{set } (\text{map } \text{fst } xs). \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x \wedge x \in A) \ xs)))$)
by (*intro sum.cong*) (*auto simp: indicator-def*)
also have ... = ($\sum x \in \text{set } (\text{map } \text{fst } xs). (\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) = x \wedge x \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0)$)
by (*intro sum.cong refl, subst sum-list-map-filter', subst sum-list-sum-nth*) *simp*
also have ... = ($\sum xa = 0..<\text{length } xs. (\sum x \in \text{set } (\text{map } \text{fst } xs). \text{if } \text{fst } (xs \ ! \ xa) = x \wedge x \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0)$)
by (*rule sum.swap*)
also have ... = ($\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) \in A \text{ then } (\sum x \in \text{set } (\text{map } \text{fst } xs). \text{if } x = \text{fst } (xs \ ! \ xa) \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0) \text{ else } 0$)
by (*auto intro!: sum.cong sum.neutral simp del: sum.delta*)
also have ... = ($\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0$)
by (*intro sum.cong refl*) (*simp-all add: sum.delta*)
also have ... = $\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) \ xs))$
by (*subst sum-list-map-filter', subst sum-list-sum-nth*) *simp-all*
finally show ?thesis .
qed

lemma *measure-pmf-of-list:*

assumes *pmf-of-list-wf* *xs*

shows $\text{measure } (\text{pmf-of-list } xs) \ A = \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) \ xs))$

using *assms* **unfolding** *pmf-of-list-wf-def Sigma-Algebra.measure-def*

by (*subst emeasure-pmf-of-list [OF assms], subst enn2real-ennreal*) (*auto intro!: sum-list-nonneg*)

lemma *sum-list-nonneg-eq-zero-iff*:
fixes $xs :: 'a :: \text{linordered-ab-group-add list}$
shows $(\bigwedge x. x \in \text{set } xs \implies x \geq 0) \implies \text{sum-list } xs = 0 \iff \text{set } xs \subseteq \{0\}$
proof (*induction xs*)
case (*Cons x xs*)
from *Cons.prem*s **have** $\text{sum-list } (x\#xs) = 0 \iff x = 0 \wedge \text{sum-list } xs = 0$
unfolding *sum-list-simps* **by** (*subst add-nonneg-eq-0-iff*) (*auto intro: sum-list-nonneg*)
with *Cons.IH Cons.prem*s **show** ?*case* **by** *simp*
qed *simp-all*

lemma *sum-list-filter-nonzero*:
 $\text{sum-list } (\text{filter } (\lambda x. x \neq 0) xs) = \text{sum-list } xs$
by (*induction xs*) *simp-all*

lemma *set-pmf-of-list-eq*:
assumes *pmf-of-list-wf xs* $\bigwedge x. x \in \text{snd } ' \text{set } xs \implies x > 0$
shows $\text{set-pmf } (\text{pmf-of-list } xs) = \text{fst } ' \text{set } xs$
proof
{
fix x **assume** $A: x \in \text{fst } ' \text{set } xs$ **and** $B: x \notin \text{set-pmf } (\text{pmf-of-list } xs)$
then obtain y **where** $(x, y) \in \text{set } xs$ **by** *auto*
from B **have** $\text{sum-list } (\text{map } \text{snd } [z \leftarrow xs. \text{fst } z = x]) = 0$
by (*simp add: pmf-pmf-of-list[OF assms(1)] set-pmf-eq*)
moreover from y **have** $y \in \text{snd } ' \{xa \in \text{set } xs. \text{fst } xa = x\}$ **by** *force*
ultimately have $y = 0$ **using** *assms(1)*
by (*subst (asm) sum-list-nonneg-eq-zero-iff*) (*auto simp: pmf-of-list-wf-def*)
with *assms(2) y* **have** *False* **by** *force*
}
thus $\text{fst } ' \text{set } xs \subseteq \text{set-pmf } (\text{pmf-of-list } xs)$ **by** *blast*
qed (*insert set-pmf-of-list[OF assms(1)], simp-all*)

lemma *pmf-of-list-remove-zeros*:
assumes *pmf-of-list-wf xs*
defines $xs' \equiv \text{filter } (\lambda z. \text{snd } z \neq 0) xs$
shows $\text{pmf-of-list-wf } xs' \text{ pmf-of-list } xs' = \text{pmf-of-list } xs$
proof –
have $\text{map } \text{snd } [z \leftarrow xs. \text{snd } z \neq 0] = \text{filter } (\lambda x. x \neq 0) (\text{map } \text{snd } xs)$
by (*induction xs*) *simp-all*
with *assms(1)* **show** *wf: pmf-of-list-wf xs'*
by (*auto simp: pmf-of-list-wf-def xs'-def sum-list-filter-nonzero*)
have $\text{sum-list } (\text{map } \text{snd } [z \leftarrow xs'. \text{fst } z = i]) = \text{sum-list } (\text{map } \text{snd } [z \leftarrow xs. \text{fst } z = i])$ **for** i
unfolding *xs'-def* **by** (*induction xs*) *simp-all*
with *assms(1) wf* **show** $\text{pmf-of-list } xs' = \text{pmf-of-list } xs$
by (*intro pmf-eqI*) (*simp-all add: pmf-pmf-of-list*)
qed

end

21 Code generation for PMFs

```
theory PMF-Impl
imports Probability-Mass-Function HOL-Library.AList-Mapping
begin
```

21.1 General code generation setup

definition *pmf-of-mapping* :: ('a, real) mapping \Rightarrow 'a pmf **where**
pmf-of-mapping m = embed-pmf (Mapping.lookup-default 0 m)

lemma *nn-integral-lookup-default*:

```
fixes m :: ('a, real) mapping
assumes finite (Mapping.keys m) All-mapping m ( $\lambda$ - x. x  $\geq$  0)
shows nn-integral (count-space UNIV) ( $\lambda$ k. ennreal (Mapping.lookup-default 0 m k)) =
ennreal ( $\sum$  k $\in$ Mapping.keys m. Mapping.lookup-default 0 m k)
```

proof –

```
have nn-integral (count-space UNIV) ( $\lambda$ k. ennreal (Mapping.lookup-default 0 m k)) =
```

```
( $\sum$  x $\in$ Mapping.keys m. ennreal (Mapping.lookup-default 0 m x)) using
assms
```

```
by (subst nn-integral-count-space'[of Mapping.keys m])
(auto simp: Mapping.lookup-default-def keys-is-none-rep Option.is-none-def)
```

```
also from assms have ... = ennreal ( $\sum$  k $\in$ Mapping.keys m. Mapping.lookup-default 0 m k)
```

```
by (intro sum-ennreal)
(auto simp: Mapping.lookup-default-def All-mapping-def split: option.splits)
```

```
finally show ?thesis .
```

qed

lemma *pmf-of-mapping*:

```
assumes finite (Mapping.keys m) All-mapping m ( $\lambda$ - p. p  $\geq$  0)
assumes ( $\sum$  x $\in$ Mapping.keys m. Mapping.lookup-default 0 m x) = 1
shows pmf (pmf-of-mapping m) x = Mapping.lookup-default 0 m x
unfolding pmf-of-mapping-def
```

proof (intro pmf-embed-pmf)

```
from assms show ( $\int$ +x. ennreal (Mapping.lookup-default 0 m x)  $\partial$ count-space UNIV) = 1
```

```
by (subst nn-integral-lookup-default) (simp-all)
```

```
qed (insert assms, simp add: All-mapping-def Mapping.lookup-default-def split: option.splits)
```

lemma *pmf-of-set-pmf-of-mapping*:

```
assumes A  $\neq$  {} set xs = A distinct xs
```

```
shows pmf-of-set A = pmf-of-mapping (Mapping.tabulate xs ( $\lambda$ -. 1 / real (length xs)))
```

(**is** ?lhs = ?rhs)
by (rule pmf-eqI, subst pmf-of-mapping)
 (insert assms, auto intro!: All-mapping-tabulate
 simp: Mapping.lookup-default-def lookup-tabulate distinct-card)

lift-definition mapping-of-pmf :: 'a pmf \Rightarrow ('a, real) mapping **is**
 $\lambda p x.$ if pmf p x = 0 then None else Some (pmf p x) .

lemma lookup-default-mapping-of-pmf:
 Mapping.lookup-default 0 (mapping-of-pmf p) x = pmf p x
by (simp add: mapping-of-pmf.abs-eq lookup-default-def Mapping.lookup.abs-eq)

context
begin

interpretation pmf-as-function .

lemma nn-integral-pmf-eq-1: $(\int^+ x. \text{ennreal } (\text{pmf } p \ x) \ \partial \text{count-space UNIV}) = 1$
by transfer simp-all
end

lemma pmf-of-mapping-mapping-of-pmf [code abstype]:
 pmf-of-mapping (mapping-of-pmf p) = p
unfolding pmf-of-mapping-def
by (rule pmf-eqI, subst pmf-embed-pmf)
 (insert nn-integral-pmf-eq-1[of p],
 auto simp: lookup-default-mapping-of-pmf split: option.splits)

lemma mapping-of-pmfI:
assumes $\bigwedge x. x \in \text{Mapping.keys } m \Rightarrow \text{Mapping.lookup } m \ x = \text{Some } (\text{pmf } p \ x)$
assumes $\text{Mapping.keys } m = \text{set-pmf } p$
shows $\text{mapping-of-pmf } p = m$
using assms **by** transfer (rule ext, auto simp: set-pmf-eq)

lemma mapping-of-pmfI':
assumes $\bigwedge x. x \in \text{Mapping.keys } m \Rightarrow \text{Mapping.lookup-default } 0 \ m \ x = \text{pmf } p \ x$
assumes $\text{Mapping.keys } m = \text{set-pmf } p$
shows $\text{mapping-of-pmf } p = m$
using assms **unfolding** Mapping.lookup-default-def
by transfer (rule ext, force simp: set-pmf-eq)

lemma return-pmf-code [code abstract]:
 mapping-of-pmf (return-pmf x) = Mapping.update x 1 Mapping.empty
by (intro mapping-of-pmfI) (auto simp: lookup-update')

lemma pmf-of-set-code-aux:
assumes $A \neq \{\}$ set xs = A distinct xs
shows $\text{mapping-of-pmf } (\text{pmf-of-set } A) = \text{Mapping.tabulate } xs \ (\lambda-. \ 1 \ / \ \text{real})$

(length xs))
using *assms*
by (*intro mapping-of-pmfI*, *subst pmf-of-set*)
 (*auto simp: lookup-tabulate distinct-card*)

definition *pmf-of-set-impl* **where**
pmf-of-set-impl A = *mapping-of-pmf* (pmf-of-set A)

lemma *pmf-of-set-impl-code-alt*:
assumes A ≠ {} *finite* A
shows *pmf-of-set-impl* A =
 (let p = 1 / real (card A)
 in *Finite-Set.fold* (λx. *Mapping.update* x p) *Mapping.empty* A)

proof –

define p **where** p = 1 / real (card A)
let ?m = *Finite-Set.fold* (λx. *Mapping.update* x p) *Mapping.empty* A
interpret *comp-fun-idem* λx. *Mapping.update* x p
by *standard* (*transfer*, *force simp: fun-eq-iff*)
have *keys: Mapping.keys* ?m = A
using *assms*(2) **by** (*induction* A *rule: finite-induct*) *simp-all*
have *lookup: Mapping.lookup* ?m x = *Some* p **if** x ∈ A **for** x
using *assms*(2) **that** **by** (*induction* A *rule: finite-induct*) (*auto simp: lookup-update*)
from *keys lookup assms* **show** ?*thesis* **unfolding** *pmf-of-set-impl-def*
by (*intro mapping-of-pmfI*) (*simp-all add: Let-def p-def*)

qed

lemma *pmf-of-set-impl-code* [*code*]:
pmf-of-set-impl (set xs) =
 (*if* xs = [] *then*
 Code.abort (*STR* "pmf-of-set of empty set") (λ-. *mapping-of-pmf* (pmf-of-set
 (set xs)))
else let xs' = *remdups* xs; p = 1 / real (length xs') *in*
 Mapping.tabulate xs' (λ-. p))
unfolding *pmf-of-set-impl-def*
using *pmf-of-set-code-aux*[*of set xs remdups xs*] **by** (*simp add: Let-def*)

lemma *pmf-of-set-code* [*code abstract*]:
mapping-of-pmf (pmf-of-set A) = *pmf-of-set-impl* A
by (*simp add: pmf-of-set-impl-def*)

lemma *pmf-of-multiset-pmf-of-mapping*:
assumes A ≠ {} *set* xs = *set-mset* A *distinct* xs
shows *mapping-of-pmf* (pmf-of-multiset A) = *Mapping.tabulate* xs (λx. *count*
 A x / real (size A))
using *assms* **by** (*intro mapping-of-pmfI*) (*auto simp: lookup-tabulate*)

definition *pmf-of-multiset-impl* **where**

pmf-of-multiset-impl A = mapping-of-pmf (pmf-of-multiset A)

lemma *pmf-of-multiset-impl-code-alt:*

assumes $A \neq \{\#\}$

shows *pmf-of-multiset-impl A =*

(let p = 1 / real (size A)

in fold-mset ($\lambda x. Mapping.map-default x 0 ((+) p)$) Mapping.empty A)

proof –

define *p* **where** $p = 1 / \text{real } (\text{size } A)$

interpret *comp-fun-commute* $\lambda x. Mapping.map-default x 0 ((+) p)$

unfolding *Mapping.map-default-def* [abs-def]

by (*standard, intro mapping-eqI ext*)

(simp-all add: o-def lookup-map-entry' lookup-default' lookup-default-def)

let $?m = \text{fold-mset } (\lambda x. Mapping.map-default x 0 ((+) p)) Mapping.empty A$

have *keys: Mapping.keys ?m = set-mset A* **by** (*induction A simp-all*)

have *lookup: Mapping.lookup-default 0 ?m x = real (count A x) * p* **for** x

by (*induction A*)

(simp-all add: lookup-map-default' lookup-default-def lookup-empty ring-distrib)

from *keys lookup assms* **show** *?thesis unfolding pmf-of-multiset-impl-def*

by (*intro mapping-of-pmfI'*) (*simp-all add: Let-def p-def*)

qed

lemma *pmf-of-multiset-impl-code* [code]:

pmf-of-multiset-impl (mset xs) =

(if xs = [] then

Code.abort (STR "pmf-of-multiset of empty multiset")

($\lambda \cdot. mapping-of-pmf (pmf-of-multiset (mset xs))$)

else let xs' = remdups xs; p = 1 / real (length xs) in

*Mapping.tabulate xs' ($\lambda x. \text{real } (\text{count } (mset xs) x) * p$)*)

using *pmf-of-multiset-pmf-of-mapping[of mset xs remdups xs]*

by (*simp add: pmf-of-multiset-impl-def*)

lemma *pmf-of-multiset-code* [code abstract]:

mapping-of-pmf (pmf-of-multiset A) = pmf-of-multiset-impl A

by (*simp add: pmf-of-multiset-impl-def*)

lemma *bernoulli-pmf-code* [code abstract]:

mapping-of-pmf (bernoulli-pmf p) =

(if p ≤ 0 then Mapping.update False 1 Mapping.empty

else if p ≥ 1 then Mapping.update True 1 Mapping.empty

else Mapping.update False (1 – p) (Mapping.update True p Mapping.empty))

by (*intro mapping-of-pmfI (auto simp: bernoulli-pmf.rep-eq lookup-update' set-pmf-eq)*)

lemma *pmf-code* [code]: $pmf p x = Mapping.lookup-default 0 (mapping-of-pmf p)$

x

unfolding *mapping-of-pmf-def Mapping.lookup-default-def*

by (*auto split: option.splits simp: id-def Mapping.lookup.abs-eq*)

lemma *set-pmf-code* [code]: $\text{set-pmf } p = \text{Mapping.keys } (\text{mapping-of-pmf } p)$
by *transfer* (*auto simp: dom-def set-pmf-eq*)

lemma *keys-mapping-of-pmf* [simp]: $\text{Mapping.keys } (\text{mapping-of-pmf } p) = \text{set-pmf } p$
by *transfer* (*auto simp: dom-def set-pmf-eq*)

definition *fold-combine-plus* **where**

fold-combine-plus = *comm-monoid-set.F* (*Mapping.combine* ((+) :: *real* \Rightarrow -))
Mapping.empty

context
begin

interpretation *fold-combine-plus*: *combine-mapping-abel-semigroup* (+) :: *real* \Rightarrow -
by *unfold-locales* (*simp-all add: add-ac*)

qualified lemma *lookup-default-fold-combine-plus*:

fixes $A :: 'b \text{ set}$ **and** $f :: 'b \Rightarrow ('a, \text{real}) \text{ mapping}$

assumes *finite A*

shows $\text{Mapping.lookup-default } 0 (\text{fold-combine-plus } f A) x =$
 $(\sum_{y \in A. \text{Mapping.lookup-default } 0 (f y) x}$

unfolding *fold-combine-plus-def* **using** *assms*

by (*induction A rule: finite-induct*)

(*simp-all add: lookup-default-empty lookup-default-neutral-combine*)

qualified lemma *keys-fold-combine-plus*:

finite A $\implies \text{Mapping.keys } (\text{fold-combine-plus } f A) = (\bigcup_{x \in A. \text{Mapping.keys } (f x))$

by (*simp add: fold-combine-plus-def fold-combine-plus.keys-fold-combine*)

qualified lemma *fold-combine-plus-code* [code]:

fold-combine-plus $g (\text{set } xs) = \text{foldr } (\lambda x. \text{Mapping.combine } (+) (g x)) (\text{remdups } xs) \text{Mapping.empty}$

by (*simp add: fold-combine-plus-def fold-combine-plus.fold-combine-code*)

private lemma *lookup-default-0-map-values*:

assumes $f x 0 = 0$

shows $\text{Mapping.lookup-default } 0 (\text{Mapping.map-values } f m) x = f x (\text{Mapping.lookup-default } 0 m x)$

unfolding *Mapping.lookup-default-def*

using *assms* **by** *transfer* (*auto split: option.splits*)

qualified lemma *mapping-of-bind-pmf*:

assumes *finite* (*set-pmf p*)

shows $\text{mapping-of-pmf } (\text{bind-pmf } p f) =$
 $\text{fold-combine-plus } (\lambda x. \text{Mapping.map-values } (\lambda \cdot. (*) (pmf\ p\ x))$
 $(\text{mapping-of-pmf } (f\ x))) (\text{set-pmf } p)$

using *assms*

by (*intro mapping-of-pmfI'*)
(auto simp: keys-fold-combine-plus lookup-default-fold-combine-plus
pmf-bind integral-measure-pmf lookup-default-0-map-values
lookup-default-mapping-of-pmf mult-ac)

lift-definition $\text{bind-pmf-aux} :: 'a\ pmf \Rightarrow ('a \Rightarrow 'b\ pmf) \Rightarrow 'a\ set \Rightarrow ('b, real)$
mapping is
 $\lambda(p :: 'a\ pmf) (f :: 'a \Rightarrow 'b\ pmf) (A::'a\ set) (x::'b).$
if $x \in (\bigcup y \in A. \text{set-pmf } (f\ y))$ *then*
 $\text{Some } (\text{measure-pmf.expectation } p (\lambda y. \text{indicator } A\ y * pmf\ (f\ y)\ x))$
else *None* .

lemma *keys-bind-pmf-aux [simp]:*
 $\text{Mapping.keys } (\text{bind-pmf-aux } p\ f\ A) = (\bigcup x \in A. \text{set-pmf } (f\ x))$
by *transfer (auto split: if-splits)*

lemma *lookup-default-bind-pmf-aux:*
 $\text{Mapping.lookup-default } 0 (\text{bind-pmf-aux } p\ f\ A)\ x =$
(if $x \in (\bigcup y \in A. \text{set-pmf } (f\ y))$ *then*
 $\text{measure-pmf.expectation } p (\lambda y. \text{indicator } A\ y * pmf\ (f\ y)\ x)$ *else* 0)
unfolding *lookup-default-def by transfer' simp-all*

lemma *lookup-default-bind-pmf-aux' [simp]:*
 $\text{Mapping.lookup-default } 0 (\text{bind-pmf-aux } p\ f\ (\text{set-pmf } p))\ x = pmf\ (\text{bind-pmf } p\ f)$
 x
unfolding *lookup-default-def*
by *transfer (auto simp: pmf-bind AE-measure-pmf-iff set-pmf-eq*
intro!: integral-cong-AE integral-eq-zero-AE)

lemma *bind-pmf-aux-correct:*
 $\text{mapping-of-pmf } (\text{bind-pmf } p\ f) = \text{bind-pmf-aux } p\ f\ (\text{set-pmf } p)$
by (*intro mapping-of-pmfI'*) *simp-all*

lemma *bind-pmf-aux-code-aux:*
assumes *finite A*
shows $\text{bind-pmf-aux } p\ f\ A =$
 $\text{fold-combine-plus } (\lambda x. \text{Mapping.map-values } (\lambda \cdot. (*) (pmf\ p\ x))$
 $(\text{mapping-of-pmf } (f\ x))) A$ (**is** *?lhs = ?rhs*)

proof (*intro mapping-eqI'[where d = 0]*)
fix x **assume** $x \in \text{Mapping.keys } ?lhs$
then obtain y **where** $y: y \in A\ x \in \text{set-pmf } (f\ y)$ **by** *auto*
hence $\text{Mapping.lookup-default } 0\ ?lhs\ x =$
 $\text{measure-pmf.expectation } p (\lambda y. \text{indicator } A\ y * pmf\ (f\ y)\ x)$
by (*auto simp: lookup-default-bind-pmf-aux*)
also from *assms* **have** $\dots = (\sum y \in A. pmf\ p\ y * pmf\ (f\ y)\ x)$

by (*subst integral-measure-pmf [of A]*)
 (*auto simp: set-pmf-eq indicator-def mult-ac split: if-splits*)
also from *assms* **have** ... = *Mapping.lookup-default 0 ?rhs x*
by (*simp add: lookup-default-fold-combine-plus lookup-default-0-map-values*
lookup-default-mapping-of-pmf)
finally show *Mapping.lookup-default 0 ?lhs x = Mapping.lookup-default 0 ?rhs*
x .
qed (*insert assms, simp-all add: keys-fold-combine-plus*)

lemma *bind-pmf-aux-code* [*code*]:
bind-pmf-aux p f (set xs) =
fold-combine-plus ($\lambda x. Mapping.map-values (\lambda \cdot. () (pmf p x))$*
(mapping-of-pmf (f x))) (set xs)
by (*rule bind-pmf-aux-code-aux simp-all*)

lemmas *bind-pmf-code* [*code abstract*] = *bind-pmf-aux-correct*

end

hide-const (**open**) *fold-combine-plus*

lift-definition *cond-pmf-impl* :: '*a pmf* \Rightarrow '*a set* \Rightarrow ('*a, real*) *mapping option is*
 $\lambda p A. \text{if } A \cap \text{set-pmf } p = \{\} \text{ then None else$
Some ($\lambda x. \text{if } x \in A \cap \text{set-pmf } p \text{ then Some (pmf } p \ x / \text{measure-pmf.prob } p \ A)$
else None) .

lemma *cond-pmf-impl-code-alt*:
assumes *finite A*
shows *cond-pmf-impl p A = (*
let C = $A \cap \text{set-pmf } p$;
prob = $(\sum x \in C. \text{pmf } p \ x)$
in if prob = 0 then
None
else
*Some (*Mapping.map-values* ($\lambda y. y / \text{prob}$)*
*(*Mapping.filter* ($\lambda k \cdot k \in C$) (*mapping-of-pmf p*))))*

proof –

define *C* **where** *C = $A \cap \text{set-pmf } p$*
define *prob* **where** *prob = $(\sum x \in C. \text{pmf } p \ x)$*
also note *C-def*
also from *assms* **have** $(\sum x \in A \cap \text{set-pmf } p. \text{pmf } p \ x) = (\sum x \in A. \text{pmf } p \ x)$
by (*intro sum.mono-neutral-left*) (*auto simp: set-pmf-eq*)
finally have *prob1*: *prob = $(\sum x \in A. \text{pmf } p \ x)$* .
hence *prob2*: *prob = measure-pmf.prob p A*
using *assms* **by** (*subst measure-measure-pmf-finite*) *simp-all*
have *prob3*: *prob = 0 $\longleftrightarrow A \cap \text{set-pmf } p = \{\}$*
by (*subst prob1, subst sum-nonneg-eq-0-iff*) (*auto simp: set-pmf-eq assms*)
from *assms* **have** *prob4*: *prob = measure-pmf.prob p C*

unfolding *prob-def* **by** (*intro measure-measure-pmf-finite [symmetric]*) (*simp-all add: C-def*)

show *?thesis*

proof (*cases prob = 0*)

case *True*

hence $A \cap \text{set-pmf } p = \{\}$ **by** (*subst (asm) prob3*)

with *True* **show** *?thesis* **by** (*simp add: Let-def prob-def C-def cond-pmf-impl.abs-eq*)

next

case *False*

hence $A: C \neq \{\}$ **unfolding** *C-def* **by** (*subst (asm) prob3*) *auto*

with *prob3* **have** *prob-nz: prob $\neq 0$* **by** (*auto simp: C-def*)

fix *x*

have *cond-pmf-impl p A =*

Some (mapping.Mapping ($\lambda x. \text{if } x \in C \text{ then}$

Some (pmf p x / measure-pmf.prob p C) else None))

(is - = Some ?m)

using *A prob2 prob4* **unfolding** *C-def* **by** *transfer (auto simp: fun-eq-iff)*

also have *?m = Mapping.map-values ($\lambda y. y / \text{prob}$)*

(Mapping.filter ($\lambda k -. k \in C$) (mapping-of-pmf p))

using *prob-nz prob4 assms* **unfolding** *C-def*

by *transfer (auto simp: fun-eq-iff set-pmf-eq)*

finally show *?thesis* **using** *False* **by** (*simp add: Let-def prob-def C-def*)

qed

qed

lemma *cond-pmf-impl-code [code]:*

cond-pmf-impl p (set xs) = (

let C = set xs \cap set-pmf p;

prob = ($\sum_{x \in C} \text{pmf } p x$)

in if prob = 0 then

None

else

Some (Mapping.map-values ($\lambda y. y / \text{prob}$)

(Mapping.filter ($\lambda k -. k \in C$) (mapping-of-pmf p))))

by (*rule cond-pmf-impl-code-alt*) *simp-all*

lemma *cond-pmf-code [code abstract]:*

mapping-of-pmf (cond-pmf p A) =

(case cond-pmf-impl p A of

None \Rightarrow Code.abort (STR "cond-pmf with set of probability 0")

($\lambda -. \text{mapping-of-pmf (cond-pmf p A)}$)

| Some m \Rightarrow m)

proof (*cases cond-pmf-impl p A*)

case (*Some m*)

hence $A: \text{set-pmf } p \cap A \neq \{\}$ **by** *transfer (auto split: if-splits)*

from *Some* **have** $B: \text{Mapping.keys } m = \text{set-pmf (cond-pmf p A)}$

by (*subst set-cond-pmf[OF A], transfer*) (*auto split: if-splits*)

with *Some A* **have** *mapping-of-pmf (cond-pmf p A) = m*

by (*intro mapping-of-pmfI* [*OF - B*], *transfer*) (*auto split: if-splits simp: pmf-cond*)
with *Some show ?thesis by simp*
qed *simp-all*

lemma *binomial-pmf-code* [*code abstract*]:
mapping-of-pmf (*binomial-pmf* *n p*) = (
 if $p < 0 \vee p > 1$ then
 Code.abort (*STR "binomial-pmf with invalid probability"*)
 (λ -. *mapping-of-pmf* (*binomial-pmf* *n p*))
 else if $p = 0$ then *Mapping.update* 0 1 *Mapping.empty*
 else if $p = 1$ then *Mapping.update* *n* 1 *Mapping.empty*
 else *Mapping.tabulate* [$0..<Suc\ n$] (λk . *real* (*n choose k*) * $p^k * (1 - p)^{n - k}$))
by (*cases* $p < 0 \vee p > 1$)
 (*simp*, *intro mapping-of-pmfI*,
auto simp: lookup-update' lookup-empty set-pmf-binomial-eq lookup-tabulate
split: if-splits)

lemma *pred-pmf-code* [*code*]:
pred-pmf *P p* = ($\forall x \in \text{set-pmf } p$. *P x*)
by (*auto simp: pred-pmf-def*)

lemma *mapping-of-pmf-pmf-of-list*:
assumes $\bigwedge x. x \in \text{snd } 'set\ xs \implies x > 0$ *sum-list* (*map snd xs*) = 1
shows *mapping-of-pmf* (*pmf-of-list xs*) =
Mapping.tabulate (*remdups* (*map fst xs*))
 (λx . *sum-list* (*map snd* (*filter* (λz . *fst z = x*) *xs*))))
proof –
from *assms* **have** *wf: pmf-of-list-wf xs* **by** (*intro pmf-of-list-wfI*) *force*
with *assms* **have** *set-pmf* (*pmf-of-list xs*) = *fst 'set xs*
by (*intro set-pmf-of-list-eq*) *auto*
with *wf* **show** *?thesis*
by (*intro mapping-of-pmfI*) (*auto simp: lookup-tabulate pmf-pmf-of-list*)
qed

lemma *mapping-of-pmf-pmf-of-list'*:
assumes *pmf-of-list-wf xs*
defines $xs' \equiv \text{filter } (\lambda z. \text{snd } z \neq 0) xs$
shows *mapping-of-pmf* (*pmf-of-list xs*) =
Mapping.tabulate (*remdups* (*map fst xs'*))
 (λx . *sum-list* (*map snd* (*filter* (λz . *fst z = x*) *xs'*))) (**is** - = *?rhs*)
proof –
have *wf: pmf-of-list-wf xs'* **unfolding** *xs'-def* **by** (*rule pmf-of-list-remove-zeros*)
fact
have *pos: $\forall x \in \text{snd } 'set\ xs'. x > 0$* **using** *assms(1)* **unfolding** *xs'-def*
by (*force simp: pmf-of-list-wf-def*)

```

from assms have pmf-of-list xs = pmf-of-list xs'
  unfolding xs'-def by (subst pmf-of-list-remove-zeros) simp-all
also from wf pos have mapping-of-pmf ... = ?rhs
  by (intro mapping-of-pmf-pmf-of-list) (auto simp: pmf-of-list-wf-def)
finally show ?thesis .
qed

```

```

lemma pmf-of-list-wf-code [code]:
  pmf-of-list-wf xs  $\longleftrightarrow$  list-all ( $\lambda z. \text{snd } z \geq 0$ ) xs  $\wedge$  sum-list (map snd xs) = 1
  by (auto simp add: pmf-of-list-wf-def list-all-def)

```

```

lemma pmf-of-list-code [code abstract]:
  mapping-of-pmf (pmf-of-list xs) = (
    if pmf-of-list-wf xs then
      let xs' = filter ( $\lambda z. \text{snd } z \neq 0$ ) xs
      in Mapping.tabulate (remdups (map fst xs'))
        ( $\lambda x. \text{sum-list$  (map snd (filter ( $\lambda z. \text{fst } z = x$ ) xs')))
    else
      Code.abort (STR "Invalid list for pmf-of-list") ( $\lambda \cdot. \text{mapping-of-pmf}$  (pmf-of-list
xs)))
  using mapping-of-pmf-pmf-of-list'[of xs] by (simp add: Let-def)

```

```

lemma mapping-of-pmf-eq-iff [simp]:
  mapping-of-pmf p = mapping-of-pmf q  $\longleftrightarrow$  p = (q :: 'a pmf)
proof (transfer, intro iffI pmf-eqI)
  fix p q :: 'a pmf and x :: 'a
  assume ( $\lambda x. \text{if pmf } p \ x = 0 \text{ then None else Some } (pmf \ p \ x) =$ 
    ( $\lambda x. \text{if pmf } q \ x = 0 \text{ then None else Some } (pmf \ q \ x)$ )
  hence (if pmf p x = 0 then None else Some (pmf p x) =
    (if pmf q x = 0 then None else Some (pmf q x)) for x
  by (simp add: fun-eq-iff)
  from this[of x] show pmf p x = pmf q x by (auto split: if-splits)
qed (simp-all cong: if-cong)

```

21.2 Code abbreviations for integrals and probabilities

Integrals and probabilities are defined for general measures, so we cannot give any code equations directly. We can, however, specialise these constants them to PMFs, give code equations for these specialised constants, and tell the code generator to unfold the original constants to the specialised ones whenever possible.

```

definition pmf-integral where
  pmf-integral p f = lebesgue-integral (measure-pmf p) (f :: -  $\Rightarrow$  real)

```

```

definition pmf-set-integral where
  pmf-set-integral p f A = lebesgue-integral (measure-pmf p) ( $\lambda x. \text{indicator } A \ x * f$ 
x :: real)

```

definition *pmf-prob* where

pmf-prob p $A = \text{measure-pmf.prob } p$ A

lemma *pmf-prob-compl*: *pmf-prob* p $(-A) = 1 - \text{pmf-prob } p$ A

using *measure-pmf.prob-compl*[of A p] **by** (*simp* *add*: *pmf-prob-def Compl-eq-Diff-UNIV*)

lemma *pmf-integral-pmf-set-integral* [code]:

pmf-integral p $f = \text{pmf-set-integral } p$ f (*set-pmf* p)

unfolding *pmf-integral-def pmf-set-integral-def*

by (*intro integral-cong-AE*) (*simp-all* *add*: *AE-measure-pmf-iff*)

lemma *pmf-prob-pmf-set-integral*:

pmf-prob p $A = \text{pmf-set-integral } p$ $(\lambda-. 1)$ A

by (*simp* *add*: *pmf-prob-def pmf-set-integral-def*)

lemma *pmf-set-integral-code-alt-finite*:

finite $A \implies \text{pmf-set-integral } p$ f $A = (\sum x \in A. \text{pmf } p$ $x * f$ $x)$

unfolding *pmf-set-integral-def*

by (*subst integral-measure-pmf*[of A]) (*auto simp*: *indicator-def mult-ac split*: *if-splits*)

lemma *pmf-set-integral-code* [code]:

pmf-set-integral p f (*set* xs) = $(\sum x \in \text{set } xs. \text{pmf } p$ $x * f$ $x)$

by (*rule pmf-set-integral-code-alt-finite*) *simp-all*

lemma *pmf-prob-code-alt-finite*:

finite $A \implies \text{pmf-prob } p$ $A = (\sum x \in A. \text{pmf } p$ $x)$

by (*simp* *add*: *pmf-prob-pmf-set-integral pmf-set-integral-code-alt-finite*)

lemma *pmf-prob-code* [code]:

pmf-prob p (*set* xs) = $(\sum x \in \text{set } xs. \text{pmf } p$ $x)$

pmf-prob p (*List.coset* xs) = $1 - (\sum x \in \text{set } xs. \text{pmf } p$ $x)$

by (*simp-all* *add*: *pmf-prob-code-alt-finite pmf-prob-compl*)

lemma *pmf-prob-code-unfold* [code-abbrev]: *pmf-prob* $p = \text{measure-pmf.prob } p$

by (*intro ext*) (*simp* *add*: *pmf-prob-def*)

lemma *pmf-integral-code-unfold* [code-abbrev]: *pmf-integral* $p = \text{measure-pmf.expectation}$

p

by (*intro ext*) (*simp* *add*: *pmf-integral-def*)

definition *pmf-of-alist* $xs = \text{embed-pmf } (\lambda x. \text{case map-of } xs$ x of *Some* $p \Rightarrow p$ | *None* $\Rightarrow 0$)

lemma *pmf-of-mapping-Mapping* [code-post]:
pmf-of-mapping (*Mapping xs*) = *pmf-of-alist xs*
unfolding *pmf-of-mapping-def Mapping.lookup-default-def* [abs-def] *pmf-of-alist-def*
by *transfer simp-all*

instantiation *pmf* :: (*equal*) *equal*
begin

definition *equal-pmf* *p q* = (*mapping-of-pmf p* = *mapping-of-pmf (q :: 'a pmf)*)

instance *by standard* (*simp add: equal-pmf-def*)
end

definition *single* :: 'a ⇒ 'a *multiset* **where**
single s = {#s#}

instantiation *pmf* :: (*random*) *random*
begin

context
includes *state-combinator-syntax term-syntax*
begin

definition
pmfify :: ('b::typerep *multiset* × (*unit* ⇒ *Code-Evaluation.term*)) ⇒
' b × (*unit* ⇒ *Code-Evaluation.term*) ⇒
' b *pmf* × (*unit* ⇒ *Code-Evaluation.term*) **where**
[*code-unfold*]: *pmfify A x* =
Code-Evaluation.valtermify pmf-of-multiset {·}
(*Code-Evaluation.valtermify* (+) {·} *A* {·})
(*Code-Evaluation.valtermify single* {·} *x*)

definition
Quickcheck-Random.random i =
Quickcheck-Random.random i ◦→ (λ*A*.
Quickcheck-Random.random i ◦→ (λ*x*. *Pair (pmfify A x)*))

instance ..

end

end

instantiation *pmf* :: (*full-exhaustive*) *full-exhaustive*
begin

definition *full-exhaustive-pmf* :: ('a *pmf* × (*unit* ⇒ *term*)) ⇒ (*bool* × *term list*)
option) ⇒ *natural* ⇒ (*bool* × *term list*) *option*

where

```
full-exhaustive-pmf f i =
  Quickcheck-Exhaustive.full-exhaustive (λA.
    Quickcheck-Exhaustive.full-exhaustive (λx. f (pmfify A x)) i) i
```

instance ..

end

end

22 Finite Maps

theory *Fin-Map*

imports *HOL-Analysis.Finite-Product-Measure* *HOL-Library.Finite-Map*
begin

The *fmap* type can be instantiated to *polish-space*, needed for the proof of projective limit. *extensional* functions are used for the representation in order to stay close to the developments of (finite) products Pi_E and their sigma-algebra Pi_M .

type-notation *fmap* ((- \Rightarrow_F /-) [22, 21] 21)

unbundle *fmap.lifting*

22.1 Domain and Application

lift-definition *domain*::('i \Rightarrow_F 'a) \Rightarrow 'i set is *dom* .

lemma *finite-domain*[*simp*, *intro*]: *finite* (*domain* *P*)
by *transfer simp*

lift-definition *proj* :: ('i \Rightarrow_F 'a) \Rightarrow 'i \Rightarrow 'a (('((-)')_F [0] 1000) is
 $\lambda f x$. if $x \in \text{dom } f$ then the ($f x$) else *undefined* .

declare [[*coercion proj*]]

lemma *extensional-proj*[*simp*, *intro*]: (*P*)_F \in *extensional* (*domain* *P*)
by *transfer (auto simp: extensional-def)*

lemma *proj-undefined*[*simp*, *intro*]: $i \notin \text{domain } P \Longrightarrow P i = \text{undefined}$
using *extensional-proj*[*of P*] **unfolding** *extensional-def* **by** *auto*

lemma *finmap-eq-iff*: $P = Q \longleftrightarrow (\text{domain } P = \text{domain } Q \wedge (\forall i \in \text{domain } P. P i = Q i))$
apply *transfer*
apply (*safe intro!*: *ext*)
subgoal for $P Q x$

by (cases $x \in \text{dom } P$; cases $P x$) (auto dest!: bspec[where $x=x$])
done

22.2 Constructor of Finite Maps

lift-definition *finmap-of*:: $'i \text{ set} \Rightarrow ('i \Rightarrow 'a) \Rightarrow ('i \Rightarrow_F 'a)$ is
 $\lambda I f x. \text{if } x \in I \wedge \text{finite } I \text{ then Some } (f x) \text{ else None}$
by (*simp add: dom-def*)

lemma *proj-finmap-of*[*simp*]:
assumes *finite inds*
shows $(\text{finmap-of inds } f)_F = \text{restrict } f \text{ inds}$
using *assms*
by *transfer force*

lemma *domain-finmap-of*[*simp*]:
assumes *finite inds*
shows $\text{domain } (\text{finmap-of inds } f) = \text{inds}$
using *assms*
by *transfer (auto split: if-splits)*

lemma *finmap-of-eq-iff*[*simp*]:
assumes *finite i finite j*
shows $\text{finmap-of } i \text{ } m = \text{finmap-of } j \text{ } n \iff i = j \wedge (\forall k \in i. m k = n k)$
using *assms* by (*auto simp: finmap-eq-iff*)

lemma *finmap-of-inj-on-extensional-finite*:

assumes *finite K*
assumes $S \subseteq \text{extensional } K$
shows *inj-on* (*finmap-of* K) S

proof (*rule inj-onI*)

fix $x y :: 'a \Rightarrow 'b$

assume $\text{finmap-of } K x = \text{finmap-of } K y$

hence $(\text{finmap-of } K x)_F = (\text{finmap-of } K y)_F$ by *simp*

moreover

assume $x \in S \ y \in S$ hence $x \in \text{extensional } K \ y \in \text{extensional } K$ using *assms*

by *auto*

ultimately

show $x = y$ using *assms* by (*simp add: extensional-restrict*)

qed

22.3 Product set of Finite Maps

This is Pi for Finite Maps, most of this is copied

definition $Pi' :: 'i \text{ set} \Rightarrow ('i \Rightarrow 'a \text{ set}) \Rightarrow ('i \Rightarrow_F 'a) \text{ set}$ **where**
 $Pi' I A = \{ P. \text{domain } P = I \wedge (\forall i. i \in I \longrightarrow (P)_F i \in A i) \}$

syntax

$-Pi' :: [pttrn, 'a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Rightarrow 'b) \text{ set} \ ((\exists \Pi'' \in \cdot / \cdot) \ 10)$

translations

$\Pi' x \in A. B == \text{CONST } Pi' A (\lambda x. B)$

22.3.1 Basic Properties of Pi'

lemma $Pi'-I[\text{intro}]$: $\text{domain } f = A \implies (\bigwedge x. x \in A \implies f x \in B x) \implies f \in Pi' A B$

by (*simp add: Pi'-def*)

lemma $Pi'-I[\text{simp}]$: $\text{domain } f = A \implies (\bigwedge x. x \in A \longrightarrow f x \in B x) \implies f \in Pi' A B$

by (*simp add: Pi'-def*)

lemma $Pi'-mem$: $f \in Pi' A B \implies x \in A \implies f x \in B x$

by (*simp add: Pi'-def*)

lemma $Pi'-iff$: $f \in Pi' I X \iff \text{domain } f = I \wedge (\forall i \in I. f i \in X i)$

unfolding $Pi'-def$ **by** *auto*

lemma $Pi'E$ [*elim*]:

$f \in Pi' A B \implies (f x \in B x \implies \text{domain } f = A \implies Q) \implies (x \notin A \implies Q) \implies Q$

by (*auto simp: Pi'-def*)

lemma *in-Pi'-cong*:

$\text{domain } f = \text{domain } g \implies (\bigwedge w. w \in A \implies f w = g w) \implies f \in Pi' A B \iff g \in Pi' A B$

by (*auto simp: Pi'-def*)

lemma $Pi'-eq\text{-empty}[\text{simp}]$:

assumes *finite A* **shows** $(Pi' A B) = \{\} \iff (\exists x \in A. B x = \{\})$

using *assms*

apply (*simp add: Pi'-def, auto*)

apply (*drule-tac x = finmap-of A (\lambda u. SOME y. y \in B u) in spec, auto*)

apply (*cut-tac P = %y. y \in B i in some-eq-ex, auto*)

done

lemma $Pi'-mono$: $(\bigwedge x. x \in A \implies B x \subseteq C x) \implies Pi' A B \subseteq Pi' A C$

by (*auto simp: Pi'-def*)

lemma $Pi-Pi'$: $\text{finite } A \implies (Pi_E A B) = \text{proj } ' Pi' A B$

apply (*auto simp: Pi'-def Pi-def extensional-def*)

apply (*rule-tac x = finmap-of A (restrict x A) in image-eqI*)

apply *auto*

done

22.4 Topological Space of Finite Maps

instantiation $fmap :: (\text{type}, \text{topological-space}) \text{topological-space}$
begin

definition *open-fmap* :: ('a \Rightarrow_F 'b) set \Rightarrow bool **where**
 [code del]: *open-fmap* = generate-topology {Pi' a b | a b. $\forall i \in a$. open (b i)}

lemma *open-Pi'I*: ($\bigwedge i. i \in I \Rightarrow$ open (A i)) \Rightarrow open (Pi' I A)
 by (auto intro: generate-topology.Basis simp: open-fmap-def)

instance using topological-space-generate-topology
 by intro-classes (auto simp: open-fmap-def class.topological-space-def)

end

lemma *open-restricted-space*:

shows open {m. P (domain m)}

proof –

have {m. P (domain m)} = ($\bigcup i \in \text{Collect } P. \{m. \text{domain } m = i\}$) **by** auto
 also have open ...

proof (rule, safe, cases)

fix i::'a set

assume finite i

hence {m. domain m = i} = Pi' i (λ -. UNIV) **by** (auto simp: Pi'-def)

also have open ... **by** (auto intro: open-Pi'I simp: finite i)

finally show open {m. domain m = i} .

next

fix i::'a set

assume \neg finite i **hence** {m. domain m = i} = {} **by** auto

also have open ... **by** simp

finally show open {m. domain m = i} .

qed

finally show ?thesis .

qed

lemma *closed-restricted-space*:

shows closed {m. P (domain m)}

using open-restricted-space[of $\lambda x. \neg P x$]

unfolding closed-def **by** (rule back-subst) auto

lemma *tendsto-proj*: (($\lambda x. x \longrightarrow a$) F \Rightarrow) (($\lambda x. (x)_F i \longrightarrow (a)_F i$) F

unfolding tendsto-def

proof safe

fix S::'b set

let ?S = Pi' (domain a) ($\lambda x. \text{if } x = i \text{ then } S \text{ else UNIV}$)

assume open S **hence** open ?S **by** (auto intro!: open-Pi'I)

moreover assume $\forall S. \text{open } S \longrightarrow a \in S \longrightarrow \text{eventually } (\lambda x. x \in S) F a i \in S$

ultimately have eventually ($\lambda x. x \in ?S$) F **by** auto

thus eventually ($\lambda x. (x)_F i \in S$) F

by eventually-elim (insert $\langle a i \in S \rangle$, force simp: Pi'-iff split: if-split-asm)

qed

lemma *continuous-proj*:

shows *continuous-on s* $(\lambda x. (x)_F i)$

unfolding *continuous-on-def* **by** (*safe intro!*: *tendsto-proj tendsto-ident-at*)

instance *fmap* :: (*type*, *first-countable-topology*) *first-countable-topology*

proof

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

have $\forall i. \exists A. \text{countable } A \wedge (\forall a \in A. x i \in a) \wedge (\forall a \in A. \text{open } a) \wedge$

$(\forall S. \text{open } S \wedge x i \in S \longrightarrow (\exists a \in A. a \subseteq S)) \wedge (\forall a b. a \in A \longrightarrow b \in A \longrightarrow a \cap b \in A)$ (**is** $\forall i. ?th i$)

proof

fix i **from** *first-countable-basis-Int-stableE*[*of x i*]

obtain A **where**

countable A

$\bigwedge C. C \in A \Longrightarrow (x)_F i \in C$

$\bigwedge C. C \in A \Longrightarrow \text{open } C$

$\bigwedge S. \text{open } S \Longrightarrow (x)_F i \in S \Longrightarrow \exists A \in A. A \subseteq S$

$\bigwedge C D. C \in A \Longrightarrow D \in A \Longrightarrow C \cap D \in A$

by *auto*

thus $?th i$ **by** (*intro exI*[**where** $x=A$]) *simp*

qed

then obtain A

where $A: \forall i. \text{countable } (A i) \wedge \text{Ball } (A i) ((\in) ((x)_F i)) \wedge \text{Ball } (A i) \text{open} \wedge$

$(\forall S. \text{open } S \wedge (x)_F i \in S \longrightarrow (\exists a \in A i. a \subseteq S)) \wedge (\forall a b. a \in A i \longrightarrow b \in A i \longrightarrow a \cap b \in A i)$

by (*auto simp: choice-iff*)

hence *open-sub*: $\bigwedge i S. i \in \text{domain } x \Longrightarrow \text{open } (S i) \Longrightarrow x i \in (S i) \Longrightarrow (\exists a \in A i. a \subseteq (S i))$ **by** *auto*

have *A-notempty*: $\bigwedge i. i \in \text{domain } x \Longrightarrow A i \neq \{\}$ **using** *open-sub*[*of -* $\lambda-. UNIV$] **by** *auto*

let $?A = (\lambda f. Pi' (\text{domain } x) f) ' (Pi_E (\text{domain } x) A)$

show $\exists A:: \text{nat} \Rightarrow ('a \Rightarrow_F 'b) \text{ set}. (\forall i. x \in (A i) \wedge \text{open } (A i)) \wedge (\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. A i \subseteq S))$

proof (*rule first-countableI*[*of ?A*], *safe*)

show *countable ?A* **using** A **by** (*simp add: countable-PiE*)

next

fix $S:: ('a \Rightarrow_F 'b) \text{ set}$ **assume** *open S* $x \in S$

thus $\exists a \in ?A. a \subseteq S$ **unfolding** *open-fmap-def*

proof (*induct rule: generate-topology.induct*)

case *UNIV* **thus** $?case$ **by** (*auto simp add: ex-in-conv PiE-eq-empty-iff A-notempty*)

next

case (*Int a b*)

then obtain $f g$ **where**

$f \in Pi_E (\text{domain } x) A$ $Pi' (\text{domain } x) f \subseteq a$ $g \in Pi_E (\text{domain } x) A$ $Pi' (\text{domain } x) g \subseteq b$

by *auto*

thus $?case$ **using** A

by (*auto simp: Pi'-iff PiE-iff extensional-def Int-stable-def*)

```

      intro!: beXI[where x= $\lambda i. f i \cap g i$ ]
next
  case (UN B)
  then obtain b where x  $\in$  b b  $\in$  B by auto
  hence  $\exists a \in ?A. a \subseteq b$  using UN by simp
  thus ?case using  $\langle b \in B \rangle$  by (metis Sup-upper2)
next
  case (Basis s)
  then obtain a b where xs: x  $\in$  Pi' a b s = Pi' a b  $\wedge i. i \in a \implies \text{open } (b i)$ 
by auto
  have  $\forall i. \exists a. (i \in \text{domain } x \wedge \text{open } (b i) \wedge (x)_F i \in b i) \longrightarrow (a \in A \ i \wedge a \subseteq b i)$ 
    using open-sub[of - b] by auto
  then obtain b'
  where  $\bigwedge i. i \in \text{domain } x \implies \text{open } (b i) \implies (x)_F i \in b i \implies (b' i \in A \ i \wedge b' i \subseteq b i)$ 
    unfolding choice-iff by auto
  with xs have  $\bigwedge i. i \in a \implies (b' i \in A \ i \wedge b' i \subseteq b i) \ Pi' a \ b' \subseteq \Pi' a \ b$ 
    by (auto simp: Pi'-iff intro!: Pi'-mono)
  thus ?case using xs
    by (intro beXI[where x=Pi' a b])
      (auto simp: Pi'-iff intro!: image-eqI[where x=restrict b' (domain x)])
qed
qed (insert A, auto simp: PiE-iff intro!: open-Pi'I)
qed

```

22.5 Metric Space of Finite Maps

```

instantiation fmap :: (type, metric-space) dist
begin

```

definition *dist-fmap* where

$\text{dist } P \ Q = \text{Max } (\text{range } (\lambda i. \text{dist } ((P)_F i) ((Q)_F i))) + (\text{if } \text{domain } P = \text{domain } Q \text{ then } 0 \text{ else } 1)$

```

instance ..
end

```

```

instantiation fmap :: (type, metric-space) uniformity-dist
begin

```

definition [code del]:

$(\text{uniformity} :: (('a, 'b) \text{fmap} \times ('a \Rightarrow_F 'b)) \text{filter}) =$
 $(\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x \ y < e\})$

instance

```

  by standard (rule uniformity-fmap-def)
end

```

declare *uniformity-Abort*[**where** 'a=(*'a* \Rightarrow_F 'b::*metric-space*), *code*]

instantiation *fmap* :: (*type*, *metric-space*) *metric-space*
begin

lemma *finite-proj-image'*: $x \notin \text{domain } P \implies \text{finite } ((P)_F \text{ ' } S)$
by (*rule finite-subset*[*of - proj P ' (domain P \cap S \cup {x})*]) *auto*

lemma *finite-proj-image*: *finite* $((P)_F \text{ ' } S)$
by (*cases* $\exists x. x \notin \text{domain } P$) (*auto intro: finite-proj-image' finite-subset*[**where** *B=domain P*])

lemma *finite-proj-diag*: *finite* $((\lambda i. d ((P)_F i) ((Q)_F i)) \text{ ' } S)$

proof –

have $(\lambda i. d ((P)_F i) ((Q)_F i)) \text{ ' } S = (\lambda(i, j). d i j) \text{ ' } ((\lambda i. ((P)_F i, (Q)_F i)) \text{ ' } S)$ **by** *auto*

moreover have $((\lambda i. ((P)_F i, (Q)_F i)) \text{ ' } S) \subseteq (\lambda i. (P)_F i) \text{ ' } S \times (\lambda i. (Q)_F i) \text{ ' } S$ **by** *auto*

moreover have *finite* ... **using** *finite-proj-image*[*of P S*] *finite-proj-image*[*of Q S*]

by (*intro finite-cartesian-product simp-all*)

ultimately show *?thesis* **by** (*simp add: finite-subset*)

qed

lemma *dist-le-1-imp-domain-eq*:

shows $\text{dist } P \ Q < 1 \implies \text{domain } P = \text{domain } Q$

by (*simp add: dist-fmap-def finite-proj-diag split: if-split-asm*)

lemma *dist-proj*:

shows $\text{dist } ((x)_F i) ((y)_F i) \leq \text{dist } x \ y$

proof –

have $\text{dist } (x \ i) (y \ i) \leq \text{Max } (\text{range } (\lambda i. \text{dist } (x \ i) (y \ i)))$

by (*simp add: Max-ge-iff finite-proj-diag*)

also have ... $\leq \text{dist } x \ y$ **by** (*simp add: dist-fmap-def*)

finally show *?thesis* .

qed

lemma *dist-fmap-lessI*:

assumes $\text{domain } P = \text{domain } Q$

assumes $0 < e$

assumes $\bigwedge i. i \in \text{domain } P \implies \text{dist } (P \ i) (Q \ i) < e$

shows $\text{dist } P \ Q < e$

proof –

have $\text{dist } P \ Q = \text{Max } (\text{range } (\lambda i. \text{dist } (P \ i) (Q \ i)))$

using *assms* **by** (*simp add: dist-fmap-def finite-proj-diag*)

also have ... $< e$

proof (*subst Max-less-iff, safe*)

fix *i*

show $\text{dist } ((P)_F i) ((Q)_F i) < e$ **using** *assms*


```

    by (cases i ∈ domain P) simp-all
  qed (simp add: finite-proj-diag)
  finally show ?thesis .
qed

instance
proof
  fix S::('a ⇒F 'b) set
  have *: open S = (∀ x∈S. ∃ e>0. ∀ y. dist y x < e → y ∈ S) (is - = ?od)
  proof
    assume open S
    thus ?od
      unfolding open-fmap-def
    proof (induct rule: generate-topology.induct)
      case UNIV thus ?case by (auto intro: zero-less-one)
    next
      case (Int a b)
      show ?case
      proof safe
        fix x assume x: x ∈ a x ∈ b
        with Int x obtain e1 e2 where
          e1>0 ∀ y. dist y x < e1 → y ∈ a e2>0 ∀ y. dist y x < e2 → y ∈ b by
force
        thus ∃ e>0. ∀ y. dist y x < e → y ∈ a ∩ b
          by (auto intro!: exI[where x=min e1 e2])
      qed
    next
      case (UN K)
      show ?case
      proof safe
        fix x X assume x ∈ X and X: X ∈ K
        with UN obtain e where e>0 ∧ y. dist y x < e → y ∈ X by force
        with X show ∃ e>0. ∀ y. dist y x < e → y ∈ ⋃ K by auto
      qed
    next
      case (Basis s) then obtain a b where s: s = Pi' a b and b: ∧ i. i ∈ a ⇒
open (b i) by auto
      show ?case
      proof safe
        fix x assume x ∈ s
        hence [simp]: finite a and a-dom: a = domain x using s by (auto simp:
Pi'-iff)
        obtain es where es: ∀ i ∈ a. es i > 0 ∧ (∀ y. dist y (proj x i) < es i → y
∈ b i)
          using b ⟨x ∈ s⟩ by atomize-elim (intro bchoice, auto simp: open-dist s)
        hence in-b: ∧ i y. i ∈ a ⇒ dist y (proj x i) < es i ⇒ y ∈ b i by auto
        show ∃ e>0. ∀ y. dist y x < e → y ∈ s
          proof (cases, rule, safe)
            assume a ≠ {}

```

```

show  $0 < \min 1 (Min (es \ 'a))$  using  $es$  by  $(auto simp: \ 'a \neq \{\})$ 
fix  $y$  assume  $d: dist\ y\ x < \min 1 (Min (es \ 'a))$ 
show  $y \in s$  unfolding  $s$ 
proof
show  $domain\ y = a$  using  $d\ s\ \ 'a \neq \{\}$  by  $(auto simp: dist-le-1-imp-domain-eq\ a-dom)$ 
  fix  $i$  assume  $i: i \in a$ 
  hence  $dist\ ((y)_F\ i)\ ((x)_F\ i) < es\ i$  using  $d$ 
  by  $(auto simp: dist-fmap-def\ \ 'a \neq \{\})$  intro!  $le-less-trans[OF\ dist-proj]$ 
  with  $i$  show  $y\ i \in b$  by  $(rule\ in-b)$ 
qed
next
assume  $\neg a \neq \{\}$ 
thus  $\exists e > 0. \forall y. dist\ y\ x < e \longrightarrow y \in s$ 
  using  $s\ \langle x \in s \rangle$  by  $(auto simp: Pi'-def\ dist-le-1-imp-domain-eq\ intro!:\ exI[where\ x=1])$ 
qed
qed
qed
next
assume  $\forall x \in S. \exists e > 0. \forall y. dist\ y\ x < e \longrightarrow y \in S$ 
then obtain  $e$  where  $e-pos: \bigwedge x. x \in S \implies e\ x > 0$  and
   $e-in: \bigwedge x\ y. x \in S \implies dist\ y\ x < e\ x \implies y \in S$ 
unfolding  $bchoice-iff$ 
by  $auto$ 
have  $S-eq: S = \bigcup \{Pi'\ a\ b \mid a\ b. \exists x \in S. domain\ x = a \wedge b = (\lambda i. ball\ (x\ i)\ (e\ x))\}$ 
proof  $safe$ 
  fix  $x$  assume  $x \in S$ 
  thus  $x \in \bigcup \{Pi'\ a\ b \mid a\ b. \exists x \in S. domain\ x = a \wedge b = (\lambda i. ball\ (x\ i)\ (e\ x))\}$ 
    using  $e-pos$  by  $(auto\ intro!:\ exI[where\ x=Pi'\ (domain\ x)\ (\lambda i. ball\ (x\ i)\ (e\ x))])$ 
next
  fix  $x\ y$ 
  assume  $y \in S$ 
moreover
  assume  $x \in (\Pi' i \in domain\ y. ball\ (y\ i)\ (e\ y))$ 
  hence  $dist\ x\ y < e\ y$  using  $e-pos\ \langle y \in S \rangle$ 
  by  $(auto simp: dist-fmap-def\ Pi'-iff\ finite-proj-diag\ dist-commute)$ 
  ultimately show  $x \in S$  by  $(rule\ e-in)$ 
qed
also have  $open \dots$ 
  unfolding  $open-fmap-def$ 
  by  $(intro\ generate-topology.UN)\ (auto\ intro: generate-topology.Basis)$ 
finally show  $open\ S$  .
qed
show  $open\ S = (\forall x \in S. \forall_F (x', y)\ in\ uniformity. x' = x \longrightarrow y \in S)$ 
unfolding  $*\ eventually-uniformity-metric$ 
by  $(simp\ del: split-paired-All\ add: dist-fmap-def\ dist-commute\ eq-commute)$ 

```

```

next
  fix P Q :: 'a ⇒F 'b
  have Max-eq-iff:  $\bigwedge A m. \text{finite } A \implies A \neq \{\} \implies (\text{Max } A = m) = (m \in A \wedge (\forall a \in A. a \leq m))$ 
  by (auto intro: Max-in Max-eqI)
  show  $\text{dist } P \ Q = 0 \iff P = Q$ 
  by (auto simp: finmap-eq-iff dist-fmap-def Max-ge-iff finite-proj-diag Max-eq-iff
      add-nonneg-eq-0-iff
      intro!: Max-eqI image-eqI[where x=undefined])
next
  fix P Q R :: 'a ⇒F 'b
  let ?dists =  $\lambda P \ Q \ i. \text{dist } ((P)_F \ i) ((Q)_F \ i)$ 
  let ?dpq = ?dists P Q and ?dpr = ?dists P R and ?dqr = ?dists Q R
  let ?dom =  $\lambda P \ Q. (\text{if } \text{domain } P = \text{domain } Q \text{ then } 0 \text{ else } 1::\text{real})$ 
  have  $\text{dist } P \ Q = \text{Max } (\text{range } ?dpq) + ?dom \ P \ Q$ 
  by (simp add: dist-fmap-def)
  also obtain t where  $t \in \text{range } ?dpq \ t = \text{Max } (\text{range } ?dpq)$  by (simp add:
  finite-proj-diag)
  then obtain i where  $\text{Max } (\text{range } ?dpq) = ?dpq \ i$  by auto
  also have  $?dpq \ i \leq ?dpr \ i + ?dqr \ i$  by (rule dist-triangle2)
  also have  $?dpr \ i \leq \text{Max } (\text{range } ?dpr)$  by (simp add: finite-proj-diag)
  also have  $?dqr \ i \leq \text{Max } (\text{range } ?dqr)$  by (simp add: finite-proj-diag)
  also have  $?dom \ P \ Q \leq ?dom \ P \ R + ?dom \ Q \ R$  by simp
  finally show  $\text{dist } P \ Q \leq \text{dist } P \ R + \text{dist } Q \ R$  by (simp add: dist-fmap-def
  ac-simps)
qed

end

```

22.6 Complete Space of Finite Maps

lemma *tendsto-finmap*:

```

fixes f :: nat ⇒ ('i ⇒F ('a :: metric-space))
assumes ind-f:  $\bigwedge n. \text{domain } (f \ n) = \text{domain } g$ 
assumes proj-g:  $\bigwedge i. i \in \text{domain } g \implies (\lambda n. (f \ n) \ i) \longrightarrow g \ i$ 
shows  $f \longrightarrow g$ 
unfolding tendsto-iff

```

proof *safe*

```

  fix e :: real assume  $0 < e$ 
  let ?dists =  $\lambda x \ i. \text{dist } ((f \ x)_F \ i) ((g)_F \ i)$ 
  have eventually  $(\lambda x. \forall i \in \text{domain } g. ?dists \ x \ i < e)$  sequentially
  using finite-domain[of g] proj-g
  proof induct
    case (insert i G)
    with  $\langle 0 < e \rangle$  have eventually  $(\lambda x. ?dists \ x \ i < e)$  sequentially by (auto simp
  add: tendsto-iff)
    moreover
    from insert have eventually  $(\lambda x. \forall i \in G. \text{dist } ((f \ x)_F \ i) ((g)_F \ i) < e)$  sequentially
  by simp

```

```

    ultimately show ?case by eventually-elim auto
  qed simp
  thus eventually ( $\lambda x. \text{dist } (f x) g < e$ ) sequentially
    by eventually-elim (auto simp add: dist-fmap-def finite-proj-diag ind-f (0 < e))
  qed

instance fmap :: (type, complete-space) complete-space
proof
  fix P::nat  $\Rightarrow$  'a  $\Rightarrow_F$  'b
  assume Cauchy P
  then obtain Nd where Nd:  $\bigwedge n. n \geq Nd \implies \text{dist } (P n) (P Nd) < 1$ 
    by (force simp: Cauchy-altdef2)
  define d where d = domain (P Nd)
  with Nd have dim:  $\bigwedge n. n \geq Nd \implies \text{domain } (P n) = d$  using dist-le-1-imp-domain-eq
  by auto
  have [simp]: finite d unfolding d-def by simp
  define p where p i n = P n i for i n
  define q where q i = lim (p i) for i
  define Q where Q = finmap-of d q
  have q:  $\bigwedge i. i \in d \implies q i = Q i$  by (auto simp add: Q-def Abs-fmap-inverse)
  {
    fix i assume i  $\in$  d
    have Cauchy (p i) unfolding Cauchy-altdef2 p-def
    proof safe
      fix e::real assume 0 < e
      with (Cauchy P) obtain N where N:  $\bigwedge n. n \geq N \implies \text{dist } (P n) (P N) < \min e 1$ 
      by (force simp: Cauchy-altdef2 min-def)
      hence  $\bigwedge n. n \geq N \implies \text{domain } (P n) = \text{domain } (P N)$  using dist-le-1-imp-domain-eq
    by auto
    with dim have dim:  $\bigwedge n. n \geq N \implies \text{domain } (P n) = d$  by (metis nat-le-linear)
    show  $\exists N. \forall n \geq N. \text{dist } ((P n) i) ((P N) i) < e$ 
    proof (safe intro!: exI[where x=N])
      fix n assume N  $\leq$  n have N  $\leq$  N by simp
      have  $\text{dist } ((P n) i) ((P N) i) \leq \text{dist } (P n) (P N)$ 
        using dim[OF (N  $\leq$  n)] dim[OF (N  $\leq$  N)] (i  $\in$  d)
        by (auto intro!: dist-proj)
      also have ... < e using N[OF (N  $\leq$  n)] by simp
      finally show  $\text{dist } ((P n) i) ((P N) i) < e$  .
    qed
  }
  qed
  hence convergent (p i) by (metis Cauchy-convergent-iff)
  hence p i  $\longrightarrow$  q i unfolding q-def convergent-def by (metis limI)
} note p = this
have P  $\longrightarrow$  Q
proof (rule metric-LIMSEQ-I)
  fix e::real assume 0 < e
  have  $\exists ni. \forall i \in d. \forall n \geq ni. \text{dist } (p i n) (q i) < e$ 
  proof (safe intro!: bchoice)

```

```

fix  $i$  assume  $i \in d$ 
from  $p[OF \langle i \in d \rangle, THEN \text{metric-LIMSEQ-D}, OF \langle 0 < e \rangle]$ 
show  $\exists no. \forall n \geq no. dist (p \ i \ n) (q \ i) < e .$ 
qed
then obtain  $ni$  where  $ni: \forall i \in d. \forall n \geq ni \ i. dist (p \ i \ n) (q \ i) < e ..$ 
define  $N$  where  $N = max \ Nd (Max (ni \ ' \ d))$ 
show  $\exists N. \forall n \geq N. dist (P \ n) Q < e$ 
proof (safe intro!:  $exI[\text{where } x=N]$ )
  fix  $n$  assume  $N \leq n$ 
  hence  $dom: domain (P \ n) = d \ domain \ Q = d \ domain (P \ n) = domain \ Q$ 
  using  $dim$  by (simp-all add: N-def Q-def dim-def Abs-fmap-inverse)
  show  $dist (P \ n) Q < e$ 
  proof (rule dist-finmap-lessI[OF dom(3) <0 < e>])
    fix  $i$ 
    assume  $i \in domain (P \ n)$ 
    hence  $ni \ i \leq Max (ni \ ' \ d)$  using  $dom$  by simp
    also have  $\dots \leq N$  by (simp add: N-def)
    finally show  $dist ((P \ n)_F \ i) ((Q)_F \ i) < e$  using  $ni \ \langle i \in domain (P \ n) \rangle \ \langle N$ 
 $\leq n \rangle \ dom$ 
    by (auto simp: p-def q N-def less-imp-le)
  qed
qed
thus convergent  $P$  by (auto simp: convergent-def)
qed

```

22.7 Second Countable Space of Finite Maps

instantiation $fmap :: (countable, second-countable-topology) \ second-countable-topology$
begin

definition $basis-proj :: 'b \ set \ set$
where $basis-proj = (SOME \ B. countable \ B \ \wedge \ topological-basis \ B)$

lemma $countable-basis-proj: countable \ basis-proj$ **and** $basis-proj: topological-basis \ basis-proj$

unfolding $basis-proj-def$ **by** (*intro is-basis countable-basis*)**+**

definition $basis-finmap :: ('a \Rightarrow_F \ 'b) \ set \ set$
where $basis-finmap = \{Pi' \ I \ S \mid I \ S. finite \ I \ \wedge \ (\forall i \in I. S \ i \in basis-proj)\}$

lemma $in-basis-finmapI:$
assumes $finite \ I$ **assumes** $\bigwedge i. i \in I \implies S \ i \in basis-proj$
shows $Pi' \ I \ S \in basis-finmap$
using $assms$ **unfolding** $basis-finmap-def$ **by** *auto*

lemma $basis-finmap-eq:$
assumes $basis-proj \neq \{\}$
shows $basis-finmap = (\lambda f. Pi' (domain \ f) (\lambda i. from-nat-into \ basis-proj ((f)_F$

```

i))) ‘
  (UNIV::('a  $\Rightarrow_F$  nat) set) (is - = ?f ‘ -)
  unfolding basis-finmap-def
proof safe
  fix I::'a set and S::'a  $\Rightarrow$  'b set
  assume finite I  $\forall i \in I. S i \in$  basis-proj
  hence  $Pi' I S = ?f$  (finmap-of I ( $\lambda x. to-nat-on$  basis-proj (S x)))
    by (force simp: Pi'-def countable-basis-proj)
  thus  $Pi' I S \in$  range ?f by simp
next
  fix x and f::'a  $\Rightarrow_F$  nat
  show  $\exists I S. (\Pi' i \in domain f. from-nat-into$  basis-proj ((f)F i)) =  $Pi' I S \wedge$ 
    finite I  $\wedge (\forall i \in I. S i \in$  basis-proj)
    using assms by (auto intro: from-nat-into)
qed

lemma basis-finmap-eq-empty: basis-proj = {}  $\implies$  basis-finmap = {Pi' {}} unde-
fined}
  by (auto simp: Pi'-iff basis-finmap-def)

lemma countable-basis-finmap: countable basis-finmap
  by (cases basis-proj = {}) (auto simp: basis-finmap-eq basis-finmap-eq-empty)

lemma finmap-topological-basis:
  topological-basis basis-finmap
proof (subst topological-basis-iff, safe)
  fix B' assume B'  $\in$  basis-finmap
  thus open B'
    by (auto intro!: open-Pi'I topological-basis-open[OF basis-proj]
      simp: topological-basis-def basis-finmap-def Let-def)
next
  fix O'::('a  $\Rightarrow_F$  'b) set and x
  assume O': open O' x  $\in$  O'
  then obtain a where a:
    x  $\in$  Pi' (domain x) a Pi' (domain x) a  $\subseteq$  O'  $\wedge i. i \in domain x \implies$  open (a i)
  unfolding open-fmap-def
  proof (atomize-elim, induct rule: generate-topology.induct)
    case (Int a b)
    let ?p= $\lambda a f. x \in$  Pi' (domain x) f  $\wedge$  Pi' (domain x) f  $\subseteq$  a  $\wedge (\forall i. i \in domain$ 
x  $\longrightarrow$  open (f i))
    from Int obtain f g where ?p a f ?p b g by auto
    thus ?case by (force intro!: exI[where x= $\lambda i. f i \cap g i$ ] simp: Pi'-def)
  next
    case (UN k)
    then obtain kk a where x  $\in$  kk kk  $\in$  k x  $\in$  Pi' (domain x) a Pi' (domain x)
a  $\subseteq$  kk
     $\wedge i. i \in domain x \implies$  open (a i)
    by force
    thus ?case by blast

```

qed (*auto simp: Pi'-def*)
have $\exists B$.
 ($\forall i \in \text{domain } x. x \ i \in B \ i \wedge B \ i \subseteq a \ i \wedge B \ i \in \text{basis-proj}$)
proof (*rule bchoice, safe*)
fix i **assume** $i \in \text{domain } x$
hence *open* ($a \ i$) $x \ i \in a \ i$ **using** a **by** *auto*
from *topological-basisE*[*OF basis-proj this*] **obtain** b'
where $b' \in \text{basis-proj } (x)_F \ i \in b' \ b' \subseteq a \ i$
by *blast*
thus $\exists y. x \ i \in y \wedge y \subseteq a \ i \wedge y \in \text{basis-proj}$ **by** *auto*
qed
then obtain B **where** $B: \forall i \in \text{domain } x. (x)_F \ i \in B \ i \wedge B \ i \subseteq a \ i \wedge B \ i \in$
basis-proj
by *auto*
define B' **where** $B' = \text{Pi}' (\text{domain } x) (\lambda i. (B \ i)::'b \ \text{set})$
have $B' \subseteq \text{Pi}' (\text{domain } x) a$ **using** B **by** (*auto intro!: Pi'-mono simp: B'-def*)
also note $\langle \dots \subseteq O' \rangle$
finally show $\exists B' \in \text{basis-finmap}. x \in B' \wedge B' \subseteq O'$ **using** B
by (*auto intro!: bexI[where x=B'] Pi'-mono in-basis-finmapI simp: B'-def*)
qed

lemma *range-enum-basis-finmap-imp-open*:
assumes $x \in \text{basis-finmap}$
shows *open* x
using *finmap-topological-basis assms* **by** (*auto simp: topological-basis-def*)

instance proof qed (*blast intro: finmap-topological-basis countable-basis-finmap*
topological-basis-imp-subbasis)

end

22.8 Polish Space of Finite Maps

instance *fmap* :: (*countable, polish-space*) *polish-space* **proof** **qed**

22.9 Product Measurable Space of Finite Maps

definition $\text{PiF } I \ M \equiv$

sigma ($\bigcup J \in I. (\text{Pi}' j \in J. \text{space } (M \ j))$) $\{(\text{Pi}' j \in J. X \ j) \mid X \ J. J \in I \wedge X \in (\text{Pi}$
 $j \in J. \text{sets } (M \ j))\}$

abbreviation

$\text{Pi}_F \ I \ M \equiv \text{PiF } I \ M$

syntax

$\text{-PiF} :: \text{pttrn} \Rightarrow 'i \ \text{set} \Rightarrow 'a \ \text{measure} \Rightarrow ('i \Rightarrow 'a) \ \text{measure} \ ((\exists \text{Pi}_F \ \text{-}\in \cdot / \cdot) \ 10)$

translations

$\text{Pi}_F \ x \in I. M == \text{CONST } \text{PiF } I \ (\%x. M)$

lemma *PiF-gen-subset*: $\{(\Pi' j \in J. X j) \mid X J. J \in I \wedge X \in (\Pi j \in J. sets (M j))\}$
 \subseteq
Pow $(\bigcup J \in I. (\Pi' j \in J. space (M j)))$
by (*auto simp*: *Pi'-def*) (*blast dest*: *sets.sets-into-space*)

lemma *space-PiF*: $space (PiF I M) = (\bigcup J \in I. (\Pi' j \in J. space (M j)))$
unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule space-measure-of*)

lemma *sets-PiF*:
 $sets (PiF I M) = sigma\text{-sets} (\bigcup J \in I. (\Pi' j \in J. space (M j)))$
 $\{(\Pi' j \in J. X j) \mid X J. J \in I \wedge X \in (\Pi j \in J. sets (M j))\}$
unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule sets-measure-of*)

lemma *sets-PiF-singleton*:
 $sets (PiF \{I\} M) = sigma\text{-sets} (\Pi' j \in I. space (M j))$
 $\{(\Pi' j \in I. X j) \mid X. X \in (\Pi j \in I. sets (M j))\}$
unfolding *sets-PiF* **by** *simp*

lemma *in-sets-PiFI*:
assumes $X = (Pi' J S) J \in I \wedge i. i \in J \implies S i \in sets (M i)$
shows $X \in sets (PiF I M)$
unfolding *sets-PiF*
using *assms* **by** *blast*

lemma *product-in-sets-PiFI*:
assumes $J \in I \wedge i. i \in J \implies S i \in sets (M i)$
shows $(Pi' J S) \in sets (PiF I M)$
unfolding *sets-PiF*
using *assms* **by** *blast*

lemma *singleton-space-subset-in-sets*:
fixes J
assumes $J \in I$
assumes *finite* J
shows $space (PiF \{J\} M) \in sets (PiF I M)$
using *assms*
by (*intro in-sets-PiFI*[**where** $J=J$ **and** $S=\lambda i. space (M i)$])
(*auto simp*: *product-def space-PiF*)

lemma *singleton-subspace-set-in-sets*:
assumes $A: A \in sets (PiF \{J\} M)$
assumes *finite* J
assumes $J \in I$
shows $A \in sets (PiF I M)$
using A [*unfolded sets-PiF*]
apply (*induct* A)
unfolding *sets-PiF*[*symmetric*] **unfolding** *space-PiF*[*symmetric*]
using *assms*
by (*auto intro*: *in-sets-PiFI intro!*: *singleton-space-subset-in-sets*)

lemma *finite-measurable-singletonI*:

assumes *finite I*
assumes $\bigwedge J. J \in I \implies \text{finite } J$
assumes *MN*: $\bigwedge J. J \in I \implies A \in \text{measurable } (PiF \{J\} M) N$
shows $A \in \text{measurable } (PiF I M) N$
unfolding *measurable-def*
proof *safe*
fix *y* **assume** $y \in \text{sets } N$
have $A -' y \cap \text{space } (PiF I M) = (\bigcup J \in I. A -' y \cap \text{space } (PiF \{J\} M))$
by (*auto simp: space-PiF*)
also have $\dots \in \text{sets } (PiF I M)$
proof (*rule sets.finite-UN*)
show *finite I* **by fact**
fix *J* **assume** $J \in I$
with *assms* **have** *finite J* **by simp**
show $A -' y \cap \text{space } (PiF \{J\} M) \in \text{sets } (PiF I M)$
by (*rule singleton-subspace-set-in-sets[OF measurable-sets[OF assms(3)]]*)
fact+
qed
finally show $A -' y \cap \text{space } (PiF I M) \in \text{sets } (PiF I M)$.
next
fix *x* **assume** $x \in \text{space } (PiF I M)$ **thus** $A x \in \text{space } N$
using *MN[of domain x]*
by (*auto simp: space-PiF measurable-space Pi'-def*)
qed

lemma *countable-finite-comprehension*:

fixes *f* :: 'a::countable set \Rightarrow -
assumes $\bigwedge s. P s \implies \text{finite } s$
assumes $\bigwedge s. P s \implies f s \in \text{sets } M$
shows $\bigcup \{f s \mid s. P s\} \in \text{sets } M$
proof -
have $\bigcup \{f s \mid s. P s\} = (\bigcup n::nat. \text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$
proof *safe*
fix *x* *X* *s* **assume** $x \in f s P s$
with *assms* **obtain** *l* **where** $s = \text{set } l$ **using** *finite-list* **by blast**
with $*$ **show** $x \in (\bigcup n. \text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$ **using**
 $\langle P s \rangle$
by (*auto intro!: exI[where x=to-nat l]*)
next
fix *x* *n* **assume** $x \in (\text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$
thus $x \in \bigcup \{f s \mid s. P s\}$ **using** *assms* **by** (*auto simp: Let-def split: if-split-asm*)
qed
hence $\bigcup \{f s \mid s. P s\} = (\bigcup n. \text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$
by simp
also have $\dots \in \text{sets } M$ **using** *assms* **by** (*auto simp: Let-def*)
finally show *thesis* .

qed

lemma *space-subset-in-sets*:

fixes $J::'a::\text{countable set set}$

assumes $J \subseteq I$

assumes $\bigwedge j. j \in J \implies \text{finite } j$

shows $\text{space } (PiF J M) \in \text{sets } (PiF I M)$

proof –

have $\text{space } (PiF J M) = \bigcup \{\text{space } (PiF \{j\} M) \mid j. j \in J\}$

unfolding *space-PiF* **by** *blast*

also have $\dots \in \text{sets } (PiF I M)$ **using** *assms*

by (*intro countable-finite-comprehension*) (*auto simp: singleton-space-subset-in-sets*)

finally show *?thesis* .

qed

lemma *subspace-set-in-sets*:

fixes $J::'a::\text{countable set set}$

assumes $A \in \text{sets } (PiF J M)$

assumes $J \subseteq I$

assumes $\bigwedge j. j \in J \implies \text{finite } j$

shows $A \in \text{sets } (PiF I M)$

using $A[\text{unfolded sets-PiF}]$

apply (*induct A*)

unfolding *sets-PiF[symmetric]* **unfolding** *space-PiF[symmetric]*

using *assms*

by (*auto intro: in-sets-PiFI intro!: space-subset-in-sets*)

lemma *countable-measurable-PiFI*:

fixes $I::'a::\text{countable set set}$

assumes $MN: \bigwedge J. J \in I \implies \text{finite } J \implies A \in \text{measurable } (PiF \{J\} M) N$

shows $A \in \text{measurable } (PiF I M) N$

unfolding *measurable-def*

proof *safe*

fix y **assume** $y \in \text{sets } N$

have $A -' y = (\bigcup \{A -' y \cap \{x. \text{domain } x = J\} \mid J. \text{finite } J\})$ **by** *auto*

{ fix $x::'a \Rightarrow_F 'b$

from *finite-list[of domain x]* **obtain** xs **where** $\text{set } xs = \text{domain } x$ **by** *auto*

hence $\exists n. \text{domain } x = \text{set } (\text{from-nat } n)$

by (*intro exI[where x=to-nat xs]*) *auto* }

hence $A -' y \cap \text{space } (PiF I M) = (\bigcup n. A -' y \cap \text{space } (PiF (\{\text{set } (\text{from-nat } n)\} \cap I) M))$

by (*auto simp: space-PiF Pi'-def*)

also have $\dots \in \text{sets } (PiF I M)$

apply (*intro sets.Int sets.countable-nat-UN subsetI, safe*)

apply (*case-tac set (from-nat i) \in I*)

apply *simp-all*

apply (*rule singleton-subspace-set-in-sets[OF measurable-sets[OF MN]]*)

using *assms* $\langle y \in \text{sets } N \rangle$

apply (*auto simp: space-PiF*)

done
finally show $A - ' y \cap \text{space } (PiF I M) \in \text{sets } (PiF I M) .$
next
fix x **assume** $x \in \text{space } (PiF I M)$ **thus** $A x \in \text{space } N$
using $MN[\text{of domain } x]$ **by** $(\text{auto simp: space-PiF measurable-space Pi'-def})$
qed

lemma measurable-PiF:
assumes $f: \bigwedge x. x \in \text{space } N \implies \text{domain } (f x) \in I \wedge (\forall i \in \text{domain } (f x). (f x) i \in \text{space } (M i))$
assumes $S: \bigwedge J S. J \in I \implies (\bigwedge i. i \in J \implies S i \in \text{sets } (M i)) \implies$
 $f - ' (Pi' J S) \cap \text{space } N \in \text{sets } N$
shows $f \in \text{measurable } N (PiF I M)$
unfolding $PiF\text{-def}$
using $PiF\text{-gen-subset}$
apply $(\text{rule measurable-measure-of})$
using f **apply force**
apply $(\text{insert } S, \text{auto})$
done

lemma restrict-sets-measurable:
assumes $A: A \in \text{sets } (PiF I M)$ **and** $J \subseteq I$
shows $A \cap \{m. \text{domain } m \in J\} \in \text{sets } (PiF J M)$
using $A[\text{unfolded sets-PiF}]$
proof $(\text{induct } A)$
case $(\text{Basic } a)$
then obtain $K S$ **where** $S: a = Pi' K S K \in I (\forall i \in K. S i \in \text{sets } (M i))$
by auto
show $?case$
proof $cases$
assume $K \in J$
hence $a \cap \{m. \text{domain } m \in J\} \in \{Pi' K X | X K. K \in J \wedge X \in (\Pi j \in K. \text{sets } (M j))\}$ **using** S
by $(\text{auto intro!: exI}[\text{where } x=K] \text{exI}[\text{where } x=S] \text{simp: Pi'-def})$
also have $\dots \subseteq \text{sets } (PiF J M)$ **unfolding** sets-PiF **by** auto
finally show $?thesis .$
next
assume $K \notin J$
hence $a \cap \{m. \text{domain } m \in J\} = \{\}$ **using** S **by** $(\text{auto simp: Pi'-def})$
also have $\dots \in \text{sets } (PiF J M)$ **by** simp
finally show $?thesis .$
qed

next
case $(\text{Union } a)$
have $\bigcup (a - ' UNIV) \cap \{m. \text{domain } m \in J\} = (\bigcup i. (a i \cap \{m. \text{domain } m \in J\}))$
by simp
also have $\dots \in \text{sets } (PiF J M)$ **using** Union **by** $(\text{intro sets.countable-nat-UN})$
 auto
finally show $?case .$

next

case (*Compl a*)
have ($\text{space } (PiF I M) - a \cap \{m. \text{domain } m \in J\} = (\text{space } (PiF J M) - (a \cap \{m. \text{domain } m \in J\}))$)
using $\langle J \subseteq I \rangle$ **by** (*auto simp: space-PiF Pi'-def*)
also have $\dots \in \text{sets } (PiF J M)$ **using** *Compl* **by** *auto*
finally show $?case$ **by** (*simp add: space-PiF*)
qed *simp*

lemma *measurable-finmap-of:*

assumes $f: \bigwedge i. (\exists x \in \text{space } N. i \in J x) \implies (\lambda x. f x i) \in \text{measurable } N (M i)$
assumes $J: \bigwedge x. x \in \text{space } N \implies J x \in I \bigwedge x. x \in \text{space } N \implies \text{finite } (J x)$
assumes $JN: \bigwedge S. \{x. J x = S\} \cap \text{space } N \in \text{sets } N$
shows $(\lambda x. \text{finmap-of } (J x) (f x)) \in \text{measurable } N (PiF I M)$
proof (*rule measurable-PiF*)
fix x **assume** $x \in \text{space } N$
with $J[\text{of } x]$ *measurable-space[OF f]*
show $\text{domain } (\text{finmap-of } (J x) (f x)) \in I \wedge$
 $(\forall i \in \text{domain } (\text{finmap-of } (J x) (f x)). (\text{finmap-of } (J x) (f x)) i \in \text{space } (M i))$
by *auto*

next

fix $K S$ **assume** $K \in I$ **and** $*$: $\bigwedge i. i \in K \implies S i \in \text{sets } (M i)$
with J **have** $eq: (\lambda x. \text{finmap-of } (J x) (f x)) -' Pi' K S \cap \text{space } N =$
 $(\text{if } \exists x \in \text{space } N. K = J x \wedge \text{finite } K \text{ then if } K = \{\} \text{ then } \{x \in \text{space } N. J x = K\}$
 $= K\}$
 $\text{else } (\bigcap i \in K. (\lambda x. f x i) -' S i \cap \{x \in \text{space } N. J x = K\}) \text{ else } \{\})$
by (*auto simp: Pi'-def*)
have $r: \{x \in \text{space } N. J x = K\} = \text{space } N \cap (\{x. J x = K\} \cap \text{space } N)$ **by**
auto
show $(\lambda x. \text{finmap-of } (J x) (f x)) -' Pi' K S \cap \text{space } N \in \text{sets } N$
unfolding *eq r*
apply (*simp del: INT-simps add:*)
apply (*intro conjI impI sets.finite-INT JN sets.Int[OF sets.top]*)
apply *simp* **apply** *assumption*
apply (*subst Int-assoc[symmetric]*)
apply (*rule sets.Int*)
apply (*intro measurable-sets[OF f] **) **apply** *force* **apply** *assumption*
apply (*intro JN*)
done

qed

lemma *measurable-PiM-finmap-of:*

assumes *finite J*
shows $\text{finmap-of } J \in \text{measurable } (Pi_M J M) (PiF \{J\} M)$
apply (*rule measurable-finmap-of*)
apply (*rule measurable-component-singleton*)
apply *simp*
apply *rule*

apply (rule \langle finite J \rangle)
apply simp
done

lemma proj-measurable-singleton:

assumes $A \in \text{sets } (M \ i)$
shows $(\lambda x. (x)_F \ i) - ' A \cap \text{space } (PiF \ \{I\} \ M) \in \text{sets } (PiF \ \{I\} \ M)$

proof cases

assume $i \in I$
hence $(\lambda x. (x)_F \ i) - ' A \cap \text{space } (PiF \ \{I\} \ M) =$
 $Pi' \ I \ (\lambda x. \text{if } x = i \text{ then } A \text{ else } \text{space } (M \ x))$
using sets.sets-into-space[OF] $\langle A \in \text{sets } (M \ i) \rangle$ assms
by (auto simp: space-PiF Pi'-def)
thus ?thesis **using** assms $\langle A \in \text{sets } (M \ i) \rangle$
by (intro in-sets-PiFI) auto

next

assume $i \notin I$
hence $(\lambda x. (x)_F \ i) - ' A \cap \text{space } (PiF \ \{I\} \ M) =$
 $(\text{if } \text{undefined} \in A \text{ then } \text{space } (PiF \ \{I\} \ M) \text{ else } \{\})$ **by** (auto simp: space-PiF
Pi'-def)
thus ?thesis **by** simp
qed

lemma measurable-proj-singleton:

assumes $i \in I$
shows $(\lambda x. (x)_F \ i) \in \text{measurable } (PiF \ \{I\} \ M) \ (M \ i)$
by (unfold measurable-def, intro CollectI conjI ballI proj-measurable-singleton
assms)
(insert $\langle i \in I \rangle$, auto simp: space-PiF)

lemma measurable-proj-countable:

fixes $I::'a::\text{countable set set}$
assumes $y \in \text{space } (M \ i)$
shows $(\lambda x. \text{if } i \in \text{domain } x \text{ then } (x)_F \ i \text{ else } y) \in \text{measurable } (PiF \ I \ M) \ (M \ i)$
proof (rule countable-measurable-PiFI)
fix J **assume** $J \in I$ finite J
show $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) \in \text{measurable } (PiF \ \{J\} \ M) \ (M \ i)$
unfolding measurable-def
proof safe
fix z **assume** $z \in \text{sets } (M \ i)$
have $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) - ' z \cap \text{space } (PiF \ \{J\} \ M) =$
 $(\lambda x. \text{if } i \in J \text{ then } (x)_F \ i \text{ else } y) - ' z \cap \text{space } (PiF \ \{J\} \ M)$
by (auto simp: space-PiF Pi'-def)
also have $\dots \in \text{sets } (PiF \ \{J\} \ M)$ **using** $\langle z \in \text{sets } (M \ i) \rangle$ \langle finite J \rangle
by (cases $i \in J$) (auto intro!: measurable-sets[OF measurable-proj-singleton])
finally show $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) - ' z \cap \text{space } (PiF \ \{J\} \ M) \in$
 $\text{sets } (PiF \ \{J\} \ M)$.
qed (insert $\langle y \in \text{space } (M \ i) \rangle$, auto simp: space-PiF Pi'-def)
qed

lemma *measurable-restrict-proj*:

assumes $J \in II$ *finite J*

shows *finmap-of J* \in *measurable* $(PiM J M)$ $(PiF II M)$

using *assms*

by $(intro$ *measurable-finmap-of measurable-component-singleton*) *auto*

lemma *measurable-proj-PiM*:

fixes $J K :: 'a::countable$ *set* **and** $I :: 'a$ *set* *set*

assumes *finite J* $J \in I$

assumes $x \in space$ $(PiM J M)$

shows *proj* \in *measurable* $(PiF \{J\} M)$ $(PiM J M)$

proof $(rule$ *measurable-PiM-single*)

show *proj* \in *space* $(PiF \{J\} M) \rightarrow (\Pi_E i \in J. space (M i))$

using *assms* **by** $(auto$ *simp* *add: space-PiM space-PiF extensional-def sets-PiF*

Pi'-def)

next

fix $A i$ **assume** $A: i \in J$ $A \in sets (M i)$

show $\{\omega \in space (PiF \{J\} M). (\omega)_F i \in A\} \in sets (PiF \{J\} M)$

proof

have $\{\omega \in space (PiF \{J\} M). (\omega)_F i \in A\} =$

$(\lambda\omega. (\omega)_F i) - 'A \cap space (PiF \{J\} M)$ **by** *auto*

also have $\dots \in sets (PiF \{J\} M)$

using *assms* A **by** $(auto$ *intro: measurable-sets[OF measurable-proj-singleton]*

simp: space-PiM)

finally show *?thesis* .

qed *simp*

qed

lemma *space-PiF-singleton-eq-product*:

assumes *finite I*

shows *space* $(PiF \{I\} M) = (\Pi' i \in I. space (M i))$

by $(auto$ *simp: product-def space-PiF assms*)

adapted from $sets (Pi_M ?I ?M) = sigma-sets (\Pi_E i \in ?I. space (?M i)) \{\{f \in \Pi_E i \in ?I. space (?M i). f i \in A\} \mid i A. i \in ?I \wedge A \in sets (?M i)\}$

lemma *sets-PiF-single*:

assumes *finite I* $I \neq \{\}$

shows *sets* $(PiF \{I\} M) =$

sigma-sets $(\Pi' i \in I. space (M i))$

$\{\{f \in \Pi' i \in I. space (M i). f i \in A\} \mid i A. i \in I \wedge A \in sets (M i)\}$

$(is - = sigma-sets ?\Omega ?R)$

unfolding *sets-PiF-singleton*

proof $(rule$ *sigma-sets-eqI*)

interpret $R: sigma-algebra ?\Omega sigma-sets ?\Omega ?R$ **by** $(rule$ *sigma-algebra-sigma-sets*) *auto*

fix A **assume** $A \in \{Pi' I X \mid X. X \in (\Pi j \in I. sets (M j))\}$

then obtain X **where** $X: A = Pi' I X X \in (\Pi j \in I. sets (M j))$ **by** *auto*

show $A \in sigma-sets ?\Omega ?R$

proof –
from $\langle I \neq \{\} \rangle X$ **have** $A = (\bigcap j \in I. \{f \in \text{space } (PiF \ \{I\} \ M). f \ j \in X \ j\})$
using *sets.sets-into-space*
by (*auto simp: space-PiF product-def*) *blast*
also have $\dots \in \text{sigma-sets } ?\Omega \ ?R$
using $X \langle I \neq \{\} \rangle$ *assms* **by** (*intro R.finite-INT*) (*auto simp: space-PiF*)
finally show $A \in \text{sigma-sets } ?\Omega \ ?R$.
qed
next
fix A **assume** $A \in ?R$
then obtain $i \ B$ **where** $A: A = \{f \in \Pi' \ i \in I. \text{space } (M \ i). f \ i \in B\} \ i \in I \ B \in$
sets $(M \ i)$
by *auto*
then have $A = (\Pi' \ j \in I. \text{if } j = i \text{ then } B \text{ else } \text{space } (M \ j))$
using *sets.sets-into-space[OF A(3)]*
apply (*auto simp: Pi'-iff split: if-split-asm*)
apply *blast*
done
also have $\dots \in \text{sigma-sets } ?\Omega \ \{Pi' \ I \ X \ |X. X \in (\Pi \ j \in I. \text{sets } (M \ j))\}$
using A
by (*intro sigma-sets.Basic*)
*(auto intro: exI[**where** $x = \lambda j. \text{if } j = i \text{ then } B \text{ else } \text{space } (M \ j)$])*
finally show $A \in \text{sigma-sets } ?\Omega \ \{Pi' \ I \ X \ |X. X \in (\Pi \ j \in I. \text{sets } (M \ j))\}$.
qed

adapted from $(\bigwedge i. i \in ?I \implies ?A \ i = ?B \ i) \implies Pi_E \ ?I \ ?A = Pi_E \ ?I \ ?B$

lemma *Pi'-cong*:
assumes *finite I*
assumes $\bigwedge i. i \in I \implies f \ i = g \ i$
shows $Pi' \ I \ f = Pi' \ I \ g$
using *assms* **by** (*auto simp: Pi'-def*)

adapted from $\llbracket \text{finite } ?I; \bigwedge i \ n \ m. \llbracket i \in ?I; n \leq m \rrbracket \implies ?A \ n \ i \subseteq ?A \ m \ i \rrbracket$
 $\implies (\bigcup_n Pi \ ?I \ (?A \ n)) = (\Pi \ i \in ?I. \bigcup_n ?A \ n \ i)$

lemma *Pi'-UN*:
fixes $A :: \text{nat} \Rightarrow 'i \Rightarrow 'a \ \text{set}$
assumes *finite I*
assumes *mono*: $\bigwedge i \ n \ m. i \in I \implies n \leq m \implies A \ n \ i \subseteq A \ m \ i$
shows $(\bigcup_n Pi' \ I \ (A \ n)) = Pi' \ I \ (\lambda i. \bigcup_n. A \ n \ i)$

proof (*intro set-eqI iffI*)
fix f **assume** $f \in Pi' \ I \ (\lambda i. \bigcup_n. A \ n \ i)$
then have $\forall i \in I. \exists n. f \ i \in A \ n \ i$ *domain* $f = I$ **by** (*auto simp: $\langle \text{finite } I \rangle$ Pi'-def*)
from *bchoice[OF this(1)]* **obtain** n **where** $n: \bigwedge i. i \in I \implies f \ i \in (A \ (n \ i) \ i)$ **by**
auto
obtain k **where** $k: \bigwedge i. i \in I \implies n \ i \leq k$
using $\langle \text{finite } I \rangle$ *finite-nat-set-iff-bounded-le[of n'I]* **by** *auto*
have $f \in Pi' \ I \ (\lambda i. A \ k \ i)$
proof
fix i **assume** $i \in I$

```

from mono[OF this, of n i k] k[OF this] n[OF this] ⟨domain f = I⟩ ⟨i ∈ I⟩
show f i ∈ A k i by (auto simp: ⟨finite I⟩)
qed (simp add: ⟨domain f = I⟩ ⟨finite I⟩)
then show f ∈ (⋃ n. Pi' I (A n)) by auto
qed (auto simp: Pi'-def ⟨finite I⟩)

```

adapted from $\llbracket \bigwedge i. i \in ?I \implies \exists S \subseteq ?E \ i. \text{countable } S \wedge ?\Omega \ i = \bigcup S; \bigwedge i. i \in ?I \implies ?E \ i \subseteq \text{Pow} \ (?\Omega \ i); \bigwedge j. j \in ?J \implies \text{finite } j; \bigcup ?J = ?I \rrbracket \implies \text{sets} \ (Pi_M \ ?I \ (\lambda i. \text{sigma} \ (?\Omega \ i) \ (?E \ i))) = \text{sets} \ (\text{sigma} \ (Pi_E \ ?I \ ?\Omega) \ \{\{f \in Pi_E \ ?I \ ?\Omega. \forall i \in j. f \ i \in A \ i\} \mid A \ j. j \in ?J \wedge A \in Pi \ j \ ?E\})$

lemma *sigma-fprod-algebra-sigma-eq*:

```

fixes E :: 'i ⇒ 'a set set and S :: 'i ⇒ nat ⇒ 'a set
assumes [simp]: finite I I ≠ {}
and S-union:  $\bigwedge i. i \in I \implies (\bigcup j. S \ i \ j) = \text{space} \ (M \ i)$ 
and S-in-E:  $\bigwedge i. i \in I \implies \text{range} \ (S \ i) \subseteq E \ i$ 
assumes E-closed:  $\bigwedge i. i \in I \implies E \ i \subseteq \text{Pow} \ (\text{space} \ (M \ i))$ 
and E-generates:  $\bigwedge i. i \in I \implies \text{sets} \ (M \ i) = \text{sigma-sets} \ (\text{space} \ (M \ i)) \ (E \ i)$ 
defines P == { Pi' I F | F.  $\forall i \in I. F \ i \in E \ i$  }
shows sets (PiF {I} M) = sigma-sets (space (PiF {I} M)) P
proof
let ?P = sigma (space (PiF {I} M)) P
from ⟨finite I⟩ [THEN ex-bij-betw-finite-nat] obtain T where bij-betw T I {0..<card I} ..
then have T:  $\bigwedge i. i \in I \implies T \ i < \text{card} \ I \wedge i. i \in I \implies \text{the-inv-into} \ I \ T \ (T \ i) = i$ 
by (auto simp add: bij-betw-def set-eq-iff image-iff the-inv-into-f-f simp del: ⟨finite I⟩)
have P-closed:  $P \subseteq \text{Pow} \ (\text{space} \ (PiF \ \{I\} \ M))$ 
using E-closed by (auto simp: space-PiF P-def Pi'-iff subset-eq)
then have space-P:  $\text{space} \ ?P = (\prod' i \in I. \text{space} \ (M \ i))$ 
by (simp add: space-PiF)
have sets (PiF {I} M) =
  sigma-sets (space ?P)  $\{\{f \in \prod' i \in I. \text{space} \ (M \ i). f \ i \in A\} \mid A. i \in I \wedge A \in \text{sets} \ (M \ i)\}$ 
using sets-PiF-single[of I M] by (simp add: space-P)
also have ...  $\subseteq \text{sets} \ (\text{sigma} \ (\text{space} \ (PiF \ \{I\} \ M)) \ P)$ 
proof (safe intro!: sets.sigma-sets-subset)
fix i A assume i ∈ I and A: A ∈ sets (M i)
have  $(\lambda x. (x)_F \ i) \in \text{measurable} \ ?P \ (\text{sigma} \ (\text{space} \ (M \ i)) \ (E \ i))$ 
proof (subst measurable-iff-measure-of)
show  $E \ i \subseteq \text{Pow} \ (\text{space} \ (M \ i))$  using ⟨i ∈ I⟩ by fact
from space-P ⟨i ∈ I⟩ show  $(\lambda x. (x)_F \ i) \in \text{space} \ ?P \rightarrow \text{space} \ (M \ i)$ 
by auto
show  $\forall A \in E \ i. (\lambda x. (x)_F \ i) -' A \cap \text{space} \ ?P \in \text{sets} \ ?P$ 
proof
fix A assume A: A ∈ E i
from T have *:  $\exists xs. \text{length} \ xs = \text{card} \ I$ 
 $\wedge (\forall j \in I. (g)_F \ j \in (\text{if } i = j \text{ then } A \text{ else } S \ j \ (xs \ ! \ T \ j)))$ 
if domain g = I and  $\forall j \in I. (g)_F \ j \in (\text{if } i = j \text{ then } A \text{ else } S \ j \ (f \ j))$  for g f

```


using *that by* (*auto intro!*: exI [*of - map* ($\lambda n. f$ (*the-inv-into* $I T n$))
 $[0..<card I]$])
from A **have** ($\lambda x. (x)_F i$) $- ' A \cap space ?P = (\Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } space (M j))$
using *E-closed* $\langle i \in I \rangle$ **by** (*auto simp*: *space-P Pi-iff subset-eq split*:
if-split-asm)
also have $\dots = (\Pi' j \in I. \bigcup n. \text{if } i = j \text{ then } A \text{ else } S j n)$
by (*intro Pi'-cong*) (*simp-all add*: *S-union*)
also have $\dots = \{g. domain\ g = I \wedge (\exists f. \forall j \in I. (g)_F j \in (\text{if } i = j \text{ then } A \text{ else } S j (f j)))\}$
by (*rule set-eqI*) (*simp del*: *if-image-distrib add*: *Pi'-iff bchoice-iff*)
also have $\dots = (\bigcup xs \in \{xs. length\ xs = card\ I\}. \Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } S j (xs ! T j))$
using $*$ **by** (*auto simp add*: *Pi'-iff split del*: *if-split*)
also have $\dots \in sets\ ?P$
proof (*safe intro!*: *sets.countable-UN*)
fix xs **show** ($\Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } S j (xs ! T j) \in sets\ ?P$)
using *A S-in-E*
by (*simp add*: *P-closed*)
(auto simp: *P-def subset-eq intro!*: exI [*of -* $\lambda j. \text{if } i = j \text{ then } A \text{ else } S j$
 $(xs ! T j)$])
qed
finally show ($\lambda x. (x)_F i$) $- ' A \cap space ?P \in sets\ ?P$
using *P-closed by simp*
qed
qed
from *measurable-sets*[*OF this, of A*] $A \langle i \in I \rangle$ *E-closed*
have ($\lambda x. (x)_F i$) $- ' A \cap space ?P \in sets\ ?P$
by (*simp add*: *E-generates*)
also have ($\lambda x. (x)_F i$) $- ' A \cap space ?P = \{f \in \Pi' i \in I. space (M i). f i \in A\}$
using *P-closed by (auto simp*: *space-PiF*)
finally show $\dots \in sets\ ?P$.
qed
finally show $sets (PiF \{I\} M) \subseteq sigma\text{-sets} (space (PiF \{I\} M)) P$
by (*simp add*: *P-closed*)
show $sigma\text{-sets} (space (PiF \{I\} M)) P \subseteq sets (PiF \{I\} M)$
using $\langle finite\ I \rangle \langle I \neq \{\} \rangle$
by (*auto intro!*: *sets.sigma-sets-subset product-in-sets-PiFI simp*: *E-generates*
P-def)
qed

lemma *product-open-generates-sets-PiF-single*:

assumes $I \neq \{\}$
assumes [*simp*]: *finite I*
shows $sets (PiF \{I\} (\lambda-. borel::'b::second-countable-topology\ measure)) =$
 $sigma\text{-sets} (space (PiF \{I\} (\lambda-. borel))) \{Pi' I F |F. (\forall i \in I. F i \in Collect$
*open)\}
proof –
from *open-countable-basisE*[*OF open-UNIV*] **obtain** $S::'b\ set\ set$*

```

  where S: S ⊆ (SOME B. countable B ∧ topological-basis B) UNIV = ⋃ S
  by auto
show ?thesis
proof (rule sigma-fprod-algebra-sigma-eq)
  show finite I by simp
  show I ≠ {} by fact
  define S' where S' = from-nat-into S
  show (⋃ j. S' j) = space borel
  using S
  apply (auto simp add: from-nat-into countable-basis-proj S'-def basis-proj-def)
  apply (metis (lifting, mono-tags) UNIV-I UnionE basis-proj-def countable-basis-proj countable-subset from-nat-into-surj)
  done
  show range S' ⊆ Collect open
  using S
  apply (auto simp add: from-nat-into countable-basis-proj S'-def)
  apply (metis UNIV-not-empty Union-empty from-nat-into subsetD topological-basis-open[OF basis-proj] basis-proj-def)
  done
  show Collect open ⊆ Pow (space borel) by simp
  show sets borel = sigma-sets (space borel) (Collect open)
  by (simp add: borel-def)
qed
qed

```

lemma *finmap-UNIV[simp]*: $(\bigcup_{J \in \text{Collect finite.}} \Pi' j \in J. \text{UNIV}) = \text{UNIV}$ by *auto*

lemma *borel-eq-PiF-borel*:

```

shows (borel :: ('i::countable ⇒F 'a::polish-space) measure) =
  PiF (Collect finite) (λ-. borel :: 'a measure)
unfolding borel-def PiF-def
proof (rule measure-eqI, clarsimp, rule sigma-sets-eqI)
  fix a:('i ⇒F 'a) set assume a ∈ Collect open hence open a by simp
  then obtain B' where B': B' ⊆ basis-finmap a = ⋃ B'
  using finmap-topological-basis by (force simp add: topological-basis-def)
  have a ∈ sigma UNIV {Pi' J X | X J. finite J ∧ X ∈ J → sigma-sets UNIV
(Collect open)}
  unfolding ⟨a = ⋃ B'⟩
  proof (rule sets.countable-Union)
    from B' countable-basis-finmap show countable B' by (metis countable-subset)
  next
    show B' ⊆ sets (sigma UNIV
    {Pi' J X | X J. finite J ∧ X ∈ J → sigma-sets UNIV (Collect open)}) (is -
    ⊆ sets ?s)
  proof
    fix x assume x ∈ B' with B' have x ∈ basis-finmap by auto
    then obtain J X where x = Pi' J X finite J X ∈ J → sigma-sets UNIV
(Collect open)

```

```

    by (auto simp: basis-finmap-def topological-basis-open[OF basis-proj])
  thus  $x \in \text{sets } ?s$  by auto
qed
qed
thus  $a \in \text{sigma-sets UNIV } \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV } (Collect\ open)\}$ 
(Collect open)}
  by simp
next
fix  $b::('i \Rightarrow_F 'a)$  set
assume  $b \in \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV } (Collect\ open)\}$ 
  hence  $b':: b \in \text{sets } (Pi_F (Collect\ finite) (\lambda-. \text{borel}))$  by (auto simp: sets-PiF
borel-def)
let  $?b = \lambda J. b \cap \{x. \text{domain } x = J\}$ 
have  $b = \bigcup ((\lambda J. ?b J) ' Collect\ finite)$  by auto
also have  $\dots \in \text{sets borel}$ 
proof (rule sets.countable-Union, safe)
fix  $J::'i$  set assume finite J
{ assume  $ef: J = \{\}$ 
  have  $?b J \in \text{sets borel}$ 
proof cases
  assume  $?b J \neq \{\}$ 
  then obtain  $f$  where  $f \in b$  domain  $f = \{\}$  using  $ef$  by auto
  hence  $?b J = \{f\}$  using  $\langle J = \{\} \rangle$ 
    by (auto simp: finmap-eq-iff)
  also have  $\{f\} \in \text{sets borel}$  by simp
  finally show  $?thesis$  .
}
qed simp
} moreover {
  assume  $J \neq (\{\}::'i \text{ set})$ 
  have  $(?b J) = b \cap \{m. \text{domain } m \in \{J\}\}$  by auto
  also have  $\dots \in \text{sets } (Pi_F \{J\} (\lambda-. \text{borel}))$ 
    using  $b'$  by (rule restrict-sets-measurable) (auto simp:  $\langle \text{finite } J \rangle$ )
  also have  $\dots = \text{sigma-sets } (space (Pi_F \{J\} (\lambda-. \text{borel})))$ 
     $\{Pi' (J) F \mid F. (\forall j \in J. F j \in \text{Collect } open)\}$ 
    (is - = sigma-sets - ?P)
  by (rule product-open-generates-sets-PiF-single[OF  $\langle J \neq \{\} \rangle \langle \text{finite } J \rangle$ ])
  also have  $\dots \subseteq \text{sigma-sets UNIV } (Collect\ open)$ 
    by (intro sigma-sets-mono'') (auto intro!: open-Pi'I simp: space-PiF)
  finally have  $(?b J) \in \text{sets borel}$  by (simp add: borel-def)
} ultimately show  $(?b J) \in \text{sets borel}$  by blast
qed (simp add: countable-Collect-finite)
finally show  $b \in \text{sigma-sets UNIV } (Collect\ open)$  by (simp add: borel-def)
qed (simp add: emeasure-sigma borel-def PiF-def)

```

22.10 Isomorphism between Functions and Finite Maps

lemma measurable-finmap-compose:

shows $(\lambda m. \text{compose } J m f) \in \text{measurable } (Pi_M (f ' J) (\lambda-. M)) (Pi_M J (\lambda-. M))$

unfolding *compose-def by measurable*

lemma *measurable-compose-inv*:

assumes *inj*: $\bigwedge j. j \in J \implies f' (f j) = j$

shows $(\lambda m. \text{compose } (f' J) m f') \in \text{measurable } (PiM J (\lambda-. M)) (PiM (f' J) (\lambda-. M))$

unfolding *compose-def by (rule measurable-restrict) (auto simp: inj)*

locale *function-to-finmap* =

fixes *J*: 'a set **and** *f* :: 'a \Rightarrow 'b::countable **and** *f'*

assumes [*simp*]: *finite J*

assumes *inv*: $i \in J \implies f' (f i) = i$

begin

to measure finmaps

definition *fm* = (*finmap-of* (f' J)) o ($\lambda g. \text{compose } (f' J) g f'$)

lemma *domain-fm[simp]*: *domain (fm x) = f' J*

unfolding *fm-def by simp*

lemma *fm-restrict[simp]*: *fm (restrict y J) = fm y*

unfolding *fm-def by (auto simp: compose-def inv intro: restrict-ext)*

lemma *fm-product*:

assumes $\bigwedge i. \text{space } (M i) = UNIV$

shows $fm - ' Pi' (f' J) S \cap \text{space } (PiM J M) = (\Pi_E j \in J. S (f j))$

using *assms*

by (*auto simp: inv fm-def compose-def space-PiM Pi'-def*)

lemma *fm-measurable*:

assumes $f' J \in N$

shows $fm \in \text{measurable } (PiM J (\lambda-. M)) (PiF N (\lambda-. M))$

unfolding *fm-def*

proof (*rule measurable-comp, rule measurable-compose-inv*)

show *finmap-of* (f' J) $\in \text{measurable } (PiM (f' J) (\lambda-. M)) (PiF N (\lambda-. M))$

using *assms by (intro measurable-finmap-of measurable-component-singleton)*

auto

qed (*simp-all add: inv*)

lemma *proj-fm*:

assumes $x \in J$

shows $fm m (f x) = m x$

using *assms by (auto simp: fm-def compose-def o-def inv)*

lemma *inj-on-compose-f'*: *inj-on* ($\lambda g. \text{compose } (f' J) g f'$) (*extensional J*)

proof (*rule inj-on-inverseI*)

fix $x::'a \Rightarrow 'c$ **assume** $x \in \text{extensional } J$

thus $(\lambda x. \text{compose } J x f) (\text{compose } (f' J) x f') = x$

by (*auto simp: compose-def inv extensional-def*)

qed

lemma *inj-on-fm*:

assumes $\bigwedge i. \text{space } (M \ i) = \text{UNIV}$
 shows *inj-on fm* (*space* ($\text{Pi}_M \ J \ M$))
 using *assms*
 apply (*auto simp: fm-def space-PiM PiE-def*)
 apply (*rule comp-inj-on*)
 apply (*rule inj-on-compose-f'*)
 apply (*rule finmap-of-inj-on-extensional-finite*)
 apply *simp*
 apply (*auto*)
 done

to measure functions

definition *mf* = ($\lambda g. \text{compose } J \ g \ f$) o *proj*

lemma *mf-fm*:

assumes $x \in \text{space } (\text{Pi}_M \ J \ (\lambda-. \ M))$
 shows *mf* (*fm* x) = x
 proof –
 have *mf* (*fm* x) \in *extensional* J
 by (*auto simp: mf-def extensional-def compose-def*)
 moreover
 have $x \in \text{extensional } J$ using *assms sets.sets-into-space*
 by (*force simp: space-PiM PiE-def*)
 moreover
 { fix i assume $i \in J$
 hence *mf* (*fm* x) $i = x \ i$
 by (*auto simp: inv mf-def compose-def fm-def*)
 }
 ultimately
 show *thesis* by (*rule extensionalityI*)
 qed

lemma *mf-measurable*:

assumes *space* $M = \text{UNIV}$
 shows *mf* \in *measurable* ($\text{PiF } \{f \ ' \ J\} \ (\lambda-. \ M)$) ($\text{PiM } J \ (\lambda-. \ M)$)
 unfolding *mf-def*
 proof (*rule measurable-comp, rule measurable-proj-PiM*)
 show $(\lambda g. \text{compose } J \ g \ f) \in \text{measurable } (\text{PiM } (f \ ' \ J) \ (\lambda x. \ M)) \ (\text{PiM } J \ (\lambda-. \ M))$
 by (*rule measurable-finmap-compose*)
 qed (*auto simp add: space-PiM extensional-def assms*)

lemma *fm-image-measurable*:

assumes *space* $M = \text{UNIV}$
 assumes $X \in \text{sets } (\text{Pi}_M \ J \ (\lambda-. \ M))$
 shows *fm* ' $X \in \text{sets } (\text{PiF } \{f \ ' \ J\} \ (\lambda-. \ M))$
 proof –

```

have  $fm \text{ ' } X = (mf) \text{ - ' } X \cap \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$ 
proof safe
  fix  $x$  assume  $x \in X$ 
  with  $mf\text{-}fm[of \ x] \text{ sets.sets-into-space}[OF \ \text{assms}(2)]$  show  $fm \ x \in mf \text{ - ' } X$  by
auto
  show  $fm \ x \in \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$  by (simp add: space-PiF assms)
next
  fix  $y \ x$ 
  assume  $x: mf \ y \in X$ 
  assume  $y: y \in \text{space } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$ 
  thus  $y \in fm \text{ ' } X$ 
  by (intro image-eqI[OF - x], unfold finmap-eq-iff)
  (auto simp: space-PiF fm-def mf-def compose-def inv Pi'-def)
qed
also have  $\dots \in \text{sets } (PiF \{f \text{ ' } J\} (\lambda\text{-. } M))$ 
  using assms
  by (intro measurable-sets[OF mf-measurable]) auto
finally show ?thesis .
qed

```

```

lemma fm-image-measurable-finite:
  assumes  $\text{space } M = UNIV$ 
  assumes  $X \in \text{sets } (Pi_M \ J \ (\lambda\text{-. } M::'c \ \text{measure}))$ 
  shows  $fm \text{ ' } X \in \text{sets } (PiF \ (\text{Collect } \text{finite}) \ (\lambda\text{-. } M::'c \ \text{measure}))$ 
  using fm-image-measurable[OF assms]
  by (rule subspace-set-in-sets) (auto simp: finite-subset)

```

measure on finmaps

```

definition mapmeasure  $M \ N = \text{distr } M \ (PiF \ (\text{Collect } \text{finite}) \ N) \ (fm)$ 

```

```

lemma sets-mapmeasure[simp]:  $\text{sets } (\text{mapmeasure } M \ N) = \text{sets } (PiF \ (\text{Collect } \text{finite}) \ N)$ 
  unfolding mapmeasure-def by simp

```

```

lemma space-mapmeasure[simp]:  $\text{space } (\text{mapmeasure } M \ N) = \text{space } (PiF \ (\text{Collect } \text{finite}) \ N)$ 
  unfolding mapmeasure-def by simp

```

```

lemma mapmeasure-PiF:
  assumes  $s1: \text{space } M = \text{space } (Pi_M \ J \ (\lambda\text{-. } N))$ 
  assumes  $s2: \text{sets } M = \text{sets } (Pi_M \ J \ (\lambda\text{-. } N))$ 
  assumes  $\text{space } N = UNIV$ 
  assumes  $X \in \text{sets } (PiF \ (\text{Collect } \text{finite}) \ (\lambda\text{-. } N))$ 
  shows  $\text{emeasure } (\text{mapmeasure } M \ (\lambda\text{-. } N)) \ X = \text{emeasure } M \ ((fm \text{ - ' } X \cap \text{extensional } J))$ 
  using assms
  by (auto simp: measurable-cong-sets[OF s2 refl] mapmeasure-def emeasure-distr fm-measurable space-PiM PiE-def)

```

```

lemma mapmeasure-PiM:
  fixes  $N::'c$  measure
  assumes  $s1$ : space  $M = \text{space } (Pi_M J (\lambda-. N))$ 
  assumes  $s2$ : sets  $M = (Pi_M J (\lambda-. N))$ 
  assumes  $N$ : space  $N = UNIV$ 
  assumes  $X$ :  $X \in \text{sets } M$ 
  shows emeasure  $M X = \text{emeasure } (\text{mapmeasure } M (\lambda-. N)) (fm \text{ ` } X)$ 
  unfolding mapmeasure-def
proof (subst emeasure-distr, subst measurable-cong-sets[OF s2 refl], rule fm-measurable)
  have  $X \subseteq \text{space } (Pi_M J (\lambda-. N))$  using assms by (simp add: sets.sets-into-space)
  from assms inj-on-fm[of  $\lambda-. N$ ] subsetD[OF this] have  $fm \text{ ` } X \cap \text{space } (Pi_M J (\lambda-. N)) = X$ 
    by (auto simp: vimage-image-eq inj-on-def)
  thus emeasure  $M X = \text{emeasure } M (fm \text{ ` } X \cap \text{space } M)$  using  $s1$ 
    by simp
  show  $fm \text{ ` } X \in \text{sets } (PiF (Collect finite) (\lambda-. N))$ 
    by (rule fm-image-measurable-finite[OF  $N X$ [simplified s2]])
qed simp

end

end

```

23 Projective Limit

```

theory Projective-Limit
imports
  Fin-Map
  Infinite-Product-Measure
  HOL-Library.Diagonal-Subsequence
begin

```

23.1 Sequences of Finite Maps in Compact Sets

```

locale finmap-seqs-into-compact =
  fixes  $K::nat \Rightarrow (nat \Rightarrow_F 'a::\text{metric-space}) \text{ set}$  and  $f::nat \Rightarrow (nat \Rightarrow_F 'a)$  and
   $M$ 
  assumes compact:  $\bigwedge n. \text{compact } (K n)$ 
  assumes f-in-K:  $\bigwedge n. K n \neq \{\}$ 
  assumes domain-K:  $\bigwedge n. k \in K n \implies \text{domain } k = \text{domain } (f n)$ 
  assumes proj-in-K:
     $\bigwedge t n m. m \geq n \implies t \in \text{domain } (f n) \implies (f m)_F t \in (\lambda k. (k)_F t) \text{ ` } K n$ 
begin

lemma proj-in-K':  $(\exists n. \forall m \geq n. (f m)_F t \in (\lambda k. (k)_F t) \text{ ` } K n)$ 
  using proj-in-K f-in-K
proof cases
  obtain  $k$  where  $k \in K (Suc 0)$  using f-in-K by auto
  assume  $\forall n. t \notin \text{domain } (f n)$ 

```

thus *?thesis*
by (*auto intro!*: *exI*[**where** $x=1$] *image-eqI*[*OF* - $\langle k \in K (Suc\ 0) \rangle$]
simp: *domain-K*[*OF* $\langle k \in K (Suc\ 0) \rangle$])
qed *blast*

lemma *proj-in-KE*:
obtains n **where** $\bigwedge m. m \geq n \implies (f\ m)_F\ t \in (\lambda k. (k)_F\ t) \text{ ‘ } K\ n$
using *proj-in-K'* **by** *blast*

lemma *compact-projset*:
shows *compact* $((\lambda k. (k)_F\ i) \text{ ‘ } K\ n)$
using *continuous-proj compact* **by** (*rule compact-continuous-image*)

end

lemma *compactE'*:
fixes $S :: 'a :: \text{metric-space set}$
assumes *compact* $S\ \forall n \geq m. f\ n \in S$
obtains $l\ r$ **where** $l \in S$ *strict-mono* $(r :: \text{nat} \Rightarrow \text{nat})\ ((f \circ r) \longrightarrow l)$ *sequentially*
proof *atomize-elim*
have *strict-mono* $((+) m)$ **by** (*simp add: strict-mono-def*)
have $\forall n. (f \circ (\lambda i. m + i))\ n \in S$ **using** *assms* **by** *auto*
from *seq-compactE*[*OF* $\langle \text{compact } S \rangle$] [*unfolded compact-eq-seq-compact-metric*] *this*
obtain $l\ r$ **where** $l \in S$ *strict-mono* $r\ (f \circ (+)\ m \circ r) \longrightarrow l$ **by** *blast*
hence $l \in S$ *strict-mono* $((\lambda i. m + i) \circ r) \wedge (f \circ ((\lambda i. m + i) \circ r)) \longrightarrow l$
using *strict-mono-o*[*OF* $\langle \text{strict-mono } ((+) m) \rangle$] $\langle \text{strict-mono } r \rangle$ **by** (*auto simp*:
o-def)
thus $\exists l\ r. l \in S \wedge \text{strict-mono } r \wedge (f \circ r) \longrightarrow l$ **by** *blast*
qed

sublocale *finmap-seqs-into-compact* \subseteq *subseqs* $\lambda n\ s. (\exists l. (\lambda i. ((f \circ s)\ i)_F\ n) \longrightarrow l)$

proof
fix n **and** $s :: \text{nat} \Rightarrow \text{nat}$
assume *strict-mono* s
from *proj-in-KE*[*of* n] **obtain** $n0$ **where** $n0: \bigwedge m. n0 \leq m \implies (f\ m)_F\ n \in (\lambda k. (k)_F\ n) \text{ ‘ } K\ n0$
by *blast*
have $\forall i \geq n0. ((f \circ s)\ i)_F\ n \in (\lambda k. (k)_F\ n) \text{ ‘ } K\ n0$
proof *safe*
fix i **assume** $n0 \leq i$
also **have** $\dots \leq s\ i$ **by** (*rule seq-suble*) *fact*
finally **have** $n0 \leq s\ i$.
with $n0$ **show** $((f \circ s)\ i)_F\ n \in (\lambda k. (k)_F\ n) \text{ ‘ } K\ n0$
by *auto*
qed
then **obtain** $l\ s$
where $l\ s \in (\lambda k. (k)_F\ n) \text{ ‘ } K\ n0$ *strict-mono* $rs\ ((\lambda i. ((f \circ s)\ i)_F\ n) \circ rs) \longrightarrow l\ s$

by (rule compactE'[OF compact-projset])
thus $\exists r'. \text{strict-mono } r' \wedge (\exists l. (\lambda i. ((f \circ (s \circ r')) i)_F n) \longrightarrow l)$ **by** (auto simp: o-def)
qed

lemma (in finmap-seqs-into-compact) diagonal-tendsto: $\exists l. (\lambda i. (f (diagseq i))_F n) \longrightarrow l$

proof –

obtain l **where** $(\lambda i. ((f \circ (diagseq \circ (+) (Suc n))) i)_F n) \longrightarrow l$

proof (atomize-elim, rule diagseq-holds)

fix $r \ s \ n$

assume *strict-mono* $(r :: \text{nat} \Rightarrow \text{nat})$

assume $\exists l. (\lambda i. ((f \circ s) i)_F n) \longrightarrow l$

then obtain l **where** $(\lambda i. (f i)_F n \circ s) \longrightarrow l$

by (auto simp: o-def)

hence $(\lambda i. (f i)_F n \circ s \circ r) \longrightarrow l$ **using** $\langle \text{strict-mono } r \rangle$

by (rule LIMSEQ-subseq-LIMSEQ)

thus $\exists l. (\lambda i. ((f \circ (s \circ r)) i)_F n) \longrightarrow l$ **by** (auto simp add: o-def)

qed

hence $(\lambda i. ((f (diagseq (i + Suc n))))_F n) \longrightarrow l$ **by** (simp add: ac-simps)

hence $(\lambda i. (f (diagseq i))_F n) \longrightarrow l$ **by** (rule LIMSEQ-offset)

thus *?thesis* ..

qed

23.2 Daniell-Kolmogorov Theorem

Existence of Projective Limit

locale *polish-projective* = *projective-family* $I \ P \ \lambda \cdot$. *borel::'a::polish-space measure*
for $I::'i$ set **and** P

begin

lemma *emeasure-lim-emb*:

assumes $X: J \subseteq I$ *finite* $J \ X \in \text{sets } (\Pi_M \ i \in J. \text{borel})$

shows $\text{lim } (\text{emb } I \ J \ X) = P \ J \ X$

proof (rule *emeasure-lim*)

write $\mu\text{-}G$ (μG)

interpret *generator*: *algebra space* $(\text{PiM } I \ (\lambda i. \text{borel}))$ *generator*

by (rule *algebra-generator*)

fix J **and** $B :: \text{nat} \Rightarrow ('i \Rightarrow 'a)$ set

assume $J: \bigwedge n. \text{finite } (J \ n) \ \bigwedge n. J \ n \subseteq I \ \bigwedge n. B \ n \in \text{sets } (\Pi_M \ i \in J \ n. \text{borel})$

incseq J

and $B: \text{decseq } (\lambda n. \text{emb } I \ (J \ n) \ (B \ n))$

and $0 < (\text{INF } i. P \ (J \ i) \ (B \ i))$ (**is** $0 < ?a$)

moreover have $?a \leq 1$

using J **by** (auto *intro!*: *INF-lower2*[of 0] *prob-space-P*[*THEN prob-space.measure-le-1*])

ultimately obtain r **where** $r: ?a = \text{ennreal } r \ 0 < r \ r \leq 1$

by (*cases* $?a$) (auto simp: *top-unique*)

define Z **where** $Z \ n = \text{emb } I \ (J \ n) \ (B \ n)$ **for** n

```

have Z-mono:  $n \leq m \implies Z\ m \subseteq Z\ n$  for  $n\ m$ 
  unfolding Z-def using B[THEN antimonoD, of  $n\ m$ ] .
have J-mono:  $\bigwedge n\ m. n \leq m \implies J\ n \subseteq J\ m$ 
  using ⟨incseq J⟩ by (force simp: incseq-def)
note [simp] = ⟨ $\bigwedge n. \text{finite } (J\ n)$ ⟩
interpret prob-space P (J i) for i using J prob-space-P by simp

have P-eq[simp]:
  sets (P (J i)) = sets ( $\prod_M\ i \in J\ i. \text{borel } \text{space } (P\ (J\ i)) = \text{space } (\prod_M\ i \in J\ i. \text{borel})$ ) for i
  using J by (auto simp: sets-P space-P)

have Z i  $\in$  generator for i
  unfolding Z-def by (auto intro!: generator.intros J)

have countable-UN-J: countable ( $\bigcup n. J\ n$ ) by (simp add: countable-finite)
define Utn where Utn = to-nat-on ( $\bigcup n. J\ n$ )
interpret function-to-finmap J n Utn from-nat-into ( $\bigcup n. J\ n$ ) for n
  by unfold-locales (auto simp: Utn-def intro: from-nat-into-to-nat-on[OF countable-UN-J])
have inj-on-Utn: inj-on Utn ( $\bigcup n. J\ n$ )
  unfolding Utn-def using countable-UN-J by (rule inj-on-to-nat-on)
hence inj-on-Utn-J:  $\bigwedge n. \text{inj-on } Utn\ (J\ n)$  by (rule subset-inj-on) auto
define P' where P' n = mapmeasure n (P (J n)) ( $\lambda \cdot. \text{borel}$ ) for n
interpret P': prob-space P' n for n
  unfolding P'-def mapmeasure-def using J
  by (auto intro!: prob-space-distr fm-measurable simp: measurable-cong-sets[OF sets-P])

let ?SUP =  $\lambda n. \text{SUP } K \in \{K. K \subseteq \text{fm } n\ ' (B\ n) \wedge \text{compact } K\}. \text{emeasure } (P'\ n)\ K$ 
  { fix n
    have emeasure (P (J n)) (B n) = emeasure (P' n) (fm n ' (B n))
      using J by (auto simp: P'-def mapmeasure-PiM space-P sets-P)
    also
    have ... = ?SUP n
    proof (rule inner-regular)
      show sets (P' n) = sets borel by (simp add: borel-eq-PiF-borel P'-def)
    next
      show fm n ' B n  $\in$  sets borel
      unfolding borel-eq-PiF-borel by (auto simp: P'-def fm-image-measurable-finite sets-P J( $\beta$ ))
    qed simp
    finally have *: emeasure (P (J n)) (B n) = ?SUP n .
    have ?SUP n  $\neq \infty$ 
      unfolding *[symmetric] by simp
    note * this
  } note R = this
have  $\forall n. \exists K. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ K \leq 2\ \text{powr } (-n)$ 

```

* $?a \wedge \text{compact } K \wedge K \subseteq \text{fm } n \text{ ' } B \ n$
proof
fix n **show** $\exists K. \text{emeasure } (P \ (J \ n)) \ (B \ n) - \text{emeasure } (P' \ n) \ K \leq \text{ennreal } (2 \ \text{powr} - \text{real } n) * ?a \wedge$
 $\text{compact } K \wedge K \subseteq \text{fm } n \text{ ' } B \ n$
unfolding $R[\text{of } n]$
proof (*rule ccontr*)
assume $H: \nexists K'. ?SUP \ n - \text{emeasure } (P' \ n) \ K' \leq \text{ennreal } (2 \ \text{powr} - \text{real } n) * ?a \wedge$
 $\text{compact } K' \wedge K' \subseteq \text{fm } n \text{ ' } B \ n$
have $?SUP \ n + 0 < ?SUP \ n + 2 \ \text{powr} \ (-n) * ?a$
using $R[\text{of } n]$ **unfolding** *ennreal-add-left-cancel-less ennreal-zero-less-mult-iff*
by (*auto intro: <0 < ?a>*)
also have $\dots = (SUP \ K \in \{K. K \subseteq \text{fm } n \text{ ' } B \ n \wedge \text{compact } K\}. \text{emeasure } (P' \ n) \ K + 2 \ \text{powr} \ (-n) * ?a)$
by (*rule ennreal-SUP-add-left[symmetric]*) *auto*
also have $\dots \leq ?SUP \ n$
proof (*intro SUP-least*)
fix K **assume** $K \in \{K. K \subseteq \text{fm } n \text{ ' } B \ n \wedge \text{compact } K\}$
with H **have** $2 \ \text{powr} \ (-n) * ?a < ?SUP \ n - \text{emeasure } (P' \ n) \ K$
by *auto*
then show $\text{emeasure } (P' \ n) \ K + (2 \ \text{powr} \ (-n)) * ?a \leq ?SUP \ n$
by (*subst (asm) less-diff-eq-ennreal*) (*auto simp: less-top[symmetric]*)
 $R(1)[\text{symmetric}] \ \text{ac-simps}$
qed
finally show *False* **by** *simp*
qed
then obtain K' **where** K' :
 $\bigwedge n. \text{emeasure } (P \ (J \ n)) \ (B \ n) - \text{emeasure } (P' \ n) \ (K' \ n) \leq \text{ennreal } (2 \ \text{powr} - \text{real } n) * ?a$
 $\bigwedge n. \text{compact } (K' \ n) \wedge \bigwedge n. K' \ n \subseteq \text{fm } n \text{ ' } B \ n$
unfolding *choice-iff* **by** *blast*
define K **where** $K \ n = \text{fm } n \text{ ' } K' \ n \cap \text{space } (Pi_M \ (J \ n) \ (\lambda-. \text{borel}))$ **for** n
have $K\text{-sets}: \bigwedge n. K \ n \in \text{sets } (Pi_M \ (J \ n) \ (\lambda-. \text{borel}))$
unfolding $K\text{-def}$
using *compact-imp-closed[OF <compact (K' -)>]*
by (*intro measurable-sets[OF fm-measurable, of - Collect finite]*)
(*auto simp: borel-eq-PiF-borel[symmetric]*)
have $K\text{-B}: \bigwedge n. K \ n \subseteq B \ n$
proof
fix $x \ n$ **assume** $x \in K \ n$
then have $\text{fm-in}: \text{fm } n \ x \in \text{fm } n \text{ ' } B \ n$
using K' **by** (*force simp: K-def*)
show $x \in B \ n$
using $\langle x \in K \ n \rangle K\text{-sets} \ \text{sets.sets-into-space } J(1,2,3)[\text{of } n] \ \text{inj-on-image-mem-iff}[OF \ \text{inj-on-fm}]$
by (*metis (no-types) Int-iff K-def fm-in space-borel*)
qed

```

define  $Z'$  where  $Z' n = \text{emb } I (J n) (K n)$  for  $n$ 
have  $Z'$ :  $\bigwedge n. Z' n \subseteq Z n$ 
  unfolding  $Z'$ -def  $Z$ -def
proof (rule prod-emb-mono, safe)
  fix  $n x$  assume  $x \in K n$ 
  hence  $\text{fm } n x \in K' n x \in \text{space } (Pi_M (J n) (\lambda-. \text{borel}))$ 
    by (simp-all add:  $K$ -def space- $P$ )
  note this(1)
  also have  $K' n \subseteq \text{fm } n ' B n$  by (simp add:  $K'$ )
  finally have  $\text{fm } n x \in \text{fm } n ' B n$  .
  thus  $x \in B n$ 
proof safe
  fix  $y$  assume  $y: y \in B n$ 
  hence  $y \in \text{space } (Pi_M (J n) (\lambda-. \text{borel}))$  using  $J$  sets.sets-into-space[of  $B n$ 
 $P (J n)$ ]
    by (auto simp add: space- $P$  sets- $P$ )
  assume  $\text{fm } n x = \text{fm } n y$ 
  note inj-onD[OF inj-on-fm[OF space-borel],
    OF  $\langle \text{fm } n x = \text{fm } n y \rangle \langle x \in \text{space} \rightarrow \langle y \in \text{space} \rangle$ ]
  with  $y$  show  $x \in B n$  by simp
qed
qed
have  $\bigwedge n. Z' n \in \text{generator}$  using  $J K'(2)$  unfolding  $Z'$ -def
  by (auto intro!: generator.intros measurable-sets[OF fm-measurable[of - Collect
finite]]
    simp:  $K$ -def borel-eq-PiF-borel[symmetric] compact-imp-closed)
define  $Y$  where  $Y n = (\bigcap i \in \{1..n\}. Z' i)$  for  $n$ 
hence  $\bigwedge n k. Y (n + k) \subseteq Y n$  by (induct-tac  $k$ ) (auto simp:  $Y$ -def)
hence  $Y$ -mono:  $\bigwedge n m. n \leq m \implies Y m \subseteq Y n$  by (auto simp: le-iff-add)
have  $Y$ - $Z'$ :  $\bigwedge n. n \geq 1 \implies Y n \subseteq Z' n$  by (auto simp:  $Y$ -def)
hence  $Y$ - $Z$ :  $\bigwedge n. n \geq 1 \implies Y n \subseteq Z n$  using  $Z'$  by auto

have  $Y$ -notempty:  $\bigwedge n. n \geq 1 \implies (Y n) \neq \{\}$ 
proof -
  fix  $n::\text{nat}$  assume  $n \geq 1$  hence  $Y n \subseteq Z n$  by fact
  have  $Y n = (\bigcap i \in \{1..n\}. \text{emb } I (J n) (\text{emb } (J n) (J i) (K i)))$  using  $J$   $J$ -mono
    by (auto simp:  $Y$ -def  $Z'$ -def)
  also have  $\dots = \text{prod-emb } I (\lambda-. \text{borel}) (J n) (\bigcap i \in \{1..n\}. \text{emb } (J n) (J i) (K i))$ 
    using  $\langle n \geq 1 \rangle$ 
    by (subst prod-emb-INT) auto
  finally
  have  $Y$ -emb:
     $Y n = \text{prod-emb } I (\lambda-. \text{borel}) (J n) (\bigcap i \in \{1..n\}. \text{prod-emb } (J n) (\lambda-. \text{borel}) (J i) (K i))$  .
  hence  $Y n \in \text{generator}$  using  $J$   $J$ -mono  $K$ -sets  $\langle n \geq 1 \rangle$ 
    by (auto simp del: prod-emb-INT intro!: generator.intros)
  have  $*$ :  $\mu G (Z n) = P (J n) (B n)$ 
    unfolding  $Z$ -def using  $J$  by (intro mu- $G$ -spec) auto

```

then have $\mu G (Z n) \neq \infty$ by auto
 note *
 moreover have *: $\mu G (Y n) = P (J n) (\bigcap i \in \{Suc\ 0..n\}. prod-emb (J n) (\lambda-. borel) (J i) (K i))$
 unfolding $Y-emb$ using $J\ J-mono\ K-sets\ \langle n \geq 1 \rangle$ by $(subst\ mu-G-spec)$ auto
 then have $\mu G (Y n) \neq \infty$ by auto
 note *
 moreover have $\mu G (Z n - Y n) =$
 $P (J n) (B n - (\bigcap i \in \{Suc\ 0..n\}. prod-emb (J n) (\lambda-. borel) (J i) (K i)))$
 unfolding $Z-def\ Y-emb\ prod-emb-Diff[symmetric]$ using $J\ J-mono\ K-sets\ \langle n \geq 1 \rangle$
 by $(subst\ mu-G-spec)$ $(auto\ intro!: sets.Diff)$
 ultimately
 have $\mu G (Z n) - \mu G (Y n) = \mu G (Z n - Y n)$
 using $J\ J-mono\ K-sets\ \langle n \geq 1 \rangle$
 by $(simp\ only: emeasure-eq-measure\ Z-def)$
 $(auto\ dest!: bspec[where\ x=n] intro!: measure-Diff[symmetric] subsetD[OF\ K-B])$
 $intro!: arg-cong[where\ f=ennreal]$
 $simp: extensional-restrict\ emeasure-eq-measure\ prod-emb-iff\ sets-P\ space-P$
 $ennreal-minus\ measure-nonneg)$
 also have subs: $Z n - Y n \subseteq (\bigcup i \in \{1..n\}. (Z i - Z' i))$
 using $\langle n \geq 1 \rangle$ unfolding $Y-def\ UN-extend-simps(7)$ by $(intro\ UN-mono\ Diff-mono\ Z-mono\ order-refl)$ auto
 have $Z n - Y n \in generator (\bigcup i \in \{1..n\}. (Z i - Z' i)) \in generator$
 using $\langle Z' - \in generator \rangle \langle Z - \in generator \rangle \langle Y - \in generator \rangle$ by auto
 hence $\mu G (Z n - Y n) \leq \mu G (\bigcup i \in \{1..n\}. (Z i - Z' i))$
 using subs generator.additive-increasing[OF positive-mu-G additive-mu-G]
 unfolding increasing-def by auto
 also have $\dots \leq (\sum i \in \{1..n\}. \mu G (Z i - Z' i))$ using $\langle Z - \in generator \rangle \langle Z' - \in generator \rangle$
 by $(intro\ generator.subadditive[OF\ positive-mu-G\ additive-mu-G])$ auto
 also have $\dots \leq (\sum i \in \{1..n\}. 2\ powr\ -real\ i * ?a)$
 proof (rule sum-mono)
 fix i assume $i \in \{1..n\}$ hence $i \leq n$ by simp
 have $\mu G (Z i - Z' i) = \mu G (prod-emb\ I (\lambda-. borel) (J i) (B i - K i))$
 unfolding $Z'-def\ Z-def$ by simp
 also have $\dots = P (J i) (B i - K i)$
 using $J\ K-sets$ by $(subst\ mu-G-spec)$ auto
 also have $\dots = P (J i) (B i) - P (J i) (K i)$
 using $K-sets\ J\ \langle K - \subseteq B \rangle$ by $(simp\ add: emeasure-Diff)$
 also have $\dots = P (J i) (B i) - P' i (K' i)$
 unfolding $K-def\ P'-def$
 by $(auto\ simp: mapmeasure-PiF\ borel-eq-PiF-borel[symmetric]\ compact-imp-closed[OF\ \langle compact\ (K' -) \rangle] space-PiM\ PiE-def)$
 also have $\dots \leq ennreal (2\ powr\ -real\ i) * ?a$ using $K'(1)[of\ i]$.
 finally show $\mu G (Z i - Z' i) \leq (2\ powr\ -real\ i) * ?a$.
 qed

also have $\dots = \text{ennreal } ((\sum_{i \in \{1..n\}} (2 \text{ powr } - \text{enn2real } i)) * \text{enn2real } ?a)$
using r **by** (*simp add: sum-distrib-right ennreal-mult[symmetric]*)
also have $\dots < \text{ennreal } (1 * \text{enn2real } ?a)$
proof (*intro ennreal-lessI mult-strict-right-mono*)
have $(\sum_{i = 1..n} 2 \text{ powr } - \text{real } i) = (\sum_{i = 1..< \text{Suc } n} (1/2) ^ i)$
by (*rule sum.cong*) (*auto simp: powr-realpow powr-divide power-divide*
powr-minus-divide)
also have $\{1..< \text{Suc } n\} = \{..< \text{Suc } n\} - \{0\}$ **by** *auto*
also have $\text{sum } ((\bigwedge (1 / 2)::\text{real})) (\{..< \text{Suc } n\} - \{0\}) =$
 $\text{sum } ((\bigwedge (1 / 2)) (\{..< \text{Suc } n\}) - 1)$ **by** (*auto simp: sum-diff1*)
also have $\dots < 1$ **by** (*subst geometric-sum*) *auto*
finally show $(\sum_{i = 1..n} 2 \text{ powr } - \text{enn2real } i) < 1$ **by** *simp*
qed (*auto simp: r enn2real-positive-iff*)
also have $\dots = ?a$ **by** (*auto simp: r*)
also have $\dots \leq \mu G (Z \ n)$
using J **by** (*auto intro: INF-lower simp: Z-def mu-G-spec*)
finally have $\mu G (Z \ n) - \mu G (Y \ n) < \mu G (Z \ n)$.
hence $R: \mu G (Z \ n) < \mu G (Z \ n) + \mu G (Y \ n)$
using $\langle \mu G (Y \ n) \neq \infty \rangle$ **by** (*auto simp: zero-less-iff-neq-zero*)
then have $\mu G (Y \ n) > 0$
by *simp*
thus $Y \ n \neq \{\}$ **using** *positive-mu-G* **by** (*auto simp add: positive-def*)
qed
hence $\forall n \in \{1..\}. \exists y. y \in Y \ n$ **by** *auto*
then obtain y **where** $y: \bigwedge n. n \geq 1 \implies y \ n \in Y \ n$ **unfolding** *bchoice-iff* **by**
force
 $\{$
fix t **and** $n \ m::\text{nat}$
assume $1 \leq n \ n \leq m$ **hence** $1 \leq m$ **by** *simp*
from $Y\text{-mono}[OF \langle m \geq n \rangle] y[OF \langle 1 \leq m \rangle]$ **have** $y \ m \in Y \ n$ **by** *auto*
also have $\dots \subseteq Z' \ n$ **using** $Y\text{-}Z'[OF \langle 1 \leq n \rangle]$.
finally
have $\text{fm } n (\text{restrict } (y \ m) (J \ n)) \in K' \ n$
unfolding $Z'\text{-def } K\text{-def } \text{prod-emb-iff}$ **by** (*simp add: Z'-def K-def prod-emb-iff*)
moreover have $\text{finmap-of } (J \ n) (\text{restrict } (y \ m) (J \ n)) = \text{finmap-of } (J \ n) (y$
 $m)$
using J **by** (*simp add: fm-def*)
ultimately have $\text{fm } n (y \ m) \in K' \ n$ **by** *simp*
 $\}$ **note** $\text{fm-in-}K'$ $=$ *this*
interpret $\text{finmap-seqs-into-compact } \lambda n. K' (Suc \ n) \ \lambda k. \text{fm } (Suc \ k) (y (Suc \ k))$
borel
proof
fix n **show** $\text{compact } (K' \ n)$ **by** *fact*
next
fix n
from $Y\text{-mono}[of \ n \ \text{Suc } n] y[of \ \text{Suc } n]$ **have** $y (Suc \ n) \in Y (Suc \ n)$ **by** *auto*
also have $\dots \subseteq Z' (Suc \ n)$ **using** $Y\text{-}Z'$ **by** *auto*
finally
have $\text{fm } (Suc \ n) (\text{restrict } (y (Suc \ n)) (J (Suc \ n))) \in K' (Suc \ n)$

```

  unfolding Z'-def K-def prod-emb-iff by (simp add: Z'-def K-def prod-emb-iff)
  thus  $K' (Suc\ n) \neq \{\}$  by auto
  fix k
  assume  $k \in K' (Suc\ n)$ 
  with  $K'[of\ Suc\ n]$  sets.sets-into-space have  $k \in fm (Suc\ n) \text{ ‘ } B (Suc\ n)$  by
  auto
  then obtain b where  $k = fm (Suc\ n) b$  by auto
  thus  $domain\ k = domain (fm (Suc\ n) (y (Suc\ n)))$ 
  by (simp-all add: fm-def)
next
  fix t and  $n\ m::nat$ 
  assume  $n \leq m$  hence  $Suc\ n \leq Suc\ m$  by simp
  assume  $t \in domain (fm (Suc\ n) (y (Suc\ n)))$ 
  then obtain j where  $j: t = Utn\ j\ j \in J (Suc\ n)$  by auto
  hence  $j \in J (Suc\ m)$  using J-mono[OF  $\langle Suc\ n \leq Suc\ m \rangle$ ] by auto
  have  $img: fm (Suc\ n) (y (Suc\ m)) \in K' (Suc\ n)$  using  $\langle n \leq m \rangle$ 
  by (intro fm-in-K') simp-all
  show  $(fm (Suc\ m) (y (Suc\ m)))_F t \in (\lambda k. (k)_F t) \text{ ‘ } K' (Suc\ n)$ 
  apply (rule image-eqI[OF - img])
  using  $\langle j \in J (Suc\ n) \rangle \langle j \in J (Suc\ m) \rangle$ 
  unfolding j by (subst proj-fm, auto)+
qed
have  $\forall t. \exists z. (\lambda i. (fm (Suc (diagseq\ i)) (y (Suc (diagseq\ i))))_F t) \longrightarrow z$ 
  using diagonal-tendsto ..
then obtain z where z:
   $\bigwedge t. (\lambda i. (fm (Suc (diagseq\ i)) (y (Suc (diagseq\ i))))_F t) \longrightarrow z\ t$ 
  unfolding choice-iff by blast
{
  fix  $n::nat$  assume  $n \geq 1$ 
  have  $\bigwedge i. domain (fm\ n\ (y (Suc (diagseq\ i)))) = domain (finmap-of (Utn \text{ ‘ } J\ n)\ z)$ 
  by simp
  moreover
  {
    fix t
    assume  $t: t \in domain (finmap-of (Utn \text{ ‘ } J\ n)\ z)$ 
    hence  $t \in Utn \text{ ‘ } J\ n$  by simp
    then obtain j where  $j: t = Utn\ j\ j \in J\ n$  by auto
    have  $(\lambda i. (fm\ n\ (y (Suc (diagseq\ i))))_F t) \longrightarrow z\ t$ 
    apply (subst (2) tendsto-iff, subst eventually-sequentially)
  }
  proof safe
    fix  $e::real$  assume  $0 < e$ 
    { fix i and  $x::'i \Rightarrow 'a$  assume  $i: i \geq n$ 
      assume  $t \in domain (fm\ n\ x)$ 
      hence  $t \in domain (fm\ i\ x)$  using J-mono[OF  $\langle i \geq n \rangle$ ] by auto
      with i have  $(fm\ i\ x)_F t = (fm\ n\ x)_F t$ 
      using j by (auto simp: proj-fm dest!: inj-onD[OF inj-on-Utn])
    }
    note index-shift = this
    have I:  $\bigwedge i. i \geq n \implies Suc (diagseq\ i) \geq n$ 

```

```

    apply (rule le-SucI)
    apply (rule order-trans) apply simp
    apply (rule seq-suble[OF subseq-diagseq])
    done
  from z
  have  $\exists N. \forall i \geq N. \text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } i)) (y (\text{Suc} (\text{diagseq } i))))_F t) (z t) < e$ 
  unfolding tendsto-iff eventually-sequentially using  $\langle 0 < e \rangle$  by auto
  then obtain N where  $N: \bigwedge i. i \geq N \implies \text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } i)) (y (\text{Suc} (\text{diagseq } i))))_F t) (z t) < e$  by auto
  show  $\exists N. \forall na \geq N. \text{dist} ((\text{fm } n (y (\text{Suc} (\text{diagseq } na))))_F t) (z t) < e$ 
  proof (rule exI[where  $x = \max N n$ ], safe)
    fix na assume  $\max N n \leq na$ 
    hence  $\text{dist} ((\text{fm } n (y (\text{Suc} (\text{diagseq } na))))_F t) (z t) = \text{dist} ((\text{fm} (\text{Suc} (\text{diagseq } na)) (y (\text{Suc} (\text{diagseq } na))))_F t) (z t)$  using t
    by (subst index-shift[OF I]) auto
    also have  $\dots < e$  using  $\langle \max N n \leq na \rangle$  by (intro N) simp
    finally show  $\text{dist} ((\text{fm } n (y (\text{Suc} (\text{diagseq } na))))_F t) (z t) < e$  .
  qed
  qed
  hence  $(\lambda i. (\text{fm } n (y (\text{Suc} (\text{diagseq } i))))_F t) \longrightarrow (\text{finmap-of } (\text{Utn } ' J n) z)_F t$ 
  by (simp add: tendsto-intros)
} ultimately
have  $(\lambda i. \text{fm } n (y (\text{Suc} (\text{diagseq } i)))) \longrightarrow \text{finmap-of } (\text{Utn } ' J n) z$ 
  by (rule tendsto-finmap)
  hence  $((\lambda i. \text{fm } n (y (\text{Suc} (\text{diagseq } i)))) o (\lambda i. i + n)) \longrightarrow \text{finmap-of } (\text{Utn } ' J n) z$ 
  by (rule LIMSEQ-subseq-LIMSEQ) (simp add: strict-mono-def)
moreover
have  $(\forall i. ((\lambda i. \text{fm } n (y (\text{Suc} (\text{diagseq } i)))) o (\lambda i. i + n)) i \in K' n)$ 
  apply (auto simp add: o-def intro!: fm-in-K'  $\langle 1 \leq n \rangle$  le-SucI)
  apply (rule le-trans)
  apply (rule le-add2)
  using seq-suble[OF subseq-diagseq]
  apply auto
  done
moreover
from  $\langle \text{compact } (K' n) \rangle$  have closed  $(K' n)$  by (rule compact-imp-closed)
ultimately
have  $\text{finmap-of } (\text{Utn } ' J n) z \in K' n$ 
  unfolding closed-sequential-limits by blast
also have  $\text{finmap-of } (\text{Utn } ' J n) z = \text{fm } n (\lambda i. z (\text{Utn } i))$ 
  unfolding finmap-eq-iff
proof clarsimp
  fix i assume  $i: i \in J n$ 
  hence  $\text{from-nat-into } (\bigcup n. J n) (\text{Utn } i) = i$ 
  unfolding Utn-def
  by (subst from-nat-into-to-nat-on[OF countable-UN-J]) auto

```



```

    with i show  $z (Utn\ i) = (fm\ n\ (\lambda i. z (Utn\ i)))_F (Utn\ i)$ 
      by (simp add: finmap-eq-iff fm-def compose-def)
    qed
  finally have  $fm\ n\ (\lambda i. z (Utn\ i)) \in K'\ n$  .
  moreover
  let  $?J = \bigcup n. J\ n$ 
  have  $(?J \cap J\ n) = J\ n$  by auto
  ultimately have  $restrict\ (\lambda i. z (Utn\ i))\ (?J \cap J\ n) \in K\ n$ 
    unfolding K-def by (auto simp: space-P space-PiM)
  hence  $restrict\ (\lambda i. z (Utn\ i))\ ?J \in Z'\ n$  unfolding Z'-def
    using J by (auto simp: prod-emb-def PiE-def extensional-def)
  also have  $\dots \subseteq Z\ n$  using Z' by simp
  finally have  $restrict\ (\lambda i. z (Utn\ i))\ ?J \in Z\ n$  .
} note in-Z = this
hence  $(\bigcap_{i \in \{1..\}} Z\ i) \neq \{\}$  by auto
thus  $(\bigcap i. Z\ i) \neq \{\}$ 
  using INT-decseq-offset[OF antimonoI[OF Z-mono]] by simp
qed fact+

```

lemma *measure-lim-emb*:

```

 $J \subseteq I \implies finite\ J \implies X \in sets\ (\prod_M\ i \in J. borel) \implies measure\ lim\ (emb\ I\ J\ X)$ 
=  $measure\ (P\ J)\ X$ 
  unfolding measure-def by (subst emeasure-lim-emb) auto

```

end

```

hide-const (open) PiF
hide-const (open) PiF
hide-const (open) Pi'
hide-const (open) finmap-of
hide-const (open) proj
hide-const (open) domain
hide-const (open) basis-finmap

```

sublocale *polish-projective* $\subseteq P$: *prob-space* *lim*

proof

```

  have  $*$ :  $emb\ I\ \{\}\ \{\lambda x. undefined\} = space\ (\prod_M\ i \in I. borel)$ 
    by (auto simp: prod-emb-def space-PiM)
  interpret prob-space P  $\{\}$ 
  using prob-space-P by simp
  show  $emeasure\ lim\ (space\ lim) = 1$ 
    using emeasure-lim-emb[of  $\{\}\ \{\lambda x. undefined\}$ ] emeasure-space-1
    by (simp add:  $*$  PiM-empty space-P)

```

qed

locale *polish-product-prob-space* =

```

  product-prob-space  $\lambda-. borel::('a::polish-space)$  measure I for  $I::'i$  set

```

sublocale *polish-product-prob-space* $\subseteq P$: *polish-projective* *I* $\lambda J. PiM\ J\ (\lambda-. borel::('a))$

measure)

..

lemma (in *polish-product-prob-space*) *limP-eq-PiM*: *lim* = *PiM I* (λ -. *borel*)
by (*rule PiM-eq*) (*auto simp: emeasure-PiM emeasure-lim-emb*)

end

24 Random Permutations

theory *Random-Permutations*

imports

HOL-Combinatorics.Multiset-Permutations

Probability-Mass-Function

begin

Choosing a set permutation (i.e. a distinct list with the same elements as the set) uniformly at random is the same as first choosing the first element of the list and then choosing the rest of the list as a permutation of the remaining set.

lemma *random-permutation-of-set*:

assumes *finite A A* \neq $\{\}$

shows *pmf-of-set (permutations-of-set A)* =
do {
x \leftarrow *pmf-of-set A*;
xs \leftarrow *pmf-of-set (permutations-of-set (A - {x}))*;
return-pmf (x#xs)
} (**is** *?lhs = ?rhs*)

proof –

from *assms* **have** *permutations-of-set A* = $(\bigcup x \in A. (\#) x \text{ ‘ permutations-of-set } (A - \{x\})$)

by (*simp add: permutations-of-set-nonempty*)

also from *assms* **have** *pmf-of-set ... = ?rhs*

by (*subst pmf-of-set-UN[where n = fact (card A - 1)]*)

(*auto simp: card-image disjoint-family-on-def map-pmf-def [symmetric] map-pmf-of-set-inj*)

finally show *?thesis* .

qed

A generic fold function that takes a function, an initial state, and a set and chooses a random order in which it then traverses the set in the same fashion as a left fold over a list. We first give a recursive definition.

function *fold-random-permutation* :: $(\text{'a} \Rightarrow \text{'b} \Rightarrow \text{'b}) \Rightarrow \text{'b} \Rightarrow \text{'a set} \Rightarrow \text{'b pmf}$

where

fold-random-permutation f x $\{\}$ = *return-pmf x*

| \neg *finite A* \implies *fold-random-permutation f x A* = *return-pmf x*

| *finite A* \implies *A* \neq $\{\}$ \implies

fold-random-permutation f x A =

```

    pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a}))
  by simp-all fastforce
termination proof (relation Wellfounded.measure (λ(-,-,A). card A))
  fix A :: 'a set and f :: 'a ⇒ 'b ⇒ 'b and x :: 'b and y :: 'a
  assume A: finite A A ≠ {} y ∈ set-pmf (pmf-of-set A)
  then have card A > 0 by (simp add: card-gt-0-iff)
  with A show ((f, f y x, A - {y}), f, x, A) ∈ Wellfounded.measure (λ(-, -, A).
card A)
  by simp
qed simp-all

```

We can now show that the above recursive definition is equivalent to choosing a random set permutation and folding over it (in any direction).

```

lemma fold-random-permutation-foldl:
  assumes finite A
  shows fold-random-permutation f x A =
    map-pmf (foldl (λx y. f y x) x) (pmf-of-set (permutations-of-set A))
using assms
proof (induction f x A rule: fold-random-permutation.induct [case-names empty
infinite remove])
  case (remove A f x)
  from remove
  have fold-random-permutation f x A =
    pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a})) by
simp
  also from remove
  have ... = pmf-of-set A ≫ (λa. map-pmf (foldl (λx y. f y x) x)
    (map-pmf ((#) a) (pmf-of-set (permutations-of-set (A - {a}))))))
  by (intro bind-pmf-cong) (simp-all add: pmf.map-comp o-def)
  also from remove have ... = map-pmf (foldl (λx y. f y x) x) (pmf-of-set
(permutations-of-set A))
  by (simp-all add: random-permutation-of-set map-bind-pmf map-pmf-def [symmetric])
  finally show ?case .
qed (simp-all add: pmf-of-set-singleton)

```

```

lemma fold-random-permutation-foldr:
  assumes finite A
  shows fold-random-permutation f x A =
    map-pmf (λxs. foldr f xs x) (pmf-of-set (permutations-of-set A))
proof -
  have fold-random-permutation f x A =
    map-pmf (foldl (λx y. f y x) x ∘ rev) (pmf-of-set (permutations-of-set A))
  using assms by (subst fold-random-permutation-foldl [OF assms])
    (simp-all add: pmf.map-comp [symmetric] map-pmf-of-set-inj)
  also have foldl (λx y. f y x) x ∘ rev = (λxs. foldr f xs x)
  by (intro ext) (simp add: foldl-conv-foldr)
  finally show ?thesis .
qed

```

lemma *fold-random-permutation-fold*:

assumes *finite A*

shows *fold-random-permutation f x A =*

map-pmf (λxs. fold f xs x) (pmf-of-set (permutations-of-set A))

by (*subst fold-random-permutation-foldl [OF assms], intro map-pmf-cong*)
(*simp-all add: foldl-conv-fold*)

lemma *fold-random-permutation-code* [*code*]:

fold-random-permutation f x (set xs) =

map-pmf (foldl (λx y. f y x) x) (pmf-of-set (permutations-of-set (set xs)))

by (*simp add: fold-random-permutation-foldl*)

We now introduce a slightly generalised version of the above fold operation that does not simply return the result in the end, but applies a monadic bind to it. This may seem somewhat arbitrary, but it is a common use case, e.g. in the Social Decision Scheme of Random Serial Dictatorship, where voters narrow down a set of possible winners in a random order and the winner is chosen from the remaining set uniformly at random.

function *fold-bind-random-permutation*

:: ('a ⇒ 'b ⇒ 'b) ⇒ ('b ⇒ 'c pmf) ⇒ 'b ⇒ 'a set ⇒ 'c pmf **where**

fold-bind-random-permutation f g x {} = g x

| *¬finite A ⇒ fold-bind-random-permutation f g x A = g x*

| *finite A ⇒ A ≠ {} ⇒*

fold-bind-random-permutation f g x A =

pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a x) (A - {a}))

by *simp-all fastforce*

termination proof (*relation Wellfounded.measure (λ(-,-,-,A). card A)*)

fix *A :: 'a set and f :: 'a ⇒ 'b ⇒ 'b and x :: 'b*

and *y :: 'a and g :: 'b ⇒ 'c pmf*

assume *A: finite A A ≠ {} y ∈ set-pmf (pmf-of-set A)*

then have *card A > 0* **by** (*simp add: card-gt-0-iff*)

with *A show ((f, g, f y x, A - {y}), f, g, x, A) ∈ Wellfounded.measure (λ(-, -, -, A). card A)*

by *simp*

qed *simp-all*

We now show that the recursive definition is equivalent to a random fold followed by a monadic bind.

lemma *fold-bind-random-permutation-altdef* [*code*]:

fold-bind-random-permutation f g x A = fold-random-permutation f x A ≫ g

proof (*induction f x A rule: fold-random-permutation.induct [case-names empty infinite remove]*)

case (*remove A f x*)

from *remove have pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a x) (A - {a})) =*

pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a})) ≫ g

by (*intro bind-pmf-cong simp-all*)

with *remove* **show** *?case* **by** (*simp* *add*: *bind-return-pmf* *bind-assoc-pmf*)
qed (*simp-all* *add*: *bind-return-pmf*)

We can now derive the following nice monadic representations of the combined fold-and-bind:

lemma *fold-bind-random-permutation-foldl*:
assumes *finite* *A*
shows *fold-bind-random-permutation* *f* *g* *x* *A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set* *A*); *g* (*foldl* ($\lambda x y. f y x$) *x* *xs*)}

using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-foldl *bind-return-pmf* *map-pmf-def*)

lemma *fold-bind-random-permutation-foldr*:
assumes *finite* *A*
shows *fold-bind-random-permutation* *f* *g* *x* *A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set* *A*); *g* (*foldr* *f* *xs* *x*)}

using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-foldr *bind-return-pmf* *map-pmf-def*)

lemma *fold-bind-random-permutation-fold*:
assumes *finite* *A*
shows *fold-bind-random-permutation* *f* *g* *x* *A* =
do {*xs* ← *pmf-of-set* (*permutations-of-set* *A*); *g* (*fold* *f* *xs* *x*)}

using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-fold *bind-return-pmf* *map-pmf-def*)

The following useful lemma allows us to swap partitioning a set w. r. t. a predicate and drawing a random permutation of that set.

lemma *partition-random-permutations*:
assumes *finite* *A*
shows *map-pmf* (*partition* *P*) (*pmf-of-set* (*permutations-of-set* *A*)) =
pair-pmf (*pmf-of-set* (*permutations-of-set* {*x* ∈ *A*. *P* *x*}))
(*pmf-of-set* (*permutations-of-set* {*x* ∈ *A*. $\neg P$ *x*})) (*is* *?lhs* = *?rhs*)

proof (*rule* *pmf-eqI*, *clarify*, *goal-cases*)
case (*1* *xs* *ys*)
show *?case*
proof (*cases* *xs* ∈ *permutations-of-set* {*x* ∈ *A*. *P* *x*} ∧ *ys* ∈ *permutations-of-set*
{*x* ∈ *A*. $\neg P$ *x*})
case *True*
let *?n1* = *card* {*x* ∈ *A*. *P* *x*} **and** *?n2* = *card* {*x* ∈ *A*. $\neg P$ *x*}
have *card-eq*: *card* *A* = *?n1* + *?n2*
proof –
have *?n1* + *?n2* = *card* ({*x* ∈ *A*. *P* *x*} ∪ {*x* ∈ *A*. $\neg P$ *x*})
using *assms* **by** (*intro* *card-Un-disjoint* [*symmetric*]) *auto*
also **have** {*x* ∈ *A*. *P* *x*} ∪ {*x* ∈ *A*. $\neg P$ *x*} = *A* **by** *blast*
finally **show** *?thesis* ..
qed

from *True* **have** *lengths* [*simp*]: *length* *xs* = *?n1* *length* *ys* = *?n2*

```

    by (auto intro!: length-finite-permutations-of-set)
  have pmf ?lhs (xs, ys) =
    real (card (permutations-of-set A ∩ partition P - ‘{(xs, ys)})) / fact
(card A)
  using assms by (auto simp: pmf-map measure-pmf-of-set)
  also have partition P - ‘{(xs, ys)} = shuffles xs ys
  using True by (intro inv-image-partition) (auto simp: permutations-of-set-def)
  also have permutations-of-set A ∩ shuffles xs ys = shuffles xs ys
  using True distinct-disjoint-shuffles[of xs ys]
  by (auto simp: permutations-of-set-def dest: set-shuffles)
  also have card (shuffles xs ys) = length xs + length ys choose length xs
  using True by (intro card-disjoint-shuffles) (auto simp: permutations-of-set-def)
  also have length xs + length ys = card A by (simp add: card-eq)
  also have real (card A choose length xs) = fact (card A) / (fact ?n1 * fact
(card A - ?n1))
  by (subst binomial-fact) (auto intro!: card-mono assms)
  also have ... / fact (card A) = 1 / (fact ?n1 * fact ?n2)
  by (simp add: field-split-simps card-eq)
  also have ... = pmf ?rhs (xs, ys) using True assms by (simp add: pmf-pair)
  finally show ?thesis .
next
case False
hence *: xs ∉ permutations-of-set {x∈A. P x} ∨ ys ∉ permutations-of-set {x∈A.
¬P x} by blast
hence eq: permutations-of-set A ∩ (partition P - ‘{(xs, ys)}) = {}
  by (auto simp: o-def permutations-of-set-def)
from * show ?thesis
  by (elim disjE) (insert assms eq, simp-all add: pmf-pair pmf-map mea-
sure-pmf-of-set)
qed
qed
end

```

25 Discrete subprobability distribution

theory *SPMF* imports

Probability-Mass-Function

HOL-Library.Complete-Partial-Order2

HOL-Library.Rewrite

begin

25.1 Auxiliary material

lemma *cSUP-singleton* [simp]: (*SUP* x∈{x}. f x :: - :: conditionally-complete-lattice)
= f x

by (metis *cSup-singleton image-empty image-insert*)

25.1.1 More about extended reals

lemma *[simp]*:

shows *ennreal-max-0*: $\text{ennreal } (\max 0 x) = \text{ennreal } x$

and *ennreal-max-0'*: $\text{ennreal } (\max x 0) = \text{ennreal } x$

by(*simp-all add: max-def ennreal-eq-0-iff*)

lemma *e2ennreal-0* *[simp]*: $e2\text{ennreal } 0 = 0$

by(*simp add: zero-ennreal-def*)

lemma *enn2real-bot* *[simp]*: $\text{enn2real } \perp = 0$

by(*simp add: bot-ennreal-def*)

lemma *continuous-at-ennreal**[continuous-intros]*: $\text{continuous } F f \implies \text{continuous } F (\lambda x. \text{ennreal } (f x))$

unfolding *continuous-def* **by** *auto*

lemma *ennreal-Sup*:

assumes *: $(\text{SUP } a \in A. \text{ennreal } a) \neq \top$

and $A \neq \{\}$

shows $\text{ennreal } (\text{Sup } A) = (\text{SUP } a \in A. \text{ennreal } a)$

proof (*rule continuous-at-Sup-mono*)

obtain *r* **where** *r*: $\text{ennreal } r = (\text{SUP } a \in A. \text{ennreal } a) \ r \geq 0$

using * **by**(*cases (SUP a ∈ A. ennreal a) simp-all*)

then show *bdd-above A*

by(*auto intro!: SUP-upper bdd-aboveI[of - r] simp add: ennreal-le-iff[symmetric]*)

qed (*auto simp: mono-def continuous-at-imp-continuous-at-within continuous-at-ennreal ennreal-leI assms*)

lemma *ennreal-SUP*:

$\llbracket (\text{SUP } a \in A. \text{ennreal } (f a)) \neq \top; A \neq \{\} \rrbracket \implies \text{ennreal } (\text{SUP } a \in A. f a) = (\text{SUP } a \in A. \text{ennreal } (f a))$

using *ennreal-Sup**[of f ‘ A]* **by** (*auto simp add: image-comp*)

lemma *ennreal-lt-0*: $x < 0 \implies \text{ennreal } x = 0$

by(*simp add: ennreal-eq-0-iff*)

25.1.2 More about 'a option

lemma *None-in-map-option-image* *[simp]*: $\text{None} \in \text{map-option } f \text{ ‘ } A \longleftrightarrow \text{None} \in A$

by *auto*

lemma *Some-in-map-option-image* *[simp]*: $\text{Some } x \in \text{map-option } f \text{ ‘ } A \longleftrightarrow (\exists y. x = f y \wedge \text{Some } y \in A)$

by(*auto intro: rev-image-eqI dest: sym*)

lemma *case-option-collapse*: $\text{case-option } x (\lambda \cdot. x) = (\lambda \cdot. x)$

by(*simp add: fun-eq-iff split: option.split*)

lemma *case-option-id*: *case-option None Some = id*
by(*rule ext*)(*simp split: option.split*)

inductive *ord-option* :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a option \Rightarrow 'b option \Rightarrow bool
for *ord* :: 'a \Rightarrow 'b \Rightarrow bool

where

None: *ord-option ord None x*
| *Some*: *ord x y \Longrightarrow ord-option ord (Some x) (Some y)*

inductive-simps *ord-option-simps* [*simp*]:

ord-option ord None x
ord-option ord x None
ord-option ord (Some x) (Some y)
ord-option ord (Some x) None

inductive-simps *ord-option-eq-simps* [*simp*]:

ord-option (=) None y
ord-option (=) (Some x) y

lemma *ord-option-reflI*: ($\bigwedge y. y \in \text{set-option } x \Longrightarrow \text{ord } y y$) \Longrightarrow *ord-option ord x*
x

by(*cases x*) *simp-all*

lemma *reflp-ord-option*: *reflp ord \Longrightarrow reflp (ord-option ord)*
by(*simp add: reflp-def ord-option-reflI*)

lemma *ord-option-trans*:

$\llbracket \text{ord-option ord } x y; \text{ord-option ord } y z;$
 $\bigwedge a b c. \llbracket a \in \text{set-option } x; b \in \text{set-option } y; c \in \text{set-option } z; \text{ord } a b; \text{ord } b c$
 $\rrbracket \Longrightarrow \text{ord } a c \rrbracket$
 $\Longrightarrow \text{ord-option ord } x z$

by(*auto elim!: ord-option.cases*)

lemma *transp-ord-option*: *transp ord \Longrightarrow transp (ord-option ord)*

unfolding *transp-def* **by**(*blast intro: ord-option-trans*)

lemma *antisymp-ord-option*: *antisymp ord \Longrightarrow antisymp (ord-option ord)*

by(*auto intro!: antisympI elim!: ord-option.cases dest: antisympD*)

lemma *ord-option-chainD*:

Complete-Partial-Order.chain (ord-option ord) Y
 \Longrightarrow *Complete-Partial-Order.chain ord {x. Some x \in Y}*

by(*rule chainI*)(*auto dest: chainD*)

definition *lub-option* :: ('a set \Rightarrow 'b) \Rightarrow 'a option set \Rightarrow 'b option

where *lub-option lub Y = (if Y \subseteq {None} then None else Some (lub {x. Some x \in Y}))*

lemma *map-lub-option*: *map-option f (lub-option lub Y) = lub-option (f \circ lub) Y*

by(*simp add: lub-option-def*)

lemma *lub-option-upper*:

assumes *Complete-Partial-Order.chain* (*ord-option ord*) Y $x \in Y$
and *lub-upper*: $\bigwedge Y x. \llbracket \text{Complete-Partial-Order.chain } \text{ord } Y; x \in Y \rrbracket \implies \text{ord } x$
(*lub Y*)
shows *ord-option ord x* (*lub-option lub Y*)
using *assms(1-2)*
by(*cases x*)(*auto simp add: lub-option-def intro: lub-upper[OF ord-option-chainD]*)

lemma *lub-option-least*:

assumes Y : *Complete-Partial-Order.chain* (*ord-option ord*) Y
and *upper*: $\bigwedge x. x \in Y \implies \text{ord-option ord } x$ y
assumes *lub-least*: $\bigwedge Y y. \llbracket \text{Complete-Partial-Order.chain } \text{ord } Y; \bigwedge x. x \in Y \implies$
ord x y $\rrbracket \implies \text{ord } (\text{lub } Y) y$
shows *ord-option ord* (*lub-option lub Y*) y
using Y
by(*cases y*)(*auto 4 3 simp add: lub-option-def intro: lub-least[OF ord-option-chainD]*
dest: upper)

lemma *lub-map-option*: *lub-option lub* (*map-option f ' Y*) = *lub-option* (*lub* \circ (\cdot)
f) Y

apply(*auto simp add: lub-option-def*)
apply(*erule notE*)
apply(*rule arg-cong[where f=lub]*)
apply(*auto intro: rev-image-eqI dest: sym*)
done

lemma *ord-option-mono*: $\llbracket \text{ord-option } A \ x \ y; \bigwedge x \ y. A \ x \ y \implies B \ x \ y \rrbracket \implies$
ord-option B x y
by(*auto elim: ord-option.cases*)

lemma *ord-option-mono' [mono]*:

($\bigwedge x \ y. A \ x \ y \longrightarrow B \ x \ y$) $\implies \text{ord-option } A \ x \ y \longrightarrow \text{ord-option } B \ x \ y$
by(*blast intro: ord-option-mono*)

lemma *ord-option-compp*: *ord-option* ($A \ OO \ B$) = *ord-option A OO ord-option B*
by(*auto simp add: fun-eq-iff elim!: ord-option.cases intro: ord-option.intros*)

lemma *ord-option-inf*: *inf* (*ord-option A*) (*ord-option B*) = *ord-option* (*inf A B*)
(**is** *?lhs = ?rhs*)

proof(*rule antisym*)
show *?lhs* \leq *?rhs* **by**(*auto elim!: ord-option.cases*)
qed(*auto elim: ord-option-mono*)

lemma *ord-option-map2*: *ord-option ord x* (*map-option f y*) = *ord-option* ($\lambda x \ y.$
ord x (*f y*)) $x \ y$
by(*auto elim: ord-option.cases*)

lemma *ord-option-map1*: $\text{ord-option ord (map-option f x) y = ord-option } (\lambda x y. \text{ord (f x) y) x y$
by(*auto elim: ord-option.cases*)

lemma *option-ord-Some1-iff*: $\text{option-ord (Some x) y} \longleftrightarrow y = \text{Some x}$
by(*auto simp add: flat-ord-def*)

25.1.3 A relator for sets that treats sets like predicates

context includes *lifting-syntax*
begin

definition *rel-pred* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow \text{bool}$
where $\text{rel-pred } R A B = (R \text{ ===> } (=)) (\lambda x. x \in A) (\lambda y. y \in B)$

lemma *rel-predI*: $(R \text{ ===> } (=)) (\lambda x. x \in A) (\lambda y. y \in B) \Longrightarrow \text{rel-pred } R A B$
by(*simp add: rel-pred-def*)

lemma *rel-predD*: $\llbracket \text{rel-pred } R A B; R x y \rrbracket \Longrightarrow x \in A \longleftrightarrow y \in B$
by(*simp add: rel-pred-def rel-fun-def*)

lemma *Collect-parametric*: $((A \text{ ===> } (=)) \text{ ===> rel-pred } A) \text{ Collect Collect}$
 — Declare this rule as *transfer-rule* only locally because it blows up the search space for *transfer* (in combination with *Collect-transfer*)
by(*simp add: rel-funI rel-predI*)

end

25.1.4 Monotonicity rules

lemma *monotone-gfp-eadd1*: $\text{monotone } (\geq) (\geq) (\lambda x. x + y :: \text{enat})$
by(*auto intro!: monotoneI*)

lemma *monotone-gfp-eadd2*: $\text{monotone } (\geq) (\geq) (\lambda y. x + y :: \text{enat})$
by(*auto intro!: monotoneI*)

lemma *mono2mono-gfp-eadd*[*THEN* *gfp.mono2mono2*, *cont-intro*, *simp*]:
shows *monotone-eadd*: $\text{monotone (rel-prod } (\geq) (\geq)) (\geq) (\lambda(x, y). x + y :: \text{enat})$
by(*simp add: monotone-gfp-eadd1 monotone-gfp-eadd2*)

lemma *eadd-gfp-partial-function-mono* [*partial-function-mono*]:
 $\llbracket \text{monotone (fun-ord } (\geq)) (\geq) f; \text{monotone (fun-ord } (\geq)) (\geq) g \rrbracket$
 $\Longrightarrow \text{monotone (fun-ord } (\geq)) (\geq) (\lambda x. f x + g x :: \text{enat})$
by(*rule mono2mono-gfp-eadd*)

lemma *mono2mono-ereal*[*THEN* *lfp.mono2mono*]:
shows *monotone-ereal*: $\text{monotone } (\leq) (\leq) \text{ereal}$
by(*rule monotoneI simp*)

lemma *mono2mono-ennreal*[*THEN* *lfp.mono2mono*]:

shows *monotone-ennreal*: *monotone* (\leq) (\leq) *ennreal*
by(*rule monotoneI*)(*simp add: ennreal-leI*)

25.1.5 Bijections

lemma *bi-unique-rel-set-bij-betw*:

assumes *unique*: *bi-unique* R

and *rel*: *rel-set* R A B

shows $\exists f. \text{bij-betw } f \ A \ B \wedge (\forall x \in A. R \ x \ (f \ x))$

proof –

from *assms* **obtain** f **where** $f: \bigwedge x. x \in A \implies R \ x \ (f \ x)$ **and** $B: \bigwedge x. x \in A \implies f \ x \in B$

apply(*atomize-elim*)

apply(*fold all-conj-distrib*)

apply(*subst choice-iff[symmetric]*)

apply(*auto dest: rel-setD1*)

done

have *inj-on* $f \ A$ **by**(*rule inj-onI*)(*auto dest!: f dest: bi-uniqueDl[OF unique]*)

moreover **have** $f \ 'A = B$ **using** *rel*

by(*auto 4 3 intro: B dest: rel-setD2 f bi-uniqueDr[OF unique]*)

ultimately **have** *bij-betw* $f \ A \ B$ **unfolding** *bij-betw-def* **..**

thus *?thesis* **using** f **by** *blast*

qed

lemma *bij-betw-rel-setD*: *bij-betw* $f \ A \ B \implies \text{rel-set } (\lambda x \ y. y = f \ x) \ A \ B$

by(*rule rel-setI*)(*auto dest: bij-betwE bij-betw-imp-surj-on[symmetric]*)

25.2 Subprobability mass function

type-synonym $'a \ \text{spmf} = 'a \ \text{option} \ \text{pmf}$

translations (*type*) $'a \ \text{spmf} \leftarrow (\text{type}) \ 'a \ \text{option} \ \text{pmf}$

definition *measure-spmf* $:: 'a \ \text{spmf} \Rightarrow 'a \ \text{measure}$

where *measure-spmf* $p = \text{distr } (\text{restrict-space } (\text{measure-pmf } p) \ (\text{range } \text{Some}))$
(*count-space UNIV*) *the*

abbreviation *spmf* $:: 'a \ \text{spmf} \Rightarrow 'a \Rightarrow \text{real}$

where *spmf* $p \ x \equiv \text{pmf } p \ (\text{Some } x)$

lemma *space-measure-spmf*: *space* (*measure-spmf* p) = *UNIV*

by(*simp add: measure-spmf-def*)

lemma *sets-measure-spmf* [*simp*, *measurable-cong*]: *sets* (*measure-spmf* p) = *sets*
(*count-space UNIV*)

by(*simp add: measure-spmf-def*)

lemma *measure-spmf-not-bot* [*simp*]: *measure-spmf* $p \neq \perp$

proof

assume *measure-spmf* $p = \perp$

hence *space* (*measure-spmf* p) = *space* \perp **by** *simp*

thus *False* **by**(*simp add: space-measure-spmf*)
qed

lemma *measurable-the-measure-pmf-Some* [*measurable, simp*]:
the \in *measurable* (*restrict-space* (*measure-pmf p*) (*range Some*)) (*count-space UNIV*)
by(*auto simp add: measurable-def sets-restrict-space space-restrict-space integral-restrict-space*)

lemma *measurable-spmf-measure1* [*simp*]: *measurable* (*measure-spmf M*) $N = UNIV$
 \rightarrow *space N*
by(*auto simp: measurable-def space-measure-spmf*)

lemma *measurable-spmf-measure2* [*simp*]: *measurable N* (*measure-spmf M*) = *measurable N* (*count-space UNIV*)
by(*intro measurable-cong-sets simp-all*)

lemma *subprob-space-measure-spmf* [*simp, intro!*]: *subprob-space* (*measure-spmf p*)
proof

show *emeasure* (*measure-spmf p*) (*space* (*measure-spmf p*)) ≤ 1
by(*simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space measure-pmf.measure-le-1*)
qed(*simp add: space-measure-spmf*)

interpretation *measure-spmf: subprob-space measure-spmf p for p*
by(*rule subprob-space-measure-spmf*)

lemma *finite-measure-spmf* [*simp*]: *finite-measure* (*measure-spmf p*)
by *unfold-locales*

lemma *spmconv-measure-spmf*: *spmconv p x = measure* (*measure-spmf p*) $\{x\}$
by(*auto simp add: measure-spmf-def measure-distr measure-restrict-space pmf.rep-eq space-restrict-space intro: arg-cong2[where f=measure]*)

lemma *emeasure-measure-spmf-conv-measure-pmf*:
emeasure (*measure-spmf p*) $A =$ *emeasure* (*measure-pmf p*) (*Some ' A*)
by(*auto simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-measure-spmf-conv-measure-pmf*:
measure (*measure-spmf p*) $A =$ *measure* (*measure-pmf p*) (*Some ' A*)
using *emeasure-measure-spmf-conv-measure-pmf[of p A]*
by(*simp add: measure-spmf.emeasure-eq-measure measure-pmf.emeasure-eq-measure*)

lemma *emeasure-spmf-map-pmf-Some* [*simp*]:
emeasure (*measure-spmf* (*map-pmf Some p*)) $A =$ *emeasure* (*measure-pmf p*) A
by(*auto simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-spmf-map-pmf-Some* [*simp*]:

$measure (measure\text{-}spmf (map\text{-}pmf\ Some\ p))\ A = measure (measure\text{-}pmf\ p)\ A$
using $emeasure\text{-}spmf\text{-}map\text{-}pmf\text{-}Some[of\ p\ A]$ **by** $(simp\ add:\ measure\text{-}spmf.emeasure\text{-}eq\text{-}measure\ measure\text{-}pmf.emeasure\text{-}eq\text{-}measure)$

lemma $nn\text{-}integral\text{-}measure\text{-}spmf$: $(\int^+ x. f\ x\ \partial measure\text{-}spmf\ p) = \int^+ x. ennreal$
 $(spmf\ p\ x) * f\ x\ \partial count\text{-}space\ UNIV$
(is $?lhs = ?rhs)$

proof –

have $?lhs = \int^+ x. pmf\ p\ x * f\ (the\ x)\ \partial count\text{-}space\ (range\ Some)$
by $(simp\ add:\ measure\text{-}spmf\text{-}def\ nn\text{-}integral\text{-}distr\ nn\text{-}integral\text{-}restrict\text{-}space\ nn\text{-}integral\text{-}measure\text{-}pmf\ nn\text{-}integral\text{-}count\text{-}space\ indicator\ ac\text{-}simps\ times\ ereal.simps(1)[symmetric]\ del:\ times\ ereal.simps(1))$
also have $\dots = \int^+ x. ennreal (spmf\ p\ (the\ x)) * f\ (the\ x)\ \partial count\text{-}space\ (range\ Some)$
by $(rule\ nn\text{-}integral\text{-}cong)\ auto$
also have $\dots = \int^+ x. spmf\ p\ (the\ (Some\ x)) * f\ (the\ (Some\ x))\ \partial count\text{-}space\ UNIV$
by $(rule\ nn\text{-}integral\text{-}bij\text{-}count\text{-}space[symmetric])(simp\ add:\ bij\text{-}betw\text{-}def)$
also have $\dots = ?rhs$ **by** $simp$
finally show $?thesis .$

qed

lemma $integral\text{-}measure\text{-}spmf$:

assumes $integrable (measure\text{-}spmf\ p)\ f$
shows $(\int x. f\ x\ \partial measure\text{-}spmf\ p) = \int x. spmf\ p\ x * f\ x\ \partial count\text{-}space\ UNIV$
proof –
have $integrable (count\text{-}space\ UNIV) (\lambda x. spmf\ p\ x * f\ x)$
using $assms$ **by** $(simp\ add:\ integrable\text{-}iff\text{-}bounded\ nn\text{-}integral\text{-}measure\text{-}spmf\ abs\text{-}mult\ ennreal\text{-}mult')$
then show $?thesis$ **using** $assms$
by $(simp\ add:\ real\text{-}lebesgue\text{-}integral\text{-}def\ nn\text{-}integral\text{-}measure\text{-}spmf\ ennreal\text{-}mult'[symmetric])$

qed

lemma $emeasure\text{-}spmf\text{-}single$: $emeasure (measure\text{-}spmf\ p)\ \{x\} = spmf\ p\ x$
by $(simp\ add:\ measure\text{-}spmf.emeasure\text{-}eq\text{-}measure\ spmf\text{-}conv\text{-}measure\text{-}spmf)$

lemma $measurable\text{-}measure\text{-}spmf[measurable]$:

$(\lambda x. measure\text{-}spmf\ (M\ x)) \in measurable (count\text{-}space\ UNIV) (subprob\text{-}algebra (count\text{-}space\ UNIV))$
by $(auto\ simp:\ space\text{-}subprob\text{-}algebra)$

lemma $nn\text{-}integral\text{-}measure\text{-}spmf\text{-}conv\text{-}measure\text{-}pmf$:

assumes $[measurable]: f \in borel\text{-}measurable (count\text{-}space\ UNIV)$
shows $nn\text{-}integral (measure\text{-}spmf\ p)\ f = nn\text{-}integral (restrict\text{-}space (measure\text{-}pmf\ p) (range\ Some)) (f \circ the)$
by $(simp\ add:\ measure\text{-}spmf\text{-}def\ nn\text{-}integral\text{-}distr\ o\text{-}def)$

lemma $measure\text{-}spmf\text{-}in\text{-}space\text{-}subprob\text{-}algebra [simp]$:

$measure\text{-}spmf\ p \in space (subprob\text{-}algebra (count\text{-}space\ UNIV))$
by $(simp\ add:\ space\text{-}subprob\text{-}algebra)$

lemma *nn-integral-spmf-neq-top*: $(\int^+ x. \text{spmf } p \ x \ \partial \text{count-space UNIV}) \neq \top$
using *nn-integral-measure-spmf*[**where** $f=\lambda. 1$, of p , symmetric] **by** *simp*

lemma *SUP-spmf-neq-top'*: $(\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)) \neq \top$
proof(*rule neq-top-trans*)
show $(\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)) \leq 1$ **by**(*rule SUP-least*)(*simp add: pmf-le-1*)
qed *simp*

lemma *SUP-spmf-neq-top*: $(\text{SUP } i. \text{ennreal } (\text{spmf } (Y \ i) \ x)) \neq \top$
proof(*rule neq-top-trans*)
show $(\text{SUP } i. \text{ennreal } (\text{spmf } (Y \ i) \ x)) \leq 1$ **by**(*rule SUP-least*)(*simp add: pmf-le-1*)
qed *simp*

lemma *SUP-emeasure-spmf-neq-top*: $(\text{SUP } p \in Y. \text{emeasure } (\text{measure-spmf } p) \ A) \neq \top$
proof(*rule neq-top-trans*)
show $(\text{SUP } p \in Y. \text{emeasure } (\text{measure-spmf } p) \ A) \leq 1$
by(*rule SUP-least*)(*simp add: measure-spmf.subprob-emeasure-le-1*)
qed *simp*

25.3 Support

definition *set-spmf* :: $'a \ \text{spmf} \Rightarrow 'a \ \text{set}$
where *set-spmf* $p = \text{set-pmf } p \gg= \text{set-option}$

lemma *set-spmf-rep-eq*: $\text{set-spmf } p = \{x. \text{measure } (\text{measure-spmf } p) \ \{x\} \neq 0\}$
proof –
have $\bigwedge x :: 'a. \text{the } -' \ \{x\} \cap \text{range } \text{Some} = \{\text{Some } x\}$ **by** *auto*
then show *?thesis*
by(*auto simp add: set-spmf-def set-pmf.rep-eq measure-spmf-def measure-distr measure-restrict-space space-restrict-space intro: rev-image-eqI*)
qed

lemma *in-set-spmf*: $x \in \text{set-spmf } p \iff \text{Some } x \in \text{set-pmf } p$
by(*simp add: set-spmf-def*)

lemma *AE-measure-spmf-iff* [*simp*]: $(\text{AE } x \ \text{in } \text{measure-spmf } p. \ P \ x) \iff (\forall x \in \text{set-spmf } p. \ P \ x)$
by(*auto 4 3 simp add: measure-spmf-def AE-distr-iff AE-restrict-space-iff AE-measure-pmf-iff set-spmf-def cong del: AE-cong*)

lemma *spmf-eq-0-set-spmf*: $\text{spmf } p \ x = 0 \iff x \notin \text{set-spmf } p$
by(*auto simp add: pmf-eq-0-set-pmf set-spmf-def intro: rev-image-eqI*)

lemma *in-set-spmf-iff-spmf*: $x \in \text{set-spmf } p \iff \text{spmf } p \ x \neq 0$
by(*auto simp add: set-spmf-def set-pmf-iff intro: rev-image-eqI*)

lemma *set-spmf-return-pmf-None* [*simp*]: *set-spmf* (*return-pmf None*) = {}
by(*auto simp add: set-spmf-def*)

lemma *countable-set-spmf* [*simp*]: *countable* (*set-spmf p*)
by(*simp add: set-spmf-def bind-UNION*)

lemma *spmf-eqI*:

assumes $\bigwedge i. \text{spmf } p \ i = \text{spmf } q \ i$
shows $p = q$
proof(*rule pmf-eqI*)
fix i
show $\text{pmf } p \ i = \text{pmf } q \ i$
proof(*cases i*)
case (*Some i'*)
thus *?thesis* **by**(*simp add: assms*)
next
case *None*
have $\text{ennreal } (\text{pmf } p \ i) = \text{measure } (\text{measure-pmf } p) \ \{i\}$ **by**(*simp add: pmf-def*)
also have $\{i\} = \text{space } (\text{measure-pmf } p) - \text{range } \text{Some}$
by(*auto simp add: None intro: ccontr*)
also have $\text{measure } (\text{measure-pmf } p) \ \dots = \text{ennreal } 1 - \text{measure } (\text{measure-pmf } p) \ (\text{range } \text{Some})$
by(*simp add: measure-pmf.prob-compl ennreal-minus[symmetric] del: space-measure-pmf*)
also have $\text{range } \text{Some} = (\bigcup x \in \text{set-spmf } p. \ \{\text{Some } x\}) \cup \text{Some } '(- \text{set-spmf } p)$
by *auto*
also have $\text{measure } (\text{measure-pmf } p) \ \dots = \text{measure } (\text{measure-pmf } p) \ (\bigcup x \in \text{set-spmf } p. \ \{\text{Some } x\})$
by(*rule measure-pmf.measure-zero-union*)(*auto simp add: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff*)
also have $\text{ennreal } \dots = \int^+ x. \ \text{measure } (\text{measure-pmf } p) \ \{\text{Some } x\} \ \partial \text{count-space } (\text{set-spmf } p)$
unfolding *measure-pmf.emmeasure-eq-measure[symmetric]*
by(*simp-all add: emeasure-UN-countable disjoint-family-on-def*)
also have $\dots = \int^+ x. \ \text{spmf } p \ x \ \partial \text{count-space } (\text{set-spmf } p)$ **by**(*simp add: pmf-def*)
also have $\dots = \int^+ x. \ \text{spmf } q \ x \ \partial \text{count-space } (\text{set-spmf } p)$ **by**(*simp add: assms*)
also have $\text{set-spmf } p = \text{set-spmf } q$ **by**(*auto simp add: in-set-spmf-iff-spmf assms*)
also have $(\int^+ x. \ \text{spmf } q \ x \ \partial \text{count-space } (\text{set-spmf } q)) = \int^+ x. \ \text{measure } (\text{measure-pmf } q) \ \{\text{Some } x\} \ \partial \text{count-space } (\text{set-spmf } q)$
by(*simp add: pmf-def*)
also have $\dots = \text{measure } (\text{measure-pmf } q) \ (\bigcup x \in \text{set-spmf } q. \ \{\text{Some } x\})$
unfolding *measure-pmf.emmeasure-eq-measure[symmetric]*
by(*simp-all add: emeasure-UN-countable disjoint-family-on-def*)
also have $\dots = \text{measure } (\text{measure-pmf } q) \ ((\bigcup x \in \text{set-spmf } q. \ \{\text{Some } x\}) \cup \text{Some } '(- \text{set-spmf } q))$
by(*rule ennreal-cong measure-pmf.measure-zero-union[symmetric]*)+(*auto simp add: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff*)
also have $((\bigcup x \in \text{set-spmf } q. \ \{\text{Some } x\}) \cup \text{Some } '(- \text{set-spmf } q)) = \text{range}$

Some by auto

also have $\text{ennreal } 1 - \text{measure } (\text{measure-pmf } q) \dots = \text{measure } (\text{measure-pmf } q)$ ($\text{space } (\text{measure-pmf } q) - \text{range } \text{Some}$)
by($\text{simp add: one-ereal-def measure-pmf.prob-compl ennreal-minus[symmetric]}$)
 $\text{del: space-measure-pmf}$)
also have $\text{space } (\text{measure-pmf } q) - \text{range } \text{Some} = \{i\}$
by($\text{auto simp add: None intro: ccontr}$)
also have $\text{measure } (\text{measure-pmf } q) \dots = \text{pmf } q \ i$ **by**(simp add: pmf-def)
finally show $?thesis$ **by** simp
qed
qed

lemma $\text{integral-measure-spmf-restrict}$:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$ **shows**
 $(\int x. f \ x \ \partial \text{measure-spmf } M) = (\int x. f \ x \ \partial \text{restrict-space } (\text{measure-spmf } M)$
 $(\text{set-spmf } M))$
by($\text{auto intro!: integral-cong-AE simp add: integral-restrict-space}$)

lemma $\text{nn-integral-measure-spmf'}$:

$(\int^+ x. f \ x \ \partial \text{measure-spmf } p) = \int^+ x. \text{ennreal } (\text{spmfs } p \ x) * f \ x \ \partial \text{count-space}$
 $(\text{set-spmf } p)$
by($\text{auto simp add: nn-integral-measure-spmf nn-integral-count-space-indicator in-set-spmf-iff-spmf}$
 $\text{intro!: nn-integral-cong split: split-indicator}$)

25.4 Functorial structure

abbreviation $\text{map-spmf} :: ('a \Rightarrow 'b) \Rightarrow 'a \ \text{spmfs} \Rightarrow 'b \ \text{spmfs}$
where $\text{map-spmf } f \equiv \text{map-pmf } (\text{map-option } f)$

context begin

local-setup $\langle \text{Local-Theory.map-background-naming } (\text{Name-Space.mandatory-path } \text{spmfs}) \rangle$

lemma map-comp : $\text{map-spmf } f (\text{map-spmf } g \ p) = \text{map-spmf } (f \circ g) \ p$
by($\text{simp add: pmf.map-comp o-def option.map-comp}$)

lemma map-id0 : $\text{map-spmf } \text{id} = \text{id}$
by($\text{simp add: pmf.map-id option.map-id0}$)

lemma map-id [simp] : $\text{map-spmf } \text{id} \ p = p$
by(simp add: map-id0)

lemma map-ident [simp] : $\text{map-spmf } (\lambda x. x) \ p = p$
by($\text{simp add: id-def[symmetric]}$)

end

lemma $\text{set-map-spmf [simp]}$: $\text{set-spmf } (\text{map-spmf } f \ p) = f \ \text{set-spmf } p$
by($\text{simp add: set-spmf-def image-bind bind-image o-def Option.option.set-map}$)

lemma *map-spmf-cong*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f x = g x \rrbracket$
 $\implies \text{map-spmf } f p = \text{map-spmf } g q$

by(*auto intro: pmf.map-cong option.map-cong simp add: in-set-spmf*)

lemma *map-spmf-cong-simp*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q = \text{simp} \implies f x = g x \rrbracket$
 $\implies \text{map-spmf } f p = \text{map-spmf } g q$

unfolding *simp-implies-def* **by**(*rule map-spmf-cong*)

lemma *map-spmf-idI*: $(\bigwedge x. x \in \text{set-spmf } p \implies f x = x) \implies \text{map-spmf } f p = p$

by(*rule map-pmf-idI map-option-idI*)+(*simp add: in-set-spmf*)

lemma *emeasure-map-spmf*:

$\text{emeasure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{emeasure } (\text{measure-spmf } p) (f - ' A)$

by(*auto simp add: measure-spmf-def emeasure-distr measurable-restrict-space1 space-restrict-space emeasure-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *measure-map-spmf*: $\text{measure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{measure } (\text{measure-spmf } p) (f - ' A)$

using *emeasure-map-spmf[of f p A]* **by**(*simp add: measure-spmf.emeasure-eq-measure*)

lemma *measure-map-spmf-conv-distr*:

$\text{measure-spmf } (\text{map-spmf } f p) = \text{distr } (\text{measure-spmf } p) (\text{count-space UNIV}) f$

by(*rule measure-eqI*)(*simp-all add: emeasure-map-spmf emeasure-distr*)

lemma *spmf-map-pmf-Some* [*simp*]: $\text{spmf } (\text{map-pmf } \text{Some } p) i = \text{pmf } p i$

by(*simp add: pmf-map-inj'*)

lemma *spmf-map-inj*: $\llbracket \text{inj-on } f (\text{set-spmf } M); x \in \text{set-spmf } M \rrbracket \implies \text{spmf } (\text{map-spmf } f M) (f x) = \text{spmf } M x$

by(*subst option.map(2)[symmetric, where f=f]*)(*rule pmf-map-inj, auto simp add: in-set-spmf inj-on-def elim!: option.inj-map-strong[rotated]*)

lemma *spmf-map-inj'*: $\text{inj } f \implies \text{spmf } (\text{map-spmf } f M) (f x) = \text{spmf } M x$

by(*subst option.map(2)[symmetric, where f=f]*)(*rule pmf-map-inj'[OF option.inj-map]*)

lemma *spmf-map-outside*: $x \notin f - ' \text{set-spmf } M \implies \text{spmf } (\text{map-spmf } f M) x = 0$

unfolding *spmf-eq-0-set-spmf* **by** *simp*

lemma *ennreal-spmf-map*: $\text{ennreal } (\text{spmf } (\text{map-spmf } f p) x) = \text{emeasure } (\text{measure-spmf } p) (f - ' \{x\})$

by(*auto simp add: ennreal-pmf-map measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space intro: arg-cong2[where f=emeasure]*)

lemma *spmf-map*: $\text{spmf } (\text{map-spmf } f p) x = \text{measure } (\text{measure-spmf } p) (f - ' \{x\})$

using *ennreal-spmf-map[of f p x]* **by**(*simp add: measure-spmf.emeasure-eq-measure*)

lemma *ennreal-spmf-map-conv-nn-integral*:

$ennreal (spmf (map-spmf f p) x) = integral^N (measure-spmf p) (indicator (f - \{x\}))$

by(*auto simp add: ennreal-pmf-map measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space intro: arg-cong2[where f=emeasure]*)

25.5 Monad operations

25.5.1 Return

abbreviation *return-spmf* :: 'a \Rightarrow 'a *spmf*

where *return-spmf* x \equiv *return-pmf* (Some x)

lemma *pmf-return-spmf*: *pmf* (*return-spmf* x) y = *indicator* {y} (Some x)

by(*fact pmf-return*)

lemma *measure-spmf-return-spmf*: *measure-spmf* (*return-spmf* x) = *Giry-Monad.return* (*count-space UNIV*) x

by(*rule measure-eqI*)(*simp-all add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space indicator-def*)

lemma *measure-spmf-return-pmf-None* [*simp*]: *measure-spmf* (*return-pmf* None) = *null-measure* (*count-space UNIV*)

by(*rule measure-eqI*)(*auto simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space indicator-eq-0-iff*)

lemma *set-return-spmf* [*simp*]: *set-spmf* (*return-spmf* x) = {x}

by(*auto simp add: set-spmf-def*)

25.5.2 Bind

definition *bind-spmf* :: 'a *spmf* \Rightarrow ('a \Rightarrow 'b *spmf*) \Rightarrow 'b *spmf*

where *bind-spmf* x f = *bind-pmf* x ($\lambda a. \text{case } a \text{ of } None \Rightarrow \text{return-pmf } None \mid \text{Some } a' \Rightarrow f a'$)

adhoc-overloading *Monad-Syntax.bind* *bind-spmf*

lemma *return-None-bind-spmf* [*simp*]: *return-pmf* None $\gg=$ (f :: 'a \Rightarrow -) = *return-pmf* None

by(*simp add: bind-spmf-def bind-return-pmf*)

lemma *return-bind-spmf* [*simp*]: *return-spmf* x $\gg=$ f = f x

by(*simp add: bind-spmf-def bind-return-pmf*)

lemma *bind-return-spmf* [*simp*]: x $\gg=$ *return-spmf* = x

proof –

have $\bigwedge a :: 'a \text{ option. } (\text{case } a \text{ of } None \Rightarrow \text{return-pmf } None \mid \text{Some } a' \Rightarrow \text{return-spmf } a') = \text{return-pmf } a$

by(*simp split: option.split*)

then show *?thesis*
by(*simp add: bind-spmf-def bind-return-pmf'*)
qed

lemma *bind-spmf-assoc [simp]*:
fixes $x :: 'a \text{ spmf}$ **and** $f :: 'a \Rightarrow 'b \text{ spmf}$ **and** $g :: 'b \Rightarrow 'c \text{ spmf}$
shows $(x \ggg f) \ggg g = x \ggg (\lambda y. f y \ggg g)$
by(*auto simp add: bind-spmf-def bind-assoc-pmf fun-eq-iff bind-return-pmf split: option.split intro: arg-cong[where f=bind-pmf x]*)

lemma *pmf-bind-spmf-None*: $\text{pmf } (p \ggg f) \text{ None} = \text{pmf } p \text{ None} + \int x. \text{pmf } (f x) \text{ None} \partial \text{measure-spmf } p$
(is *?lhs = ?rhs*)

proof –

let $?f = \lambda x. \text{pmf } (\text{case } x \text{ of None} \Rightarrow \text{return-pmf None} \mid \text{Some } x \Rightarrow f x) \text{ None}$
have $?lhs = \int x. ?f x \partial \text{measure-pmf } p$
by(*simp add: bind-spmf-def pmf-bind*)
also have $\dots = \int x. ?f \text{ None} * \text{indicator } \{\text{None}\} x + ?f x * \text{indicator } (\text{range } \text{Some}) x \partial \text{measure-pmf } p$
by(*rule Bochner-Integration.integral-cong*)(*auto simp add: indicator-def*)
also have $\dots = (\int x. ?f \text{ None} * \text{indicator } \{\text{None}\} x \partial \text{measure-pmf } p) + (\int x. ?f x * \text{indicator } (\text{range } \text{Some}) x \partial \text{measure-pmf } p)$
by(*rule Bochner-Integration.integral-add*)(*auto 4 3 intro: integrable-real-mult-indicator measure-pmf.integrable-const-bound[where B=1] simp add: AE-measure-pmf-iff pmf-le-1*)
also have $\dots = \text{pmf } p \text{ None} + \int x. \text{indicator } (\text{range } \text{Some}) x * \text{pmf } (f (the x)) \text{ None} \partial \text{measure-pmf } p$
by(*auto simp add: measure-measure-pmf-finite indicator-eq-0-iff intro!: Bochner-Integration.integral-cong*)
also have $\dots = ?rhs$ **unfolding** *measure-spmf-def*
by(*subst integral-distr*)(*auto simp add: integral-restrict-space*)
finally show *?thesis* .

qed

lemma *spmf-bind*: $\text{spmf } (p \ggg f) y = \int x. \text{spmf } (f x) y \partial \text{measure-spmf } p$
unfolding *measure-spmf-def*
by(*subst integral-distr*)(*auto simp add: bind-spmf-def pmf-bind integral-restrict-space indicator-eq-0-iff intro!: Bochner-Integration.integral-cong split: option.split*)

lemma *ennreal-spmf-bind*: $\text{ennreal } (\text{spmf } (p \ggg f) x) = \int^+ y. \text{spmf } (f y) x \partial \text{measure-spmf } p$
by(*auto simp add: bind-spmf-def ennreal-pmf-bind nn-integral-measure-spmf-conv-measure-pmf nn-integral-restrict-space intro: nn-integral-cong split: split-indicator option.split*)

lemma *measure-spmf-bind-pmf*: $\text{measure-spmf } (p \ggg f) = \text{measure-pmf } p \ggg \text{measure-spmf } \circ f$
(is *?lhs = ?rhs*)

proof(*rule measure-eqI*)

show *sets ?lhs = sets ?rhs*

by(*simp add: sets-bind[where N=count-space UNIV] space-measure-spmf*)

```

next
  fix A :: 'a set
  have emeasure ?lhs A =  $\int^+ x.$  emeasure (measure-spmf (f x)) A  $\partial$ measure-pmf
  p
  by(simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
  bind-spmf-def)
  also have ... = emeasure ?rhs A
  by(simp add: emeasure-bind[where N=count-space UNIV] space-measure-spmf
  space-subprob-algebra)
  finally show emeasure ?lhs A = emeasure ?rhs A .
qed

```

```

lemma measure-spmf-bind: measure-spmf (p  $\ggg$  f) = measure-spmf p  $\ggg$  mea-
  sure-spmf o f
  (is ?lhs = ?rhs)

```

```

proof(rule measure-eqI)
  show sets ?lhs = sets ?rhs
  by(simp add: sets-bind[where N=count-space UNIV] space-measure-spmf)
next
  fix A :: 'a set
  let ?A = the -' A  $\cap$  range Some
  have emeasure ?lhs A =  $\int^+ x.$  emeasure (measure-pmf (case x of None  $\Rightarrow$ 
  return-pmf None | Some x  $\Rightarrow$  f x)) ?A  $\partial$ measure-pmf p
  by(simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
  bind-spmf-def)
  also have ... =  $\int^+ x.$  emeasure (measure-pmf (f (the x))) ?A * indicator
  (range Some) x  $\partial$ measure-pmf p
  by(rule nn-integral-cong)(auto split: option.split simp add: indicator-def)
  also have ... =  $\int^+ x.$  emeasure (measure-spmf (f x)) A  $\partial$ measure-spmf p
  by(simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space emea-
  sure-distr space-restrict-space emeasure-restrict-space)
  also have ... = emeasure ?rhs A
  by(simp add: emeasure-bind[where N=count-space UNIV] space-measure-spmf
  space-subprob-algebra)
  finally show emeasure ?lhs A = emeasure ?rhs A .
qed

```

```

lemma map-spmf-bind-spmf: map-spmf f (bind-spmf p g) = bind-spmf p (map-spmf
  f o g)
by(auto simp add: bind-spmf-def map-bind-pmf fun-eq-iff split: option.split intro:
  arg-cong2[where f=bind-pmf])

```

```

lemma bind-map-spmf: map-spmf f p  $\ggg$  g = p  $\ggg$  g o f
by(simp add: bind-spmf-def bind-map-pmf o-def cong del: option.case-cong-weak)

```

```

lemma spmf-bind-leI:
  assumes  $\bigwedge y. y \in \text{set-spmf } p \implies \text{spmf } (f y) x \leq r$ 
  and  $0 \leq r$ 
  shows  $\text{spmf } (\text{bind-spmf } p f) x \leq r$ 

```

proof –

have $\text{ennreal } (\text{spmf } (\text{bind-spmf } p \ f) \ x) = \int^+ y. \text{spmf } (f \ y) \ x \ \partial \text{measure-spmf } p$
by(*rule ennreal-spmf-bind*)
also have $\dots \leq \int^+ y. r \ \partial \text{measure-spmf } p$ **by**(*rule nn-integral-mono-AE*)(*simp add: assms*)
also have $\dots \leq r$ **using** *assms measure-spmf.emmeasure-space-le-1*
by(*auto simp add: measure-spmf.emmeasure-eq-measure intro!: mult-left-le*)
finally show *?thesis* **using** *assms(2)* **by**(*simp*)
qed

lemma *map-spmf-conv-bind-spmf*: $\text{map-spmf } f \ p = (p \ggg (\lambda x. \text{return-spmf } (f \ x)))$
by(*simp add: map-pmf-def bind-spmf-def*)(*rule bind-pmf-cong, simp-all split: option.split*)

lemma *bind-spmf-cong*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f \ x = g \ x \rrbracket$
 $\implies \text{bind-spmf } p \ f = \text{bind-spmf } q \ g$
by(*auto simp add: bind-spmf-def in-set-spmf intro: bind-pmf-cong option.case-cong*)

lemma *bind-spmf-cong-simp*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q = \text{simp} \implies f \ x = g \ x \rrbracket$
 $\implies \text{bind-spmf } p \ f = \text{bind-spmf } q \ g$
by(*simp add: simp-implies-def cong: bind-spmf-cong*)

lemma *set-bind-spmf*: $\text{set-spmf } (M \ggg f) = \text{set-spmf } M \ggg (\text{set-spmf } \circ f)$
by(*auto simp add: set-spmf-def bind-spmf-def bind-UNION split: option.splits*)

lemma *bind-spmf-const-return-None* [*simp*]: $\text{bind-spmf } p \ (\lambda _. \text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$
by(*simp add: bind-spmf-def case-option-collapse*)

lemma *bind-commute-spmf*:

$\text{bind-spmf } p \ (\lambda x. \text{bind-spmf } q \ (f \ x)) = \text{bind-spmf } q \ (\lambda y. \text{bind-spmf } p \ (\lambda x. f \ x \ y))$
(is ?lhs = ?rhs)

proof –

let $?f = \lambda x \ y. \text{case } x \ \text{of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } a \Rightarrow (\text{case } y \ \text{of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } b \Rightarrow f \ a \ b)$
have $?lhs = p \ggg (\lambda x. q \ggg ?f \ x)$
unfolding *bind-spmf-def* **by**(*rule bind-pmf-cong[OF refl]*)(*simp split: option.split*)
also have $\dots = q \ggg (\lambda y. p \ggg (\lambda x. ?f \ x \ y))$ **by**(*rule bind-commute-pmf*)
also have $\dots = ?rhs$ **unfolding** *bind-spmf-def*
by(*rule bind-pmf-cong[OF refl]*)(*auto split: option.split, metis bind-spmf-const-return-None bind-spmf-def*)
finally show *?thesis* .
qed

25.6 Relator

abbreviation *rel-spmf* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \ \text{spmf} \Rightarrow 'b \ \text{spmf} \Rightarrow \text{bool}$

where $rel\text{-}spmf\ R \equiv rel\text{-}pmf\ (rel\text{-}option\ R)$

lemma $rel\text{-}pmf\text{-}mono$:

$\llbracket rel\text{-}pmf\ A\ f\ g; \bigwedge x\ y.\ A\ x\ y \implies B\ x\ y \rrbracket \implies rel\text{-}pmf\ B\ f\ g$
using $pmf.rel\text{-}mono$ [of $A\ B$] **by**($simp\ add: le\text{-}fun\text{-}def$)

lemma $rel\text{-}spmf\text{-}mono$:

$\llbracket rel\text{-}spmf\ A\ f\ g; \bigwedge x\ y.\ A\ x\ y \implies B\ x\ y \rrbracket \implies rel\text{-}spmf\ B\ f\ g$
apply($erule\ rel\text{-}pmf\text{-}mono$)
using $option.rel\text{-}mono$ [of $A\ B$] **by**($simp\ add: le\text{-}fun\text{-}def$)

lemma $rel\text{-}spmf\text{-}mono\text{-}strong$:

$\llbracket rel\text{-}spmf\ A\ f\ g; \bigwedge x\ y.\ \llbracket A\ x\ y; x \in set\text{-}spmf\ f; y \in set\text{-}spmf\ g \rrbracket \implies B\ x\ y \rrbracket \implies rel\text{-}spmf\ B\ f\ g$
apply($erule\ pmf.rel\text{-}mono\text{-}strong$)
apply($erule\ option.rel\text{-}mono\text{-}strong$)
apply($auto\ simp\ add: in\text{-}set\text{-}spmf$)
done

lemma $rel\text{-}spmf\text{-}reflI$: $(\bigwedge x.\ x \in set\text{-}spmf\ p \implies P\ x\ x) \implies rel\text{-}spmf\ P\ p\ p$
by($rule\ rel\text{-}pmf\text{-}reflI$)($auto\ simp\ add: set\text{-}spmf\text{-}def\ intro: rel\text{-}option\text{-}reflI$)

lemma $rel\text{-}spmfI$ [$intro?$]:

$\llbracket \bigwedge x\ y.\ (x, y) \in set\text{-}spmf\ pq \implies P\ x\ y; map\text{-}spmf\ fst\ pq = p; map\text{-}spmf\ snd\ pq = q \rrbracket$
 $\implies rel\text{-}spmf\ P\ p\ q$
by($rule\ rel\text{-}pmf.intros$ [**where** $pq = map\text{-}pmf\ (\lambda x.\ case\ x\ of\ None \Rightarrow (None, None) \mid Some\ (a, b) \Rightarrow (Some\ a, Some\ b))\ pq$])
 $(auto\ simp\ add: pmf.map\text{-}comp\ o\text{-}def\ in\text{-}set\text{-}spmf\ split: option.splits\ intro: pmf.map\text{-}cong)$

lemma $rel\text{-}spmfE$ [$elim?$, $consumes\ 1$, $case\text{-}names\ rel\text{-}spmf$]:

assumes $rel\text{-}spmf\ P\ p\ q$
obtains pq **where**
 $\bigwedge x\ y.\ (x, y) \in set\text{-}spmf\ pq \implies P\ x\ y$
 $p = map\text{-}spmf\ fst\ pq$
 $q = map\text{-}spmf\ snd\ pq$
using $assms$
proof($cases\ rule: rel\text{-}pmf.cases$ [$consumes\ 1$, $case\text{-}names\ rel\text{-}pmf$])
case ($rel\text{-}pmf\ pq$)
let $?pq = map\text{-}pmf\ (\lambda(a, b).\ case\ (a, b)\ of\ (Some\ x, Some\ y) \Rightarrow Some\ (x, y) \mid _ \Rightarrow None)\ pq$
have $\bigwedge x\ y.\ (x, y) \in set\text{-}spmf\ ?pq \implies P\ x\ y$
by($auto\ simp\ add: in\text{-}set\text{-}spmf\ split: option.split\text{-}asm\ dest: rel\text{-}pmf(1)$)
moreover
have $\bigwedge x.\ (x, None) \in set\text{-}pmf\ pq \implies x = None$ **by**($auto\ dest!: rel\text{-}pmf(1)$)
then **have** $p = map\text{-}spmf\ fst\ ?pq$ **using** $rel\text{-}pmf(2)$
by($auto\ simp\ add: pmf.map\text{-}comp\ split\text{-}beta\ intro!: pmf.map\text{-}cong\ split: option.split$)
moreover

have $\bigwedge y. (None, y) \in \text{set-pmf } pq \implies y = None$ **by** (*auto dest!: rel-pmf(1)*)
then have $q = \text{map-spmf snd } ?pq$ **using** *rel-pmf(3)*
by (*auto simp add: pmf.map-comp split-beta intro!: pmf.map-cong split: option.split*)
ultimately show thesis ..
qed

lemma *rel-spmf-simps*:

$\text{rel-spmf } R \ p \ q \longleftrightarrow (\exists pq. (\forall (x, y) \in \text{set-spmf } pq. R \ x \ y) \wedge \text{map-spmf fst } pq = p \wedge \text{map-spmf snd } pq = q)$
by (*auto intro: rel-spmfI elim!: rel-spmfE*)

lemma *spm-f-rel-map*:

shows *spm-f-rel-map1*: $\bigwedge R \ f \ x. \text{rel-spmf } R \ (\text{map-spmf } f \ x) = \text{rel-spmf } (\lambda x. R \ (f \ x)) \ x$
and *spm-f-rel-map2*: $\bigwedge R \ x \ g \ y. \text{rel-spmf } R \ x \ (\text{map-spmf } g \ y) = \text{rel-spmf } (\lambda x \ y. R \ x \ (g \ y)) \ x \ y$
by (*simp-all add: fun-eq-iff pmf.rel-map option.rel-map[abs-def]*)

lemma *spm-f-rel-conversep*: $\text{rel-spmf } R^{-1-1} = (\text{rel-spmf } R)^{-1-1}$
by (*simp add: option.rel-conversep pmf.rel-conversep*)

lemma *spm-f-rel-eq*: $\text{rel-spmf } (=) = (=)$
by (*simp add: pmf.rel-eq option.rel-eq*)

context includes *lifting-syntax*
begin

lemma *bind-spmf-parametric* [*transfer-rule*]:

$(\text{rel-spmf } A \ ==\>\ (A \ ==\>\ \text{rel-spmf } B) \ ==\>\ \text{rel-spmf } B)$ *bind-spmf bind-spmf*
unfolding *bind-spmf-def[abs-def]* **by** *transfer-prover*

lemma *return-spmf-parametric*: $(A \ ==\>\ \text{rel-spmf } A)$ *return-spmf return-spmf*
by *transfer-prover*

lemma *map-spmf-parametric*: $((A \ ==\>\ B) \ ==\>\ \text{rel-spmf } A \ ==\>\ \text{rel-spmf } B)$ *map-spmf map-spmf*
by *transfer-prover*

lemma *rel-spmf-parametric*:

$((A \ ==\>\ B \ ==\>\ (=)) \ ==\>\ \text{rel-spmf } A \ ==\>\ \text{rel-spmf } B \ ==\>\ (=))$
rel-spmf rel-spmf
by *transfer-prover*

lemma *set-spmf-parametric* [*transfer-rule*]:

$(\text{rel-spmf } A \ ==\>\ \text{rel-set } A)$ *set-spmf set-spmf*
unfolding *set-spmf-def[abs-def]* **by** *transfer-prover*

lemma *return-spmf-None-parametric*:

(*rel-spmf A*) (*return-pmf None*) (*return-pmf None*)
by *simp*

end

lemma *rel-spmf-bindI*:

$\llbracket \text{rel-spmf } R \text{ } p \text{ } q; \bigwedge x y. R \text{ } x \text{ } y \implies \text{rel-spmf } P \text{ } (f \text{ } x) \text{ } (g \text{ } y) \rrbracket$
 $\implies \text{rel-spmf } P \text{ } (p \ggg f) \text{ } (q \ggg g)$

by(*fact bind-spmf-parametric*[*THEN rel-funD*, *THEN rel-funD*, *OF - rel-funI*])

lemma *rel-spmf-bind-reflI*:

$(\bigwedge x. x \in \text{set-spmf } p \implies \text{rel-spmf } P \text{ } (f \text{ } x) \text{ } (g \text{ } x)) \implies \text{rel-spmf } P \text{ } (p \ggg f) \text{ } (p \ggg g)$

by(*rule rel-spmf-bindI*[**where** $R = \lambda x y. x = y \wedge x \in \text{set-spmf } p$])(*auto intro: rel-spmf-reflI*)

lemma *rel-pmf-return-pmfI*: $P \text{ } x \text{ } y \implies \text{rel-pmf } P \text{ } (\text{return-pmf } x) \text{ } (\text{return-pmf } y)$

by(*rule rel-pmf.intros*[**where** $pq = \text{return-pmf } (x, y)$])(*simp-all*)

context includes *lifting-syntax*

begin

We do not yet have a relator for *'a measure*, so we combine *Sigma-Algebra.measure* and *measure-pmf*

lemma *measure-pmf-parametric*:

$(\text{rel-pmf } A \implies \text{rel-pred } A \implies (=)) (\lambda p. \text{measure } (\text{measure-pmf } p)) (\lambda q. \text{measure } (\text{measure-pmf } q))$

proof(*rule rel-funI*)+

fix $p \text{ } q \text{ } X \text{ } Y$

assume $\text{rel-pmf } A \text{ } p \text{ } q$ **and** $\text{rel-pred } A \text{ } X \text{ } Y$

from *this*(1) **obtain** pq **where** $A: \bigwedge x y. (x, y) \in \text{set-pmf } pq \implies A \text{ } x \text{ } y$

and $p: p = \text{map-pmf } \text{fst } pq$ **and** $q: q = \text{map-pmf } \text{snd } pq$ **by** *cases auto*

show $\text{measure } p \text{ } X = \text{measure } q \text{ } Y$ **unfolding** $p \text{ } q$ *measure-map-pmf*

by(*rule measure-pmf.finite-measure-eq-AE*)(*auto simp add: AE-measure-pmf-iff dest!: A rel-predD*[*OF <rel-pred - ->*])

qed

lemma *measure-spmf-parametric*:

$(\text{rel-spmf } A \implies \text{rel-pred } A \implies (=)) (\lambda p. \text{measure } (\text{measure-spmf } p)) (\lambda q. \text{measure } (\text{measure-spmf } q))$

unfolding *measure-measure-spmf-conv-measure-pmf*[*abs-def*]

apply(*rule rel-funI*)+

apply(*erule measure-pmf-parametric*[*THEN rel-funD*, *THEN rel-funD*])

apply(*auto simp add: rel-pred-def rel-fun-def elim: option.rel-cases*)

done

end

25.7 From 'a pmf to 'a spmf**definition** $spmf\text{-of}\text{-}pmf :: 'a\ pmf \Rightarrow 'a\ spmf$ **where** $spmf\text{-of}\text{-}pmf = map\text{-}pmf\ Some$ **lemma** $set\text{-}spmf\text{-}spmf\text{-of}\text{-}pmf [simp]: set\text{-}spmf (spmf\text{-of}\text{-}pmf\ p) = set\text{-}pmf\ p$ **by**($auto\ simp\ add: spmf\text{-of}\text{-}pmf\text{-}def\ set\text{-}spmf\text{-}def\ bind\text{-}image\ o\text{-}def$)**lemma** $spmf\text{-}spmf\text{-of}\text{-}pmf [simp]: spmf (spmf\text{-of}\text{-}pmf\ p)\ x = pmf\ p\ x$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def$)**lemma** $pmf\text{-}spmf\text{-of}\text{-}pmf\text{-}None [simp]: pmf (spmf\text{-of}\text{-}pmf\ p)\ None = 0$ **using** $ennreal\text{-}pmf\text{-}map[of\ Some\ p\ None]$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def$)**lemma** $emeasure\text{-}spmf\text{-of}\text{-}pmf [simp]: emeasure (measure\text{-}spmf (spmf\text{-of}\text{-}pmf\ p))$ $A = emeasure (measure\text{-}pmf\ p)\ A$ **by**($simp\ add: emeasure\text{-}measure\text{-}spmf\text{-}conv\text{-}measure\text{-}pmf\ spmf\text{-of}\text{-}pmf\text{-}def\ inj\text{-}vimage\text{-}image\text{-}eq$)**lemma** $measure\text{-}spmf\text{-}spmf\text{-of}\text{-}pmf [simp]: measure\text{-}spmf (spmf\text{-of}\text{-}pmf\ p) = measure\text{-}pmf\ p$ **by**($rule\ measure\text{-}eqI$) $simp\text{-}all$ **lemma** $map\text{-}spmf\text{-of}\text{-}pmf [simp]: map\text{-}spmf\ f (spmf\text{-of}\text{-}pmf\ p) = spmf\text{-of}\text{-}pmf (map\text{-}pmf\ f\ p)$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def\ pmf.\text{map}\text{-}comp\ o\text{-}def$)**lemma** $rel\text{-}spmf\text{-}spmf\text{-of}\text{-}pmf [simp]: rel\text{-}spmf\ R (spmf\text{-of}\text{-}pmf\ p) (spmf\text{-of}\text{-}pmf\ q) = rel\text{-}pmf\ R\ p\ q$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def\ pmf.\text{rel}\text{-}map$)**lemma** $spmf\text{-of}\text{-}pmf\text{-}return\text{-}pmf [simp]: spmf\text{-of}\text{-}pmf (return\text{-}pmf\ x) = return\text{-}spmf\ x$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def$)**lemma** $bind\text{-}spmf\text{-of}\text{-}pmf [simp]: bind\text{-}spmf (spmf\text{-of}\text{-}pmf\ p)\ f = bind\text{-}pmf\ p\ f$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def\ bind\text{-}spmf\text{-}def\ bind\text{-}map\text{-}pmf$)**lemma** $set\text{-}spmf\text{-}bind\text{-}pmf: set\text{-}spmf (bind\text{-}pmf\ p\ f) = Set.\text{bind} (set\text{-}pmf\ p) (set\text{-}spmf\ \circ\ f)$ **unfolding** $bind\text{-}spmf\text{-of}\text{-}pmf[symmetric]$ **by**($subst\ set\text{-}bind\text{-}spmf$) $simp$ **lemma** $spmf\text{-of}\text{-}pmf\text{-}bind: spmf\text{-of}\text{-}pmf (bind\text{-}pmf\ p\ f) = bind\text{-}pmf\ p (\lambda x. spmf\text{-of}\text{-}pmf (f\ x))$ **by**($simp\ add: spmf\text{-of}\text{-}pmf\text{-}def\ map\text{-}bind\text{-}pmf$)**lemma** $bind\text{-}pmf\text{-}return\text{-}spmf: p \gg (\lambda x. return\text{-}spmf (f\ x)) = spmf\text{-of}\text{-}pmf (map\text{-}pmf\ f\ p)$ **by**($simp\ add: map\text{-}pmf\text{-}def\ spmf\text{-of}\text{-}pmf\text{-}bind$)

25.8 Weight of a subprobability

abbreviation *weight-spmf* :: 'a spmf \Rightarrow real

where *weight-spmf* $p \equiv \text{measure } (\text{measure-spmf } p) (\text{space } (\text{measure-spmf } p))$

lemma *weight-spmf-def*: *weight-spmf* $p = \text{measure } (\text{measure-spmf } p) \text{ UNIV}$

by(*simp* *add*: *space-measure-spmf*)

lemma *weight-spmf-le-1*: *weight-spmf* $p \leq 1$

by(*simp* *add*: *measure-spmf.subprob-measure-le-1*)

lemma *weight-return-spmf* [*simp*]: *weight-spmf* (*return-spmf* x) = 1

by(*simp* *add*: *measure-spmf-return-spmf* *measure-return*)

lemma *weight-return-pmf-None* [*simp*]: *weight-spmf* (*return-pmf* *None*) = 0

by(*simp*)

lemma *weight-map-spmf* [*simp*]: *weight-spmf* (*map-spmf* f p) = *weight-spmf* p

by(*simp* *add*: *weight-spmf-def* *measure-map-spmf*)

lemma *weight-spmf-of-pmf* [*simp*]: *weight-spmf* (*spmf-of-pmf* p) = 1

using *measure-pmf.prob-space*[*of* p] **by**(*simp* *add*: *spmf-of-pmf-def* *weight-spmf-def*)

lemma *weight-spmf-nonneg*: *weight-spmf* $p \geq 0$

by(*fact* *measure-nonneg*)

lemma (**in** *finite-measure*) *integrable-weight-spmf* [*simp*]:

$(\lambda x. \text{weight-spmf } (f x)) \in \text{borel-measurable } M \implies \text{integrable } M (\lambda x. \text{weight-spmf } (f x))$

by(*rule* *integrable-const-bound*[**where** $B=1$])(*simp-all* *add*: *weight-spmf-nonneg* *weight-spmf-le-1*)

lemma *weight-spmf-eq-nn-integral-spmf*: *weight-spmf* $p = \int^+ x. \text{spmf } p x \partial \text{count-space}$
UNIV

by(*simp* *add*: *measure-measure-spmf-conv-measure-pmf* *space-measure-spmf* *measure-pmf.emeasure-eq-measure*[*symmetric*] *nn-integral-pmf*[*symmetric*] *embed-measure-count-space*[*symmetric*] *inj-on-def* *nn-integral-embed-measure* *measurable-embed-measure1*)

lemma *weight-spmf-eq-nn-integral-support*:

weight-spmf $p = \int^+ x. \text{spmf } p x \partial \text{count-space } (\text{set-spmf } p)$

unfolding *weight-spmf-eq-nn-integral-spmf*

by(*auto* *simp* *add*: *nn-integral-count-space-indicator* *in-set-spmf-iff-spmf* *intro!*: *nn-integral-cong* *split*: *split-indicator*)

lemma *pmf-None-eq-weight-spmf*: *pmf* $p \text{ None} = 1 - \text{weight-spmf } p$

proof –

have *weight-spmf* $p = \int^+ x. \text{spmf } p x \partial \text{count-space}$ UNIV **by**(*rule* *weight-spmf-eq-nn-integral-spmf*)

also have $\dots = \int^+ x. \text{ennreal } (\text{pmf } p x) * \text{indicator } (\text{range } \text{Some}) x \partial \text{count-space}$

UNIV

by(*simp* *add*: *nn-integral-count-space-indicator*[*symmetric*] *embed-measure-count-space*[*symmetric*] *nn-integral-embed-measure* *measurable-embed-measure1*)

also have $\dots + \text{pmf } p \text{ None} = \int^+ x. \text{ennreal } (\text{pmf } p \ x) * \text{indicator } (\text{range } \text{Some}) \ x + \text{ennreal } (\text{pmf } p \ \text{None}) * \text{indicator } \{\text{None}\} \ x \ \partial \text{count-space } \text{UNIV}$
by(subst nn-integral-add)(simp-all add: max-def)
also have $\dots = \int^+ x. \text{pmf } p \ x \ \partial \text{count-space } \text{UNIV}$
by(rule nn-integral-cong)(auto split: split-indicator)
also have $\dots = 1$ **by** (simp add: nn-integral-pmf)
finally show ?thesis **by**(simp add: ennreal-plus[symmetric] del: ennreal-plus)
qed

lemma *weight-spmf-conv-pmf-None*: $\text{weight-spmf } p = 1 - \text{pmf } p \ \text{None}$
by(simp add: pmf-None-eq-weight-spmf)

lemma *weight-spmf-le-0*: $\text{weight-spmf } p \leq 0 \iff \text{weight-spmf } p = 0$
by(rule measure-le-0-iff)

lemma *weight-spmf-lt-0*: $\neg \text{weight-spmf } p < 0$
by(simp add: not-less weight-spmf-nonneg)

lemma *spmfm-le-weight*: $\text{spmfm } p \ x \leq \text{weight-spmf } p$
proof –

have $\text{ennreal } (\text{spmfm } p \ x) \leq \text{weight-spmf } p$
unfolding *weight-spmf-eq-nn-integral-spmfm* **by**(rule nn-integral-ge-point) simp
then show ?thesis **by** simp
qed

lemma *weight-spmf-eq-0*: $\text{weight-spmf } p = 0 \iff p = \text{return-pmf } \text{None}$
by(auto intro!: pmf-eqI simp add: pmf-None-eq-weight-spmfm split: split-indicator)(metis not-Some-eq pmfm-le-0-iff spmfm-le-weight)

lemma *weight-bind-spmfm*: $\text{weight-spmfm } (x \ggg f) = \text{lebesgue-integral } (\text{measure-spmfm } x) \ (\text{weight-spmfm } \circ f)$
unfolding *weight-spmfm-def*
by(simp add: measure-spmfm-bind o-def measure-spmfm.measure-bind[where $N = \text{count-space } \text{UNIV}$])

lemma *rel-spmfm-weightD*: $\text{rel-spmfm } A \ p \ q \implies \text{weight-spmfm } p = \text{weight-spmfm } q$
by(erule rel-spmfmE) simp

lemma *rel-spmfm-bij-betw*:

assumes f : *bij-betw* f (*set-spmfm* p) (*set-spmfm* q)
and eq : $\bigwedge x. x \in \text{set-spmfm } p \implies \text{spmfm } p \ x = \text{spmfm } q \ (f \ x)$
shows *rel-spmfm* $(\lambda x \ y. f \ x = y) \ p \ q$

proof –

let ? f = *map-option* f

have weq : $\text{ennreal } (\text{weight-spmfm } p) = \text{ennreal } (\text{weight-spmfm } q)$

unfolding *weight-spmfm-eq-nn-integral-support*

by(subst nn-integral-bij-count-space[OF f , symmetric])(rule nn-integral-cong-AE, simp add: eq AE-count-space)

```

then have None ∈ set-pmf p ↔ None ∈ set-pmf q
  by(simp add: pmf-None-eq-weight-spmf set-pmf-iff)
with f have bij-betw (map-option f) (set-pmf p) (set-pmf q)
  apply(auto simp add: bij-betw-def in-set-spmf inj-on-def intro: option.expand)
  apply(rename-tac [!] x)
  apply(case-tac [!] x)
  apply(auto iff: in-set-spmf)
  done
then have rel-pmf (λx y. ?f x = y) p q
  by(rule rel-pmf-bij-betw)(case-tac x, simp-all add: weq[simplified] eq in-set-spmf
  pmf-None-eq-weight-spmf)
  thus ?thesis by(rule pmf.rel-mono-strong)(auto intro!: rel-optionI simp add: Option.is-none-def)
qed

```

25.9 From density to spmfs

context fixes $f :: 'a \Rightarrow \text{real}$ **begin**

definition *embed-spmf* :: $'a$ *spmf*

where *embed-spmf* = *embed-pmf* (λx. case x of None ⇒ 1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV) | Some x' ⇒ max 0 (f x'))

context

assumes *prob*: (∫⁺ x. ennreal (f x) ∂count-space UNIV) ≤ 1
begin

lemma *nn-integral-embed-spmf-eq-1*:

(∫⁺ x. ennreal (case x of None ⇒ 1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV) | Some x' ⇒ max 0 (f x')) ∂count-space UNIV) = 1
(**is** ?lhs = - **is** (∫⁺ x. ?f x ∂?M) = -)

proof -

have ?lhs = ∫⁺ x. ?f x * indicator {None} x + ?f x * indicator (range Some) x ∂?M

by(rule nn-integral-cong)(auto split: split-indicator)

also have ... = (1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV)) + ∫⁺ x. ?f x * indicator (range Some) x ∂?M

(**is** - = ?None + ?Some)

by(subst nn-integral-add)(simp-all add: AE-count-space max-def le-diff-eq real-le-ereal-iff one-ereal-def[symmetric] prob split: option.split)

also have ?Some = ∫⁺ x. ?f x ∂count-space (range Some)

by(simp add: nn-integral-count-space-indicator)

also have count-space (range Some) = embed-measure (count-space UNIV) Some
by(simp add: embed-measure-count-space)

also have (∫⁺ x. ?f x ∂...) = ∫⁺ x. ennreal (f x) ∂count-space UNIV

by(subst nn-integral-embed-measure)(simp-all add: measurable-embed-measure1)

also have ?None + ... = 1 **using** *prob*

by(auto simp add: ennreal-minus[symmetric] ennreal-1[symmetric] ennreal-enn2real-if top-unique simp del: ennreal-1)(simp add: diff-add-self-ennreal)

finally show *?thesis* .
qed

lemma *pmf-embed-spmf-None*: *pmf embed-spmf None = 1 - enn2real (∫⁺ x. ennreal (f x) ∂count-space UNIV)*
unfolding *embed-spmf-def*
apply(*subst pmf-embed-pmf*)
subgoal using *prob by(simp add: field-simps enn2real-leI split: option.split)*
apply(*rule nn-integral-embed-spmf-eq-1*)
apply *simp*
done

lemma *pmf-embed-spmf [simp]*: *pmf embed-spmf x = max 0 (f x)*
unfolding *embed-spmf-def*
apply(*subst pmf-embed-pmf*)
subgoal using *prob by(simp add: field-simps enn2real-leI split: option.split)*
apply(*rule nn-integral-embed-spmf-eq-1*)
apply *simp*
done

end

end

lemma *embed-spmf-K-0[simp]*: *embed-spmf (λ-. 0) = return-pmf None (is ?lhs = ?rhs)*
by(*rule spmf-eqI(simp add: zero-ereal-def[symmetric])*)

25.10 Ordering on spmfs

rel-pmf does not preserve a ccpo structure. Counterexample by Saheb-Djahromi: Take prefix order over *bool llist* and the set *range (λn :: nat. uniform (llist-n n))* where *llist-n* is the set of all *llists* of length *n* and *uniform* returns a uniform distribution over the given set. The set forms a chain in *ord-pmf lprefix*, but it has not an upper bound. Any upper bound may contain only infinite lists in its support because otherwise it is not greater than the *n+1*-st element in the chain where *n* is the length of the finite list. Moreover its support must contain all infinite lists, because otherwise there is a finite list all of whose finite extensions are not in the support - a contradiction to the upper bound property. Hence, the support is uncountable, but pmf’s only have countable support.

However, if all chains in the ccpo are finite, then it should preserve the ccpo structure.

abbreviation *ord-spmf* :: (*'a ⇒ 'a ⇒ bool*) ⇒ *'a spmf ⇒ 'a spmf ⇒ bool*
where *ord-spmf ord* ≡ *rel-pmf (ord-option ord)*

locale *ord-spmf-syntax* **begin**

notation *ord-spmf* (**infix** \sqsubseteq_1 60)
end

lemma *ord-spmf-map-spmf1*: *ord-spmf* *R* (*map-spmf* *f* *p*) = *ord-spmf* ($\lambda x. R (f x)$)
p
by(*simp* *add*: *pmf.rel-map*[*abs-def*] *ord-option-map1*[*abs-def*])

lemma *ord-spmf-map-spmf2*: *ord-spmf* *R* *p* (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x y. R x (f y)$) *p* *q*
by(*simp* *add*: *pmf.rel-map* *ord-option-map2*)

lemma *ord-spmf-map-spmf12*: *ord-spmf* *R* (*map-spmf* *f* *p*) (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x y. R (f x) (f y)$) *p* *q*
by(*simp* *add*: *pmf.rel-map* *ord-option-map1*[*abs-def*] *ord-option-map2*)

lemmas *ord-spmf-map-spmf* = *ord-spmf-map-spmf1* *ord-spmf-map-spmf2* *ord-spmf-map-spmf12*

context **fixes** *ord* :: 'a \Rightarrow 'a \Rightarrow bool (**structure**) **begin**
interpretation *ord-spmf-syntax* .

lemma *ord-spmfI*:

$\llbracket \bigwedge x y. (x, y) \in \text{set-spmf } pq \implies \text{ord } x y; \text{map-spmf } \text{fst } pq = p; \text{map-spmf } \text{snd } pq = q \rrbracket$
 $\implies p \sqsubseteq q$

by(*rule* *rel-pmf.intros*[**where** *pq=map-pmf* ($\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow (\text{None}, \text{None}) \mid \text{Some } (a, b) \Rightarrow (\text{Some } a, \text{Some } b)$) *pq*])
(*auto* *simp* *add*: *pmf.map-comp* *o-def* *in-set-spmf* *split*: *option.splits* *intro*: *pmf.map-cong*)

lemma *ord-spmf-None* [*simp*]: *return-pmf* *None* $\sqsubseteq x$

by(*rule* *rel-pmf.intros*[**where** *pq=map-pmf* (*Pair* *None*) *x*])(*auto* *simp* *add*: *pmf.map-comp* *o-def*)

lemma *ord-spmf-reflI*: ($\bigwedge x. x \in \text{set-spmf } p \implies \text{ord } x x$) $\implies p \sqsubseteq p$

by(*rule* *rel-pmf-reflI* *ord-option-reflI*)+(*auto* *simp* *add*: *in-set-spmf*)

lemma *rel-spmf-inf*:

assumes $p \sqsubseteq q$

and $q \sqsubseteq p$

and *refl*: *reflp* *ord*

and *trans*: *transp* *ord*

shows *rel-spmf* (*inf* *ord* *ord*⁻¹⁻¹) *p* *q*

proof –

from $\langle p \sqsubseteq q \rangle \langle q \sqsubseteq p \rangle$

have *rel-pmf* (*inf* (*ord-option* *ord*) (*ord-option* *ord*)⁻¹⁻¹) *p* *q*

by(*rule* *rel-pmf-inf*)(*blast* *intro*: *reflp-ord-option* *transp-ord-option* *refl* *trans*)+

also have *inf* (*ord-option* *ord*) (*ord-option* *ord*)⁻¹⁻¹ = *rel-option* (*inf* *ord* *ord*⁻¹⁻¹)

by(*auto* *simp* *add*: *fun-eq-iff* *elim*: *ord-option.cases* *option.rel-cases*)

finally show *?thesis* .

qed

end

lemma *ord-spmf-return-spmf2*: $\text{ord-spmf } R \ p \ (\text{return-spmf } y) \longleftrightarrow (\forall x \in \text{set-spmf } p. R \ x \ y)$
by(*auto simp add: rel-pmf-return-pmf2 in-set-spmf ord-option.simps intro: ccontr*)

lemma *ord-spmf-mono*: $\llbracket \text{ord-spmf } A \ p \ q; \bigwedge x \ y. A \ x \ y \implies B \ x \ y \rrbracket \implies \text{ord-spmf } B \ p \ q$
by(*erule rel-pmf-mono*)(*erule ord-option-mono*)

lemma *ord-spmf-compp*: $\text{ord-spmf } (A \ OO \ B) = \text{ord-spmf } A \ OO \ \text{ord-spmf } B$
by(*simp add: ord-option-compp pmf.rel-compp*)

lemma *ord-spmf-bindI*:
assumes *pq*: $\text{ord-spmf } R \ p \ q$
and *fg*: $\bigwedge x \ y. R \ x \ y \implies \text{ord-spmf } P \ (f \ x) \ (g \ y)$
shows $\text{ord-spmf } P \ (p \ggg f) \ (q \ggg g)$
unfolding *bind-spmf-def* **using** *pq*
by(*rule rel-pmf-bindI*)(*auto split: option.split intro: fg*)

lemma *ord-spmf-bind-refI*:
 $(\bigwedge x. x \in \text{set-spmf } p \implies \text{ord-spmf } R \ (f \ x) \ (g \ x)) \implies \text{ord-spmf } R \ (p \ggg f) \ (p \ggg g)$
by(*rule ord-spmf-bindI*[**where** $R = \lambda x \ y. x = y \wedge x \in \text{set-spmf } p$])(*auto intro: ord-spmf-refI*)

lemma *ord-pmf-increaseI*:
assumes *le*: $\bigwedge x. \text{spmfs } p \ x \leq \text{spmfs } q \ x$
and *refl*: $\bigwedge x. x \in \text{set-spmfs } p \implies R \ x \ x$
shows $\text{ord-spmfs } R \ p \ q$
proof (*rule rel-pmf.intros*)
define *pq* **where** $pq = \text{embed-pmf}$
 $(\lambda(x, y). \text{case } x \ \text{of } \text{Some } x' \Rightarrow (\text{case } y \ \text{of } \text{Some } y' \Rightarrow \text{if } x' = y' \ \text{then } \text{spmfs } p \ x' \ \text{else } 0 \mid \text{None} \Rightarrow 0) \mid \text{None} \Rightarrow (\text{case } y \ \text{of } \text{None} \Rightarrow \text{pmfs } q \ \text{None} \mid \text{Some } y' \Rightarrow \text{spmfs } q \ y' - \text{spmfs } p \ y'))$
(is $- = \text{embed-pmf } ?f$)
have *nonneg*: $\bigwedge xy. ?f \ xy \geq 0$
by(*clarsimp simp add: le field-simps split: option.split*)
have *integral*: $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) = 1$ (**is** *nn-integral* $?M - = -$)
proof $-$
have $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) =$
 $\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } \{(None, None)\} \ xy +$
 $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy +$
 $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (\text{Some } x, \text{Some } x))) \ xy \ \partial ?M$
by(*rule nn-integral-cong*)(*auto split: split-indicator option.splits if-split-asm*)
also have $\dots = (\int^+ xy. ?f \ xy * \text{indicator } \{(None, None)\} \ xy \ \partial ?M) +$
 $(\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy \ \partial ?M)$

```

+
  (∫+ xy. ennreal (?f xy) * indicator (range (λx. (Some x, Some x))) xy ∂?M)
  (is - = ?None + ?Some2 + ?Some)
  by(subst nn-integral-add)(simp-all add: nn-integral-add AE-count-space le-diff-eq
le split: option.split)
  also have ?None = pmf q None by simp
  also have ?Some2 = ∫+ x. ennreal (spmf q x) - spmf p x ∂count-space UNIV
  by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1 ennreal-minus)
  also have ... = (∫+ x. spmf q x ∂count-space UNIV) - (∫+ x. spmf p x
∂count-space UNIV)
  (is - = ?Some2' - ?Some2'')
  by(subst nn-integral-diff)(simp-all add: le nn-integral-spmf-neq-top)
  also have ?Some = ∫+ x. spmf p x ∂count-space UNIV
  by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1)
  also have pmf q None + (?Some2' - ?Some2'') + ... = pmf q None +
?Some2'
  by(auto simp add: diff-add-self-ennreal le intro!: nn-integral-mono)
  also have ... = ∫+ x. ennreal (pmf q x) * indicator {None} x + ennreal (pmf
q x) * indicator (range Some) x ∂count-space UNIV
  by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator[symmetric]
embed-measure-count-space[symmetric] nn-integral-embed-measure measurable-embed-measure1)
  also have ... = ∫+ x. pmf q x ∂count-space UNIV
  by(rule nn-integral-cong)(auto split: split-indicator)
  also have ... = 1 by(simp add: nn-integral-pmf)
  finally show ?thesis .
qed
note f = nonneg integral

{ fix x y
  assume (x, y) ∈ set-pmf pq
  hence ?f (x, y) ≠ 0 unfolding pq-def by(simp add: set-embed-pmf[OF f])
  then show ord-option R x y
  by(simp add: spmf-eq-0-set-spmf refl split: option.split-asm if-split-asm) }

have weight-le: weight-spmf p ≤ weight-spmf q
  by(subst ennreal-le-iff[symmetric])(auto simp add: weight-spmf-eq-nn-integral-spmf
intro!: nn-integral-mono le)

show map-pmf fst pq = p
proof(rule pmf-eqI)
  fix i
  have ennreal (pmf (map-pmf fst pq) i) = (∫+ y. pmf pq (i, y) ∂count-space
UNIV)
  unfolding pq-def ennreal-pmf-map
  apply(simp add: embed-pmf.rep-eq[OF f] o-def emeasure-density nn-integral-count-space-indicator[symmet
  apply(subst pmf-embed-pmf[OF f])
  apply(rule nn-integral-bij-count-space[symmetric])

```



```

    apply(auto simp add: bij-betw-def inj-on-def)
  done
  also have ... = pmf p i
  proof(cases i)
    case (Some x)
    have  $(\int^+ y. \text{pmf } pq \text{ (Some } x, y) \partial \text{count-space UNIV}) = \int^+ y. \text{pmf } p \text{ (Some } x) * \text{indicator } \{\text{Some } x\} y \partial \text{count-space UNIV}$ 
      by(rule nn-integral-cong)(simp add: pq-def pmf-embed-pmf[OF f] split: option.split)
    then show ?thesis using Some by simp
  next
    case None
    have  $(\int^+ y. \text{pmf } pq \text{ (None, } y) \partial \text{count-space UNIV}) = (\int^+ y. \text{ennreal (pmf } pq \text{ (None, Some (the } y))) * \text{indicator (range Some)} y + \text{ennreal (pmf } pq \text{ (None, None)) * indicator } \{\text{None}\} y \partial \text{count-space UNIV})$ 
      by(rule nn-integral-cong)(auto split: split-indicator)
    also have ... =  $(\int^+ y. \text{ennreal (pmf } pq \text{ (None, Some (the } y))) \partial \text{count-space (range Some)} + \text{pmf } pq \text{ (None, None)})$ 
      by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator)
    also have ... =  $(\int^+ y. \text{ennreal (spm f } q \text{ } y) - \text{ennreal (spm f } p \text{ } y) \partial \text{count-space UNIV}) + \text{pmf } q \text{ None}$ 
      by(simp add: pq-def pmf-embed-pmf[OF f] embed-measure-count-space[symmetric] nn-integral-embed-measure measurable-embed-measure1 ennreal-minus)
    also have  $(\int^+ y. \text{ennreal (spm f } q \text{ } y) - \text{ennreal (spm f } p \text{ } y) \partial \text{count-space UNIV}) = (\int^+ y. \text{spm f } q \text{ } y \partial \text{count-space UNIV}) - (\int^+ y. \text{spm f } p \text{ } y \partial \text{count-space UNIV})$ 
      by(subst nn-integral-diff)(simp-all add: AE-count-space le nn-integral-spmf-neq-top split: split-indicator)
    also have ... =  $\text{pmf } p \text{ None} - \text{pmf } q \text{ None}$ 
      by(simp add: pmf-None-eq-weight-spmf weight-spmf-eq-nn-integral-spmf[symmetric] ennreal-minus)
    also have ... =  $\text{ennreal (pmf } p \text{ None)} - \text{ennreal (pmf } q \text{ None)}$  by(simp add: ennreal-minus)
    finally show ?thesis using None weight-le
      by(auto simp add: diff-add-self-ennreal pmf-None-eq-weight-spmf intro: ennreal-leI)
  qed
  finally show  $\text{pmf (map-pmf fst } pq) i = \text{pmf } p \text{ } i$  by simp
  qed

  show  $\text{map-pmf snd } pq = q$ 
  proof(rule pmf-eqI)
    fix i
    have  $\text{ennreal (pmf (map-pmf snd } pq) i) = (\int^+ x. \text{pmf } pq \text{ (} x, i) \partial \text{count-space UNIV})$ 
      unfolding pq-def ennreal-pmf-map

```

```

apply(simp add: embed-pmf.rep-eq[OF f] o-def emeasure-density nn-integral-count-space-indicator[symmetric])
apply(subst pmf-embed-pmf[OF f])
apply(rule nn-integral-bij-count-space[symmetric])
apply(auto simp add: bij-betw-def inj-on-def)
done
also have ... = ennreal (pmf q i)
proof(cases i)
  case None
    have ( $\int^+ x. \text{pmf } pq (x, \text{None}) \partial \text{count-space UNIV}$ ) =  $\int^+ x. \text{pmf } q \text{ None} * \text{indicator } \{\text{None} :: 'a \text{ option}\} x \partial \text{count-space UNIV}$ 
      by(rule nn-integral-cong)(simp add: pq-def pmf-embed-pmf[OF f] split: option.split)
    then show ?thesis using None by simp
  next
    case (Some y)
    have ( $\int^+ x. \text{pmf } pq (x, \text{Some } y) \partial \text{count-space UNIV}$ ) =
      ( $\int^+ x. \text{ennreal } (\text{pmf } pq (x, \text{Some } y)) * \text{indicator } (\text{range } \text{Some}) x + \text{ennreal } (\text{pmf } pq (\text{None}, \text{Some } y)) * \text{indicator } \{\text{None}\} x \partial \text{count-space UNIV}$ )
      by(rule nn-integral-cong)(auto split: split-indicator)
    also have ... = ( $\int^+ x. \text{ennreal } (\text{pmf } pq (x, \text{Some } y)) * \text{indicator } (\text{range } \text{Some}) x \partial \text{count-space UNIV}$ ) +  $\text{pmf } pq (\text{None}, \text{Some } y)$ 
      by(subst nn-integral-add)(simp-all)
    also have ... = ( $\int^+ x. \text{ennreal } (\text{spmf } p y) * \text{indicator } \{\text{Some } y\} x \partial \text{count-space UNIV}$ ) + ( $\text{spmf } q y - \text{spmf } p y$ )
      by(auto simp add: pq-def pmf-embed-pmf[OF f] one-ereal-def[symmetric] simp del: nn-integral-indicator-singleton intro!: arg-cong2[where f=(+)] nn-integral-cong split: option.split)
    also have ... =  $\text{spmf } q y$  by(simp add: ennreal-minus[symmetric] le)
    finally show ?thesis using Some by simp
  qed
finally show pmf (map-pmf snd pq) i = pmf q i by simp
qed
qed

```

lemma ord-spmf-eq-leD:

```

assumes ord-spmf (=) p q
shows spmf p x ≤ spmf q x
proof(cases x ∈ set-spmf p)
  case False
    thus ?thesis by(simp add: in-set-spmf-iff-spmf)
  next
    case True
    from assms obtain pq
      where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{ord-option } (=) x y$ 
      and p:  $p = \text{map-pmf fst } pq$ 
      and q:  $q = \text{map-pmf snd } pq$  by cases auto
    have  $\text{ennreal } (\text{spmf } p x) = \text{integral}^N pq (\text{indicator } (\text{fst} - \{ \text{Some } x \}))$ 
      using p by(simp add: ennreal-pmf-map)

```

also have $\dots = \text{integral}^N pq$ (*indicator* $\{(Some\ x, Some\ x)\}$)
by(*rule nn-integral-cong-AE*)(*auto simp add: AE-measure-pmf-iff-split: split-indicator*
dest: pq)
also have $\dots \leq \text{integral}^N pq$ (*indicator* $(snd - \{Some\ x\})$)
by(*rule nn-integral-mono*) *simp*
also have $\dots = \text{ennreal} (spm\ f\ q\ x)$ **using** q **by**(*simp add: ennreal-pmf-map*)
finally show *?thesis* **by** *simp*
qed

lemma *ord-spmf-eqD-set-spmf*: $\text{ord-spmf} (=) p\ q \implies \text{set-spmf } p \subseteq \text{set-spmf } q$
by(*rule subsetI*)(*drule-tac x=x in ord-spmf-eq-leD, auto simp add: in-set-spmf-iff-spmf*)

lemma *ord-spmf-eqD-emeasure*:

$\text{ord-spmf} (=) p\ q \implies \text{emeasure} (\text{measure-spmf } p) A \leq \text{emeasure} (\text{measure-spmf } q) A$

by(*auto intro!: nn-integral-mono split: split-indicator dest: ord-spmf-eq-leD simp*
add: nn-integral-measure-spmf nn-integral-indicator[symmetric])

lemma *ord-spmf-eqD-measure-spmf*: $\text{ord-spmf} (=) p\ q \implies \text{measure-spmf } p \leq \text{measure-spmf } q$

by (*subst le-measure*) (*auto simp: ord-spmf-eqD-emeasure*)

25.11 CCPO structure for the flat ccpo *ord-option* (=)

context *fixes* $Y :: 'a\ \text{spm}\ f\ \text{set}$ **begin**

definition *lub-spmf* :: $'a\ \text{spm}\ f$

where $\text{lub-spmf} = \text{embed-spmf} (\lambda x. \text{enn2real} (\text{SUP } p \in Y. \text{ennreal} (\text{spm}\ f\ p\ x)))$
 — We go through *ennreal* to have a sensible definition even if Y is empty.

lemma *lub-spmf-empty* [*simp*]: $\text{SPMF.lub-spmf } \{\} = \text{return-pmf } None$

by(*simp add: SPMF.lub-spmf-def bot-ereal-def*)

context *assumes* *chain*: *Complete-Partial-Order.chain* ($\text{ord-spmf} (=)$) Y **begin**

lemma *chain-ord-spmf-eqD*: *Complete-Partial-Order.chain* (\leq) $((\lambda p\ x. \text{ennreal} (\text{spm}\ f\ p\ x)) \text{ ' } Y)$

(*is Complete-Partial-Order.chain* - ($?f \text{ ' } -$))

proof(*rule chainI*)

fix $f\ g$

assume $f \in ?f \text{ ' } Y\ g \in ?f \text{ ' } Y$

then obtain $p\ q$ **where** $f: f = ?f\ p\ p \in Y$ **and** $g: g = ?f\ q\ q \in Y$ **by** *blast*

from *chain* $\langle p \in Y \rangle \langle q \in Y \rangle$ **have** $\text{ord-spmf} (=) p\ q \vee \text{ord-spmf} (=) q\ p$ **by**(*rule chainD*)

thus $f \leq g \vee g \leq f$

proof

assume $\text{ord-spmf} (=) p\ q$

hence $\bigwedge x. \text{spm}\ f\ p\ x \leq \text{spm}\ f\ q\ x$ **by**(*rule ord-spmf-eq-leD*)

hence $f \leq g$ **unfolding** $f\ g$ **by**(*auto intro: le-funI*)

```

    thus ?thesis ..
  next
    assume ord-spmf (=) q p
    hence  $\bigwedge x. \text{spm}f\ q\ x \leq \text{spm}f\ p\ x$  by(rule ord-spmf-eq-leD)
    hence  $g \leq f$  unfolding f g by(auto intro: le-funI)
    thus ?thesis ..
  qed
qed

lemma ord-spmf-eq-pmf-None-eq:
  assumes le: ord-spmf (=) p q
  and None: pmf p None = pmf q None
  shows p = q
proof(rule spmf-eqI)
  fix i
  from le have le':  $\bigwedge x. \text{spm}f\ p\ x \leq \text{spm}f\ q\ x$  by(rule ord-spmf-eq-leD)
  have ( $\int^+ x. \text{ennreal}(\text{spm}f\ q\ x) - \text{spm}f\ p\ x\ \partial\text{count-space UNIV}$ ) =
    ( $\int^+ x. \text{spm}f\ q\ x\ \partial\text{count-space UNIV}$ ) - ( $\int^+ x. \text{spm}f\ p\ x\ \partial\text{count-space UNIV}$ )
  by(subst nn-integral-diff)(simp-all add: AE-count-space le' nn-integral-spmf-neq-top)
  also have ... = (1 - pmf q None) - (1 - pmf p None) unfolding pmf-None-eq-weight-spmf
    by(simp add: weight-spmf-eq-nn-integral-spmf[symmetric] ennreal-minus)
  also have ... = 0 using None by simp
  finally have  $\bigwedge x. \text{spm}f\ q\ x \leq \text{spm}f\ p\ x$ 
    by(simp add: nn-integral-0-iff-AE AE-count-space ennreal-minus ennreal-eq-0-iff)
  with le' show spmf p i = spmf q i by(rule antisym)
qed

lemma ord-spmf-eqD-pmf-None:
  assumes ord-spmf (=) x y
  shows pmf x None  $\geq$  pmf y None
using assms
apply cases
apply(clarsimp simp only: ennreal-le-iff[symmetric, OF pmf-nonneg] ennreal-pmf-map)
apply(fastforce simp add: AE-measure-pmf-iff intro!: nn-integral-mono-AE)
done

Chains on 'a spmf maintain countable support. Thanks to Johannes Hölzl
for the proof idea.

lemma spmf-chain-countable: countable ( $\bigcup_{p \in Y. \text{set-spmf } p}$ )
proof(cases Y = {})
  case Y: False
  show ?thesis
proof(cases  $\exists x \in Y. \forall y \in Y. \text{ord-spmf } (=) y x$ )
  case True
  then obtain x where x:  $x \in Y$  and upper:  $\bigwedge y. y \in Y \implies \text{ord-spmf } (=) y x$ 
by blast
  hence ( $\bigcup_{x \in Y. \text{set-spmf } x}$ )  $\subseteq \text{set-spmf } x$  by(auto dest: ord-spmf-eqD-set-spmf)
  thus ?thesis by(rule countable-subset) simp
next

```

```

case False
define  $N :: 'a \text{ option pmf} \Rightarrow \text{real}$  where  $N \ p = \text{pmf } p \ \text{None}$  for  $p$ 

have  $N\text{-less-imp-le-spmf}$ :  $\llbracket x \in Y; y \in Y; N \ y < N \ x \rrbracket \Longrightarrow \text{ord-spmf } (=) \ x \ y$ 
for  $x \ y$ 
using  $\text{chainD}[OF \ \text{chain}, \ \text{of } x \ y] \ \text{ord-spmf-eqD-pmf-None}[\text{of } x \ y] \ \text{ord-spmf-eqD-pmf-None}[\text{of } y \ x]$ 
by (auto simp: N-def)
have  $N\text{-eq-imp-eq}$ :  $\llbracket x \in Y; y \in Y; N \ y = N \ x \rrbracket \Longrightarrow x = y$  for  $x \ y$ 
using  $\text{chainD}[OF \ \text{chain}, \ \text{of } x \ y] \ \text{by}(\text{auto simp add: N-def dest: ord-spmf-eq-pmf-None-eq})$ 

have  $NC$ :  $N \ 'Y \neq \{\}$  bdd-below ( $N \ 'Y$ )
using  $\langle Y \neq \{\} \rangle$  by(auto intro!: bdd-belowI[of - 0] simp: N-def)
have  $NC\text{-less}$ :  $\text{Inf } (N \ 'Y) < N \ x$  if  $x \in Y$  for  $x$  unfolding  $c\text{Inf-less-iff}[OF \ NC]$ 
proof(rule ccontr)
assume **:  $\neg (\exists y \in N \ 'Y. y < N \ x)$ 
{ fix  $y$ 
assume  $y \in Y$ 
with ** consider  $N \ x < N \ y \mid N \ x = N \ y$  by(auto simp add: not-less le-less)
hence  $\text{ord-spmf } (=) \ y \ x$  using  $\langle y \in Y \rangle \langle x \in Y \rangle$ 
by cases(auto dest: N-less-imp-le-spmf N-eq-imp-eq intro: ord-spmf-refl) }
with False  $\langle x \in Y \rangle$  show False by blast
qed

from  $NC$  have  $\text{Inf } (N \ 'Y) \in \text{closure } (N \ 'Y)$  by (intro closure-contains-Inf)
then obtain  $X'$  where  $\bigwedge n. X' \ n \in N \ 'Y$  and  $X': X' \longrightarrow \text{Inf } (N \ 'Y)$ 
unfolding closure-sequential by auto
then obtain  $X$  where  $X: \bigwedge n. X \ n \in Y$  and  $X' = (\lambda n. N \ (X \ n))$  unfolding
image-iff Bex-def by metis

with  $X'$  have  $\text{seq: } (\lambda n. N \ (X \ n)) \longrightarrow \text{Inf } (N \ 'Y)$  by simp
have  $(\bigcup x \in Y. \text{set-spmf } x) \subseteq (\bigcup n. \text{set-spmf } (X \ n))$ 
proof(rule UN-least)
fix  $x$ 
assume  $x \in Y$ 
from order-tendstoD(2)[OF seq NC-less[OF  $\langle x \in Y \rangle$ ]]
obtain  $i$  where  $N \ (X \ i) < N \ x$  by (auto simp: eventually-sequentially)
thus  $\text{set-spmf } x \subseteq (\bigcup n. \text{set-spmf } (X \ n))$  using  $X \ \langle x \in Y \rangle$ 
by(blast dest: N-less-imp-le-spmf ord-spmf-eqD-set-spmf)
qed
thus ?thesis by(rule countable-subset) simp
qed
qed simp

lemma lub-spmf-subprob:  $(\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)) \ \partial \text{count-space } UNIV) \leq 1$ 
proof(cases  $Y = \{\}$ )
case True

```

```

thus ?thesis by(simp add: bot-ennreal)
next
  case False
  let ?B =  $\bigcup_{p \in Y} \text{set-spmf } p$ 
  have countable: countable ?B by(rule spmf-chain-countable)

  have ( $\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)) \ \partial \text{count-space UNIV}$ ) =
    ( $\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x) * \text{indicator } ?B \ x) \ \partial \text{count-space UNIV}$ )
    by (intro nn-integral-cong arg-cong [of - - Sup]) (auto split: split-indicator simp
  add: spmf-eq-0-set-spmf)
  also have ... = ( $\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)) \ \partial \text{count-space } ?B$ )
    unfolding ennreal-indicator[symmetric] using False
    by(subst SUP-mult-right-ennreal[symmetric])(simp add: ennreal-indicator nn-integral-count-space-indicator)
  also have ... = ( $\text{SUP } p \in Y. \int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ ) using False -
  countable
    by(rule nn-integral-monotone-convergence-SUP-countable)(rule chain-ord-spmf-eqD)
  also have ...  $\leq 1$ 
  proof(rule SUP-least)
    fix p
    assume p  $\in Y$ 
    have ( $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ ) =  $\int^+ x. \text{ennreal } (\text{spmf } p \ x) * \text{indicator}$ 
  ?B x  $\partial \text{count-space UNIV}$ 
      by(simp add: nn-integral-count-space-indicator)
    also have ... =  $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space UNIV}$ 
    by(rule nn-integral-cong)(auto split: split-indicator simp add: spmf-eq-0-set-spmf
  ⟨p  $\in Y$ ⟩)
    also have ...  $\leq 1$ 
      by(simp add: weight-spmf-eq-nn-integral-spmf[symmetric] weight-spmf-le-1)
    finally show ( $\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } ?B$ )  $\leq 1$  .
  qed
  finally show ?thesis .
qed

lemma spmf-lub-spmf:
  assumes Y  $\neq \{\}$ 
  shows spmf lub-spmf x = ( $\text{SUP } p \in Y. \text{spmf } p \ x$ )
proof -
  from assms obtain p where p  $\in Y$  by auto
  have spmf lub-spmf x = max 0 (enn2real (SUP p  $\in Y. \text{ennreal } (\text{spmf } p \ x)$ ))
unfolding lub-spmf-def
    by(rule spmf-embed-spmf)(simp del: SUP-eq-top-iff Sup-eq-top-iff add: en-
  nreal-enn2real-if SUP-spmf-neq-top' lub-spmf-subprob)
  also have ... = enn2real (SUP p  $\in Y. \text{ennreal } (\text{spmf } p \ x)$ )
    by(rule max-absorb2)(simp)
  also have ... = enn2real (ennreal (SUP p  $\in Y. \text{spmf } p \ x$ )) using assms
    by(subst ennreal-SUP[symmetric])(simp-all add: SUP-spmf-neq-top' del: SUP-eq-top-iff
  Sup-eq-top-iff)
  also have 0  $\leq$  ( $\bigcup_{p \in Y} \text{spmf } p \ x$ ) using assms
    by(auto intro!: cSUP-upper2 bdd-aboveI[where M=1] simp add: pmf-le-1)

```

then have $\text{enn2real } (\text{ennreal } (\text{SUP } p \in Y. \text{ spmf } p x)) = (\text{SUP } p \in Y. \text{ spmf } p x)$
by(rule *enn2real-ennreal*)
finally show *?thesis* .
qed

lemma *ennreal-spmf-lub-spmf*: $Y \neq \{\}$ $\implies \text{ennreal } (\text{spmfm } \text{lub-spmf } x) = (\text{SUP } p \in Y. \text{ennreal } (\text{spmfm } p x))$
unfolding *spmfm-lub-spmf* **by**(subst *ennreal-SUP*)(*simp-all add: SUP-spmfm-neq-top'*
del: SUP-eq-top-iff Sup-eq-top-iff)

lemma *lub-spmfm-upper*:
assumes $p: p \in Y$
shows $\text{ord-spmfm } (=) p \text{ lub-spmfm}$
proof(rule *ord-pmfm-increaseI*)
fix x
from p **have** [*simp*]: $Y \neq \{\}$ **by** *auto*
from p **have** $\text{ennreal } (\text{spmfm } p x) \leq (\text{SUP } p \in Y. \text{ennreal } (\text{spmfm } p x))$ **by**(rule *SUP-upper*)
also have $\dots = \text{ennreal } (\text{spmfm } \text{lub-spmfm } x)$ **using** p
by(subst *spmfm-lub-spmfm*)(*auto simp add: ennreal-SUP SUP-spmfm-neq-top' simp del: SUP-eq-top-iff Sup-eq-top-iff*)
finally show $\text{spmfm } p x \leq \text{spmfm } \text{lub-spmfm } x$ **by** *simp*
qed *simp*

lemma *lub-spmfm-least*:
assumes $z: \bigwedge x. x \in Y \implies \text{ord-spmfm } (=) x z$
shows $\text{ord-spmfm } (=) \text{lub-spmfm } z$
proof(cases $Y = \{\}$)
case *nonempty*: *False*
show *?thesis*
proof(rule *ord-pmfm-increaseI*)
fix x
from *nonempty* **obtain** p **where** $p: p \in Y$ **by** *auto*
have $\text{ennreal } (\text{spmfm } \text{lub-spmfm } x) = (\text{SUP } p \in Y. \text{ennreal } (\text{spmfm } p x))$
by(subst *spmfm-lub-spmfm*)(*auto simp add: ennreal-SUP SUP-spmfm-neq-top' nonempty simp del: SUP-eq-top-iff Sup-eq-top-iff*)
also have $\dots \leq \text{ennreal } (\text{spmfm } z x)$ **by**(rule *SUP-least*)(*simp add: ord-spmfm-eq-leD z*)
finally show $\text{spmfm } \text{lub-spmfm } x \leq \text{spmfm } z x$ **by** *simp*
qed *simp*
qed *simp*

lemma *set-lub-spmfm*: $\text{set-spmfm } \text{lub-spmfm} = (\bigcup p \in Y. \text{set-spmfm } p)$ (**is** *?lhs = ?rhs*)
proof(cases $Y = \{\}$)
case [*simp*]: *False*
show *?thesis*
proof(rule *set-eqI*)
fix x
have $x \in ?lhs \iff \text{ennreal } (\text{spmfm } \text{lub-spmfm } x) > 0$

```

    by(simp-all add: in-set-spmf-iff-spmf less-le)
  also have ...  $\longleftrightarrow (\exists p \in Y. \text{ennreal} (\text{spm}f\ p\ x) > 0)$ 
    by(simp add: ennreal-spmf-lub-spmf less-SUP-iff)
  also have ...  $\longleftrightarrow x \in ?rhs$ 
    by(auto simp add: in-set-spmf-iff-spmf less-le)
  finally show  $x \in ?lhs \longleftrightarrow x \in ?rhs$  .
qed
qed simp

lemma emeasure-lub-spmf:
  assumes  $Y: Y \neq \{\}$ 
  shows  $\text{emeasure} (\text{measure-spmf}\ \text{lub-spmf})\ A = (\text{SUP}\ y \in Y. \text{emeasure} (\text{measure-spmf}\ y)\ A)$ 
    (is  $?lhs = ?rhs$ )
  proof -
    let  $?M = \text{count-space} (\text{set-spmf}\ \text{lub-spmf})$ 
    have  $?lhs = \int^+ x. \text{ennreal} (\text{spm}f\ \text{lub-spmf}\ x) * \text{indicator}\ A\ x\ \partial ?M$ 
      by(auto simp add: nn-integral-indicator[symmetric] nn-integral-measure-spmf')
    also have ...  $= \int^+ x. (\text{SUP}\ y \in Y. \text{ennreal} (\text{spm}f\ y\ x) * \text{indicator}\ A\ x)\ \partial ?M$ 
      unfolding ennreal-indicator[symmetric]
      by(simp add: spmf-lub-spmf assms ennreal-SUP[OF SUP-spmf-neq-top'] SUP-mult-right-ennreal)
    also from assms have ...  $= (\text{SUP}\ y \in Y. \int^+ x. \text{ennreal} (\text{spm}f\ y\ x) * \text{indicator}\ A\ x\ \partial ?M)$ 
      proof(rule nn-integral-monotone-convergence-SUP-countable)
        have  $(\lambda i\ x. \text{ennreal} (\text{spm}f\ i\ x) * \text{indicator}\ A\ x) \uparrow Y = (\lambda f\ x. f\ x * \text{indicator}\ A\ x)$ 
           $(\lambda p\ x. \text{ennreal} (\text{spm}f\ p\ x)) \uparrow Y$ 
          by(simp add: image-image)
        also have  $\text{Complete-Partial-Order.chain} (\leq) \dots$  using  $\text{chain-ord-spmf-eqD}$ 
          by(rule chain-imageI)(auto simp add: le-fun-def split: split-indicator)
        finally show  $\text{Complete-Partial-Order.chain} (\leq) ((\lambda i\ x. \text{ennreal} (\text{spm}f\ i\ x) * \text{indicator}\ A\ x) \uparrow Y)$  .
      proof simp
        also have ...  $= (\text{SUP}\ y \in Y. \int^+ x. \text{ennreal} (\text{spm}f\ y\ x) * \text{indicator}\ A\ x\ \partial \text{count-space}\ \text{UNIV})$ 
          by(auto simp add: nn-integral-count-space-indicator set-lub-spmf spmf-eq-0-set-spmf
            split: split-indicator intro!: arg-cong [of - - Sup] image-cong nn-integral-cong)
        also have ...  $= ?rhs$ 
          by(auto simp add: nn-integral-indicator[symmetric] nn-integral-measure-spmf)
        finally show  $?thesis$  .
      qed
    qed
  qed

```

```

lemma measure-lub-spmf:
  assumes  $Y: Y \neq \{\}$ 
  shows  $\text{measure} (\text{measure-spmf}\ \text{lub-spmf})\ A = (\text{SUP}\ y \in Y. \text{measure} (\text{measure-spmf}\ y)\ A)$ 
    (is  $?lhs = ?rhs$ )
  proof -
    have  $\text{ennreal}\ ?lhs = \text{ennreal}\ ?rhs$ 
      using emeasure-lub-spmf[OF assms] SUP-emeasure-spmf-neq-top[of A Y] Y
      unfolding measure-spmf.emeasure-eq-measure by(subst ennreal-SUP)

```


moreover have $0 \leq ?rhs$ **using** Y
by(*auto intro!*: *cSUP-upper2 bdd-aboveI*[**where** $M=1$] *measure-spmf.subprob-measure-le-1*)
ultimately show *?thesis* **by**(*simp*)
qed

lemma *weight-lub-spmf*:
assumes $Y: Y \neq \{\}$
shows *weight-spmf lub-spmf* = ($SUP\ y \in Y. \text{weight-spmf } y$)
unfolding *weight-spmf-def* **by**(*rule measure-lub-spmf*) *fact*

lemma *measure-spmf-lub-spmf*:
assumes $Y: Y \neq \{\}$
shows *measure-spmf lub-spmf* = ($SUP\ p \in Y. \text{measure-spmf } p$) (**is** *?lhs = ?rhs*)
proof(*rule measure-eqI*)
from *assms* **obtain** p **where** $p: p \in Y$ **by** *auto*
from *chain* **have** *chain'*: *Complete-Partial-Order.chain* (\leq) (*measure-spmf ' Y*)
by(*rule chain-imageI*)(*rule ord-spmf-eqD-measure-spmf*)
show *sets ?lhs = sets ?rhs*
using Y **by** (*subst sets-SUP*) *auto*
show *emeasure ?lhs A = emeasure ?rhs A* **for** A
using *chain' Y p* **by** (*subst emeasure-SUP-chain*) (*auto simp: emeasure-lub-spmf*)
qed

end

end

lemma *partial-function-definitions-spmf*: *partial-function-definitions* (*ord-spmf (=)*)
lub-spmf
(**is** *partial-function-definitions ?R -*)
proof
fix x **show** $?R\ x\ x$ **by**(*simp add: ord-spmf-refl*)
next
fix $x\ y\ z$
assume $?R\ x\ y\ ?R\ y\ z$
with *transp-ord-option*[*OF transp-equality*] **show** $?R\ x\ z$ **by**(*rule transp-rel-pmf*[*THEN transpD*])
next
fix $x\ y$
assume $?R\ x\ y\ ?R\ y\ x$
thus $x = y$
by(*rule rel-pmf-antisym*)(*simp-all add: reflp-ord-option transp-ord-option anti-symp-ord-option*)
next
fix $Y\ x$
assume *Complete-Partial-Order.chain* $?R\ Y\ x \in Y$
then **show** $?R\ x$ (*lub-spmf Y*)
by(*rule lub-spmf-upper*)
next

```

fix  $Y z$ 
assume Complete-Partial-Order.chain  $?R Y \wedge x. x \in Y \implies ?R x z$ 
then show  $?R (\text{lub-spmf } Y) z$ 
  by(cases  $Y = \{\}$ )(simp-all add: lub-spmf-least)
qed

```

```

lemma ccpo-spmf: class.ccpo lub-spmf (ord-spmf (=)) (mk-less (ord-spmf (=)))
by(rule ccpo partial-function-definitions-spmf)+

```

```

interpretation spmf: partial-function-definitions ord-spmf (=) lub-spmf
  rewrites lub-spmf  $\{\} \equiv \text{return-pmf None}$ 
by(rule partial-function-definitions-spmf) simp

```

```

declaration  $\langle \text{Partial-Function.init } \textit{spmf term} \langle \textit{spmf.fixp-fun} \rangle$ 
   $\textit{term} \langle \textit{spmf.mono-body} \rangle @\{\textit{thm } \textit{spmf.fixp-rule-uc}\} @\{\textit{thm } \textit{spmf.fixp-induct-uc}\}$ 
   $\textit{NONE} \rangle$ 

```

```

declare spmf.leq-refl[simp]
declare admissible-leI[OF ccpo-spmf, cont-intro]

```

```

abbreviation mono-spmf  $\equiv \textit{monotone (fun-ord (ord-spmf (=))) (ord-spmf (=))}$ 

```

```

lemma lub-spmf-const [simp]: lub-spmf  $\{p\} = p$ 
by(rule spmf-eqI)(simp add: spmf-lub-spmf[OF ccpo.chain-singleton[OF ccpo-spmf]])

```

```

lemma bind-spmf-mono':
  assumes fg: ord-spmf (=)  $f g$ 
  and hk:  $\wedge x :: 'a. \textit{ord-spmf (=)} (h x) (k x)$ 
  shows ord-spmf (=)  $(f \gg h) (g \gg k)$ 
unfolding bind-spmf-def using assms(1)
by(rule rel-pmf-bindI)(auto split: option.split simp add: hk)

```

```

lemma bind-spmf-mono [partial-function-mono]:
  assumes mf: mono-spmf  $B$  and mg:  $\wedge y. \textit{mono-spmf} (\lambda f. C y f)$ 
  shows mono-spmf  $(\lambda f. \textit{bind-spmf} (B f) (\lambda y. C y f))$ 
proof (rule monotoneI)
  fix  $f g :: 'a \Rightarrow 'b \textit{ spmf}$ 
  assume fg: fun-ord (ord-spmf (=))  $f g$ 
  with mf have ord-spmf (=)  $(B f) (B g)$  by (rule monotoneD[of - - - f g])
  moreover from mg have  $\wedge y'. \textit{ord-spmf (=)} (C y' f) (C y' g)$ 
  by (rule monotoneD) (rule fg)
  ultimately show ord-spmf (=)  $(\textit{bind-spmf} (B f) (\lambda y. C y f)) (\textit{bind-spmf} (B g) (\lambda y'. C y' g))$ 
  by(rule bind-spmf-mono')
qed

```

```

lemma monotone-bind-spmf1: monotone (ord-spmf (=)) (ord-spmf (=))  $(\lambda y. \textit{bind-spmf} y g)$ 
by(rule monotoneI)(simp add: bind-spmf-mono' ord-spmf-reflI)

```

lemma *monotone-bind-spmf2*:

assumes $g: \bigwedge x. \text{monotone ord } (\text{ord-spmf } (=)) (\lambda y. g \ y \ x)$

shows $\text{monotone ord } (\text{ord-spmf } (=)) (\lambda y. \text{bind-spmf } p \ (g \ y))$

by(*rule monotoneI*)(*auto intro: bind-spmf-mono' monotoneD[OF g] ord-spmf-reflI*)

lemma *bind-lub-spmf*:

assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf* (=)) *Y*

shows $\text{bind-spmf } (\text{lub-spmf } Y) \ f = \text{lub-spmf } ((\lambda p. \text{bind-spmf } p \ f) \ ' Y) \ (\text{is } ?lhs = ?rhs)$

proof(*cases* $Y = \{\}$)

case $Y: \text{False}$

show *?thesis*

proof(*rule spmf-eqI*)

fix i

have *chain'*: *Complete-Partial-Order.chain* (\leq) ($(\lambda p \ x. \text{ennreal } (\text{spmf } p \ x \ * \ \text{spmf } (f \ x) \ i)) \ ' Y$)

using *chain* **by**(*rule chain-imageI*)(*auto simp add: le-fun-def dest: ord-spmf-eq-leD intro: mult-right-mono*)

have *chain''*: *Complete-Partial-Order.chain* (*ord-spmf* (=)) ($(\lambda p. p \ \gg\! = \ f) \ ' Y$)

using *chain* **by**(*rule chain-imageI*)(*auto intro!: monotoneI bind-spmf-mono' ord-spmf-reflI*)

let $?M = \text{count-space } (\text{set-spmf } (\text{lub-spmf } Y))$

have $\text{ennreal } (\text{spmf } ?lhs \ i) = \int^+ x. \text{ennreal } (\text{spmf } (\text{lub-spmf } Y) \ x) \ * \ \text{ennreal } (\text{spmf } (f \ x) \ i) \ \partial ?M$

by(*auto simp add: ennreal-spmf-lub-spmf ennreal-spmf-bind nn-integral-measure-spmf'*)

also have $\dots = \int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x \ * \ \text{spmf } (f \ x) \ i)) \ \partial ?M$

by(*subst ennreal-spmf-lub-spmf[OF chain Y](subst SUP-mult-right-ennreal, simp-all add: ennreal-mult Y)*)

also have $\dots = (\text{SUP } p \in Y. \int^+ x. \text{ennreal } (\text{spmf } p \ x \ * \ \text{spmf } (f \ x) \ i) \ \partial ?M)$

using $Y \ \text{chain}'$ **by**(*rule nn-integral-monotone-convergence-SUP-countable*)

simp

also have $\dots = (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } (\text{bind-spmf } p \ f) \ i))$

by(*auto simp add: ennreal-spmf-bind nn-integral-measure-spmf nn-integral-count-space-indicator set-lub-spmf[OF chain] in-set-spmf-iff-spmf ennreal-mult intro!: arg-cong [of - - Sup] image-cong nn-integral-cong split: split-indicator*)

also have $\dots = \text{ennreal } (\text{spmf } ?rhs \ i)$ **using** *chain''* **by**(*simp add: ennreal-spmf-lub-spmf Y image-comp*)

finally show $\text{spmf } ?lhs \ i = \text{spmf } ?rhs \ i$ **by** *simp*

qed

qed *simp*

lemma *map-lub-spmf*:

Complete-Partial-Order.chain (*ord-spmf* (=)) *Y*

$\implies \text{map-spmf } f \ (\text{lub-spmf } Y) = \text{lub-spmf } (\text{map-spmf } f \ ' Y)$

unfolding *map-spmf-conv-bind-spmf[abs-def]* **by**(*simp add: bind-lub-spmf o-def*)

lemma *mcont-bind-spmf1*: *mcont* *lub-spmf* (*ord-spmf* (=)) *lub-spmf* (*ord-spmf* (=)) ($\lambda y. \text{bind-spmf } y \ f$)

using *monotone-bind-spmf1* **by**(rule *mcontI*)(rule *contI*, *simp add: bind-lub-spmf*)

lemma *bind-lub-spmf2*:

assumes *chain*: *Complete-Partial-Order.chain* ord *Y*

and *g*: $\bigwedge y. \text{monotone ord (ord-spmf (=)) (g y)}$

shows $\text{bind-spmf } x (\lambda y. \text{lub-spmf (g y ' Y)}) = \text{lub-spmf ((}\lambda p. \text{bind-spmf } x (\lambda y. \text{g y p})) \text{' Y)}$

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

proof(cases *Y = {}*)

case *Y: False*

show *?thesis*

proof(rule *spmf-eqI*)

fix *i*

have *chain'*: $\bigwedge y. \text{Complete-Partial-Order.chain (ord-spmf (=)) (g y ' Y)}$

using *chain g [THEN monotoneD]* **by**(rule *chain-imageI*)

have *chain''*: *Complete-Partial-Order.chain* (\leq) (($\lambda p y. \text{ennreal (spmf } x \ y \ * \ \text{spmf (g y p) i})}$) ' *Y*)

using *chain* **by**(rule *chain-imageI*)(*auto simp add: le-fun-def dest: ord-spmf-eq-leD monotoneD[OF g] intro!: mult-left-mono*)

have *chain'''*: *Complete-Partial-Order.chain* (*ord-spmf (=)*) (($\lambda p. \text{bind-spmf } x (\lambda y. \text{g y p})$) ' *Y*)

using *chain* **by**(rule *chain-imageI*)(rule *monotone-bind-spmf2*[*OF g, THEN monotoneD*])

have $\text{ennreal (spmf ?lhs i)} = \int^+ y. (\text{SUP } p \in Y. \text{ennreal (spmf } x \ y \ * \ \text{spmf (g y p) i)}) \ \partial \text{count-space (set-spmf } x)$

by(*simp add: ennreal-spmf-bind ennreal-spmf-lub-spmf[OF chain'] Y nn-integral-measure-spmf' SUP-mult-left-ennreal ennreal-mult image-comp*)

also have $\dots = (\text{SUP } p \in Y. \int^+ y. \text{ennreal (spmf } x \ y \ * \ \text{spmf (g y p) i)}) \ \partial \text{count-space (set-spmf } x)$

unfolding *nn-integral-measure-spmf'* **using** *Y chain''*

by(rule *nn-integral-monotone-convergence-SUP-countable*) *simp*

also have $\dots = (\text{SUP } p \in Y. \text{ennreal (spmf (bind-spmf } x (\lambda y. \text{g y p}) i))}$

by(*simp add: ennreal-spmf-bind nn-integral-measure-spmf' ennreal-mult*)

also have $\dots = \text{ennreal (spmf ?rhs i)}$ **using** *chain'''*

by(*auto simp add: ennreal-spmf-lub-spmf Y image-comp*)

finally show $\text{spmf ?lhs i} = \text{spmf ?rhs i}$ **by** *simp*

qed

qed *simp*

lemma *mcont-bind-spmf [cont-intro]*:

assumes *f*: *mcont luba orda lub-spmf (ord-spmf (=)) f*

and *g*: $\bigwedge y. \text{mcont luba orda lub-spmf (ord-spmf (=)) (g y)}$

shows $\text{mcont luba orda lub-spmf (ord-spmf (=)) (\lambda x. \text{bind-spmf (f } x) (\lambda y. \text{g y } x))}$

proof(rule *spmf.mcont2mcont'[OF - - f]*)

fix *z*

show $\text{mcont lub-spmf (ord-spmf (=)) lub-spmf (ord-spmf (=)) (\lambda x. \text{bind-spmf } x (\lambda y. \text{g y } z))}$

```

  by(rule mcont-bind-spmf1)
next
  fix x
  let ?f =  $\lambda z. \text{bind-spmf } x (\lambda y. g y z)$ 
  have monotone orda ( $\text{ord-spmf } (=)$ ) ?f using mcont-mono[OF g] by(rule mono-
tone-bind-spmf2)
  moreover have cont luba orda lub-spmf ( $\text{ord-spmf } (=)$ ) ?f
  proof(rule contI)
    fix Y
    assume chain: Complete-Partial-Order.chain orda Y and Y:  $Y \neq \{\}$ 
    have bind-spmf x ( $\lambda y. g y (\text{luba } Y)$ ) = bind-spmf x ( $\lambda y. \text{lub-spmf } (g y \text{ ‘ } Y)$ )
      by(rule bind-spmf-cong)(simp-all add: mcont-contD[OF g chain Y])
    also have ... = lub-spmf (( $\lambda p. x \gg (\lambda y. g y p)$ ) ‘ Y) using chain
      by(rule bind-lub-spmf2)(rule mcont-mono[OF g])
    finally show bind-spmf x ( $\lambda y. g y (\text{luba } Y)$ ) = ... .
  qed
  ultimately show mcont luba orda lub-spmf ( $\text{ord-spmf } (=)$ ) ?f by(rule mcontI)
qed

```

lemma *bind-pmf-mono* [*partial-function-mono*]:
 $(\bigwedge y. \text{mono-spmf } (\lambda f. C y f)) \implies \text{mono-spmf } (\lambda f. \text{bind-pmf } p (\lambda x. C x f))$
using *bind-spmf-mono*[of $\lambda-. \text{spm-f-of-pmf } p C$] **by** *simp*

lemma *map-spmf-mono* [*partial-function-mono*]: $\text{mono-spmf } B \implies \text{mono-spmf } (\lambda g. \text{map-spmf } f (B g))$
unfolding *map-spmf-conv-bind-spmf* **by**(rule *bind-spmf-mono*) *simp-all*

lemma *mcont-map-spmf* [*cont-intro*]:
 $\text{mcont luba orda lub-spmf } (\text{ord-spmf } (=)) g$
 $\implies \text{mcont luba orda lub-spmf } (\text{ord-spmf } (=)) (\lambda x. \text{map-spmf } f (g x))$
unfolding *map-spmf-conv-bind-spmf* **by**(rule *mcont-bind-spmf*) *simp-all*

lemma *monotone-set-spmf*: $\text{monotone } (\text{ord-spmf } (=)) (\subseteq) \text{set-spmf}$
by(rule *monotoneI*)(rule *ord-spmf-eqD-set-spmf*)

lemma *cont-set-spmf*: $\text{cont lub-spmf } (\text{ord-spmf } (=)) \text{Union } (\subseteq) \text{set-spmf}$
by(rule *contI*)(*subst set-lub-spmf*; *simp*)

lemma *mcont2mcont-set-spmf*[*THEN mcont2mcont, cont-intro*]:
shows *mcont-set-spmf*: $\text{mcont lub-spmf } (\text{ord-spmf } (=)) \text{Union } (\subseteq) \text{set-spmf}$
by(rule *mcontI monotone-set-spmf cont-set-spmf*)+

lemma *monotone-spmf*: $\text{monotone } (\text{ord-spmf } (=)) (\leq) (\lambda p. \text{spm-f } p x)$
by(rule *monotoneI*)(*simp add: ord-spmf-eq-leD*)

lemma *cont-spmf*: $\text{cont lub-spmf } (\text{ord-spmf } (=)) \text{Sup } (\leq) (\lambda p. \text{spm-f } p x)$
by(rule *contI*)(*simp add: spmf-lub-spmf*)

lemma *mcont-spmf*: $\text{mcont lub-spmf } (\text{ord-spmf } (=)) \text{Sup } (\leq) (\lambda p. \text{spm-f } p x)$

by(rule mcontI monotone-spmf cont-spmf)+

lemma cont-ennreal-spmf: cont lub-spmf (ord-spmf (=)) Sup (\leq) (λp . ennreal (spm f p x))

by(rule contI)(simp add: ennreal-spmf-lub-spmf)

lemma mcont2mcont-ennreal-spmf [THEN mcont2mcont, cont-intro]:

shows mcont-ennreal-spmf: mcont lub-spmf (ord-spmf (=)) Sup (\leq) (λp . ennreal (spm f p x))

by(rule mcontI mono2mono-ennreal monotone-spmf cont-ennreal-spmf)+

lemma nn-integral-map-spmf [simp]: nn-integral (measure-spmf (map-spmf f p))
g = nn-integral (measure-spmf p) (g \circ f)

by(auto 4 3 simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space
intro: nn-integral-cong split: split-indicator)

25.11.1 Admissibility of rel-spmf

lemma rel-spmf-measureD:

assumes rel-spmf R p q

shows measure (measure-spmf p) A \leq measure (measure-spmf q) {y. $\exists x \in A$. R x y} (is ?lhs \leq ?rhs)

proof –

have ?lhs = measure (measure-pmf p) (Some ‘A) **by**(simp add: measure-measure-spmf-conv-measure-pmf)

also have ... \leq measure (measure-pmf q) {y. $\exists x \in \text{Some ‘A}$. rel-option R x y}

using assms **by**(rule rel-pmf-measureD)

also have ... = ?rhs **unfolding** measure-measure-spmf-conv-measure-pmf

by(rule arg-cong2[**where** f=measure])(auto simp add: option-rel-SomeI)

finally show ?thesis .

qed

locale rel-spmf-characterisation =

assumes rel-pmf-measureI:

$\bigwedge (R :: 'a \text{ option} \Rightarrow 'b \text{ option} \Rightarrow \text{bool}) p q$.

$(\bigwedge A$. measure (measure-pmf p) A \leq measure (measure-pmf q) {y. $\exists x \in A$. R x y})

\implies rel-pmf R p q

— This assumption is shown to hold in general in the AFP entry *MFMC-Countable*.

begin

context fixes R :: 'a \Rightarrow 'b \Rightarrow bool **begin**

lemma rel-spmf-measureI:

assumes eq1: $\bigwedge A$. measure (measure-spmf p) A \leq measure (measure-spmf q) {y. $\exists x \in A$. R x y}

assumes eq2: weight-spmf q \leq weight-spmf p

shows rel-spmf R p q

proof(rule rel-pmf-measureI)

fix A :: 'a option set

```

define A' where A' = the ' (A  $\cap$  range Some)
define A'' where A'' = A  $\cap$  {None}
have A: A = Some ' A'  $\cup$  A'' Some ' A'  $\cap$  A'' = {}
  unfolding A'-def A''-def by(auto 4 3 intro: rev-image-eqI)
  have measure (measure-pmf p) A = measure (measure-pmf p) (Some ' A') +
  measure (measure-pmf p) A''
  by(simp add: A measure-pmf.finite-measure-Union)
also have measure (measure-pmf p) (Some ' A') = measure (measure-spmf p) A'
  by(simp add: measure-measure-spmf-conv-measure-pmf)
also have ...  $\leq$  measure (measure-spmf q) {y.  $\exists x \in A'$ . R x y} by(rule eq1)
also (ord-eq-le-trans[OF - add-right-mono])
have ... = measure (measure-pmf q) {y.  $\exists x \in A'$ . rel-option R (Some x) y}
  unfolding measure-measure-spmf-conv-measure-pmf
  by(rule arg-cong2[where f=measure])(auto simp add: A'-def option-rel-Some1)
also
{ have weight-spmf p  $\leq$  measure (measure-spmf q) {y.  $\exists x$ . R x y}
  using eq1[of UNIV] unfolding weight-spmf-def by simp
  also have ...  $\leq$  weight-spmf q unfolding weight-spmf-def
  by(rule measure-spmf.finite-measure-mono) simp-all
  finally have weight-spmf p = weight-spmf q using eq2 by simp }
then have measure (measure-pmf p) A'' = measure (measure-pmf q) (if None  $\in$ 
A then {None} else {})
  unfolding A''-def by(simp add: pmf-None-eq-weight-spmf measure-pmf-single)
  also have measure (measure-pmf q) {y.  $\exists x \in A'$ . rel-option R (Some x) y} + ...
= measure (measure-pmf q) {y.  $\exists x \in A$ . rel-option R x y}
  by(subst measure-pmf.finite-measure-Union[symmetric])
  (auto 4 3 intro!: arg-cong2[where f=measure] simp add: option-rel-Some1
option-rel-Some2 A'-def intro: rev-beI elim: option.rel-cases)
  finally show measure (measure-pmf p) A  $\leq$  ... .
qed

```

lemma *admissible-rel-spmf*:

```

ccpo.admissible (prod-lub lub-spmf lub-spmf) (rel-prod (ord-spmf (=)) (ord-spmf
(=))) (case-prod (rel-spmf R))
(is ccpo.admissible ?lub ?ord ?P)

```

proof(rule ccpo.admissibleI)

fix Y

assume chain: Complete-Partial-Order.chain ?ord Y

and Y: Y \neq {}

and R: $\forall (p, q) \in Y$. rel-spmf R p q

from R **have** R: $\bigwedge p q$. (p, q) \in Y \implies rel-spmf R p q **by** auto

have chain1: Complete-Partial-Order.chain (ord-spmf (=)) (fst ' Y)

and chain2: Complete-Partial-Order.chain (ord-spmf (=)) (snd ' Y)

using chain **by**(rule chain-imageI; clarsimp)+

from Y **have** Y1: fst ' Y \neq {} **and** Y2: snd ' Y \neq {} **by** auto

have rel-spmf R (lub-spmf (fst ' Y)) (lub-spmf (snd ' Y))

proof(rule rel-spmf-measureI)

show weight-spmf (lub-spmf (snd ' Y)) \leq weight-spmf (lub-spmf (fst ' Y))

by(*auto simp add: weight-lub-spmf chain1 chain2 Y rel-spmf-weightD*[*OF R, symmetric*]) *intro!*: *cSUP-least intro: cSUP-upper2*[*OF bdd-aboveI2*[*OF weight-spmf-le-1*]])

fix *A*
have *measure (measure-spmf (lub-spmf (fst ‘ Y))) A = (SUP y∈fst ‘ Y. measure (measure-spmf y) A)*
using *chain1 Y1* **by**(*rule measure-lub-spmf*)
also have $\dots \leq (SUP y \in \text{snd ‘ } Y. \text{measure (measure-spmf } y) \{y. \exists x \in A. R x y\})$ **using** *Y1*
by(*rule cSUP-least*)(*auto intro! cSUP-upper2*[*OF bdd-aboveI2*[*OF measure-spmf.subprob-measure-le-1*]] *rel-spmf-measureD* *R*)
also have $\dots = \text{measure (measure-spmf (lub-spmf (snd ‘ Y))) \{y. \exists x \in A. R x y\}}$
using *chain2 Y2* **by**(*rule measure-lub-spmf*[*symmetric*])
finally show *measure (measure-spmf (lub-spmf (fst ‘ Y))) A ≤ \dots .*
qed
then show *?P (?lub Y)* **by**(*simp add: prod-lub-def*)
qed

lemma *admissible-rel-spmf-mcont* [*cont-intro*]:

$\llbracket \text{mcont lub ord lub-spmf (ord-spmf (=)) } f; \text{mcont lub ord lub-spmf (ord-spmf (=)) } g \rrbracket$

$\implies \text{ccpo.admissible lub ord } (\lambda x. \text{rel-spmf } R (f x) (g x))$

by(*rule admissible-subst*[*OF admissible-rel-spmf, where f=λx. (f x, g x), simplified*])(*rule mcont-Pair*)

context includes *lifting-syntax*

begin

lemma *fixp-spmf-parametric'*:

assumes *f: λx. monotone (ord-spmf (=)) (ord-spmf (=)) F*

and *g: λx. monotone (ord-spmf (=)) (ord-spmf (=)) G*

and *param: (rel-spmf R ===> rel-spmf R) F G*

shows $(\text{rel-spmf } R) (\text{ccpo.fixp lub-spmf (ord-spmf (=)) } F) (\text{ccpo.fixp lub-spmf (ord-spmf (=)) } G)$

by(*rule parallel-fixp-induct*[*OF ccpo-spmf ccpo-spmf - f g*])(*auto intro: param*[*THEN rel-funD*])

lemma *fixp-spmf-parametric*:

assumes *f: λx. mono-spmf (λf. F f x)*

and *g: λx. mono-spmf (λf. G f x)*

and *param: ((A ===> rel-spmf R) ===> A ===> rel-spmf R) F G*

shows $(A ===> \text{rel-spmf } R) (\text{spm.f.fixp-fun } F) (\text{spm.f.fixp-fun } G)$

using *f g*

proof(*rule parallel-fixp-induct-1-1*[*OF partial-function-definitions-spmf partial-function-definitions-spmf - - reflexive reflexive, where P=(A ===> rel-spmf R)*])

show $\text{ccpo.admissible (prod-lub (fun-lub lub-spmf) (fun-lub lub-spmf)) (rel-prod (fun-ord (ord-spmf (=))) (fun-ord (ord-spmf (=)))) (\lambda x. (A ===> \text{rel-spmf } R) (\text{fst } x) (\text{snd } x))$


```

unfolding rel-fun-def
apply(rule admissible-all admissible-imp admissible-rel-spmf-mcont)+
apply(rule spmf.mcont2mcont[OF mcont-call])
apply(rule mcont-fst)
apply(rule spmf.mcont2mcont[OF mcont-call])
apply(rule mcont-snd)
done
show (A ==> rel-spmf R) (λ-. lub-spmf {}) (λ-. lub-spmf {}) by auto
show (A ==> rel-spmf R) (F f) (G g) if (A ==> rel-spmf R) f g for f g
using that by(rule rel-funD[OF param])
qed

end

end

end

```

25.12 Restrictions on spmfs

definition *restrict-spmf* :: 'a spmf ⇒ 'a set ⇒ 'a spmf (**infixl** 1 110)
where $p \upharpoonright A = \text{map-pmf } (\lambda x. x \gg= (\lambda y. \text{if } y \in A \text{ then Some } y \text{ else None})) p$

lemma *set-restrict-spmf [simp]*: $\text{set-spmf } (p \upharpoonright A) = \text{set-spmf } p \cap A$
by(fastforce simp add: restrict-spmf-def set-spmf-def split: bind-splits if-split-asm)

lemma *restrict-map-spmf*: $\text{map-spmf } f p \upharpoonright A = \text{map-spmf } f (p \upharpoonright (f^{-1} A))$
by(simp add: restrict-spmf-def pmf.map-comp o-def map-option-bind bind-map-option if-distrib cong del: if-weak-cong)

lemma *restrict-restrict-spmf [simp]*: $p \upharpoonright A \upharpoonright B = p \upharpoonright (A \cap B)$
by(auto simp add: restrict-spmf-def pmf.map-comp o-def intro!: pmf.map-comp bind-option-cong)

lemma *restrict-spmf-empty [simp]*: $p \upharpoonright \{\} = \text{return-pmf None}$
by(simp add: restrict-spmf-def)

lemma *restrict-spmf-UNIV [simp]*: $p \upharpoonright \text{UNIV} = p$
by(simp add: restrict-spmf-def)

lemma *spmfs-restrict-spmf-outside [simp]*: $x \notin A \implies \text{spmfs } (p \upharpoonright A) x = 0$
by(simp add: spmf-eq-0-set-spmf)

lemma *emeasure-restrict-spmf [simp]*:
 $\text{emeasure } (\text{measure-spmf } (p \upharpoonright A)) X = \text{emeasure } (\text{measure-spmf } p) (X \cap A)$
by(auto simp add: restrict-spmf-def measure-spmf-def emeasure-distr measurable-restrict-space1
emeasure-restrict-space space-restrict-space intro: arg-cong2[**where** f=emeasure]
split: bind-splits if-split-asm)

lemma *measure-restrict-spmf [simp]*:

$measure (measure\text{-}spmf (p \upharpoonright A)) X = measure (measure\text{-}spmf p) (X \cap A)$
using *emeasure-restrict-spmf*[of $p A X$]
by(*simp only: measure-spmf.emeasure-eq-measure ennreal-inj measure-nonneg*)

lemma *spmf-restrict-spmf*: $spmf (p \upharpoonright A) x = (if x \in A then\ spmf\ p\ x\ else\ 0)$
by(*simp add: spmf-conv-measure-spmf*)

lemma *spmf-restrict-spmf-inside* [*simp*]: $x \in A \implies spmf (p \upharpoonright A) x = spmf\ p\ x$
by(*simp add: spmf-restrict-spmf*)

lemma *pmf-restrict-spmf-None*: $pmf (p \upharpoonright A) None = pmf\ p\ None + measure (measure\text{-}spmf\ p) (- A)$

proof –

have [*simp*]: $None \notin Some\ '(- A)$ **by** *auto*
have $(\lambda x. x \gg (\lambda y. if\ y \in A\ then\ Some\ y\ else\ None)) -\{None\} = \{None\} \cup (Some\ '(- A))$
by(*auto split: bind-splits if-split-asm*)
then show *?thesis unfolding ereal.inject[symmetric]*
by(*simp add: restrict-spmf-def ennreal-pmf-map emeasure-pmf-single del: ereal.inject (simp add: pmf.rep-eq measure-pmf.finite-measure-Union[symmetric] measure-measure-spmf-conv-measure-pmf measure-pmf.emeasure-eq-measure)*)
qed

lemma *restrict-spmf-trivial*: $(\bigwedge x. x \in set\text{-}spmf\ p \implies x \in A) \implies p \upharpoonright A = p$
by(*rule spmf-eqI*)(*auto simp add: spmf-restrict-spmf spmf-eq-0-set-spmf*)

lemma *restrict-spmf-trivial'*: $set\text{-}spmf\ p \subseteq A \implies p \upharpoonright A = p$
by(*rule restrict-spmf-trivial*) *blast*

lemma *restrict-return-spmf*: $return\text{-}spmf\ x \upharpoonright A = (if\ x \in A\ then\ return\text{-}spmf\ x\ else\ return\text{-}pmf\ None)$
by(*simp add: restrict-spmf-def*)

lemma *restrict-return-spmf-inside* [*simp*]: $x \in A \implies return\text{-}spmf\ x \upharpoonright A = return\text{-}spmf\ x$
by(*simp add: restrict-return-spmf*)

lemma *restrict-return-spmf-outside* [*simp*]: $x \notin A \implies return\text{-}spmf\ x \upharpoonright A = return\text{-}pmf\ None$
by(*simp add: restrict-return-spmf*)

lemma *restrict-spmf-return-pmf-None* [*simp*]: $return\text{-}pmf\ None \upharpoonright A = return\text{-}pmf\ None$
by(*simp add: restrict-spmf-def*)

lemma *restrict-bind-pmf*: $bind\text{-}pmf\ p\ g \upharpoonright A = p \gg (\lambda x. g\ x \upharpoonright A)$
by(*simp add: restrict-spmf-def map-bind-pmf o-def*)

lemma *restrict-bind-spmf*: $bind\text{-}spmf\ p\ g \upharpoonright A = p \gg (\lambda x. g\ x \upharpoonright A)$

by(*auto simp add: bind-spmf-def restrict-bind-pmf cong del: option.case-cong-weak cong: option.case-cong intro!: bind-pmf-cong split: option.split*)

lemma *bind-restrict-pmf: bind-pmf (p | A) g = p \gg (λx . if $x \in \text{Some } A$ then $g\ x$ else $g\ \text{None}$)*

by(*auto simp add: restrict-spmf-def bind-map-pmf fun-eq-iff split: bind-split intro: arg-cong2[where f=bind-pmf]*)

lemma *bind-restrict-spmf: bind-spmf (p | A) g = p \gg (λx . if $x \in A$ then $g\ x$ else return-pmf None)*

by(*auto simp add: bind-spmf-def bind-restrict-pmf fun-eq-iff intro: arg-cong2[where f=bind-pmf] split: option.split*)

lemma *spmf-map-restrict: spmf (map-spmf fst (p | (snd - ' {y}))) x = spmf p (x, y)*

by(*subst spmf-map)(auto intro: arg-cong2[where f=measure] simp add: spmf-conv-measure-spmf)*

lemma *measure-eqI-restrict-spmf:*

assumes *rel-spmf R (restrict-spmf p A) (restrict-spmf q B)*

shows *measure (measure-spmf p) A = measure (measure-spmf q) B*

proof –

from *assms have weight-spmf (restrict-spmf p A) = weight-spmf (restrict-spmf q B)* **by**(*rule rel-spmf-weightD*)

thus *?thesis* **by**(*simp add: weight-spmf-def*)

qed

25.13 Subprobability distributions of sets

definition *spmf-of-set :: 'a set \Rightarrow 'a spmf*

where

spmf-of-set A = (if finite A \wedge A \neq {} then spmf-of-pmf (pmf-of-set A) else return-pmf None)

lemma *spmf-of-set: spmf (spmf-of-set A) x = indicator A x / card A*

by(*auto simp add: spmf-of-set-def*)

lemma *pmf-spmf-of-set-None [simp]: pmf (spmf-of-set A) None = indicator {A}. infinite A \vee A = {}*

by(*simp add: spmf-of-set-def*)

lemma *set-spmf-of-set: set-spmf (spmf-of-set A) = (if finite A then A else {})*

by(*simp add: spmf-of-set-def*)

lemma *set-spmf-of-set-finite [simp]: finite A \implies set-spmf (spmf-of-set A) = A*

by(*simp add: set-spmf-of-set*)

lemma *spmf-of-set-singleton: spmf-of-set {x} = return-spmf x*

by(*simp add: spmf-of-set-def pmf-of-set-singleton*)

lemma *map-spmf-of-set-inj-on* [simp]:

inj-on f A \implies *map-spmf f (spmf-of-set A) = spmf-of-set (f ‘ A)*

by(*auto simp add: spmf-of-set-def map-pmf-of-set-inj dest: finite-imageD*)

lemma *spmf-of-pmf-pmf-of-set* [simp]:

\llbracket *finite A; A \neq {}* $\rrbracket \implies$ *spmf-of-pmf (pmf-of-set A) = spmf-of-set A*

by(*simp add: spmf-of-set-def*)

lemma *weight-spmf-of-set*:

weight-spmf (spmf-of-set A) = (if finite A \wedge A \neq {} then 1 else 0)

by(*auto simp only: spmf-of-set-def weight-spmf-of-pmf weight-return-pmf-None split: if-split*)

lemma *weight-spmf-of-set-finite* [simp]: \llbracket *finite A; A \neq {}* $\rrbracket \implies$ *weight-spmf (spmf-of-set A) = 1*

by(*simp add: weight-spmf-of-set*)

lemma *weight-spmf-of-set-infinite* [simp]: *infinite A* \implies *weight-spmf (spmf-of-set A) = 0*

by(*simp add: weight-spmf-of-set*)

lemma *measure-spmf-spmf-of-set*:

measure-spmf (spmf-of-set A) = (if finite A \wedge A \neq {} then measure-pmf (pmf-of-set A) else null-measure (count-space UNIV))

by(*simp add: spmf-of-set-def del: spmf-of-pmf-pmf-of-set*)

lemma *emeasure-spmf-of-set*:

emeasure (measure-spmf (spmf-of-set S)) A = card (S \cap A) / card S

by(*auto simp add: measure-spmf-spmf-of-set emeasure-pmf-of-set*)

lemma *measure-spmf-of-set*:

measure (measure-spmf (spmf-of-set S)) A = card (S \cap A) / card S

by(*auto simp add: measure-spmf-spmf-of-set measure-pmf-of-set*)

lemma *nn-integral-spmf-of-set*: *nn-integral (measure-spmf (spmf-of-set A)) f = sum f A / card A*

by(*cases finite A*)(*auto simp add: spmf-of-set-def nn-integral-pmf-of-set card-gt-0-iff simp del: spmf-of-pmf-pmf-of-set*)

lemma *integral-spmf-of-set*: *integral^L (measure-spmf (spmf-of-set A)) f = sum f A / card A*

by(*clarsimp simp add: spmf-of-set-def integral-pmf-of-set card-gt-0-iff simp del: spmf-of-pmf-pmf-of-set*)

notepad begin — *pmf-of-set* is not fully parametric.

define *R* :: *nat* \Rightarrow *nat* \Rightarrow *bool* **where** *R x y* \longleftrightarrow (*x* \neq 0 \longrightarrow *y* = 0) **for** *x y*

define *A* :: *nat set* **where** *A* = {0, 1}

define *B* :: *nat set* **where** *B* = {0, 1, 2}

have *rel-set R A B unfolding R-def[abs-def] A-def B-def rel-set-def* **by** *auto*

```

have  $\neg$  rel-pmf R (pmf-of-set A) (pmf-of-set B)
proof
  assume rel-pmf R (pmf-of-set A) (pmf-of-set B)
  then obtain pq where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R x y$ 
    and 1: map-pmf fst pq = pmf-of-set A
    and 2: map-pmf snd pq = pmf-of-set B
    by cases auto
  have pmf (pmf-of-set B) 1 = 1 / 3 by (simp add: B-def)
  have pmf (pmf-of-set B) 2 = 1 / 3 by (simp add: B-def)

  have 2 / 3 = pmf (pmf-of-set B) 1 + pmf (pmf-of-set B) 2 by (simp add:
B-def)
  also have ... = measure (measure-pmf (pmf-of-set B)) ( $\{1\} \cup \{2\}$ )
  by (subst measure-pmf.finite-measure-Union) (simp-all add: measure-pmf-single)
  also have ... = emeasure (measure-pmf pq) (snd - '  $\{2, 1\}$ )
  unfolding 2 [symmetric] measure-pmf.emeasure-eq-measure [symmetric] by (simp)
  also have ... = emeasure (measure-pmf pq)  $\{(0, 2), (0, 1)\}$ 
  by (rule emeasure-eq-AE) (auto simp add: AE-measure-pmf-iff R-def dest!: pq)
  also have ...  $\leq$  emeasure (measure-pmf pq) (fst - '  $\{0\}$ )
  by (rule emeasure-mono) auto
  also have ... = emeasure (measure-pmf (pmf-of-set A))  $\{0\}$ 
  unfolding 1 [symmetric] by simp
  also have ... = pmf (pmf-of-set A) 0
  by (simp add: measure-pmf-single measure-pmf.emeasure-eq-measure)
  also have pmf (pmf-of-set A) 0 = 1 / 2 by (simp add: A-def)
  finally show False by (subst (asm) ennreal-le-iff; simp)
qed
end

```

```

lemma rel-pmf-of-set-bij:
  assumes f: bij-betw f A B
  and A:  $A \neq \{\}$  finite A
  and B:  $B \neq \{\}$  finite B
  and R:  $\bigwedge x. x \in A \implies R x (f x)$ 
  shows rel-pmf R (pmf-of-set A) (pmf-of-set B)
proof (rule pmf.rel-mono-strong)
  define AB where  $AB = (\lambda x. (x, f x)) ' A$ 
  define R' where  $R' x y \iff (x, y) \in AB$  for x y
  have  $(x, y) \in AB$  if  $(x, y) \in \text{set-pmf} (pmf-of-set AB)$  for x y
  using that by (auto simp add: AB-def A)
  moreover have map-pmf fst (pmf-of-set AB) = pmf-of-set A
  by (simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp
o-def)
  moreover
  from f have [simp]: inj-on f A by (rule bij-betw-imp-inj-on)
  from f have [simp]:  $f ' A = B$  by (rule bij-betw-imp-surj-on)
  have map-pmf snd (pmf-of-set AB) = pmf-of-set B
  by (simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp
o-def)

```

(*simp add: map-pmf-of-set-inj A*)
ultimately show *rel-pmf* ($\lambda x y. (x, y) \in AB$) (*pmf-of-set A*) (*pmf-of-set B*) ..
qed(*auto intro: R*)

lemma *rel-spmf-of-set-bij*:
assumes *f: bij-betw f A B*
and *R: $\bigwedge x. x \in A \implies R x (f x)$*
shows *rel-spmf R (spmf-of-set A) (spmf-of-set B)*
proof –
have *finite A \longleftrightarrow finite B* **using** *f* **by**(*rule bij-betw-finite*)
moreover have *A = {} \longleftrightarrow B = {}* **using** *f* **by**(*auto dest: bij-betw-empty2*
bij-betw-empty1)
ultimately show *?thesis* **using** *assms*
by(*auto simp add: spmf-of-set-def simp del: spmf-of-pmf-pmf-of-set intro: rel-pmf-of-set-bij*)
qed

context includes *lifting-syntax*
begin

lemma *rel-spmf-of-set*:
assumes *bi-unique R*
shows (*rel-set R \implies rel-spmf R*) *spmf-of-set spmf-of-set*
proof
fix *A B*
assume *R: rel-set R A B*
with *assms* **obtain** *f* **where** *bij-betw f A B* **and** *f: $\bigwedge x. x \in A \implies R x (f x)$*
by(*auto dest: bi-unique-rel-set-bij-betw*)
then show *rel-spmf R (spmf-of-set A) (spmf-of-set B)* **by**(*rule rel-spmf-of-set-bij*)
qed

end

lemma *map-mem-spmf-of-set*:
assumes *finite B B \neq {}*
shows *map-spmf ($\lambda x. x \in A$) (spmf-of-set B) = spmf-of-pmf (bernoulli-pmf*
(card (A \cap B) / card B))
(is ?lhs = ?rhs)
proof(*rule spmf-eqI*)
fix *i*
have *ennreal (spmf ?lhs i) = card (B \cap ($\lambda x. x \in A$) - ‘{i}) / (card B)*
by(*subst ennreal-spmf-map*)(*simp add: measure-spmf-spmf-of-set assms emea-*
sure-pmf-of-set)
also have $\dots =$ (*if i then card (B \cap A) / card B else card (B - A) / card B*)
by(*auto intro: arg-cong[where f=card]*)
also have $\dots =$ (*if i then card (B \cap A) / card B else (card B - card (B \cap A))*
/ card B)
by(*auto simp add: card-Diff-subset-Int assms*)
also have $\dots =$ *ennreal (spmf ?rhs i)*
by(*simp add: assms card-gt-0-iff field-simps card-mono Int-commute of-nat-diff*)

finally show $\text{spmf } ?lhs \ i = \text{spmf } ?rhs \ i$ **by** *simp*
qed

abbreviation $\text{coin-spmf} :: \text{bool} \ \text{spmf}$
where $\text{coin-spmf} \equiv \text{spmf-of-set UNIV}$

lemma $\text{map-eq-const-coin-spmf}$: $\text{map-spmf } ((=) \ c) \ \text{coin-spmf} = \text{coin-spmf}$

proof –

have $\text{inj } ((\longleftrightarrow) \ c) \ \text{range } ((\longleftrightarrow) \ c) = \text{UNIV}$ **by**(*auto intro: inj-onI*)

then show *?thesis* **by** *simp*

qed

lemma $\text{bind-coin-spmf-eq-const}$: $\text{coin-spmf} \gg= (\lambda x :: \text{bool}. \text{return-spmf } (b = x))$
 $= \text{coin-spmf}$

using $\text{map-eq-const-coin-spmf}$ **unfolding** $\text{map-spmf-conv-bind-spmf}$ **by** *simp*

lemma $\text{bind-coin-spmf-eq-const}'$: $\text{coin-spmf} \gg= (\lambda x :: \text{bool}. \text{return-spmf } (x = b))$
 $= \text{coin-spmf}$

by(*rewrite in - = \sqsupset bind-coin-spmf-eq-const[symmetric, of b](auto intro: bind-spmf-cong)*)

25.14 Losslessness

definition $\text{lossless-spmf} :: 'a \ \text{spmf} \Rightarrow \text{bool}$

where $\text{lossless-spmf } p \longleftrightarrow \text{weight-spmf } p = 1$

lemma $\text{lossless-iff-pmf-None}$: $\text{lossless-spmf } p \longleftrightarrow \text{pmf } p \ \text{None} = 0$

by(*simp add: lossless-spmf-def pmf-None-eq-weight-spmf*)

lemma $\text{lossless-return-spmf [iff]}$: $\text{lossless-spmf } (\text{return-spmf } x)$

by(*simp add: lossless-iff-pmf-None*)

lemma $\text{lossless-return-pmf-None [iff]}$: $\neg \text{lossless-spmf } (\text{return-pmf } \text{None})$

by(*simp add: lossless-iff-pmf-None*)

lemma $\text{lossless-map-spmf [simp]}$: $\text{lossless-spmf } (\text{map-spmf } f \ p) \longleftrightarrow \text{lossless-spmf } p$

by(*auto simp add: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma $\text{lossless-bind-spmf [simp]}$:

$\text{lossless-spmf } (p \gg= f) \longleftrightarrow \text{lossless-spmf } p \wedge (\forall x \in \text{set-spmf } p. \text{lossless-spmf } (f \ x))$

by(*simp add: lossless-iff-pmf-None pmf-bind-spmf-None add-nonneg-eq-0-iff integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-spmf.integrable-const-bound[where B=1] pmf-le-1*)

lemma $\text{lossless-weight-spmfD}$: $\text{lossless-spmf } p \Longrightarrow \text{weight-spmf } p = 1$

by(*simp add: lossless-spmf-def*)

lemma $\text{lossless-iff-set-pmf-None}$:

$lossless\text{-}spmf\ p \longleftrightarrow None \notin set\text{-}pmf\ p$
by (*simp add: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma *lossless-spmf-of-set* [*simp*]: $lossless\text{-}spmf\ (spmf\text{-of}\text{-}set\ A) \longleftrightarrow finite\ A \wedge A \neq \{\}$
by(*auto simp add: lossless-spmf-def weight-spmf-of-set*)

lemma *lossless-spmf-spmf-of-spmf* [*simp*]: $lossless\text{-}spmf\ (spmf\text{-of}\text{-}pmf\ p)$
by(*simp add: lossless-spmf-def*)

lemma *lossless-spmf-bind-pmf* [*simp*]:
 $lossless\text{-}spmf\ (bind\text{-}pmf\ p\ f) \longleftrightarrow (\forall x \in set\text{-}pmf\ p. lossless\text{-}spmf\ (f\ x))$
by(*simp add: lossless-iff-pmf-None pmf-bind integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-pmf.integrable-const-bound[where B=1] AE-measure-pmf-iff pmf-le-1*)

lemma *lossless-spmf-conv-spmf-of-pmf*: $lossless\text{-}spmf\ p \longleftrightarrow (\exists p'. p = spmf\text{-of}\text{-}pmf\ p')$

proof

assume *lossless-spmf p*

hence *: $\bigwedge y. y \in set\text{-}pmf\ p \implies \exists x. y = Some\ x$

by(*case-tac y*)(*simp-all add: lossless-iff-set-pmf-None*)

let *?p = map-pmf the p*

have $p = spmf\text{-of}\text{-}pmf\ ?p$

proof(*rule spmf-eqI*)

fix *i*

have $ennreal\ (pmf\ (map\text{-}pmf\ the\ p)\ i) = \int^+ x. indicator\ (the\ \text{-}\{i\})\ x\ \partial p$

by(*simp add: ennreal-pmf-map*)

also have $\dots = \int^+ x. indicator\ \{i\}\ x\ \partial measure\text{-}spmf\ p$ **unfolding** *measure-spmf-def*

by(*subst nn-integral-distr*)(*auto simp add: nn-integral-restrict-space AE-measure-pmf-iff simp del: nn-integral-indicator intro!: nn-integral-cong-AE split: split-indicator dest!: **)

also have $\dots = spmf\ p\ i$ **by**(*simp add: emeasure-spmf-single*)

finally show $spmf\ p\ i = spmf\ (spmf\text{-of}\text{-}pmf\ ?p)\ i$ **by** *simp*

qed

thus $\exists p'. p = spmf\text{-of}\text{-}pmf\ p' ..$

qed *auto*

lemma *spmf-False-conv-True*: $lossless\text{-}spmf\ p \implies spmf\ p\ False = 1 - spmf\ p\ True$
by(*clarsimp simp add: lossless-spmf-conv-spmf-of-pmf pmf-False-conv-True*)

lemma *spmf-True-conv-False*: $lossless\text{-}spmf\ p \implies spmf\ p\ True = 1 - spmf\ p\ False$
by(*simp add: spmf-False-conv-True*)

lemma *bind-eq-return-spmf*:

$bind\text{-}spmf\ p\ f = return\text{-}spmf\ x \longleftrightarrow (\forall y \in set\text{-}spmf\ p. f\ y = return\text{-}spmf\ x) \wedge lossless\text{-}spmf\ p$

by(*auto simp add: bind-spmf-def bind-eq-return-pmf in-set-spmf lossless-iff-pmf-None*)

pmf-eq-0-set-pmf iff del: not-None-eq split: option.split)

lemma *rel-spmf-return-spmf2*:

rel-spmf R p (return-spmf x) \longleftrightarrow lossless-spmf p \wedge ($\forall a \in \text{set-spmf } p. R a x$)
by(*auto simp add: lossless-iff-set-pmf-None rel-pmf-return-pmf2 option-rel-Some2 in-set-spmf, metis in-set-spmf not-None-eq*)

lemma *rel-spmf-return-spmf1*:

rel-spmf R (return-spmf x) p \longleftrightarrow lossless-spmf p \wedge ($\forall a \in \text{set-spmf } p. R x a$)
using *rel-spmf-return-spmf2*[of R^{-1-1}] **by**(*simp add: spmf-rel-conversep*)

lemma *rel-spmf-bindI1*:

assumes *f*: $\bigwedge x. x \in \text{set-spmf } p \implies \text{rel-spmf } R (f x) q$
and *p*: *lossless-spmf p*
shows *rel-spmf R (bind-spmf p f) q*

proof –

fix *x* :: 'a
have *rel-spmf R (bind-spmf p f) (bind-spmf (return-spmf x) ($\lambda \cdot$. q))*
by(*rule rel-spmf-bindI*[**where** $R = \lambda x \cdot x \in \text{set-spmf } p$])(*simp-all add: rel-spmf-return-spmf2 p f*)
then show *?thesis* **by** *simp*
qed

lemma *rel-spmf-bindI2*:

$\llbracket \bigwedge x. x \in \text{set-spmf } q \implies \text{rel-spmf } R p (f x); \text{lossless-spmf } q \rrbracket$
 $\implies \text{rel-spmf } R p (\text{bind-spmf } q f)$
using *rel-spmf-bindI1*[of *q conversep R f p*] **by**(*simp add: spmf-rel-conversep*)

25.15 Scaling

definition *scale-spmf* :: *real* \Rightarrow 'a *spmf* \Rightarrow 'a *spmf*

where

scale-spmf r p = *embed-spmf* ($\lambda x. \text{min} (\text{inverse} (\text{weight-spmf } p)) (\text{max } 0 r) * \text{spmf } p x$)

lemma *scale-spmf-le-1*:

$(\int^+ x. \text{min} (\text{inverse} (\text{weight-spmf } p)) (\text{max } 0 r) * \text{spmf } p x \text{ } \partial \text{count-space UNIV}) \leq 1$ (**is** *?lhs* \leq -)

proof –

have *?lhs* = $\text{min} (\text{inverse} (\text{weight-spmf } p)) (\text{max } 0 r) * \int^+ x. \text{spmf } p x \text{ } \partial \text{count-space UNIV}$

by(*subst nn-integral-cmult*[*symmetric*])(*simp-all add: weight-spmf-nonneg max-def min-def ennreal-mult*)

also have $\dots \leq 1$ **unfolding** *weight-spmf-eq-nn-integral-spmf*[*symmetric*]

by(*simp add: min-def max-def weight-spmf-nonneg order.strict-iff-order field-simps ennreal-mult*[*symmetric*])

finally show *?thesis* .

qed

lemma *spmf-scale-spmf*: $\text{spmf } (\text{scale-spmf } r \ p) \ x = \max 0 \ (\min \ (\text{inverse } (\text{weight-spmf } p)) \ r) \ * \ \text{spmf } p \ x$ (**is** ?lhs = ?rhs)
unfolding *scale-spmf-def*
apply(*subst spmf-embed-spmf*[*OF scale-spmf-le-1*])
apply(*simp add: max-def min-def weight-spmf-le-0 field-simps weight-spmf-nonneg not-le order.strict-iff-order*)
apply(*metis antisym-conv order-trans weight-spmf-nonneg zero-le-mult-iff zero-le-one*)
done

lemma *real-inverse-le-1-iff*: **fixes** $x :: \text{real}$
shows $\llbracket 0 \leq x; x \leq 1 \rrbracket \implies 1 / x \leq 1 \longleftrightarrow x = 1 \vee x = 0$
by *auto*

lemma *spmf-scale-spmf'*: $r \leq 1 \implies \text{spmf } (\text{scale-spmf } r \ p) \ x = \max 0 \ r \ * \ \text{spmf } p \ x$
using *real-inverse-le-1-iff*[*OF weight-spmf-nonneg weight-spmf-le-1, of p*]
by(*auto simp add: spmf-scale-spmf max-def min-def field-simps*)(*metis pmf-le-0-iff spmf-le-weight*)

lemma *scale-spmf-neg*: $r \leq 0 \implies \text{scale-spmf } r \ p = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf' max-def*)

lemma *scale-spmf-return-None* [*simp*]: $\text{scale-spmf } r \ (\text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf*)

lemma *scale-spmf-conv-bind-bernoulli*:
assumes $r \leq 1$
shows $\text{scale-spmf } r \ p = \text{bind-pmf } (\text{bernoulli-pmf } r) \ (\lambda b. \text{if } b \ \text{then } p \ \text{else } \text{return-pmf } \text{None})$ (**is** ?lhs = ?rhs)
proof(*rule spmf-eqI*)
fix x
have $\text{ennreal } (\text{spmf } ?lhs \ x) = \text{ennreal } (\text{spmf } ?rhs \ x)$ **using** *assms*
unfolding *spmf-scale-spmf ennreal-pmf-bind nn-integral-measure-pmf UNIV-bool bernoulli-pmf.rep-eq*
apply(*auto simp add: nn-integral-count-space-finite max-def min-def field-simps real-inverse-le-1-iff*[*OF weight-spmf-nonneg weight-spmf-le-1*] *weight-spmf-lt-0 not-le ennreal-mult*[*symmetric*])
apply (*metis pmf-le-0-iff spmf-le-weight*)
apply (*metis pmf-le-0-iff spmf-le-weight*)
apply (*meson le-divide-eq-1-pos measure-spmf.subprob-measure-le-1 not-less order-trans weight-spmf-le-0*)
by (*meson divide-le-0-1-iff less-imp-le order-trans weight-spmf-le-0*)
thus $\text{spmf } ?lhs \ x = \text{spmf } ?rhs \ x$ **by** *simp*
qed

lemma *nn-integral-spmf*: $(\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } A) = \text{emeasure } (\text{measure-spmf } p) \ A$
apply(*simp add: measure-spmf-def emeasure-distr emeasure-restrict-space space-restrict-space*)

```

nn-integral-pmf[symmetric])
apply(rule nn-integral-bij-count-space[where  $g=Some$ ])
apply(auto simp add: bij-betw-def)
done

```

```

lemma measure-spmf-scale-spmf: measure-spmf (scale-spmf r p) = scale-measure
(min (inverse (weight-spmf p)) r) (measure-spmf p)
apply(rule measure-eqI)
apply simp
apply(simp add: nn-integral-spmf[symmetric] spmf-scale-spmf)
apply(subst nn-integral-cmult[symmetric])
apply(auto simp add: max-def min-def ennreal-mult[symmetric] not-le ennreal-lt-0)
done

```

```

lemma measure-spmf-scale-spmf':
 $r \leq 1 \implies$  measure-spmf (scale-spmf r p) = scale-measure r (measure-spmf p)
unfolding measure-spmf-scale-spmf
apply(cases weight-spmf p > 0)
apply(simp add: min.absorb2 field-simps weight-spmf-le-1 mult-le-one)
apply(clarsimp simp add: weight-spmf-le-0 min-def scale-spmf-neg weight-spmf-eq-0
not-less)
done

```

```

lemma scale-spmf-1 [simp]: scale-spmf 1 p = p
apply(rule spmf-eqI)
apply(simp add: spmf-scale-spmf max-def min-def order.strict-iff-order field-simps
weight-spmf-nonneg)
apply(metis antisym-conv divide-le-eq-1 less-imp-le pmf-nonneg spmf-le-weight weight-spmf-nonneg
weight-spmf-le-1)
done

```

```

lemma scale-spmf-0 [simp]: scale-spmf 0 p = return-pmf None
by(rule spmf-eqI)(simp add: spmf-scale-spmf min-def max-def weight-spmf-le-0)

```

```

lemma bind-scale-spmf:
assumes  $r \leq 1$ 
shows bind-spmf (scale-spmf r p) f = bind-spmf p ( $\lambda x.$  scale-spmf r (f x))
(is ?lhs = ?rhs)
proof(rule spmf-eqI)
fix x
have ennreal (spmf ?lhs x) = ennreal (spmf ?rhs x) using r
by(simp add: ennreal-spmf-bind measure-spmf-scale-spmf' nn-integral-scale-measure
spmf-scale-spmf')
(simp add: ennreal-mult ennreal-lt-0 nn-integral-cmult max-def min-def)
thus spmf ?lhs x = spmf ?rhs x by simp
qed

```

```

lemma scale-bind-spmf:
assumes  $r \leq 1$ 

```

shows $\text{scale-spmf } r \ (\text{bind-spmf } p \ f) = \text{bind-spmf } p \ (\lambda x. \text{scale-spmf } r \ (f \ x))$
(is $?lhs = ?rhs$)
proof(*rule spmf-eqI*)
fix x
have $\text{ennreal } (\text{spmf } ?lhs \ x) = \text{ennreal } (\text{spmf } ?rhs \ x)$ **using** *assms*
unfolding $\text{spmf-scale-spmf}'[OF \ \text{assms}]$
by(*simp add: ennreal-mult ennreal-spmf-bind spmf-scale-spmf' nn-integral-cmult max-def min-def*)
thus $\text{spmf } ?lhs \ x = \text{spmf } ?rhs \ x$ **by** *simp*
qed

lemma *bind-spmf-const*: $\text{bind-spmf } p \ (\lambda x. \ q) = \text{scale-spmf } (\text{weight-spmf } p) \ q$ **(is** $?lhs = ?rhs$)
proof(*rule spmf-eqI*)
fix x
have $\text{ennreal } (\text{spmf } ?lhs \ x) = \text{ennreal } (\text{spmf } ?rhs \ x)$
using *measure-spmf.subprob-measure-le-1[of p space (measure-spmf p)]*
by(*subst ennreal-spmf-bind)(simp add: spmf-scale-spmf' weight-spmf-le-1 ennreal-mult mult.commute max-def min-def measure-spmf.emmeasure-eq-measure*)
thus $\text{spmf } ?lhs \ x = \text{spmf } ?rhs \ x$ **by** *simp*
qed

lemma *map-scale-spmf*: $\text{map-spmf } f \ (\text{scale-spmf } r \ p) = \text{scale-spmf } r \ (\text{map-spmf } f \ p)$ **(is** $?lhs = ?rhs$)
proof(*rule spmf-eqI*)
fix i
show $\text{spmf } ?lhs \ i = \text{spmf } ?rhs \ i$ **unfolding** *spmf-scale-spmf*
by(*subst (1 2) spmf-map)(auto simp add: measure-spmf-scale-spmf max-def min-def ennreal-lt-0*)
qed

lemma *set-scale-spmf*: $\text{set-spmf } (\text{scale-spmf } r \ p) = (\text{if } r > 0 \ \text{then } \text{set-spmf } p \ \text{else } \{\})$
apply(*auto simp add: in-set-spmf-iff-spmf spmf-scale-spmf*)
apply(*simp add: max-def min-def not-le weight-spmf-lt-0 weight-spmf-eq-0 split: if-split-asm*)
done

lemma *set-scale-spmf'* [*simp*]: $0 < r \implies \text{set-spmf } (\text{scale-spmf } r \ p) = \text{set-spmf } p$
by(*simp add: set-scale-spmf*)

lemma *rel-spmf-scaleI*:
assumes $r > 0 \implies \text{rel-spmf } A \ p \ q$
shows $\text{rel-spmf } A \ (\text{scale-spmf } r \ p) \ (\text{scale-spmf } r \ q)$
proof(*cases r > 0*)
case *True*
from *assms[OF this]* **show** *?thesis*
by(*rule rel-spmfE)(auto simp add: map-scale-spmf[symmetric] spmf-rel-map True intro: rel-spmf-refI*)

qed(*simp add: not-less scale-spmf-neg*)

lemma *weight-scale-spmf*: $\text{weight-spmf } (\text{scale-spmf } r \ p) = \min \ 1 \ (\max \ 0 \ r \ * \ \text{weight-spmf } p)$

proof –

have *ennreal* ($\text{weight-spmf } (\text{scale-spmf } r \ p)$) = $\min \ 1 \ (\max \ 0 \ r \ * \ \text{ennreal } (\text{weight-spmf } p))$

unfolding *weight-spmf-eq-nn-integral-spmf*

apply(*simp add: spmf-scale-spmf ennreal-mult zero-ereal-def[symmetric] nn-integral-cmult*)

apply(*auto simp add: weight-spmf-eq-nn-integral-spmf[symmetric] field-simps min-def max-def not-le weight-spmf-lt-0 ennreal-mult[symmetric]*)

subgoal **by**(*subst (asm) ennreal-mult[symmetric], meson divide-less-0-1-iff le-less-trans not-le weight-spmf-lt-0, simp+, meson not-le pos-divide-le-eq weight-spmf-le-0*)

subgoal **by**(*cases r ≥ 0)(simp-all add: ennreal-mult[symmetric] weight-spmf-nonneg ennreal-lt-0, meson le-less-trans not-le pos-divide-le-eq zero-less-divide-1-iff)*)

done

thus *?thesis* **by**(*auto simp add: min-def max-def ennreal-mult[symmetric] split: if-split-asm*)

qed

lemma *weight-scale-spmf'* [*simp*]:

$\llbracket 0 \leq r; r \leq 1 \rrbracket \implies \text{weight-spmf } (\text{scale-spmf } r \ p) = r \ * \ \text{weight-spmf } p$

by(*simp add: weight-scale-spmf max-def min-def)(metis antisym-conv mult-left-le order-trans weight-spmf-le-1)*)

lemma *pmf-scale-spmf-None*:

$\text{pmf } (\text{scale-spmf } k \ p) \ \text{None} = 1 - \min \ 1 \ (\max \ 0 \ k \ * \ (1 - \text{pmf } p \ \text{None}))$

unfolding *pmf-None-eq-weight-spmf* **by**(*simp add: weight-scale-spmf*)

lemma *scale-scale-spmf*:

$\text{scale-spmf } r \ (\text{scale-spmf } r' \ p) = \text{scale-spmf } (r \ * \ \max \ 0 \ (\min \ (\text{inverse } (\text{weight-spmf } p)) \ r')) \ p$

(*is ?lhs = ?rhs*)

proof(*cases weight-spmf p > 0*)

case *False*

thus *?thesis*

by (*simp add: weight-spmf-eq-0 zero-less-measure-iff*)

next

case *True*

show *?thesis*

proof(*rule spmf-eqI*)

fix *i*

have $\max \ 0 \ (\min \ (1 / \text{weight-spmf } p) \ r') \ * \ \max \ 0 \ (\min \ (1 / \min \ 1 \ (\text{weight-spmf } p \ * \ \max \ 0 \ r')) \ r) =$

$\max \ 0 \ (\min \ (1 / \text{weight-spmf } p) \ (r \ * \ \max \ 0 \ (\min \ (1 / \text{weight-spmf } p) \ r')))$

using *True*

by(*simp add: field-simps max-def min.absorb-iff2[symmetric]*)(*auto simp add: min-def field-simps zero-le-mult-iff*)

then show *spmf ?lhs i = spmf ?rhs i*

apply (*subst spmf-scale-spmf*) +
by (*metis (no-types, opaque-lifting) inverse-eq-divide mult.commute mult.left-commute weight-scale-spmf*)
qed
qed

lemma *scale-scale-spmf' [simp]*:
 $\llbracket 0 \leq r; r \leq 1; 0 \leq r'; r' \leq 1 \rrbracket$
 $\implies \text{scale-spmf } r (\text{scale-spmf } r' p) = \text{scale-spmf } (r * r') p$
apply(*cases weight-spmf p > 0*)
apply(*auto simp add: scale-scale-spmf min-def max-def field-simps not-le weight-spmf-lt-0 weight-spmf-eq-0 not-less weight-spmf-le-0*)
apply(*subgoal-tac 1 = r'*)
apply (*metis (no-types) div-by-1 eq-iff measure-spmf.subprob-measure-le-1 mult.commute mult-cancel-right1*)
apply(*meson eq-iff le-divide-eq-1-pos measure-spmf.subprob-measure-le-1 mult-imp-div-pos-le order.trans*)
done

lemma *scale-spmf-eq-same: scale-spmf r p = p \longleftrightarrow weight-spmf p = 0 \vee r = 1 \vee r \geq 1 \wedge weight-spmf p = 1*
(is ?lhs \longleftrightarrow ?rhs)

proof
assume *?lhs*
hence *weight-spmf (scale-spmf r p) = weight-spmf p* **by** *simp*
hence ***: *min 1 (max 0 r * weight-spmf p) = weight-spmf p* **by**(*simp add: weight-scale-spmf*)
hence ****: *weight-spmf p = 0 \vee r \geq 1* **by**(*auto simp add: min-def max-def split: if-split-asm*)
show *?rhs*
proof(*cases weight-spmf p = 0*)
case *False*
with **** **have** *r \geq 1* **by** *simp*
with *** **False** **have** *r = 1 \vee weight-spmf p = 1* **by**(*simp add: max-def min-def not-le split: if-split-asm*)
with *r \geq 1* **show** *?thesis* **by** *simp*
qed *simp*
qed(*auto intro!: spmf-eqI simp add: spmf-scale-spmf, metis pmf-le-0-iff spmf-le-weight*)

lemma *map-const-spmf-of-set*:
 $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{map-spmf } (\lambda \cdot. c) (\text{spmof-of-set } A) = \text{return-spmf } c$
by(*simp add: map-spmf-conv-bind-spmf bind-spmf-const*)

25.16 Conditional spmfs

lemma *set-pmf-Int-Some: set-pmf p \cap Some 'A = {} \longleftrightarrow set-spmf p \cap A = {}*
by(*auto simp add: in-set-spmf*)

lemma *measure-spmf-zero-iff: measure (measure-spmf p) A = 0 \longleftrightarrow set-spmf p*

$\cap A = \{\}$

unfolding *measure-measure-spmf-conv-measure-pmf* **by** (*simp add: measure-pmf-zero-iff set-pmf-Int-Some*)

definition *cond-spmf* $:: 'a \text{ spmf} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ spmf}$

where *cond-spmf* $p \ A = (\text{if } \text{set-spmf } p \ \cap \ A = \{\} \text{ then } \text{return-pmf } \text{None} \text{ else } \text{cond-pmf } p \ (\text{Some } 'A))$

lemma *set-cond-spmf* [*simp*]: $\text{set-spmf } (\text{cond-spmf } p \ A) = \text{set-spmf } p \ \cap \ A$

by (*auto* $\not\llcorner$ *simp add: cond-spmf-def in-set-spmf iff: set-cond-pmf [THEN set-eq-iff [THEN iffD1], THEN spec, rotated]*)

lemma *cond-map-spmf* [*simp*]: $\text{cond-spmf } (\text{map-spmf } f \ p) \ A = \text{map-spmf } f \ (\text{cond-spmf } p \ (f \ - 'A))$

proof –

have *map-option* $f \ - ' \text{Some } 'A = \text{Some } 'f \ - 'A$ **by** *auto*

moreover have *set-pmf* $p \ \cap \ \text{map-option } f \ - ' \text{Some } 'A \neq \{\}$ **if** *Some* $x \in \text{set-pmf } p \ f \ x \in A$ **for** x

using *that by auto*

ultimately show *?thesis* **by** (*auto simp add: cond-spmf-def in-set-spmf cond-map-pmf*)

qed

lemma *spmf-cond-spmf* [*simp*]:

spmf $(\text{cond-spmf } p \ A) \ x = (\text{if } x \in A \text{ then } \text{spmf } p \ x / \text{measure } (\text{measure-spmf } p) \ A \text{ else } 0)$

by (*auto simp add: cond-spmf-def pmf-cond set-pmf-Int-Some [symmetric] measure-measure-spmf-conv-measure-measure-pmf-zero-iff*)

lemma *bind-eq-return-pmf-None*:

bind-spmf $p \ f = \text{return-pmf } \text{None} \iff (\forall x \in \text{set-spmf } p. f \ x = \text{return-pmf } \text{None})$

by (*auto simp add: bind-spmf-def bind-eq-return-pmf in-set-spmf split: option.splits*)

lemma *return-pmf-None-eq-bind*:

return-pmf $\text{None} = \text{bind-spmf } p \ f \iff (\forall x \in \text{set-spmf } p. f \ x = \text{return-pmf } \text{None})$

using *bind-eq-return-pmf-None [of p f]* **by** *auto*

25.17 Product spmf

definition *pair-spmf* $:: 'a \text{ spmf} \Rightarrow 'b \text{ spmf} \Rightarrow ('a \times 'b) \text{ spmf}$

where *pair-spmf* $p \ q = \text{bind-pmf } (\text{pair-pmf } p \ q) \ (\lambda xy. \text{case } xy \text{ of } (\text{Some } x, \text{Some } y) \Rightarrow \text{return-spmf } (x, y) \mid - \Rightarrow \text{return-pmf } \text{None})$

lemma *map-fst-pair-spmf* [*simp*]: $\text{map-spmf } \text{fst} \ (\text{pair-spmf } p \ q) = \text{scale-spmf } (\text{weight-spmf } q) \ p$

unfolding *bind-spmf-const* [*symmetric*]

apply (*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*)

apply (*subst bind-commute-pmf*)

apply (*auto intro!: bind-pmf-cong [OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse option.case-distrib [where h=map-spmf -] op-*

tion.case-distrib[symmetric] case-option-id split: option.split cong del: option.case-cong-weak)
done

lemma *map-snd-pair-spmf [simp]: map-spmf snd (pair-spmf p q) = scale-spmf (weight-spmf p) q*

unfolding *bind-spmf-const[symmetric]*

apply(*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*

cong del: option.case-cong-weak)

apply(*auto intro!: bind-pmf-cong[OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse option.case-distrib[where h=map-spmf-] option.case-distrib[symmetric] case-option-id split: option.split cong del: option.case-cong-weak*)

done

lemma *set-pair-spmf [simp]: set-spmf (pair-spmf p q) = set-spmf p × set-spmf q*

by(*auto 4 3 simp add: pair-spmf-def set-spmf-bind-pmf bind-UNION in-set-spmf intro: rev-beqI split: option.splits*)

lemma *spmf-pair [simp]: spmf (pair-spmf p q) (x, y) = spmf p x * spmf q y (is ?lhs = ?rhs)*

proof –

have *ennreal ?lhs = ∫⁺ a. ∫⁺ b. indicator {(x, y)} (a, b) ∂measure-spmf q*
∂measure-spmf p

unfolding *measure-spmf-def pair-spmf-def ennreal-pmf-bind nn-integral-pair-pmf'*

by(*auto simp add: zero-ereal-def[symmetric] nn-integral-distr nn-integral-restrict-space nn-integral-multc[symmetric] intro!: nn-integral-cong split: option.split split-indicator*)

also have *... = ∫⁺ a. (∫⁺ b. indicator {y} b ∂measure-spmf q) * indicator {x}*
a ∂measure-spmf p

by(*subst nn-integral-multc[symmetric])(auto intro!: nn-integral-cong split: split-indicator)*

also have *... = ennreal ?rhs* **by**(*simp add: emeasure-spmf-single max-def ennreal-mult mult.commute*)

finally show *?thesis* **by** *simp*

qed

lemma *pair-map-spmf2: pair-spmf p (map-spmf f q) = map-spmf (apsnd f) (pair-spmf p q)*

by(*auto simp add: pair-spmf-def pair-map-pmf2 bind-map-pmf map-bind-pmf intro: bind-pmf-cong split: option.split*)

lemma *pair-map-spmf1: pair-spmf (map-spmf f p) q = map-spmf (apfst f) (pair-spmf p q)*

by(*auto simp add: pair-spmf-def pair-map-pmf1 bind-map-pmf map-bind-pmf intro: bind-pmf-cong split: option.split*)

lemma *pair-map-spmf: pair-spmf (map-spmf f p) (map-spmf g q) = map-spmf (map-prod f g) (pair-spmf p q)*

unfolding *pair-map-spmf2 pair-map-spmf1 spmf.map-comp* **by**(*simp add: apfst-def apsnd-def o-def prod.map-comp*)

lemma *pair-spmf-alt-def*: $\text{pair-spmf } p \ q = \text{bind-spmf } p \ (\lambda x. \text{bind-spmf } q \ (\lambda y. \text{return-spmf } (x, y)))$
by(*auto simp add: pair-spmf-def pair-pmf-def bind-spmf-def bind-assoc-pmf bind-return-pmf split: option.split intro: bind-pmf-cong*)

lemma *weight-pair-spmf [simp]*: $\text{weight-spmf } (\text{pair-spmf } p \ q) = \text{weight-spmf } p \ * \ \text{weight-spmf } q$
unfolding *pair-spmf-alt-def* **by**(*simp add: weight-bind-spmf o-def*)

lemma *pair-scale-spmf1*:
 $r \leq 1 \implies \text{pair-spmf } (\text{scale-spmf } r \ p) \ q = \text{scale-spmf } r \ (\text{pair-spmf } p \ q)$
by(*simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf*)

lemma *pair-scale-spmf2*:
 $r \leq 1 \implies \text{pair-spmf } p \ (\text{scale-spmf } r \ q) = \text{scale-spmf } r \ (\text{pair-spmf } p \ q)$
by(*simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf*)

lemma *pair-spmf-return-None1 [simp]*: $\text{pair-spmf } (\text{return-pmf } \text{None}) \ p = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*clarsimp*)

lemma *pair-spmf-return-None2 [simp]*: $\text{pair-spmf } p \ (\text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$
by(*rule spmf-eqI*)(*clarsimp*)

lemma *pair-spmf-return-spmf1*: $\text{pair-spmf } (\text{return-spmf } x) \ q = \text{map-spmf } (\text{Pair } x) \ q$
by(*rule spmf-eqI*)(*auto split: split-indicator simp add: spmf-map-inj' inj-on-def intro: spmf-map-outside*)

lemma *pair-spmf-return-spmf2*: $\text{pair-spmf } p \ (\text{return-spmf } y) = \text{map-spmf } (\lambda x. (x, y)) \ p$
by(*rule spmf-eqI*)(*auto split: split-indicator simp add: inj-on-def intro!: spmf-map-outside spmf-map-inj'[symmetric]*)

lemma *pair-spmf-return-spmf [simp]*: $\text{pair-spmf } (\text{return-spmf } x) \ (\text{return-spmf } y) = \text{return-spmf } (x, y)$
by(*simp add: pair-spmf-return-spmf1*)

lemma *rel-pair-spmf-prod*:
 $\text{rel-spmf } (\text{rel-prod } A \ B) \ (\text{pair-spmf } p \ q) \ (\text{pair-spmf } p' \ q') \longleftrightarrow$
 $\text{rel-spmf } A \ (\text{scale-spmf } (\text{weight-spmf } q) \ p) \ (\text{scale-spmf } (\text{weight-spmf } q') \ p') \wedge$
 $\text{rel-spmf } B \ (\text{scale-spmf } (\text{weight-spmf } p) \ q) \ (\text{scale-spmf } (\text{weight-spmf } p') \ q')$
(is *?lhs* \longleftrightarrow *?rhs* **is** \longleftrightarrow *?A* \wedge *?B* **is** \longleftrightarrow *rel-spmf - ?p* *?p'* \wedge *rel-spmf - ?q* *?q'*)
proof(*intro iffI conjI*)
assume *?rhs*
then obtain *pq pq'* **where** *p*: $\text{map-spmf } \text{fst } pq = ?p$ **and** *p'*: $\text{map-spmf } \text{snd } pq = ?p'$

```

and  $q$ :  $\text{map-spmf fst } pq' = ?q$  and  $q'$ :  $\text{map-spmf snd } pq' = ?q'$ 
and  $*$ :  $\bigwedge x x'. (x, x') \in \text{set-spmf } pq \implies A x x'$ 
and  $**$ :  $\bigwedge y y'. (y, y') \in \text{set-spmf } pq' \implies B y y'$  by( $\text{auto elim!}$ :  $\text{rel-spmfE}$ )
let  $?f = \lambda((x, x'), (y, y')). ((x, y), (x', y'))$ 
let  $?r = 1 / (\text{weight-spmf } p * \text{weight-spmf } q)$ 
let  $?pq = \text{scale-spmf } ?r (\text{map-spmf } ?f (\text{pair-spmf } pq \text{ } pq'))$ 

{ fix  $p :: 'x \text{ spmf}$  and  $q :: 'y \text{ spmf}$ 
  assume  $\text{weight-spmf } q \neq 0$ 
  and  $\text{weight-spmf } p \neq 0$ 
  and  $1 / (\text{weight-spmf } p * \text{weight-spmf } q) \leq \text{weight-spmf } p * \text{weight-spmf } q$ 
  hence  $1 \leq (\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q)$ 
  by( $\text{simp add: pos-divide-le-eq order.strict-iff-order weight-spmf-nonneg}$ )
  moreover have  $(\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q) \leq (1 * 1) * (1 * 1)$ 
  by( $\text{intro mult-mono}$ )( $\text{simp-all add: weight-spmf-nonneg weight-spmf-le-1}$ )
  ultimately have  $(\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q) = 1$  by  $\text{simp}$ 
  hence  $*$ :  $\text{weight-spmf } p * \text{weight-spmf } q = 1$ 
  by( $\text{metis antisym-conv less-le mult-less-cancel-left1 weight-pair-spmf weight-spmf-le-1 weight-spmf-nonneg}$ )
  hence  $**$ :  $\text{weight-spmf } p = 1$  by( $\text{metis antisym-conv mult-left-le weight-spmf-le-1 weight-spmf-nonneg}$ )
  moreover from  $**$  have  $\text{weight-spmf } q = 1$  by  $\text{simp}$ 
  moreover note  $\text{calculation}$  }
note  $\text{full} = \text{this}$ 

show  $?lhs$ 
proof
  have [ $\text{simp}$ ]:  $\text{fst} \circ ?f = \text{map-prod fst fst}$  by( $\text{simp add: fun-eq-iff}$ )
  have  $\text{map-spmf fst } ?pq = \text{scale-spmf } ?r (\text{pair-spmf } ?p \text{ } ?q)$ 
  by( $\text{simp add: pair-map-spmf[symmetric] p q map-scale-spmf spmf.map-comp}$ )
  also have  $\dots = \text{pair-spmf } p \text{ } q$  using  $\text{full[of } p \text{ } q]$ 
  by( $\text{simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg}$ )
  ( $\text{auto simp add: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg weight-spmf-eq-0}$ )
  finally show  $\text{map-spmf fst } ?pq = \dots$ 

  have [ $\text{simp}$ ]:  $\text{snd} \circ ?f = \text{map-prod snd snd}$  by( $\text{simp add: fun-eq-iff}$ )
  from  $\langle ?rhs \rangle$  have  $\text{eq: weight-spmf } p * \text{weight-spmf } q = \text{weight-spmf } p' * \text{weight-spmf } q'$ 
  by( $\text{auto dest!}$ :  $\text{rel-spmf-weightD simp add: weight-spmf-le-1 weight-spmf-nonneg}$ )

  have  $\text{map-spmf snd } ?pq = \text{scale-spmf } ?r (\text{pair-spmf } ?p' \text{ } ?q')$ 
  by( $\text{simp add: pair-map-spmf[symmetric] p' q' map-scale-spmf spmf.map-comp}$ )
  also have  $\dots = \text{pair-spmf } p' \text{ } q'$  using  $\text{full[of } p' \text{ } q'] \text{ eq}$ 
  by( $\text{simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg}$ )
  ( $\text{auto simp add: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg weight-spmf-eq-0}$ )

```

```

finally show map-spmf snd ?pq = ... .
qed(auto simp add: set-scale-spmf split: if-split-asm dest: * ** )
next
assume ?lhs
then obtain pq where pq: map-spmf fst pq = pair-spmf p q
  and pq': map-spmf snd pq = pair-spmf p' q'
  and *:  $\bigwedge x y x' y'. ((x, y), (x', y')) \in \text{set-spmf } pq \implies A x x' \wedge B y y'$ 
  by(auto elim: rel-spmfE)

show ?A
proof
  let ?f =  $(\lambda((x, y), (x', y')). (x, x'))$ 
  let ?pq = map-spmf ?f pq
  have [simp]:  $\text{fst} \circ ?f = \text{fst} \circ \text{fst}$  by(simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf q) p using pq
  by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]:  $\text{snd} \circ ?f = \text{fst} \circ \text{snd}$  by(simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf q') p' using pq'
  by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])
qed(auto dest: * )

show ?B
proof
  let ?f =  $(\lambda((x, y), (x', y')). (y, y'))$ 
  let ?pq = map-spmf ?f pq
  have [simp]:  $\text{fst} \circ ?f = \text{snd} \circ \text{fst}$  by(simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf p) q using pq
  by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]:  $\text{snd} \circ ?f = \text{snd} \circ \text{snd}$  by(simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf p') q' using pq'
  by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])
qed(auto dest: * )
qed

```

lemma pair-pair-spmf:

```

pair-spmf (pair-spmf p q) r = map-spmf ( $\lambda(x, (y, z)). ((x, y), z)$ ) (pair-spmf p
(pair-spmf q r))
by(simp add: pair-spmf-alt-def map-spmf-conv-bind-spmf)

```

lemma pair-commute-spmf:

```

pair-spmf p q = map-spmf ( $\lambda(y, x). (x, y)$ ) (pair-spmf q p)
unfolding pair-spmf-alt-def by(subst bind-commute-spmf)(simp add: map-spmf-conv-bind-spmf)

```

25.18 Assertions

definition assert-spmf :: $\text{bool} \Rightarrow \text{unit spmf}$

where assert-spmf b = (if b then return-spmf () else return-pmf None)

lemma *assert-spmf-simps* [*simp*]:
assert-spmf True = return-spmf ()
assert-spmf False = return-pmf None
by(*simp-all add: assert-spmf-def*)

lemma *in-set-assert-spmf* [*simp*]: $x \in \text{set-spmf } (\text{assert-spmf } p) \longleftrightarrow p$
by(*cases p simp-all*)

lemma *set-spmf-assert-spmf-eq-empty* [*simp*]: $\text{set-spmf } (\text{assert-spmf } b) = \{\} \longleftrightarrow \neg b$
by(*cases b simp-all*)

lemma *lossless-assert-spmf* [*iff*]: $\text{lossless-spmf } (\text{assert-spmf } b) \longleftrightarrow b$
by(*cases b simp-all*)

25.19 Try

definition *try-spmf* :: $'a \text{ spmf} \Rightarrow 'a \text{ spmf} \Rightarrow 'a \text{ spmf}$ (*TRY - ELSE - [0,60] 59*)
where *try-spmf* $p \ q = \text{bind-pmf } p \ (\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow q \mid \text{Some } y \Rightarrow \text{return-spmf } y)$

lemma *try-spmf-lossless* [*simp*]:
assumes *lossless-spmf p*
shows *TRY p ELSE q = p*
proof –
have *TRY p ELSE q = bind-pmf p return-pmf* **unfolding** *try-spmf-def* **using** *assms*
by(*auto simp add: lossless-iff-set-pmf-None split: option.split intro: bind-pmf-cong*)
thesis *?thesis* **by**(*simp add: bind-return-pmf'*)
qed

lemma *try-spmf-return-spmf1*: *TRY return-spmf x ELSE q = return-spmf x*
by(*simp add: try-spmf-def bind-return-pmf*)

lemma *try-spmf-return-None* [*simp*]: *TRY return-pmf None ELSE q = q*
by(*simp add: try-spmf-def bind-return-pmf*)

lemma *try-spmf-return-pmf-None2* [*simp*]: *TRY p ELSE return-pmf None = p*
by(*simp add: try-spmf-def option.case-distrib[symmetric] bind-return-pmf' case-option-id*)

lemma *map-try-spmf*: $\text{map-spmf } f \ (\text{try-spmf } p \ q) = \text{try-spmf } (\text{map-spmf } f \ p)$
 $(\text{map-spmf } f \ q)$
by(*simp add: try-spmf-def map-bind-pmf bind-map-pmf option.case-distrib[where h=map-spmf f] o-def cong del: option.case-cong-weak*)

lemma *try-spmf-bind-pmf*: *TRY (bind-pmf p f) ELSE q = bind-pmf p (\lambda x. TRY (f x) ELSE q)*
by(*simp add: try-spmf-def bind-assoc-pmf*)

lemma *try-spmf-bind-spmf-lossless*:

$lossless-spmf\ p \implies TRY\ (bind-spmf\ p\ f)\ ELSE\ q = bind-spmf\ p\ (\lambda x. TRY\ (f\ x)\ ELSE\ q)$

by(*auto simp add: try-spmf-def bind-spmf-def bind-assoc-pmf bind-return-pmf lossless-iff-set-pmf-None intro!: bind-pmf-cong split: option.split*)

lemma *try-spmf-bind-out*:

$lossless-spmf\ p \implies bind-spmf\ p\ (\lambda x. TRY\ (f\ x)\ ELSE\ q) = TRY\ (bind-spmf\ p\ f)\ ELSE\ q$

by(*simp add: try-spmf-bind-spmf-lossless*)

lemma *lossless-try-spmf [simp]*:

$lossless-spmf\ (TRY\ p\ ELSE\ q) \longleftrightarrow lossless-spmf\ p \vee lossless-spmf\ q$

by(*auto simp add: try-spmf-def in-set-spmf lossless-iff-set-pmf-None split: option.splits*)

context includes *lifting-syntax*

begin

lemma *try-spmf-parametric [transfer-rule]*:

$(rel-spmf\ A\ ==>\ rel-spmf\ A\ ==>\ rel-spmf\ A)\ try-spmf\ try-spmf$

unfolding *try-spmf-def[abs-def]* **by** *transfer-prover*

end

lemma *try-spmf-cong*:

$\llbracket p = p'; \neg lossless-spmf\ p' \implies q = q' \rrbracket \implies TRY\ p\ ELSE\ q = TRY\ p'\ ELSE\ q'$

unfolding *try-spmf-def*

by(*rule bind-pmf-cong*)(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

lemma *rel-spmf-try-spmf*:

$\llbracket rel-spmf\ R\ p\ p'; \neg lossless-spmf\ p' \implies rel-spmf\ R\ q\ q' \rrbracket$

$\implies rel-spmf\ R\ (TRY\ p\ ELSE\ q)\ (TRY\ p'\ ELSE\ q')$

unfolding *try-spmf-def*

apply(*rule rel-pmf-bindI[where R= $\lambda x\ y. rel-option\ R\ x\ y \wedge x \in set-pmf\ p \wedge y \in set-pmf\ p'$]*)

apply(*erule pmf.rel-mono-strong; simp*)

apply(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

done

lemma *spmf-try-spmf*:

$spmf\ (TRY\ p\ ELSE\ q)\ x = spmf\ p\ x + pmf\ p\ None * spmf\ q\ x$

proof –

have $ennreal\ (spmf\ (TRY\ p\ ELSE\ q)\ x) = \int^+ y. ennreal\ (spmf\ q\ x) * indicator\ \{None\}\ y + indicator\ \{Some\ x\}\ y\ \partial measure-pmf\ p$

unfolding *try-spmf-def ennreal-pmf-bind* **by**(*rule nn-integral-cong*)(*simp split: option.split split-indicator*)

also have $\dots = (\int^+ y. ennreal\ (spmf\ q\ x) * indicator\ \{None\}\ y\ \partial measure-pmf\ p) + \int^+ y. indicator\ \{Some\ x\}\ y\ \partial measure-pmf\ p$


```

have good: rel-fun ?A (=) (λx. A x ∧ ¬ bad1 x) (λy. B y ∧ ¬ bad2 y) by(auto
simp add: rel-fun-def)
from assms have 1: measure (measure-spmf p) {x. A x ∧ ¬ bad1 x} = measure
(measure-spmf q) {y. B y ∧ ¬ bad2 y}
by(rule measure-spmf-parametric[THEN rel-funD, THEN rel-funD])(rule Col-
lect-parametric[THEN rel-funD, OF good])

have bad: rel-fun ?A (=) bad1 bad2 by(simp add: rel-fun-def)
show 2: ?bad using assms
by(rule measure-spmf-parametric[THEN rel-funD, THEN rel-funD])(rule Col-
lect-parametric[THEN rel-funD, OF bad])

let ?μp = measure (measure-spmf p) and ?μq = measure (measure-spmf q)
have {x. A x ∧ bad1 x} ∪ {x. A x ∧ ¬ bad1 x} = {x. A x}
and {y. B y ∧ bad2 y} ∪ {y. B y ∧ ¬ bad2 y} = {y. B y} by auto
then have |?μp {x. A x} - ?μq {x. B x}| = |?μp ({x. A x ∧ bad1 x} ∪ {x. A
x ∧ ¬ bad1 x}) - ?μq ({y. B y ∧ bad2 y} ∪ {y. B y ∧ ¬ bad2 y})|
by simp
also have ... = |?μp {x. A x ∧ bad1 x} + ?μp {x. A x ∧ ¬ bad1 x} - ?μq {y.
B y ∧ bad2 y} - ?μq {y. B y ∧ ¬ bad2 y}|
by(subst (1 2) measure-Union)(auto)
also have ... = |?μp {x. A x ∧ bad1 x} - ?μq {y. B y ∧ bad2 y}| using 1 by
simp
also have ... ≤ max (?μp {x. A x ∧ bad1 x}) (?μq {y. B y ∧ bad2 y})
by(rule abs-leI)(auto simp add: max-def not-le, simp-all only: add-increasing
measure-nonneg mult-2)
also have ... ≤ max (?μp {x. bad1 x}) (?μq {y. bad2 y})
by(rule max.mono; rule measure-spmf.finite-measure-mono; auto)
also note 2[symmetric]
finally show ?fundamental by simp
qed
end

```

26 Indexed products of PMFs

theory Product-PMF

imports Probability-Mass-Function Independent-Family
begin

Conflicting notation from *HOL–Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** *abs'-summable'-on* 46)

26.1 Preliminaries

lemma *pmf-expectation-eq-infsetsum*: *measure-pmf.expectation p f = infsetsum*
*(λx. pmf p x * f x) UNIV*

unfolding *infsetsum-def measure-pmf-eq-density* **by** (*subst integral-density*) *simp-all*

lemma *measure-pmf-prob-product*:
assumes *countable A countable B*
shows *measure-pmf.prob (pair-pmf M N) (A × B) = measure-pmf.prob M A * measure-pmf.prob N B*
proof –
have *measure-pmf.prob (pair-pmf M N) (A × B) = (∑_{a(a, b) ∈ A × B.} pmf M a * pmf N b)*
by (*auto intro!*: *infsetsum-cong simp add: measure-pmf-conv-infsetsum pmf-pair*)
also have *... = measure-pmf.prob M A * measure-pmf.prob N B*
using *assms* **by** (*subst infsetsum-product*) (*auto simp add: measure-pmf-conv-infsetsum*)
finally show *?thesis*
by *simp*
qed

26.2 Definition

In analogy to Pi_M , we define an indexed product of PMFs. In the literature, this is typically called taking a vector of independent random variables. Note that the components do not have to be identically distributed.

The operation takes an explicit index set A and a function f that maps each element from A to a PMF and defines the product measure $\bigotimes_{i \in A} f(i)$, which is represented as a ($'a \Rightarrow 'b$) *pmf*.

Note that unlike Pi_M , this only works for *finite* index sets. It could be extended to countable sets and beyond, but the construction becomes somewhat more involved.

definition *Pi-pmf* :: $'a \text{ set} \Rightarrow 'b \Rightarrow ('a \Rightarrow 'b \text{ pmf}) \Rightarrow ('a \Rightarrow 'b) \text{ pmf}$ **where**
 $Pi\text{-}pmf\ A\ dflt\ p =$
 $embed\text{-}pmf\ (\lambda f. \text{if } (\forall x. x \notin A \longrightarrow f\ x = dflt) \text{ then } \prod_{x \in A}. pmf\ (p\ x)\ (f\ x) \text{ else } 0)$

A technical subtlety that needs to be addressed is this: Intuitively, the functions in the support of a product distribution have domain A . However, since HOL is a total logic, these functions must still return *some* value for inputs outside A . The product measure Pi_M simply lets these functions return *undefined* in these cases. We chose a different solution here, which is to supply a default value *dflt* that is returned in these cases.

As one possible application, one could model the result of n different independent coin tosses as $Pi\text{-}pmf\ \{0::'a..<n\}\ False\ (\lambda. bernoulli\text{-}pmf\ (1 / 2))$. This returns a function of type $nat \Rightarrow bool$ that maps every natural number below n to the result of the corresponding coin toss, and every other natural number to *False*.

lemma *pmf-Pi*:
assumes *A: finite A*
shows $pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ f =$
 $(\text{if } (\forall x. x \notin A \longrightarrow f\ x = dflt) \text{ then } \prod_{x \in A}. pmf\ (p\ x)\ (f\ x) \text{ else } 0)$
unfolding *Pi-pmf-def*

proof (*rule pmf-embed-pmf, goal-cases*)
case 2
define S **where** $S = \{f. \forall x. x \notin A \longrightarrow f x = dflt\}$
define B **where** $B = (\lambda x. \text{set-pmf } (p x))$

have *neutral-left*: $(\prod_{x \in A}. \text{pmf } (p x) (f x)) = 0$
if $f \in \text{PiE } A B - (\lambda f. \text{restrict } f A) ' S$ **for** f
proof –
have *restrict* $(\lambda x. \text{if } x \in A \text{ then } f x \text{ else } dflt) A \in (\lambda f. \text{restrict } f A) ' S$
by (*intro imageI*) (*auto simp: S-def*)
also have *restrict* $(\lambda x. \text{if } x \in A \text{ then } f x \text{ else } dflt) A = f$
using *that* **by** (*auto simp: PiE-def Pi-def extensional-def fun-eq-iff*)
finally show *?thesis* **using** *that* **by** *blast*

qed
have *neutral-right*: $(\prod_{x \in A}. \text{pmf } (p x) (f x)) = 0$
if $f \in (\lambda f. \text{restrict } f A) ' S - \text{PiE } A B$ **for** f
proof –
from *that* **obtain** f' **where** $f': f = \text{restrict } f' A f' \in S$ **by** *auto*
moreover from *this* **and** *that* **have** *restrict* $f' A \notin \text{PiE } A B$ **by** *simp*
then obtain x **where** $x \in A$ $\text{pmf } (p x) (f' x) = 0$ **by** (*auto simp: B-def set-pmf-eq*)
with f' **and** A **show** *?thesis* **by** *auto*

qed

have $(\lambda f. \prod_{x \in A}. \text{pmf } (p x) (f x)) \text{abs-summable-on } \text{PiE } A B$
by (*intro abs-summable-on-prod-PiE A*) (*auto simp: B-def*)
also have *?this* $\longleftrightarrow (\lambda f. \prod_{x \in A}. \text{pmf } (p x) (f x)) \text{abs-summable-on } (\lambda f. \text{restrict } f A) ' S$
by (*intro abs-summable-on-cong-neutral neutral-left neutral-right*) *auto*
also have $\dots \longleftrightarrow (\lambda f. \prod_{x \in A}. \text{pmf } (p x) (\text{restrict } f A x)) \text{abs-summable-on } S$
by (*rule abs-summable-on-reindex-iff [symmetric]*) (*force simp: inj-on-def fun-eq-iff S-def*)
also have $\dots \longleftrightarrow (\lambda f. \text{if } \forall x. x \notin A \longrightarrow f x = dflt \text{ then } \prod_{x \in A}. \text{pmf } (p x) (f x) \text{ else } 0)$
 $\text{abs-summable-on UNIV}$
by (*intro abs-summable-on-cong-neutral*) (*auto simp: S-def*)
finally have *summable*: \dots .

have $1 = (\prod_{x \in A}. 1::\text{real})$ **by** *simp*
also have $(\prod_{x \in A}. 1) = (\prod_{x \in A}. \sum_{y \in B} \text{pmf } (p x) y)$
unfolding *B-def* **by** (*subst infsetsum-pmf-eq-1*) *auto*
also have $(\prod_{x \in A}. \sum_{y \in B} \text{pmf } (p x) y) = (\sum_{f \in \text{PiE } A B}. \prod_{x \in A}. \text{pmf } (p x) (f x))$
by (*intro infsetsum-prod-PiE [symmetric] A*) (*auto simp: B-def*)
also have $\dots = (\sum_{f \in (\lambda f. \text{restrict } f A) ' S}. \prod_{x \in A}. \text{pmf } (p x) (f x))$ **using** A
by (*intro infsetsum-cong-neutral neutral-left neutral-right refl*)
also have $\dots = (\sum_{f \in S}. \prod_{x \in A}. \text{pmf } (p x) (\text{restrict } f A x))$
by (*rule infsetsum-reindex*) (*force simp: inj-on-def fun-eq-iff S-def*)
also have $\dots = (\sum_{f \in S}. \prod_{x \in A}. \text{pmf } (p x) (f x))$

by (*intro infsetsum-cong*) (*auto simp: S-def*)
also have $\dots = (\sum_{af}. \text{if } \forall x. x \notin A \longrightarrow f x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (p x) (f x) \text{ else } 0)$
by (*intro infsetsum-cong-neutral*) (*auto simp: S-def*)
also have $\text{ennreal } \dots = (\int^+ f. \text{ennreal } (\text{if } \forall x. x \notin A \longrightarrow f x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (p x) (f x) \text{ else } 0) \partial \text{count-space UNIV})$
by (*intro nn-integral-conv-infsetsum [symmetric] summable*) (*auto simp: prod-nonneg*)
finally show *?case by simp*
qed (*auto simp: prod-nonneg*)

lemma *Pi-pmf-cong*:

assumes $A = A' \text{ dflt} = \text{dflt}' \wedge x. x \in A \implies f x = f' x$
shows $\text{Pi-pmf } A \text{ dflt } f = \text{Pi-pmf } A' \text{ dflt}' f'$
proof –
have $(\lambda fa. \text{if } \forall x. x \notin A \longrightarrow fa x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (f x) (fa x) \text{ else } 0) =$
 $(\lambda f. \text{if } \forall x. x \notin A' \longrightarrow f x = \text{dflt}' \text{ then } \prod_{x \in A'}. \text{pmf } (f' x) (f x) \text{ else } 0)$
using *assms by (intro ext) (auto intro!: prod.cong)*
thus *?thesis*
by (*simp only: Pi-pmf-def*)
qed

lemma *pmf-Pi'*:

assumes $\text{finite } A \wedge x. x \notin A \implies f x = \text{dflt}$
shows $\text{pmf } (\text{Pi-pmf } A \text{ dflt } p) f = (\prod_{x \in A}. \text{pmf } (p x) (f x))$
using *assms by (subst pmf-Pi) auto*

lemma *pmf-Pi-outside*:

assumes $\text{finite } A \exists x. x \notin A \wedge f x \neq \text{dflt}$
shows $\text{pmf } (\text{Pi-pmf } A \text{ dflt } p) f = 0$
using *assms by (subst pmf-Pi) auto*

lemma *pmf-Pi-empty [simp]*: $\text{Pi-pmf } \{\} \text{ dflt } p = \text{return-pmf } (\lambda-. \text{dflt})$

by (*intro pmf-eqI, subst pmf-Pi*) (*auto simp: indicator-def*)

lemma *set-Pi-pmf-subset*: $\text{finite } A \implies \text{set-pmf } (\text{Pi-pmf } A \text{ dflt } p) \subseteq \{f. \forall x. x \notin A \longrightarrow f x = \text{dflt}\}$

by (*auto simp: set-pmf-eq pmf-Pi*)

26.3 Dependent product sets with a default

The following describes a dependent product of sets where the functions are required to return the default value *dflt* outside their domain, in analogy to *PiE*, which uses *undefined*.

definition *PiE-dflt*

where $\text{PiE-dflt } A \text{ dflt } B = \{f. \forall x. (x \in A \longrightarrow f x \in B x) \wedge (x \notin A \longrightarrow f x = \text{dflt})\}$

lemma *restrict-PiE-dflt*: $(\lambda h. \text{restrict } h A) \text{ ` PiE-dflt } A \text{ dflt } B = \text{PiE } A B$

proof (*intro equalityI subsetI*)

```

fix  $h$  assume  $h \in (\lambda h. \text{restrict } h \ A) \text{ ‘ } PiE\text{-dflt } A \ \text{dflt } B$ 
thus  $h \in PiE \ A \ B$ 
  by (auto simp: PiE-dflt-def)
next
  fix  $h$  assume  $h: h \in PiE \ A \ B$ 
  hence restrict  $(\lambda x. \text{if } x \in A \ \text{then } h \ x \ \text{else } \text{dflt}) \ A \in (\lambda h. \text{restrict } h \ A) \text{ ‘ } PiE\text{-dflt}$ 
A dflt B
    by (intro imageI) (auto simp: PiE-def extensional-def PiE-dflt-def)
  also have restrict  $(\lambda x. \text{if } x \in A \ \text{then } h \ x \ \text{else } \text{dflt}) \ A = h$ 
    using  $h$  by (auto simp: fun-eq-iff)
  finally show  $h \in (\lambda h. \text{restrict } h \ A) \text{ ‘ } PiE\text{-dflt } A \ \text{dflt } B .$ 
qed

```

```

lemma dflt-image-PiE:  $(\lambda h \ x. \text{if } x \in A \ \text{then } h \ x \ \text{else } \text{dflt}) \text{ ‘ } PiE \ A \ B = PiE\text{-dflt}$ 
A dflt B
  (is  $?f \text{ ‘ } ?X = ?Y$ )
proof (intro equalityI subsetI)
  fix  $h$  assume  $h \in ?f \text{ ‘ } ?X$ 
  thus  $h \in ?Y$ 
    by (auto simp: PiE-dflt-def PiE-def)
next
  fix  $h$  assume  $h: h \in ?Y$ 
  hence  $?f \ (\text{restrict } h \ A) \in ?f \text{ ‘ } ?X$ 
    by (intro imageI) (auto simp: PiE-def extensional-def PiE-dflt-def)
  also have  $?f \ (\text{restrict } h \ A) = h$ 
    using  $h$  by (auto simp: fun-eq-iff PiE-dflt-def)
  finally show  $h \in ?f \text{ ‘ } ?X .$ 
qed

```

```

lemma finite-PiE-dflt [intro]:
  assumes finite  $A \ \bigwedge x. x \in A \implies \text{finite } (B \ x)$ 
  shows finite  $(PiE\text{-dflt } A \ d \ B)$ 
proof –
  have  $PiE\text{-dflt } A \ d \ B = (\lambda f \ x. \text{if } x \in A \ \text{then } f \ x \ \text{else } d) \text{ ‘ } PiE \ A \ B$ 
    by (rule dflt-image-PiE [symmetric])
  also have finite ...
    by (intro finite-imageI finite-PiE assms)
  finally show thesis .
qed

```

```

lemma card-PiE-dflt:
  assumes finite  $A \ \bigwedge x. x \in A \implies \text{finite } (B \ x)$ 
  shows card  $(PiE\text{-dflt } A \ d \ B) = (\prod x \in A. \text{card } (B \ x))$ 
proof –
  from assms have  $(\prod x \in A. \text{card } (B \ x)) = \text{card } (PiE \ A \ B)$ 
    by (intro card-PiE [symmetric]) auto
  also have  $PiE \ A \ B = (\lambda f. \text{restrict } f \ A) \text{ ‘ } PiE\text{-dflt } A \ d \ B$ 
    by (rule restrict-PiE-dflt [symmetric])
  also have card ... = card  $(PiE\text{-dflt } A \ d \ B)$ 

```

by (intro card-image) (force simp: inj-on-def restrict-def fun-eq-iff PiE-dflt-def)
 finally show ?thesis ..
 qed

lemma *PiE-dflt-empty-iff* [simp]: $PiE-dflt A dflt B = \{\}$ \longleftrightarrow $(\exists x \in A. B x = \{\})$
 by (simp add: dflt-image-PiE [symmetric] PiE-eq-empty-iff)

lemma *set-Pi-pmf-subset'*:
 assumes finite A
 shows $set-pmf (Pi-pmf A dflt p) \subseteq PiE-dflt A dflt (set-pmf \circ p)$
 using assms by (auto simp: set-pmf-eq pmf-Pi PiE-dflt-def)

lemma *set-Pi-pmf*:
 assumes finite A
 shows $set-pmf (Pi-pmf A dflt p) = PiE-dflt A dflt (set-pmf \circ p)$
proof (rule equalityI)
 show $PiE-dflt A dflt (set-pmf \circ p) \subseteq set-pmf (Pi-pmf A dflt p)$
proof safe
 fix f assume f: $f \in PiE-dflt A dflt (set-pmf \circ p)$
 hence $pmf (Pi-pmf A dflt p) f = (\prod x \in A. pmf (p x) (f x))$
 using assms by (auto simp: pmf-Pi PiE-dflt-def)
 also have ... > 0
 using f by (intro prod-pos) (auto simp: PiE-dflt-def set-pmf-eq)
 finally show $f \in set-pmf (Pi-pmf A dflt p)$
 by (auto simp: set-pmf-eq)
 qed
qed (use set-Pi-pmf-subset'[OF assms, of dflt p] in auto)

The probability of an independent combination of events is precisely the product of the probabilities of each individual event.

lemma *measure-Pi-pmf-PiE-dflt*:
 assumes [simp]: finite A
 shows $measure-pmf.prob (Pi-pmf A dflt p) (PiE-dflt A dflt B) =$
 $(\prod x \in A. measure-pmf.prob (p x) (B x))$
proof –
define B' where $B' = (\lambda x. B x \cap set-pmf (p x))$
have $measure-pmf.prob (Pi-pmf A dflt p) (PiE-dflt A dflt B) =$
 $(\sum_a h \in PiE-dflt A dflt B. pmf (Pi-pmf A dflt p) h)$
 by (rule measure-pmf-conv-infsetsum)
also have ... $= (\sum_a h \in PiE-dflt A dflt B. \prod x \in A. pmf (p x) (h x))$
 by (intro infsetsum-cong, subst pmf-Pi') (auto simp: PiE-dflt-def)
also have ... $= (\sum_a h \in (\lambda h. restrict h A) ' PiE-dflt A dflt B. \prod x \in A. pmf (p x) (h x))$
 by (subst infsetsum-reindex) (force simp: inj-on-def PiE-dflt-def fun-eq-iff)+
also have $(\lambda h. restrict h A) ' PiE-dflt A dflt B = PiE A B$
 by (rule restrict-PiE-dflt)
also have $(\sum_a h \in PiE A B. \prod x \in A. pmf (p x) (h x)) = (\sum_a h \in PiE A B'. \prod x \in A. pmf (p x) (h x))$
 by (intro infsetsum-cong-neutral) (auto simp: B'-def set-pmf-eq)

also have $(\sum_{a \in h \in \text{PiE } A} B'. \prod_{x \in A}. \text{pmf } (p \ x) \ (h \ x)) = (\prod_{x \in A}. \text{infsetsum } (\text{pmf } (p \ x)) \ (B' \ x))$
by $(\text{intro infsetsum-prod-PiE})$ $(\text{auto simp: } B'\text{-def})$
also have $\dots = (\prod_{x \in A}. \text{infsetsum } (\text{pmf } (p \ x)) \ (B \ x))$
by $(\text{intro prod.cong infsetsum-cong-neutral})$ $(\text{auto simp: } B'\text{-def set-pmf-eq})$
also have $\dots = (\prod_{x \in A}. \text{measure-pmf.prob } (p \ x) \ (B \ x))$
by $(\text{subst measure-pmf-conv-infsetsum})$ (rule refl)
finally show $?thesis$.
qed

lemma measure-Pi-pmf-Pi :

fixes $t::\text{nat}$

assumes $[\text{simp}]$: $\text{finite } A$

shows $\text{measure-pmf.prob } (\text{Pi-pmf } A \ \text{dflt } p) \ (\text{Pi } A \ B) =$
 $(\prod_{x \in A}. \text{measure-pmf.prob } (p \ x) \ (B \ x))$ **(is** $?lhs = ?rhs$ **)**

proof –

have $?lhs = \text{measure-pmf.prob } (\text{Pi-pmf } A \ \text{dflt } p) \ (\text{PiE-dflt } A \ \text{dflt } B)$

by $(\text{intro measure-prob-cong-0})$

$(\text{auto simp: PiE-dflt-def PiE-def intro!: pmf-Pi-outside})+$

also have $\dots = ?rhs$

using assms **by** $(\text{simp add: measure-Pi-pmf-PiE-dflt})$

finally show $?thesis$

by simp

qed

26.4 Common PMF operations on products

Pi-pmf distributes over the ‘bind’ operation in the Giry monad:

lemma Pi-pmf-bind :

assumes $\text{finite } A$

shows $\text{Pi-pmf } A \ d \ (\lambda x. \text{bind-pmf } (p \ x) \ (q \ x)) =$
 $\text{do } \{f \leftarrow \text{Pi-pmf } A \ d' \ p; \text{Pi-pmf } A \ d \ (\lambda x. q \ x \ (f \ x))\}$ **(is** $?lhs = ?rhs$ **)**

proof $(\text{rule pmf-eqI, goal-cases})$

case $(1 \ f)$

show $?case$

proof $(\text{cases } \exists x \in -A. f \ x \neq d)$

case False

define B **where** $B = (\lambda x. \text{set-pmf } (p \ x))$

have $[\text{simp}]$: $\text{countable } (B \ x)$ **for** x **by** $(\text{auto simp: } B\text{-def})$

{

fix $x :: 'a$

have $(\lambda a. \text{pmf } (p \ x) \ a * 1)$ $\text{abs-summable-on } B \ x$

by $(\text{simp add: pmf-abs-summable})$

moreover have $\text{norm } (\text{pmf } (p \ x) \ a * 1) \geq \text{norm } (\text{pmf } (p \ x) \ a * \text{pmf } (q \ x \ a) \ (f \ x))$ **for** a

unfolding norm-mult **by** $(\text{intro mult-left-mono})$ $(\text{auto simp: pmf-le-1})$

ultimately have $(\lambda a. \text{pmf } (p \ x) \ a * \text{pmf } (q \ x \ a) \ (f \ x))$ $\text{abs-summable-on } B \ x$

by $(\text{rule abs-summable-on-comparison-test})$

```

} note summable = this

have pmf ?rhs f = (∑ a.g. pmf (Pi-pmf A d' p) g * (∏ x∈A. pmf (q x (g x))
(f x)))
  by (subst pmf-bind, subst pmf-Pi')
    (insert assms False, simp-all add: pmf-expectation-eq-infsetsum)
also have ... = (∑ a.g∈PiE-dflt A d' B.
  pmf (Pi-pmf A d' p) g * (∏ x∈A. pmf (q x (g x)) (f x)))
unfolding B-def
  using assms by (intro infsetsum-cong-neutral) (auto simp: pmf-Pi PiE-dflt-def
set-pmf-eq)
  also have ... = (∑ a.g∈PiE-dflt A d' B.
    (∏ x∈A. pmf (p x) (g x) * pmf (q x (g x)) (f x)))
    using assms by (intro infsetsum-cong) (auto simp: pmf-Pi PiE-dflt-def
prod.distrib)
  also have ... = (∑ a.g∈(λg. restrict g A) ' PiE-dflt A d' B.
    (∏ x∈A. pmf (p x) (g x) * pmf (q x (g x)) (f x)))
    by (subst infsetsum-reindex) (force simp: PiE-dflt-def inj-on-def fun-eq-iff)+
  also have (λg. restrict g A) ' PiE-dflt A d' B = PiE A B
    by (rule restrict-PiE-dflt)
  also have (∑ a.g∈... (∏ x∈A. pmf (p x) (g x) * pmf (q x (g x)) (f x))) =
    (∏ x∈A. ∑ a.a∈B x. pmf (p x) a * pmf (q x a) (f x))
    using assms summable by (subst infsetsum-prod-PiE) simp-all
  also have ... = (∏ x∈A. ∑ a.a. pmf (p x) a * pmf (q x a) (f x))
    by (intro prod.cong infsetsum-cong-neutral) (auto simp: B-def set-pmf-eq)
  also have ... = pmf ?lhs f
  using False assms by (subst pmf-Pi') (simp-all add: pmf-bind pmf-expectation-eq-infsetsum)
  finally show ?thesis ..
next
case True
have pmf ?rhs f =
  measure-pmf.expectation (Pi-pmf A d' p) (λx. pmf (Pi-pmf A d (λx.a. q
xa (x xa))) f)
  using assms by (simp add: pmf-bind)
  also have ... = measure-pmf.expectation (Pi-pmf A d' p) (λx. 0)
  using assms True by (intro Bochner-Integration.integral-cong pmf-Pi-outside)
auto
  also have ... = pmf ?lhs f
  using assms True by (subst pmf-Pi-outside) auto
  finally show ?thesis ..
qed
qed

```

lemma *Pi-pmf-return-pmf* [*simp*]:

```

assumes finite A
shows Pi-pmf A dflt (λx. return-pmf (f x)) = return-pmf (λx. if x ∈ A then f
x else dflt)
  using assms by (intro pmf-eqI) (auto simp: pmf-Pi simp: indicator-def split:
if-splits)

```

Analogously any componentwise mapping can be pulled outside the product:

lemma *Pi-pmf-map*:

assumes [*simp*]: *finite A* **and** *f dflt = dflt'*

shows $Pi\text{-}pmf\ A\ dflt'\ (\lambda x. map\text{-}pmf\ f\ (g\ x)) = map\text{-}pmf\ (\lambda h. f\ \circ\ h)\ (Pi\text{-}pmf\ A\ dflt\ g)$

proof –

have $Pi\text{-}pmf\ A\ dflt'\ (\lambda x. map\text{-}pmf\ f\ (g\ x)) = Pi\text{-}pmf\ A\ dflt'\ (\lambda x. g\ x \gg (\lambda x. return\text{-}pmf\ (f\ x)))$

using *assms* **by** (*simp add: map-pmf-def Pi-pmf-bind*)

also have $\dots = Pi\text{-}pmf\ A\ dflt\ g \gg (\lambda h. return\text{-}pmf\ (\lambda x. if\ x \in A\ then\ f\ (h\ x)\ else\ dflt'))$

by (*subst Pi-pmf-bind[where d' = dflt'] auto*)

also have $\dots = map\text{-}pmf\ (\lambda h. f\ \circ\ h)\ (Pi\text{-}pmf\ A\ dflt\ g)$

unfolding *map-pmf-def* **using** *set-Pi-pmf-subset'[of A dflt g]*

by (*intro bind-pmf-cong refl arg-cong[of - - return-pmf]*)

(*auto dest: simp: fun-eq-iff PiE-dflt-def assms(2)*)

finally show *?thesis* .

qed

We can exchange the default value in a product of PMFs like this:

lemma *Pi-pmf-default-swap*:

assumes *finite A*

shows $map\text{-}pmf\ (\lambda f\ x. if\ x \in A\ then\ f\ x\ else\ dflt')\ (Pi\text{-}pmf\ A\ dflt\ p) = Pi\text{-}pmf\ A\ dflt'\ p$ (*is ?lhs = ?rhs*)

proof (*rule pmf-eqI, goal-cases*)

case (*1 f*)

let *?B = ($\lambda f\ x. if\ x \in A\ then\ f\ x\ else\ dflt'$) - '{f} \cap PiE-dflt A dflt (\lambda-. UNIV)*

show *?case*

proof (*cases \exists x \in -A. f x \neq dflt'*)

case *False*

let *?f' = $\lambda x. if\ x \in A\ then\ f\ x\ else\ dflt$*

from *False* **have** *pmf ?lhs f = measure-pmf.prob (Pi-pmf A dflt p) ?B*

using *assms* **unfolding** *pmf-map*

by (*intro measure-prob-cong-0*) (*auto simp: PiE-dflt-def pmf-Pi-outside*)

also from *False* **have** *?B = {?f'}*

by (*auto simp: fun-eq-iff PiE-dflt-def*)

also have $measure\text{-}pmf.\text{prob}\ (Pi\text{-}pmf\ A\ dflt\ p)\ \{?f'\} = pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ ?f'$

by (*simp add: measure-pmf-single*)

also have $\dots = pmf\ ?rhs\ f$

using *False assms* **by** (*subst (1 2) pmf-Pi*) *auto*

finally show *?thesis* .

next

case *True*

have $pmf\ ?lhs\ f = measure\text{-}pmf.\text{prob}\ (Pi\text{-}pmf\ A\ dflt\ p)\ ?B$

using *assms* **unfolding** *pmf-map*

by (*intro measure-prob-cong-0*) (*auto simp: PiE-dflt-def pmf-Pi-outside*)

also from *True* **have** *?B = {}* **by** *auto*

also have $measure\text{-}pmf.\text{prob}\ (Pi\text{-}pmf\ A\ dflt\ p)\ \dots = 0$

```

    by simp
    also have 0 = pmf ?rhs f
      using True assms by (intro pmf-Pi-outside [symmetric]) auto
    finally show ?thesis .
  qed
qed

```

The following rule allows reindexing the product:

lemma *Pi-pmf-bij-betw*:

```

  assumes finite A bij-betw h A B  $\wedge x. x \notin A \implies h x \notin B$ 
  shows Pi-pmf A dflt ( $\lambda-. f$ ) = map-pmf ( $\lambda g. g \circ h$ ) (Pi-pmf B dflt ( $\lambda-. f$ ))
    (is ?lhs = ?rhs)
  proof -
    have B: finite B
      using assms bij-betw-finite by auto
    have pmf ?lhs g = pmf ?rhs g for g
      proof (cases  $\forall a. a \notin A \longrightarrow g a = dflt$ )
        case True
          define h' where h' = the-inv-into A h
          have h': h' (h x) = x if x  $\in$  A for x
          unfolding h'-def using that assms by (auto simp add: bij-betw-def the-inv-into-f-f)
          have h: h (h' x) = x if x  $\in$  B for x
          unfolding h'-def using that assms f-the-inv-into-f-bij-betw by fastforce
          have pmf ?rhs g = measure-pmf.prob (Pi-pmf B dflt ( $\lambda-. f$ )) (( $\lambda g. g \circ h$ ) -'
            {g})
          unfolding pmf-map by simp
          also have ... = measure-pmf.prob (Pi-pmf B dflt ( $\lambda-. f$ ))
            ((( $\lambda g. g \circ h$ ) -' {g})  $\cap$  PiE-dflt B dflt ( $\lambda-. UNIV$ ))
          using B by (intro measure-prob-cong-0) (auto simp: PiE-dflt-def pmf-Pi-outside)
          also have ... = pmf (Pi-pmf B dflt ( $\lambda-. f$ )) ( $\lambda x. \text{if } x \in B \text{ then } g (h' x) \text{ else } dflt$ )
        qed
      proof -
        have (if h x  $\in$  B then g (h' (h x)) else dflt) = g x for x
          using h' assms True by (cases x  $\in$  A) (auto simp add: bij-betwE)
        then have ( $\lambda g. g \circ h$ ) -' {g}  $\cap$  PiE-dflt B dflt ( $\lambda-. UNIV$ ) =
          {( $\lambda x. \text{if } x \in B \text{ then } g (h' x) \text{ else } dflt$ )}
          using assms h' h True unfolding PiE-dflt-def by auto
        then show ?thesis
          by (simp add: measure-pmf-single)
      qed
    also have ... = pmf (Pi-pmf A dflt ( $\lambda-. f$ )) g
      using B assms True h'-def
      by (auto simp add: pmf-Pi intro!: prod.reindex-bij-betw bij-betw-the-inv-into)
    finally show ?thesis
      by simp
  next
    case False
    have pmf ?rhs g = infsetsum (pmf (Pi-pmf B dflt ( $\lambda-. f$ ))) (( $\lambda g. g \circ h$ ) -' {g})
      using assms by (auto simp add: measure-pmf-conv-infsetsum pmf-map)

```



```

also have ... = infsetsum ( $\lambda\cdot. 0$ ) (( $\lambda g x. g (h x)$ ) - ‘ $\{g\}$ )
using B False assms by (intro infsetsum-cong pmf-Pi-outside) fastforce+
also have ... = 0
by simp
finally show ?thesis
using assms False by (auto simp add: pmf-Pi pmf-map)
qed
then show ?thesis
by (rule pmf-eqI)
qed

```

A product of uniform random choices is again a uniform distribution.

lemma *Pi-pmf-of-set*:

```

assumes finite A  $\wedge x. x \in A \implies \text{finite } (B x) \wedge x. x \in A \implies B x \neq \{\}$ 
shows Pi-pmf A d ( $\lambda x. \text{pmf-of-set } (B x)$ ) = pmf-of-set (PiE-dflt A d B) (is
?lhs = ?rhs)

```

proof (*rule pmf-eqI, goal-cases*)

case (*1 f*)

show *?case*

proof (*cases* $\exists x. x \notin A \wedge f x \neq d$)

case *True*

hence *pmf ?lhs f = 0*

using *assms* **by** (*intro pmf-Pi-outside*) (*auto simp: PiE-dflt-def*)

also from *True* **have** $f \notin \text{PiE-dflt } A d B$

by (*auto simp: PiE-dflt-def*)

hence $0 = \text{pmf } ?rhs f$

using *assms* **by** (*subst pmf-of-set*) *auto*

finally show *?thesis* .

next

case *False*

hence *pmf ?lhs f* = ($\prod x \in A. \text{pmf } (\text{pmf-of-set } (B x)) (f x)$)

using *assms* **by** (*subst pmf-Pi'*) *auto*

also have ... = ($\prod x \in A. \text{indicator } (B x) (f x) / \text{real } (\text{card } (B x))$)

by (*intro prod.cong refl, subst pmf-of-set*) (*use assms False in auto*)

also have ... = ($\prod x \in A. \text{indicator } (B x) (f x) / \text{real } (\prod x \in A. \text{card } (B x))$)

by (*subst prod-dividef*) *simp-all*

also have ($\prod x \in A. \text{indicator } (B x) (f x) :: \text{real}$) = *indicator (PiE-dflt A d B) f*

using *assms False* **by** (*auto simp: indicator-def PiE-dflt-def*)

also have ($\prod x \in A. \text{card } (B x)$) = *card (PiE-dflt A d B)*

using *assms* **by** (*intro card-PiE-dflt [symmetric]*) *auto*

also have *indicator (PiE-dflt A d B) f / ... = pmf ?rhs f*

using *assms* **by** (*intro pmf-of-set [symmetric]*) *auto*

finally show *?thesis* .

qed

qed

26.5 Merging and splitting PMF products

The following lemma shows that we can add a single PMF to a product:

lemma *Pi-pmf-insert*:

assumes *finite A x ∉ A*
shows $Pi\text{-}pmf\ (insert\ x\ A)\ dflt\ p = map\text{-}pmf\ (\lambda(y,f). f(x:=y))\ (pair\text{-}pmf\ (p\ x)\ (Pi\text{-}pmf\ A\ dflt\ p))$
proof (*intro pmf-eqI*)
fix *f*
let $?M = pair\text{-}pmf\ (p\ x)\ (Pi\text{-}pmf\ A\ dflt\ p)$
have $pmf\ (map\text{-}pmf\ (\lambda(y, f). f(x := y))\ ?M)\ f =$
 $measure\text{-}pmf.\text{prob}\ ?M\ ((\lambda(y, f). f(x := y)) - \{f\})$
by (*subst pmf-map*) *auto*
also have $((\lambda(y, f). f(x := y)) - \{f\}) = (\bigcup y'. \{(f\ x, f(x := y'))\})$
by (*auto simp: fun-upd-def fun-eq-iff*)
also have $measure\text{-}pmf.\text{prob}\ ?M \dots = measure\text{-}pmf.\text{prob}\ ?M\ \{(f\ x, f(x := dflt))\}$
using *assms* **by** (*intro measure-prob-cong-0*) (*auto simp: pmf-pair pmf-Pi split: if-splits*)
also have $\dots = pmf\ (p\ x)\ (f\ x) * pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ (f(x := dflt))$
by (*simp add: measure-pmf-single pmf-pair pmf-Pi*)
also have $\dots = pmf\ (Pi\text{-}pmf\ (insert\ x\ A)\ dflt\ p)\ f$
proof (*cases ∃ y. y ∉ insert x A → f y = dflt*)
case *True*
with *assms* **have** $pmf\ (p\ x)\ (f\ x) * pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ (f(x := dflt)) =$
 $pmf\ (p\ x)\ (f\ x) * (\prod xa \in A. pmf\ (p\ xa)\ ((f(x := dflt))\ xa))$
by (*subst pmf-Pi'*) *auto*
also have $(\prod xa \in A. pmf\ (p\ xa)\ ((f(x := dflt))\ xa)) = (\prod xa \in A. pmf\ (p\ xa)\ (f\ xa))$
using *assms* **by** (*intro prod.cong*) *auto*
also have $pmf\ (p\ x)\ (f\ x) * \dots = pmf\ (Pi\text{-}pmf\ (insert\ x\ A)\ dflt\ p)\ f$
using *assms* **True** **by** (*subst pmf-Pi'*) *auto*
finally show *?thesis* .
qed (*insert assms, auto simp: pmf-Pi*)
finally show $\dots = pmf\ (map\text{-}pmf\ (\lambda(y, f). f(x := y))\ ?M)\ f ..$
qed

lemma *Pi-pmf-insert'*:

assumes *finite A x ∉ A*
shows $Pi\text{-}pmf\ (insert\ x\ A)\ dflt\ p =$
 $do\ \{y \leftarrow p\ x; f \leftarrow Pi\text{-}pmf\ A\ dflt\ p; return\text{-}pmf\ (f(x := y))\}$
using *assms*
by (*subst Pi-pmf-insert*)
(auto simp add: map-pmf-def pair-pmf-def case-prod-beta' bind-return-pmf bind-assoc-pmf)

lemma *Pi-pmf-singleton*:

$Pi\text{-}pmf\ \{x\}\ dflt\ p = map\text{-}pmf\ (\lambda a\ b. if\ b = x\ then\ a\ else\ dflt)\ (p\ x)$
proof –
have $Pi\text{-}pmf\ \{x\}\ dflt\ p = map\text{-}pmf\ (fun\text{-}upd\ (\lambda-. dflt)\ x)\ (p\ x)$
by (*subst Pi-pmf-insert*) (*simp-all add: pair-return-pmf2 pmf.map-comp o-def*)
also have $fun\text{-}upd\ (\lambda-. dflt)\ x = (\lambda z\ y. if\ y = x\ then\ z\ else\ dflt)$
by (*simp add: fun-upd-def fun-eq-iff*)

finally show *?thesis* .
qed

Projecting a product of PMFs onto a component yields the expected result:

lemma *Pi-pmf-component:*
assumes *finite A*
shows $\text{map-pmf } (\lambda f. f x) (\text{Pi-pmf } A \text{ dflt } p) = (\text{if } x \in A \text{ then } p x \text{ else return-pmf } dflt)$
proof (*cases* $x \in A$)
case *True*
define A' **where** $A' = A - \{x\}$
from *assms* **and** *True* **have** $A': A = \text{insert } x A'$
by (*auto simp: A'-def*)
from *assms* **have** $\text{map-pmf } (\lambda f. f x) (\text{Pi-pmf } A \text{ dflt } p) = p x$ **unfolding** A'
by (*subst Pi-pmf-insert*)
(auto simp: A'-def pmf.map-comp o-def case-prod-unfold map-fst-pair-pmf)
with *True* **show** *?thesis* **by** *simp*
next
case *False*
have $\text{map-pmf } (\lambda f. f x) (\text{Pi-pmf } A \text{ dflt } p) = \text{map-pmf } (\lambda -. dflt) (\text{Pi-pmf } A \text{ dflt } p)$
using *assms False set-Pi-pmf-subset[of A dflt p]*
by (*intro pmf.map-cong refl*) (*auto simp: set-pmf-eq pmf-Pi-outside*)
with *False* **show** *?thesis* **by** *simp*
qed

We can take merge two PMF products on disjoint sets like this:

lemma *Pi-pmf-union:*
assumes *finite A finite B A ∩ B = {}*
shows $\text{Pi-pmf } (A \cup B) \text{ dflt } p =$
 $\text{map-pmf } (\lambda(f,g) x. \text{if } x \in A \text{ then } f x \text{ else } g x)$
 $(\text{pair-pmf } (\text{Pi-pmf } A \text{ dflt } p) (\text{Pi-pmf } B \text{ dflt } p))$ (**is** $= \text{map-pmf } (?h A)$
 $(?q A)$)
using *assms(1,3)*
proof (*induction rule: finite-induct*)
case (*insert* $x A$)
have $\text{map-pmf } (?h (\text{insert } x A)) (?q (\text{insert } x A)) =$
 $\text{do } \{v \leftarrow p x; (f, g) \leftarrow \text{pair-pmf } (\text{Pi-pmf } A \text{ dflt } p) (\text{Pi-pmf } B \text{ dflt } p);$
 $\text{return-pmf } (\lambda y. \text{if } y \in \text{insert } x A \text{ then } (f(x := v)) y \text{ else } g y)\}$
by (*subst Pi-pmf-insert*)
(insert insert.hyprs insert.premis,
simp-all add: pair-pmf-def map-bind-pmf bind-map-pmf bind-assoc-pmf
bind-return-pmf)
also have $\dots = \text{do } \{v \leftarrow p x; (f, g) \leftarrow ?q A; \text{return-pmf } ((?h A (f,g))(x := v))\}$
by (*intro bind-pmf-cong refl*) (*auto simp: fun-eq-iff*)
also have $\dots = \text{do } \{v \leftarrow p x; f \leftarrow \text{map-pmf } (?h A) (?q A); \text{return-pmf } (f(x := v))\}$
by (*simp add: bind-map-pmf map-bind-pmf case-prod-unfold cong: if-cong*)
also have $\dots = \text{do } \{v \leftarrow p x; f \leftarrow \text{Pi-pmf } (A \cup B) \text{ dflt } p; \text{return-pmf } (f(x :=$

```

v))}
  using insert.hyps and insert.premis by (intro bind-pmf-cong insert.IH [symmetric]
refl) auto
  also have ... = Pi-pmf (insert x (A ∪ B)) dflt p
    by (subst Pi-pmf-insert)
      (insert assms insert.hyps insert.premis, auto simp: pair-pmf-def map-bind-pmf)
  also have insert x (A ∪ B) = insert x A ∪ B
    by simp
  finally show ?case ..
qed (simp-all add: case-prod-unfold map-snd-pair-pmf)

```

We can also project a product to a subset of the indices by mapping all the other indices to the default value:

```

lemma Pi-pmf-subset:
  assumes finite A A' ⊆ A
  shows Pi-pmf A' dflt p = map-pmf (λf x. if x ∈ A' then f x else dflt) (Pi-pmf
A dflt p)
proof -
  let ?P = pair-pmf (Pi-pmf A' dflt p) (Pi-pmf (A - A') dflt p)
  from assms have [simp]: finite A'
    by (blast dest: finite-subset)
  from assms have A = A' ∪ (A - A')
    by blast
  also have Pi-pmf ... dflt p = map-pmf (λ(f,g) x. if x ∈ A' then f x else g x) ?P
    using assms by (intro Pi-pmf-union) auto
  also have map-pmf (λf x. if x ∈ A' then f x else dflt) ... = map-pmf fst ?P
    unfolding map-pmf-comp o-def case-prod-unfold
    using set-Pi-pmf-subset[of A' dflt p] by (intro map-pmf-cong refl) (auto simp:
fun-eq-iff)
  also have ... = Pi-pmf A' dflt p
    by (simp add: map-fst-pair-pmf)
  finally show ?thesis ..
qed

```

```

lemma Pi-pmf-subset':
  fixes f :: 'a ⇒ 'b pmf
  assumes finite A B ⊆ A ∧ x. x ∈ A - B ⇒ f x = return-pmf dflt
  shows Pi-pmf A dflt f = Pi-pmf B dflt f
proof -
  have Pi-pmf (B ∪ (A - B)) dflt f =
    map-pmf (λ(f, g) x. if x ∈ B then f x else g x)
      (pair-pmf (Pi-pmf B dflt f) (Pi-pmf (A - B) dflt f))
    using assms by (intro Pi-pmf-union) (auto dest: finite-subset)
  also have Pi-pmf (A - B) dflt f = Pi-pmf (A - B) dflt (λ-. return-pmf dflt)
    using assms by (intro Pi-pmf-cong) auto
  also have ... = return-pmf (λ-. dflt)
    using assms by simp
  also have map-pmf (λ(f, g) x. if x ∈ B then f x else g x)
    (pair-pmf (Pi-pmf B dflt f) (return-pmf (λ-. dflt))) =

```

$map\text{-}pmf (\lambda f x. \text{if } x \in B \text{ then } f x \text{ else } dflt) (Pi\text{-}pmf B dflt f)$
by (*simp add: map-pmf-def pair-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'*)
also have $\dots = Pi\text{-}pmf B dflt f$
using *assms* **by** (*intro Pi-pmf-default-swap*) (*auto dest: finite-subset*)
also have $B \cup (A - B) = A$
using *assms* **by** *auto*
finally show *?thesis* .
qed

lemma *Pi-pmf-if-set*:
assumes *finite A*
shows $Pi\text{-}pmf A dflt (\lambda x. \text{if } b x \text{ then } f x \text{ else } return\text{-}pmf dflt) =$
 $Pi\text{-}pmf \{x \in A. b x\} dflt f$
proof –
have $Pi\text{-}pmf A dflt (\lambda x. \text{if } b x \text{ then } f x \text{ else } return\text{-}pmf dflt) =$
 $Pi\text{-}pmf \{x \in A. b x\} dflt (\lambda x. \text{if } b x \text{ then } f x \text{ else } return\text{-}pmf dflt)$
using *assms* **by** (*intro Pi-pmf-subset'*) *auto*
also have $\dots = Pi\text{-}pmf \{x \in A. b x\} dflt f$
by (*intro Pi-pmf-cong*) *auto*
finally show *?thesis* .
qed

lemma *Pi-pmf-if-set'*:
assumes *finite A*
shows $Pi\text{-}pmf A dflt (\lambda x. \text{if } b x \text{ then } return\text{-}pmf dflt \text{ else } f x) =$
 $Pi\text{-}pmf \{x \in A. \neg b x\} dflt f$
proof –
have $Pi\text{-}pmf A dflt (\lambda x. \text{if } b x \text{ then } return\text{-}pmf dflt \text{ else } f x) =$
 $Pi\text{-}pmf \{x \in A. \neg b x\} dflt (\lambda x. \text{if } b x \text{ then } return\text{-}pmf dflt \text{ else } f x)$
using *assms* **by** (*intro Pi-pmf-subset'*) *auto*
also have $\dots = Pi\text{-}pmf \{x \in A. \neg b x\} dflt f$
by (*intro Pi-pmf-cong*) *auto*
finally show *?thesis* .
qed

Lastly, we can delete a single component from a product:

lemma *Pi-pmf-remove*:
assumes *finite A*
shows $Pi\text{-}pmf (A - \{x\}) dflt p = map\text{-}pmf (\lambda f. f(x := dflt)) (Pi\text{-}pmf A dflt$
 $p)$
proof –
have $Pi\text{-}pmf (A - \{x\}) dflt p =$
 $map\text{-}pmf (\lambda f xa. \text{if } xa \in A - \{x\} \text{ then } f xa \text{ else } dflt) (Pi\text{-}pmf A dflt p)$
using *assms* **by** (*intro Pi-pmf-subset*) *auto*
also have $\dots = map\text{-}pmf (\lambda f. f(x := dflt)) (Pi\text{-}pmf A dflt p)$
using *set-Pi-pmf-subset[of A dflt p] assms*
by (*intro map-pmf-cong refl*) (*auto simp: fun-eq-iff*)
finally show *?thesis* .
qed

26.6 Additional properties

lemma *nn-integral-prod-Pi-pmf*:

assumes *finite A*

shows $nn\text{-integral } (Pi\text{-pmf } A \text{ dflt } p) (\lambda y. \prod x \in A. f \ x \ (y \ x)) = (\prod x \in A. nn\text{-integral } (p \ x) \ (f \ x))$

using *assms*

proof (*induction rule: finite-induct*)

case (*insert x A*)

have $nn\text{-integral } (Pi\text{-pmf } (insert \ x \ A) \ \text{dflt } \ p) (\lambda y. \prod z \in insert \ x \ A. f \ z \ (y \ z)) = (\int^+ a. \int^+ b. f \ x \ a \ * \ (\prod z \in A. f \ z \ (if \ z = x \ then \ a \ else \ b \ z))) \ \partial Pi\text{-pmf } A \ \text{dflt } p \ \partial p \ x)$

using *insert by (auto simp: Pi-pmf-insert case-prod-unfold nn-integral-pair-pmf' cong: if-cong)*

also have $(\lambda a \ b. \prod z \in A. f \ z \ (if \ z = x \ then \ a \ else \ b \ z)) = (\lambda a \ b. \prod z \in A. f \ z \ (b \ z))$

by (*intro ext prod.cong (use insert.hyps in auto)*)

also have $(\int^+ a. \int^+ b. f \ x \ a \ * \ (\prod z \in A. f \ z \ (b \ z))) \ \partial Pi\text{-pmf } A \ \text{dflt } p \ \partial p \ x = (\int^+ y. f \ x \ y \ \partial(p \ x)) \ * \ (\int^+ y. (\prod z \in A. f \ z \ (y \ z))) \ \partial(Pi\text{-pmf } A \ \text{dflt } p)$

by (*simp add: nn-integral-multc nn-integral-cmult*)

also have $(\int^+ y. (\prod z \in A. f \ z \ (y \ z))) \ \partial(Pi\text{-pmf } A \ \text{dflt } p) = (\prod x \in A. nn\text{-integral } (p \ x) \ (f \ x))$

by (*rule insert.IH*)

also have $(\int^+ y. f \ x \ y \ \partial(p \ x)) \ * \ \dots = (\prod x \in insert \ x \ A. nn\text{-integral } (p \ x) \ (f \ x))$

using *insert.hyps by simp*

finally show *?case .*

qed *auto*

lemma *integrable-prod-Pi-pmf*:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{real\text{-normed-field, second-countable-topology, banach}\}$

assumes *finite A and $\bigwedge x. x \in A \implies integrable \ (measure\text{-pmf } (p \ x)) \ (f \ x)$*

shows $integrable \ (measure\text{-pmf } (Pi\text{-pmf } A \ \text{dflt } p)) (\lambda h. \prod x \in A. f \ x \ (h \ x))$

proof (*intro integrableI-bounded*)

have $(\int^+ x. ennreal \ (norm \ (\prod xa \in A. f \ xa \ (x \ xa))) \ \partial measure\text{-pmf } (Pi\text{-pmf } A \ \text{dflt } p)) =$

$(\int^+ x. (\prod y \in A. ennreal \ (norm \ (f \ y \ (x \ y)))) \ \partial measure\text{-pmf } (Pi\text{-pmf } A \ \text{dflt } p))$

by (*simp flip: prod-norm prod-ennreal*)

also have $\dots = (\prod x \in A. \int^+ a. ennreal \ (norm \ (f \ x \ a)) \ \partial measure\text{-pmf } (p \ x))$

by (*intro nn-integral-prod-Pi-pmf fact*)

also have $(\int^+ a. ennreal \ (norm \ (f \ i \ a)) \ \partial measure\text{-pmf } (p \ i)) \neq top$ **if** $i: i \in A$ **for** i

using *assms(2)[OF i] by (simp add: integrable-iff-bounded)*

hence $(\prod x \in A. \int^+ a. ennreal \ (norm \ (f \ x \ a)) \ \partial measure\text{-pmf } (p \ x)) \neq top$

by (*subst ennreal-prod-eq-top auto*)

finally show $(\int^+ x. ennreal \ (norm \ (\prod xa \in A. f \ xa \ (x \ xa))) \ \partial measure\text{-pmf } (Pi\text{-pmf } A \ \text{dflt } p)) < \infty$

by (*simp add: top.not-eq-extremum*)

qed *auto*

lemma *expectation-prod-Pi-pmf*:

fixes $f :: - \Rightarrow - \Rightarrow \text{real}$

assumes *finite A*

assumes $\bigwedge x. x \in A \implies \text{integrable } (\text{measure-pmf } (p \ x)) \ (f \ x)$

assumes $\bigwedge x \ y. x \in A \implies y \in \text{set-pmf } (p \ x) \implies f \ x \ y \geq 0$

shows $\text{measure-pmf.expectation } (Pi\text{-pmf } A \ \text{dflt } p) \ (\lambda y. \prod_{x \in A}. f \ x \ (y \ x)) =$
 $(\prod_{x \in A}. \text{measure-pmf.expectation } (p \ x) \ (\lambda v. f \ x \ v))$

proof –

have *nonneg*: $\text{measure-pmf.expectation } (p \ x) \ (f \ x) \geq 0$ **if** $x \in A$ **for** x

using *that by* (*intro Bochner-Integration.integral-nonneg-AE AE-pmfI assms*)

have *nonneg'*: $0 \leq \text{measure-pmf.expectation } (Pi\text{-pmf } A \ \text{dflt } p) \ (\lambda y. \prod_{x \in A}. f \ x \ (y \ x))$

by (*intro Bochner-Integration.integral-nonneg-AE AE-pmfI assms prod-nonneg*)
(use assms in ‹auto simp: set-Pi-pmf PiE-dflt-def›)

have *ennreal* ($\text{measure-pmf.expectation } (Pi\text{-pmf } A \ \text{dflt } p) \ (\lambda y. \prod_{x \in A}. f \ x \ (y \ x))$)

=

nn-integral ($Pi\text{-pmf } A \ \text{dflt } p$) ($\lambda y. \text{ennreal } (\prod_{x \in A}. f \ x \ (y \ x))$) **using** *assms*

by (*intro nn-integral-eq-integral [symmetric] assms integrable-prod-Pi-pmf*)

(auto simp: AE-measure-pmf-iff set-Pi-pmf PiE-dflt-def prod-nonneg)

also have $\dots = \text{nn-integral } (Pi\text{-pmf } A \ \text{dflt } p) \ (\lambda y. (\prod_{x \in A}. \text{ennreal } (f \ x \ (y \ x))))$

by (*intro nn-integral-cong-AE AE-pmfI prod-ennreal [symmetric]*)

(use assms(1) in ‹auto simp: set-Pi-pmf PiE-dflt-def intro!: assms(3)›)

also have $\dots = (\prod_{x \in A}. \int^+ a. \text{ennreal } (f \ x \ a) \ \partial \text{measure-pmf } (p \ x))$

by (*rule nn-integral-prod-Pi-pmf*) *fact+*

also have $\dots = (\prod_{x \in A}. \text{ennreal } (\text{measure-pmf.expectation } (p \ x) \ (f \ x)))$

by (*intro prod.cong nn-integral-eq-integral assms AE-pmfI*) *auto*

also have $\dots = \text{ennreal } (\prod_{x \in A}. \text{measure-pmf.expectation } (p \ x) \ (f \ x))$

by (*intro prod-ennreal nonneg*)

finally show *?thesis*

using *nonneg nonneg'* **by** (*subst (asm) ennreal-inj*) (*auto intro!: prod-nonneg*)

qed

lemma *indep-vars-Pi-pmf*:

assumes *fin: finite I*

shows *prob-space.indep-vars* ($\text{measure-pmf } (Pi\text{-pmf } I \ \text{dflt } p)$)

$(\lambda \cdot. \text{count-space UNIV}) \ (\lambda x \ f. f \ x) \ I$

proof (*cases I = {}*)

case *True*

show *?thesis*

by (*subst prob-space.indep-vars-def [OF measure-pmf.prob-space-axioms]*,

subst prob-space.indep-sets-def [OF measure-pmf.prob-space-axioms]) (*simp-all*

add: True)

next

case [*simp*]: *False*

show *?thesis*

proof (*subst prob-space.indep-vars-iff-distr-eq-PiM'*)

show *distr* ($\text{measure-pmf } (Pi\text{-pmf } I \ \text{dflt } p)$) ($Pi_M \ I$) ($\lambda i. \text{count-space UNIV}$)

$(\lambda x. \text{restrict } x \ I) =$

```

       $Pi_M I (\lambda i. \text{distr } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p)) (\text{count-space } UNIV) (\lambda f. f i))$ 
    proof (rule product-sigma-finite.PiM-eqI, goal-cases)
      case 1
        interpret product-prob-space  $\lambda i. \text{distr } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p))$ 
        ( $\text{count-space } UNIV$ ) ( $\lambda f. f i$ )
        by (intro product-prob-spaceI prob-space.prob-space-distr measure-pmf.prob-space-axioms)
            simp-all
        show ?case by unfold-locales
      next
        case 3
          have sets ( $Pi_M I (\lambda i. \text{distr } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p)) (\text{count-space } UNIV) (\lambda f. f i))$ ) =
            sets ( $Pi_M I (\lambda \cdot. \text{count-space } UNIV)$ )
          by (intro sets-PiM-cong) simp-all
          thus ?case by simp
        next
          case ( $\not\leq A$ )
          have  $Pi_E I A \in \text{sets } (Pi_M I (\lambda i. \text{count-space } UNIV))$ 
          using  $\not\leq$  by (intro sets-PiM-I-finite fin) auto
          hence  $\text{emeasure } (\text{distr } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p)) (Pi_M I (\lambda i. \text{count-space } UNIV)))$ 
            ( $\lambda x. \text{restrict } x I$ ) ( $Pi_E I A$ ) =
             $\text{emeasure } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p)) ((\lambda x. \text{restrict } x I) -' Pi_E I A)$ 
          using  $\not\leq$  by (subst emeasure-distr) (auto simp: space-PiM)
          also have ... =  $\text{emeasure } (\text{measure-pmf } (Pi\text{-pmf } I \text{ dflt } p)) (Pi_E\text{-dflt } I \text{ dflt } A)$ 
          by (intro emeasure-eq-AE AE-pmfI) (auto simp: PiE-dflt-def set-Pi-pmf fin)
          also have ... =  $(\prod_{i \in I}. \text{emeasure } (\text{measure-pmf } (p i)) (A i))$ 
          by (simp add: measure-pmf.emeasure-eq-measure measure-Pi-pmf-PiE-dflt
            fin prod-enreal)
          also have ... =  $(\prod_{i \in I}. \text{emeasure } (\text{measure-pmf } (\text{map-pmf } (\lambda f. f i) (Pi\text{-pmf } I \text{ dflt } p))) (A i))$ 
          by (intro prod.cong refl, subst Pi-pmf-component) (auto simp: fin)
          finally show ?case
          by (simp add: map-pmf-rep-eq)
        qed fact+
      qed (simp-all add: measure-pmf.prob-space-axioms)
    qed

```

lemma

```

fixes  $h :: 'a :: \text{comm-monoid-add} \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$ 
assumes fin: finite I
assumes integrable:  $\bigwedge i. i \in I \implies \text{integrable } (\text{measure-pmf } (D i)) h$ 
shows integrable-sum-Pi-pmf:  $\text{integrable } (Pi\text{-pmf } I \text{ dflt } D) (\lambda g. \sum_{i \in I}. h (g i))$ 
and expectation-sum-Pi-pmf:
       $\text{measure-pmf.expectation } (Pi\text{-pmf } I \text{ dflt } D) (\lambda g. \sum_{i \in I}. h (g i)) =$ 
       $(\sum_{i \in I}. \text{measure-pmf.expectation } (D i) h)$ 
proof –
  have integrable':  $\text{integrable } (Pi\text{-pmf } I \text{ dflt } D) (\lambda g. h (g i))$  if  $i: i \in I$  for  $i$ 

```



```

proof –
  have integrable (D i) h
    using i by (rule assms)
  also have  $D\ i = \text{map-pmf } (\lambda g. g\ i) (Pi\text{-pmf } I\ \text{dflt } D)$ 
    by (subst Pi-pmf-component) (use fin i in auto)
  finally show integrable (measure-pmf (Pi-pmf I dflt D)) ( $\lambda x. h\ (x\ i)$ )
    by simp
qed
thus integrable (Pi-pmf I dflt D) ( $\lambda g. \sum_{i \in I}. h\ (g\ i)$ )
  by (intro Bochner-Integration.integrable-sum)

have measure-pmf.expectation (Pi-pmf I dflt D) ( $\lambda x. \sum_{i \in I}. h\ (x\ i)$ ) =
  ( $\sum_{i \in I}. \text{measure-pmf.expectation } (\text{map-pmf } (\lambda x. x\ i) (Pi\text{-pmf } I\ \text{dflt } D))\ h$ )
  using integrable' by (subst Bochner-Integration.integral-sum) auto
also have  $\dots = (\sum_{i \in I}. \text{measure-pmf.expectation } (D\ i)\ h)$ 
  by (intro sum.cong refl, subst Pi-pmf-component) (use fin in auto)
finally show measure-pmf.expectation (Pi-pmf I dflt D) ( $\lambda g. \sum_{i \in I}. h\ (g\ i)$ ) =
  ( $\sum_{i \in I}. \text{measure-pmf.expectation } (D\ i)\ h$ ) .
qed

```

26.7 Applications

Choosing a subset of a set uniformly at random is equivalent to tossing a fair coin independently for each element and collecting all the elements that came up heads.

lemma *pmf-of-set-Pow-conv-bernoulli*:

```

assumes finite (A :: 'a set)
shows map-pmf ( $\lambda b. \{x \in A. b\ x\}$ ) (Pi-pmf A P ( $\lambda-. \text{bernoulli-pmf } (1/2)$ )) =
pmf-of-set (Pow A)
proof –
  have Pi-pmf A P ( $\lambda-. \text{bernoulli-pmf } (1/2)$ ) = pmf-of-set (PiE-dflt A P ( $\lambda x. UNIV$ ))
    using assms by (simp add: bernoulli-pmf-half-conv-pmf-of-set Pi-pmf-of-set)
  also have map-pmf ( $\lambda b. \{x \in A. b\ x\}$ )  $\dots = \text{pmf-of-set } (Pow\ A)$ 
    proof –
      have bij-betw ( $\lambda b. \{x \in A. b\ x\}$ ) (PiE-dflt A P ( $\lambda-. UNIV$ )) (Pow A)
        by (rule bij-betwI[of - - -  $\lambda B\ b. \text{if } b \in A \text{ then } b \in B \text{ else } P$ ]) (auto simp add: PiE-dflt-def)
      then show ?thesis
        using assms by (intro map-pmf-of-set-bij-betw) auto
    qed
  finally show ?thesis
    by simp
qed

```

A binomial distribution can be seen as the number of successes in n independent coin tosses.

```

lemma binomial-pmf-altdef':
  fixes  $A :: 'a \text{ set}$ 
  assumes finite A and card A = n and p: p ∈ {0..1}
  shows  $\text{binomial-pmf } n \text{ } p =$ 
     $\text{map-pmf } (\lambda f. \text{card } \{x \in A. f \ x\}) (\text{Pi-pmf } A \ \text{dflt } (\lambda-. \text{bernoulli-pmf } p))$  (is
     $\text{?lhs} = \text{?rhs})$ 
  proof –
    from assms have  $\text{?lhs} = \text{binomial-pmf } (\text{card } A) \text{ } p$ 
      by simp
    also have  $\dots = \text{?rhs}$ 
    using assms(1)
    proof (induction rule: finite-induct)
      case empty
      with  $p$  show  $\text{?case}$  by (simp add: binomial-pmf-0)
    next
      case (insert x A)
      from insert.hyps have  $\text{card } (\text{insert } x \ A) = \text{Suc } (\text{card } A)$ 
        by simp
      also have  $\text{binomial-pmf } \dots \text{ } p = \text{do } \{$ 
         $b \leftarrow \text{bernoulli-pmf } p;$ 
         $f \leftarrow \text{Pi-pmf } A \ \text{dflt } (\lambda-. \text{bernoulli-pmf } p);$ 
         $\text{return-pmf } ((\text{if } b \text{ then } 1 \text{ else } 0) + \text{card } \{y \in A. f \ y\})$ 
         $\}$ 
        using  $p$  by (simp add: binomial-pmf-Suc insert.IH bind-map-pmf)
      also have  $\dots = \text{do } \{$ 
         $b \leftarrow \text{bernoulli-pmf } p;$ 
         $f \leftarrow \text{Pi-pmf } A \ \text{dflt } (\lambda-. \text{bernoulli-pmf } p);$ 
         $\text{return-pmf } (\text{card } \{y \in \text{insert } x \ A. (f(x := b)) \ y\})$ 
         $\}$ 
      proof (intro bind-pmf-cong refl, goal-cases)
        case ( $1 \ b \ f$ )
        have  $(\text{if } b \text{ then } 1 \text{ else } 0) + \text{card } \{y \in A. f \ y\} = \text{card } ((\text{if } b \text{ then } \{x\} \text{ else } \{\}) \cup$ 
           $\{y \in A. f \ y\})$ 
          using insert.hyps by auto
          also have  $(\text{if } b \text{ then } \{x\} \text{ else } \{\}) \cup \{y \in A. f \ y\} = \{y \in \text{insert } x \ A. (f(x := b))$ 
             $y\}$ 
            using insert.hyps by auto
          finally show  $\text{?case}$  by simp
        qed
        also have  $\dots = \text{map-pmf } (\lambda f. \text{card } \{y \in \text{insert } x \ A. f \ y\})$ 
           $(\text{Pi-pmf } (\text{insert } x \ A) \ \text{dflt } (\lambda-. \text{bernoulli-pmf } p))$ 
          using insert.hyps by (subst Pi-pmf-insert) (simp-all add: pair-pmf-def map-bind-pmf)
          finally show  $\text{?case}$  .
        qed
        finally show  $\text{?thesis}$  .
      qed
    end

```

27 Hoeffding’s Lemma and Hoeffding’s Inequality

```
theory Hoeffding
  imports Product-PMF Independent-Family
begin
```

Hoeffding’s inequality shows that a sum of bounded independent random variables is concentrated around its mean, with an exponential decay of the tail probabilities.

27.1 Hoeffding’s Lemma

```
lemma convex-on-exp:
  fixes l :: real
  assumes l ≥ 0
  shows convex-on UNIV (λx. exp(l*x))
  using assms
  by (intro convex-on-realI[where f' = λx. l * exp (l * x)])
    (auto intro!: derivative-eq-intros mult-left-mono)
```

```
lemma mult-const-minus-self-real-le:
  fixes x :: real
  shows x * (c - x) ≤ c2 / 4
proof -
  have x * (c - x) = -(x - c / 2)2 + c2 / 4
    by (simp add: field-simps power2-eq-square)
  also have ... ≤ 0 + c2 / 4
    by (intro add-mono) auto
  finally show ?thesis by simp
qed
```

```
lemma Hoeffdings-lemma-aux:
  fixes h p :: real
  assumes h ≥ 0 and p ≥ 0
  defines L ≡ (λh. -h * p + ln (1 + p * (exp h - 1)))
  shows L h ≤ h2 / 8
proof (cases h = 0)
  case False
  hence h: h > 0
  using ⟨h ≥ 0⟩ by simp
  define L' where L' = (λh. -p + p * exp h / (1 + p * (exp h - 1)))
  define L'' where L'' = (λh. -(p2) * exp h * exp h / (1 + p * (exp h - 1))2 +
    p * exp h / (1 + p * (exp h - 1)))
  define Ls where Ls = (λn. [L, L', L''] ! n)

  have [simp]: L 0 = 0 L' 0 = 0
    by (auto simp: L-def L'-def)

  have L': (L has-real-derivative L' x) (at x) if x ∈ {0..h} for x
```

```

proof –
  have  $1 + p * (\exp x - 1) > 0$ 
    using  $\langle p \geq 0 \rangle$  that by (intro add-pos-nonneg mult-nonneg-nonneg) auto
  thus ?thesis
  unfolding L-def L'-def by (auto intro!: derivative-eq-intros)
qed

have L'': (L' has-real-derivative L'' x) (at x) if  $x \in \{0..h\}$  for x
proof –
  have *:  $1 + p * (\exp x - 1) > 0$ 
    using  $\langle p \geq 0 \rangle$  that by (intro add-pos-nonneg mult-nonneg-nonneg) auto
  show ?thesis
  unfolding L'-def L''-def
  by (insert *, (rule derivative-eq-intros refl | simp)+) (auto simp: divide-simps;
algebra)
qed

have diff:  $\forall m t. m < 2 \wedge 0 \leq t \wedge t \leq h \longrightarrow (Ls\ m\ \text{has-real-derivative}\ Ls\ (Suc\ m)\ t)\ (at\ t)$ 
  using L' L'' by (auto simp: Ls-def nth-Cons split: nat.splits)
from Taylor[of 2 Ls L 0 h 0 h, OF - - diff]
  obtain t where  $t \in \{0 <..<h\}$   $L\ h = L''\ t * h^2 / 2$ 
  using  $\langle h > 0 \rangle$  by (auto simp: Ls-def lessThan-nat-numeral)
define u where  $u = p * \exp t / (1 + p * (\exp t - 1))$ 

have  $L''\ t = u * (1 - u)$ 
  by (simp add: L''-def u-def divide-simps; algebra)
also have  $\dots \leq 1 / 4$ 
  using mult-const-minus-self-real-le[of u 1] by simp
finally have  $L''\ t \leq 1 / 4$  .

note t(2)
also have  $L''\ t * h^2 / 2 \leq (1 / 4) * h^2 / 2$ 
  using  $\langle L''\ t \leq 1 / 4 \rangle$  by (intro mult-right-mono divide-right-mono) auto
finally show  $L\ h \leq h^2 / 8$  by simp
qed (auto simp: L-def)

locale interval-bounded-random-variable = prob-space +
  fixes f :: 'a  $\Rightarrow$  real and a b :: real
  assumes random-variable [measurable]: random-variable borel f
  assumes AE-in-interval: AE x in M. f x  $\in \{a..b\}$ 
begin

lemma integrable [intro]: integrable M f
proof (rule integrable-const-bound)
  show AE x in M. norm (f x)  $\leq \max |a| |b|$ 
  by (intro eventually-mono[OF AE-in-interval]) auto
qed (fact random-variable)

```

We first show Hoeffding’s lemma for distributions whose expectation is 0. The general case will easily follow from this later.

lemma *Hoeffdings-lemma-nn-integral-0:*

assumes $l > 0$ **and** $E0$: *expectation* $f = 0$

shows *nn-integral* $M (\lambda x. \exp (l * f x)) \leq \text{ennreal } (\exp (l^2 * (b - a)^2 / 8))$

proof (*cases* $AE\ x\ \text{in}\ M. f\ x = 0$)

case *True*

hence *nn-integral* $M (\lambda x. \exp (l * f x)) = \text{nn-integral } M (\lambda x. \text{ennreal } 1)$

by (*intro nn-integral-cong-AE*) *auto*

also have $\dots = \text{ennreal } (\text{expectation } (\lambda \cdot. 1))$

by (*intro nn-integral-eq-integral*) *auto*

finally show *?thesis* **by** (*simp add: prob-space*)

next

case *False*

have $a < 0$

proof (*rule ccontr*)

assume a : $\neg(a < 0)$

have $AE\ x\ \text{in}\ M. f\ x = 0$

proof (*subst integral-nonneg-eq-0-iff-AE [symmetric]*)

show $AE\ x\ \text{in}\ M. f\ x \geq 0$

using *AE-in-interval* **by** *eventually-elim* (*use a in auto*)

qed (*use E0 in <auto simp: id-def integrable>*)

with *False* **show** *False* **by** *contradiction*

qed

have $b > 0$

proof (*rule ccontr*)

assume b : $\neg(b > 0)$

have $AE\ x\ \text{in}\ M. -f\ x = 0$

proof (*subst integral-nonneg-eq-0-iff-AE [symmetric]*)

show $AE\ x\ \text{in}\ M. -f\ x \geq 0$

using *AE-in-interval* **by** *eventually-elim* (*use b in auto*)

qed (*use E0 in <auto simp: id-def integrable>*)

with *False* **show** *False* **by** *simp*

qed

have $a < b$

using $\langle a < 0 \rangle \langle b > 0 \rangle$ **by** *linarith*

define p **where** $p = -a / (b - a)$

define L **where** $L = (\lambda t. -t * p + \ln (1 - p + p * \exp t))$

define z **where** $z = l * (b - a)$

have $z > 0$

unfolding *z-def* **using** $\langle a < b \rangle \langle l > 0 \rangle$ **by** *auto*

have $p > 0$

using $\langle a < 0 \rangle \langle a < b \rangle$ **unfolding** *p-def* **by** (*intro divide-pos-pos*) *auto*

have $(\int^+ x. \exp (l * f x) \partial M) \leq$

$(\int^+ x. (b - f x) / (b - a) * \exp (l * a) + (f x - a) / (b - a) * \exp (l * b))$

∂M)

proof (*intro nn-integral-mono-AE eventually-mono[OF AE-in-interval] ennreal-leI*)
fix x **assume** $x: f x \in \{a..b\}$
define y **where** $y = (b - f x) / (b - a)$
have $y: y \in \{0..1\}$
using $x \langle a < b \rangle$ **by** (*auto simp: y-def*)
have *conv: convex-on UNIV* ($\lambda x. \exp(l * x)$)
using $\langle l > 0 \rangle$ **by** (*intro convex-on-exp*) *auto*
have $\exp(l * ((1 - y) * b + y * a)) \leq (1 - y) * \exp(l * b) + y * \exp(l * a)$
using $y \langle l > 0 \rangle$ **by** (*intro convex-onD[OF convex-on-exp]*) *auto*
also have $(1 - y) * b + y * a = f x$
using $\langle a < b \rangle$ **by** (*simp add: y-def divide-simps*) (*simp add: algebra-simps*)?
also have $1 - y = (f x - a) / (b - a)$
using $\langle a < b \rangle$ **by** (*simp add: field-simps y-def*)
finally show $\exp(l * f x) \leq (b - f x) / (b - a) * \exp(l * a) + (f x - a) / (b - a) * \exp(l * b)$
by (*simp add: y-def*)
qed
also have $\dots = (\int^+ x. \text{ennreal } (b - f x) * \exp(l * a) / (b - a) + \text{ennreal } (f x - a) * \exp(l * b) / (b - a) \partial M)$
using $\langle a < 0 \rangle \langle b > 0 \rangle$
by (*intro nn-integral-cong-AE eventually-mono[OF AE-in-interval]*) (*simp add: ennreal-plus ennreal-mult flip: divide-ennreal*)
also have $\dots = ((\int^+ x. \text{ennreal } (b - f x) \partial M) * \text{ennreal } (\exp(l * a)) + (\int^+ x. \text{ennreal } (f x - a) \partial M) * \text{ennreal } (\exp(l * b))) / \text{ennreal } (b - a)$
by (*simp add: nn-integral-add nn-integral-divide nn-integral-multc add-divide-distrib-ennreal*)
also have $(\int^+ x. \text{ennreal } (b - f x) \partial M) = \text{ennreal } (\text{expectation } (\lambda x. b - f x))$
by (*intro nn-integral-eq-integral Bochner-Integration.integrable-diff eventually-mono[OF AE-in-interval] integrable-const integrable*) *auto*
also have $\text{expectation } (\lambda x. b - f x) = b$
using *assms* **by** (*subst Bochner-Integration.integral-diff*) (*auto simp: prob-space*)
also have $(\int^+ x. \text{ennreal } (f x - a) \partial M) = \text{ennreal } (\text{expectation } (\lambda x. f x - a))$
by (*intro nn-integral-eq-integral Bochner-Integration.integrable-diff eventually-mono[OF AE-in-interval] integrable-const integrable*) *auto*
also have $\text{expectation } (\lambda x. f x - a) = (-a)$
using *assms* **by** (*subst Bochner-Integration.integral-diff*) (*auto simp: prob-space*)
also have $(\text{ennreal } b * (\exp(l * a)) + \text{ennreal } (-a) * (\exp(l * b))) / (b - a) = \text{ennreal } (b * \exp(l * a) - a * \exp(l * b)) / \text{ennreal } (b - a)$
using $\langle a < 0 \rangle \langle b > 0 \rangle$
by (*simp flip: ennreal-mult ennreal-plus add: mult-nonpos-nonneg divide-ennreal mult-mono*)
also have $b * \exp(l * a) - a * \exp(l * b) = \exp(L z) * (b - a)$
proof –
have *pos: 1 - p + p * exp z > 0*
proof –
have $\exp z > 1$ **using** $\langle l > 0 \rangle$ **and** $\langle a < b \rangle$
by (*subst one-less-exp-iff*) (*auto simp: z-def intro!: mult-pos-pos*)

```

hence  $(\exp z - 1) * p \geq 0$ 
  unfolding p-def using  $\langle a < 0 \rangle$  and  $\langle a < b \rangle$ 
  by (intro mult-nonneg-nonneg divide-nonneg-pos) auto
thus ?thesis
  by (simp add: algebra-simps)
qed

have  $\exp (L z) * (b - a) = \exp (-z * p) * (1 - p + p * \exp z) * (b - a)$ 
  using pos by (simp add: exp-add L-def exp-diff exp-minus divide-simps)
also have  $\dots = b * \exp (l * a) - a * \exp (l * b)$  using  $\langle a < b \rangle$ 
  by (simp add: p-def z-def divide-simps) (simp add: exp-diff algebra-simps)?
finally show ?thesis by simp
qed
also have  $\text{ennreal} (\exp (L z) * (b - a)) / \text{ennreal} (b - a) = \text{ennreal} (\exp (L z))$ 
  using  $\langle a < b \rangle$  by (simp add: divide-ennreal)
also have  $L z = -z * p + \ln (1 + p * (\exp z - 1))$ 
  by (simp add: L-def algebra-simps)
also have  $\dots \leq z^2 / 8$ 
  unfolding L-def by (rule Hoeffdings-lemma-aux[where p = p]) (use  $\langle z > 0 \rangle$ 
 $\langle p > 0 \rangle$  in simp-all)
  hence  $\text{ennreal} (\exp (-z * p + \ln (1 + p * (\exp z - 1)))) \leq \text{ennreal} (\exp (z^2 / 8))$ 
  by (intro ennreal-leI) auto
finally show ?thesis
  by (simp add: z-def power-mult-distrib)
qed

context
begin

interpretation shift: interval-bounded-random-variable M  $\lambda x. f x - \mu a - \mu b - \mu$ 
  rewrites  $b - \mu - (a - \mu) \equiv b - a$ 
  by unfold-locales (auto intro!: eventually-mono[OF AE-in-interval])

lemma expectation-shift: expectation  $(\lambda x. f x - \text{expectation } f) = 0$ 
  by (subst Bochner-Integration.integral-diff) (auto simp: integrable prob-space)

lemmas Hoeffdings-lemma-nn-integral = shift.Hoeffdings-lemma-nn-integral-0[OF
  - expectation-shift]

end

end

```

27.2 Hoeffding’s Inequality

Consider n independent real random variables X_1, \dots, X_n that each almost surely lie in a compact interval $[a_i, b_i]$. Hoeffding’s inequality states that the

distribution of the sum of the X_i is tightly concentrated around the sum of the expected values: the probability of it being above or below the sum of the expected values by more than some ε decreases exponentially with ε .

```

locale indep-interval-bounded-random-variables = prob-space +
  fixes  $I :: 'b$  set and  $X :: 'b \Rightarrow 'a \Rightarrow real$ 
  fixes  $a\ b :: 'b \Rightarrow real$ 
  assumes fin: finite  $I$ 
  assumes indep: indep-vars ( $\lambda$ -. borel)  $X\ I$ 
  assumes AE-in-interval:  $\bigwedge i. i \in I \implies AE\ x\ in\ M. X\ i\ x \in \{a\ i..b\ i\}$ 
begin

```

```

lemma random-variable [measurable]:
  assumes  $i: i \in I$ 
  shows random-variable borel ( $X\ i$ )
  using  $i$  indep unfolding indep-vars-def by blast

```

```

lemma bounded-random-variable [intro]:
  assumes  $i: i \in I$ 
  shows interval-bounded-random-variable  $M\ (X\ i)\ (a\ i)\ (b\ i)$ 
  by unfold-locales (use AE-in-interval[OF  $i$ ]  $i$  in auto)

```

end

```

locale Hoeffding-ineq = indep-interval-bounded-random-variables +
  fixes  $\mu :: real$ 
  defines  $\mu \equiv (\sum i \in I. expectation\ (X\ i))$ 
begin

```

```

theorem Hoeffding-ineq-ge:
  assumes  $\varepsilon \geq 0$ 
  assumes  $(\sum i \in I. (b\ i - a\ i)^2) > 0$ 
  shows prob  $\{x \in space\ M. (\sum i \in I. X\ i\ x) \geq \mu + \varepsilon\} \leq exp\ (-2 * \varepsilon^2 / (\sum i \in I. (b\ i - a\ i)^2))$ 

```

```

proof (cases  $\varepsilon = 0$ )
  case [simp]: True
  have prob  $\{x \in space\ M. (\sum i \in I. X\ i\ x) \geq \mu + \varepsilon\} \leq 1$ 
    by simp
  thus ?thesis by simp

```

```

next
  case False
  with  $\langle \varepsilon \geq 0 \rangle$  have  $\varepsilon: \varepsilon > 0$ 
    by auto

```

```

  define  $d$  where  $d = (\sum i \in I. (b\ i - a\ i)^2)$ 
  define  $l :: real$  where  $l = 4 * \varepsilon / d$ 
  have  $d: d > 0$ 
    using assms by (simp add: d-def)
  have  $l: l > 0$ 

```


using ε *d* **by** (*simp add: l-def*)
define μ' **where** $\mu' = (\lambda i. \text{expectation } (X\ i))$

have $\{x \in \text{space } M. (\sum i \in I. X\ i\ x) \geq \mu + \varepsilon\} = \{x \in \text{space } M. (\sum i \in I. X\ i\ x) - \mu \geq \varepsilon\}$
by (*simp add: algebra-simps*)
hence *ennreal* (*prob* $\{x \in \text{space } M. (\sum i \in I. X\ i\ x) \geq \mu + \varepsilon\}$) = *emeasure* *M* ...
by (*simp add: emeasure-eq-measure*)
also have ... $\leq \text{ennreal } (\exp (-l * \varepsilon)) * (\int^+ x \in \text{space } M. \exp (l * ((\sum i \in I. X\ i\ x) - \mu))) \partial M)$
by (*intro Chernoff-ineq-nn-integral-ge l*) *auto*
also have $(\lambda x. (\sum i \in I. X\ i\ x) - \mu) = (\lambda x. (\sum i \in I. X\ i\ x - \mu' i))$
by (*simp add: μ -def sum-subtractf μ' -def*)
also have $(\int^+ x \in \text{space } M. \exp (l * ((\sum i \in I. X\ i\ x - \mu' i))) \partial M) = (\int^+ x. (\prod i \in I. \text{ennreal } (\exp (l * (X\ i\ x - \mu' i)))) \partial M)$
by (*intro nn-integral-cong*)
(simp-all add: sum-distrib-left ring-distrib exp-diff exp-sum fin prod-ennreal)
also have ... = $(\prod i \in I. \int^+ x. \text{ennreal } (\exp (l * (X\ i\ x - \mu' i))) \partial M)$
by (*intro indep-vars-nn-integral fin indep-vars-compose2[OF indep]*) *auto*
also have *ennreal* ($\exp (-l * \varepsilon)$) * ... $\leq \text{ennreal } (\exp (-l * \varepsilon)) * (\prod i \in I. \text{ennreal } (\exp (l^2 * (b\ i - a\ i)^2 / 8)))$
proof (*intro mult-left-mono prod-mono-ennreal*)
fix *i* **assume** *i*: $i \in I$
from *i* **interpret** *interval-bounded-random-variable* *M* *X* *i* *a* *b* *i* ..
show $(\int^+ x. \text{ennreal } (\exp (l * (X\ i\ x - \mu' i))) \partial M) \leq \text{ennreal } (\exp (l^2 * (b\ i - a\ i)^2 / 8))$
unfolding μ' -*def* **by** (*rule Hoeffdings-lemma-nn-integral*) *fact+*
qed *auto*
also have ... = *ennreal* ($\exp (-l * \varepsilon) * (\prod i \in I. \exp (l^2 * (b\ i - a\ i)^2 / 8))$)
by (*simp add: prod-ennreal prod-nonneg flip: ennreal-mult*)
also have $\exp (-l * \varepsilon) * (\prod i \in I. \exp (l^2 * (b\ i - a\ i)^2 / 8)) = \exp (d * l^2 / 8 - l * \varepsilon)$
by (*simp add: exp-diff exp-minus sum-divide-distrib sum-distrib-left sum-distrib-right exp-sum fin divide-simps mult-ac d-def*)
also have $d * l^2 / 8 - l * \varepsilon = -2 * \varepsilon^2 / d$
using *d* **by** (*simp add: l-def field-simps power2-eq-square*)
finally show ?*thesis*
by (*subst (asm) ennreal-le-iff*) (*simp-all add: d-def*)
qed

corollary *Hoeffding-ineq-le*:

assumes $\varepsilon: \varepsilon \geq 0$
assumes $(\sum i \in I. (b\ i - a\ i)^2) > 0$
shows *prob* $\{x \in \text{space } M. (\sum i \in I. X\ i\ x) \leq \mu - \varepsilon\} \leq \exp (-2 * \varepsilon^2 / (\sum i \in I. (b\ i - a\ i)^2))$
proof –
interpret *flip: Hoeffding-ineq* *M* *I* $\lambda i\ x. -X\ i\ x\ \lambda i. -b\ i\ \lambda i. -a\ i - \mu$
proof *unfold-locales*
fix *i* **assume** *i* $i \in I$

```

then interpret interval-bounded-random-variable  $M X i a i b i ..$ 
show  $AE x \text{ in } M. - X i x \in \{- b i .. - a i\}$ 
  by (intro eventually-mono[OF AE-in-interval]) auto
qed (auto simp: fin  $\mu$ -def sum-negf intro: indep-vars-compose2[OF indep])

have  $prob \{x \in space M. (\sum i \in I. X i x) \leq \mu - \varepsilon\} = prob \{x \in space M. (\sum i \in I. -X i x) \geq -\mu + \varepsilon\}$ 
  by (simp add: sum-negf algebra-simps)
also have  $\dots \leq exp (-2 * \varepsilon^2 / (\sum i \in I. (b i - a i)^2))$ 
  using flip.Hoeffding-ineq-ge[OF  $\varepsilon$ ] assms(2) by simp
finally show ?thesis .
qed

corollary Hoeffding-ineq-abs-ge:
  assumes  $\varepsilon: \varepsilon \geq 0$ 
  assumes  $(\sum i \in I. (b i - a i)^2) > 0$ 
  shows  $prob \{x \in space M. |(\sum i \in I. X i x) - \mu| \geq \varepsilon\} \leq 2 * exp (-2 * \varepsilon^2 / (\sum i \in I. (b i - a i)^2))$ 
proof -
  have  $\{x \in space M. |(\sum i \in I. X i x) - \mu| \geq \varepsilon\} = \{x \in space M. (\sum i \in I. X i x) \geq \mu + \varepsilon\} \cup \{x \in space M. (\sum i \in I. X i x) \leq \mu - \varepsilon\}$ 
  by auto
  also have  $prob \dots \leq prob \{x \in space M. (\sum i \in I. X i x) \geq \mu + \varepsilon\} + prob \{x \in space M. (\sum i \in I. X i x) \leq \mu - \varepsilon\}$ 
  by (intro measure-Un-le) auto
  also have  $\dots \leq exp (-2 * \varepsilon^2 / (\sum i \in I. (b i - a i)^2)) + exp (-2 * \varepsilon^2 / (\sum i \in I. (b i - a i)^2))$ 
  by (intro add-mono Hoeffding-ineq-ge Hoeffding-ineq-le assms)
  finally show ?thesis by simp
qed

end

```

27.3 Hoeffding’s inequality for i.i.d. bounded random variables

If we have n even identically-distributed random variables, the statement of Hoeffding’s lemma simplifies a bit more: it shows that the probability that the average of the X_i is more than ε above the expected value is no greater than $e^{\frac{-2n\varepsilon^2}{(b-a)^2}}$.

This essentially gives us a more concrete version of the weak law of large numbers: the law states that the probability vanishes for $n \rightarrow \infty$ for any $\varepsilon > 0$. Unlike Hoeffding’s inequality, it does not assume the variables to have bounded support, but it does not provide concrete bounds.

```

locale iid-interval-bounded-random-variables = prob-space +
  fixes  $I :: 'b \text{ set}$  and  $X :: 'b \Rightarrow 'a \Rightarrow real$  and  $Y :: 'a \Rightarrow real$ 

```

```

fixes  $a\ b :: \text{real}$ 
assumes  $\text{fin}: \text{finite } I$ 
assumes  $\text{indep}: \text{indep-vars } (\lambda\cdot. \text{borel})\ X\ I$ 
assumes  $\text{distr-X}: i \in I \implies \text{distr } M\ \text{borel } (X\ i) = \text{distr } M\ \text{borel } Y$ 
assumes  $\text{rv-Y } [\text{measurable}]: \text{random-variable } \text{borel } Y$ 
assumes  $\text{AE-in-interval}: \text{AE } x\ \text{in } M. Y\ x \in \{a..b\}$ 
begin

lemma  $\text{random-variable } [\text{measurable}]$ :
  assumes  $i: i \in I$ 
  shows  $\text{random-variable } \text{borel } (X\ i)$ 
  using  $i\ \text{indep}\ \text{unfolding}\ \text{indep-vars-def}\ \text{by}\ \text{blast}$ 

sublocale  $X: \text{indep-interval-bounded-random-variables } M\ I\ X\ \lambda\cdot. a\ \lambda\cdot. b$ 
proof
  fix  $i$  assume  $i: i \in I$ 
  have  $\text{AE } x\ \text{in } M. Y\ x \in \{a..b\}$ 
    by  $(\text{fact } \text{AE-in-interval})$ 
  also have  $?this \longleftrightarrow (\text{AE } x\ \text{in } \text{distr } M\ \text{borel } Y. x \in \{a..b\})$ 
    by  $(\text{subst } \text{AE-distr-iff})\ \text{auto}$ 
  also have  $\text{distr } M\ \text{borel } Y = \text{distr } M\ \text{borel } (X\ i)$ 
    using  $i\ \text{by}\ (\text{simp } \text{add}: \text{distr-X})$ 
  also have  $(\text{AE } x\ \text{in } \dots\ x \in \{a..b\}) \longleftrightarrow (\text{AE } x\ \text{in } M. X\ i\ x \in \{a..b\})$ 
    using  $i\ \text{by}\ (\text{subst } \text{AE-distr-iff})\ \text{auto}$ 
  finally show  $\text{AE } x\ \text{in } M. X\ i\ x \in \{a..b\} .$ 
qed  $(\text{simp-all } \text{add}: \text{fin } \text{indep})$ 

lemma  $\text{expectation-X } [\text{simp}]$ :
  assumes  $i: i \in I$ 
  shows  $\text{expectation } (X\ i) = \text{expectation } Y$ 
proof  $-$ 
  have  $\text{expectation } (X\ i) = \text{lebesgue-integral } (\text{distr } M\ \text{borel } (X\ i))\ (\lambda x. x)$ 
    using  $i\ \text{by}\ (\text{intro } \text{integral-distr } [\text{symmetric}])\ \text{auto}$ 
  also have  $\text{distr } M\ \text{borel } (X\ i) = \text{distr } M\ \text{borel } Y$ 
    using  $i\ \text{by}\ (\text{rule } \text{distr-X})$ 
  also have  $\text{lebesgue-integral } \dots\ (\lambda x. x) = \text{expectation } Y$ 
    by  $(\text{rule } \text{integral-distr})\ \text{auto}$ 
  finally show  $\text{expectation } (X\ i) = \text{expectation } Y .$ 
qed

end

locale  $\text{Hoeffding-ineq-iid} = \text{iid-interval-bounded-random-variables } +$ 
  fixes  $\mu :: \text{real}$ 
  defines  $\mu \equiv \text{expectation } Y$ 
begin

sublocale  $X: \text{Hoeffding-ineq } M\ I\ X\ \lambda\cdot. a\ \lambda\cdot. b\ \text{real } (\text{card } I) * \mu$ 

```

by *unfold-locales (simp-all add: μ -def)*

corollary

assumes $\varepsilon: \varepsilon \geq 0$

assumes $a < b \ I \neq \{\}$

defines $n \equiv \text{card } I$

shows *Hoeffding-ineq-ge*:

$$\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \geq n * \mu + \varepsilon\} \leq \exp(-2 * \varepsilon^2 / (n * (b - a)^2)) \text{ (is ?le)}$$

and *Hoeffding-ineq-le*:

$$\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \leq n * \mu - \varepsilon\} \leq \exp(-2 * \varepsilon^2 / (n * (b - a)^2)) \text{ (is ?ge)}$$

and *Hoeffding-ineq-abs-ge*:

$$\text{prob } \{x \in \text{space } M. |(\sum_{i \in I}. X \ i \ x) - n * \mu| \geq \varepsilon\} \leq 2 * \exp(-2 * \varepsilon^2 / (n * (b - a)^2)) \text{ (is ?abs-ge)}$$

proof –

have *pos*: $(\sum_{i \in I}. (b - a)^2) > 0$

using $\langle a < b \rangle \ \langle I \neq \{\} \rangle$ *fin* **by** (*intro sum-pos*) *auto*

show *?le*

using *X.Hoeffding-ineq-ge[OF ε pos]* **by** (*simp add: n-def*)

show *?ge*

using *X.Hoeffding-ineq-le[OF ε pos]* **by** (*simp add: n-def*)

show *?abs-ge*

using *X.Hoeffding-ineq-abs-ge[OF ε pos]* **by** (*simp add: n-def*)

qed

lemma

assumes $\varepsilon: \varepsilon \geq 0$

assumes $a < b \ I \neq \{\}$

defines $n \equiv \text{card } I$

shows *Hoeffding-ineq-ge'*:

$$\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) / n \geq \mu + \varepsilon\} \leq \exp(-2 * n * \varepsilon^2 / (b - a)^2) \text{ (is ?ge)}$$

and *Hoeffding-ineq-le'*:

$$\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) / n \leq \mu - \varepsilon\} \leq \exp(-2 * n * \varepsilon^2 / (b - a)^2) \text{ (is ?le)}$$

and *Hoeffding-ineq-abs-ge'*:

$$\text{prob } \{x \in \text{space } M. |(\sum_{i \in I}. X \ i \ x) / n - \mu| \geq \varepsilon\} \leq 2 * \exp(-2 * n * \varepsilon^2 / (b - a)^2) \text{ (is ?abs-ge)}$$

proof –

have $n > 0$

using *assms fin* **by** (*auto simp: field-simps*)

have $\varepsilon': \varepsilon * n \geq 0$

using $\langle n > 0 \rangle \ \langle \varepsilon \geq 0 \rangle$ **by** *auto*

have *eq*: $-(2 * (\varepsilon * \text{real } n)^2 / (\text{real } (\text{card } I) * (b - a)^2)) = - (2 * \text{real } n * \varepsilon^2 / (b - a)^2)$

using $\langle n > 0 \rangle$ **by** (*simp add: power2-eq-square divide-simps n-def*)

have $\{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) / n \geq \mu + \varepsilon\} =$

```

    {x∈space M. (∑ i∈I. X i x) ≥ μ * n + ε * n}
  using ⟨n > 0⟩ by (intro Collect-cong conj-cong refl) (auto simp: field-simps)
with Hoeffding-ineq-ge[OF ε' ⟨a < b⟩ ⟨I ≠ {}⟩] ⟨n > 0⟩ eq show ?ge
  by (simp add: n-def mult-ac)

have {x∈space M. (∑ i∈I. X i x) / n ≤ μ - ε} =
  {x∈space M. (∑ i∈I. X i x) ≤ μ * n - ε * n}
  using ⟨n > 0⟩ by (intro Collect-cong conj-cong refl) (auto simp: field-simps)
with Hoeffding-ineq-le[OF ε' ⟨a < b⟩ ⟨I ≠ {}⟩] ⟨n > 0⟩ eq show ?le
  by (simp add: n-def mult-ac)

have {x∈space M. |(∑ i∈I. X i x) / n - μ| ≥ ε} =
  {x∈space M. |(∑ i∈I. X i x) - μ * n| ≥ ε * n}
  using ⟨n > 0⟩ by (intro Collect-cong conj-cong refl) (auto simp: field-simps)
with Hoeffding-ineq-abs-ge[OF ε' ⟨a < b⟩ ⟨I ≠ {}⟩] ⟨n > 0⟩ eq show ?abs-ge
  by (simp add: n-def mult-ac)
qed

end

```

27.4 Hoeffding’s Inequality for the Binomial distribution

We can now apply Hoeffding’s inequality to the Binomial distribution, which can be seen as the sum of n i.i.d. coin flips (the support of each of which is contained in $[0, 1]$).

```

locale binomial-distribution =

```

```

  fixes n :: nat and p :: real

```

```

  assumes p: p ∈ {0..1}

```

```

begin

```

```

context

```

```

  fixes coins :: (nat ⇒ bool) pmf and μ

```

```

  assumes n: n > 0

```

```

  defines coins ≡ Pi-pmf {..<n} False (λ-. bernoulli-pmf p)

```

```

begin

```

```

lemma coins-component:

```

```

  assumes i: i < n

```

```

  shows distr (measure-pmf coins) borel (λf. if f i then 1 else 0) =

```

```

    distr (measure-pmf (bernoulli-pmf p)) borel (λb. if b then 1 else 0)

```

```

proof -

```

```

  have distr (measure-pmf coins) borel (λf. if f i then 1 else 0) =

```

```

    distr (measure-pmf (map-pmf (λf. f i) coins)) borel (λb. if b then 1 else 0)

```

```

  unfolding map-pmf-rep-eq by (subst distr-distr) (auto simp: o-def)

```

```

  also have map-pmf (λf. f i) coins = bernoulli-pmf p

```

```

  unfolding coins-def using i by (subst Pi-pmf-component) auto

```

```

  finally show ?thesis

```

```

  unfolding map-pmf-rep-eq .

```

```

qed

```

lemma *prob-binomial-pmf-conv-coins*:

*measure-pmf.prob (binomial-pmf n p) {x. P (real x)} =
measure-pmf.prob coins {x. P ($\sum i < n$. if x i then 1 else 0)}*

proof –

have *eq1*: ($\sum i < n$. if x i then 1 else 0) = real (card {i ∈ {.. n }. x i}) **for** x

proof –

have ($\sum i < n$. if x i then 1 else (0::real)) = ($\sum i \in \{i \in \{.. n \}. x i\}$. 1)

by (*intro sum.mono-neutral-cong-right*) *auto*

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

qed

have *eq2*: *binomial-pmf n p* = *map-pmf* (λv . card {i ∈ {.. n }. v i}) *coins*

unfolding *coins-def* **by** (*rule binomial-pmf-altdef'*) (*use p in auto*)

show *?thesis*

by (*subst eq2*) (*simp-all add: eq1*)

qed

interpretation *Hoeffding-ineq-iid*

coins {.. n } λi f. if f i then 1 else 0 λf . if f 0 then 1 else 0 0 1 p

proof *unfold-locales*

show *prob-space.indep-vars (measure-pmf coins) (λ -. borel) (λi f. if f i then 1 else 0) {.. n }*

unfolding *coins-def*

by (*intro prob-space.indep-vars-compose2[OF - indep-vars-Pi-pmf]*)
(*auto simp: measure-pmf.prob-space-axioms*)

next

have *measure-pmf.expectation coins* (λf . if f 0 then 1 else 0 :: real) =

measure-pmf.expectation (map-pmf (λf . f 0) coins) (λb . if b then 1 else 0 :: real)

by (*simp add: coins-def*)

also have *map-pmf* (λf . f 0) *coins* = *bernoulli-pmf p*

using n **by** (*simp add: coins-def Pi-pmf-component*)

also have *measure-pmf.expectation* ... (λb . if b then 1 else 0) = p

using p **by** *simp*

finally show p \equiv *measure-pmf.expectation coins* (λf . if f 0 then 1 else 0) **by** *simp*

qed (*auto simp: coins-component*)

corollary

fixes ε :: real

assumes ε : $\varepsilon \geq 0$

shows *prob-ge: measure-pmf.prob (binomial-pmf n p) {x. $x \geq n * p + \varepsilon$ } $\leq \exp$
($-2 * \varepsilon^2 / n$)*

and *prob-le: measure-pmf.prob (binomial-pmf n p) {x. $x \leq n * p - \varepsilon$ } $\leq \exp$
($-2 * \varepsilon^2 / n$)*

and *prob-abs-ge:*

*measure-pmf.prob (binomial-pmf n p) {x. $|x - n * p| \geq \varepsilon$ } $\leq 2 * \exp$ ($-2 * \varepsilon^2 / n$)*

proof –

```

have [simp]: {..n} ≠ {}
using n by auto
show measure-pmf.prob (binomial-pmf n p) {x. x ≥ n * p + ε} ≤ exp (-2 * ε2 / n)
using Hoeffding-ineq-ge[of ε] by (subst prob-binomial-pmf-conv-coins) (use
assms in simp-all)
show measure-pmf.prob (binomial-pmf n p) {x. x ≤ n * p - ε} ≤ exp (-2 * ε2 / n)
using Hoeffding-ineq-le[of ε] by (subst prob-binomial-pmf-conv-coins) (use
assms in simp-all)
show measure-pmf.prob (binomial-pmf n p) {x. |x - n * p| ≥ ε} ≤ 2 * exp (-2 * ε2 / n)
using Hoeffding-ineq-abs-ge[of ε]
by (subst prob-binomial-pmf-conv-coins) (use assms in simp-all)
qed

```

corollary

```

fixes ε :: real
assumes ε: ε ≥ 0
shows prob-ge': measure-pmf.prob (binomial-pmf n p) {x. x / n ≥ p + ε} ≤ exp (-2 * n * ε2)
and prob-le': measure-pmf.prob (binomial-pmf n p) {x. x / n ≤ p - ε} ≤ exp (-2 * n * ε2)
and prob-abs-ge':
measure-pmf.prob (binomial-pmf n p) {x. |x / n - p| ≥ ε} ≤ 2 * exp (-2 * n * ε2)

```

proof –

```

have [simp]: {..n} ≠ {}
using n by auto
show measure-pmf.prob (binomial-pmf n p) {x. x / n ≥ p + ε} ≤ exp (-2 * n * ε2)
using Hoeffding-ineq-ge'[of ε] by (subst prob-binomial-pmf-conv-coins) (use
assms in simp-all)
show measure-pmf.prob (binomial-pmf n p) {x. x / n ≤ p - ε} ≤ exp (-2 * n * ε2)
using Hoeffding-ineq-le'[of ε] by (subst prob-binomial-pmf-conv-coins) (use
assms in simp-all)
show measure-pmf.prob (binomial-pmf n p) {x. |x / n - p| ≥ ε} ≤ 2 * exp (-2 * n * ε2)
using Hoeffding-ineq-abs-ge'[of ε]
by (subst prob-binomial-pmf-conv-coins) (use assms in simp-all)

```

qed

end

end

27.5 Tail bounds for the negative binomial distribution

Since the tail probabilities of a negative Binomial distribution are equal to the tail probabilities of some Binomial distribution, we can obtain tail bounds for the negative Binomial distribution through the Hoeffding tail bounds for the Binomial distribution.

context

fixes $p\ q :: \text{real}$
assumes $p: p \in \{0 < .. < 1\}$
defines $q \equiv 1 - p$

begin

lemma *prob-neg-binomial-pmf-ge-bound:*

fixes $n :: \text{nat}$ **and** $k :: \text{real}$
defines $\mu \equiv \text{real } n * q / p$
assumes $k: k \geq 0$
shows $\text{measure-pmf.prob (neg-binomial-pmf } n\ p) \{x. \text{real } x \geq \mu + k\}$
 $\leq \exp (- 2 * p \wedge 3 * k^2 / (n + p * k))$

proof –

consider $n = 0 \mid p = 1 \mid n > 0\ p \neq 1$

by *blast*

thus *?thesis*

proof *cases*

assume [*simp*]: $n = 0$

show *?thesis* **using** k

by (*simp add: indicator-def* μ -*def*)

next

assume [*simp*]: $p = 1$

show *?thesis* **using** k

by (*auto simp add: indicator-def* μ -*def* q -*def*)

next

assume $n: n > 0$ **and** $p \neq 1$

from $\langle p \neq 1 \rangle$ **and** p **have** $p: p \in \{0 < .. < 1\}$

by *auto*

from p **have** $q: q \in \{0 < .. < 1\}$

by (*auto simp: q-def*)

define $k1$ **where** $k1 = \mu + k$

have $k1: k1 \geq \mu$

using k **by** (*simp add: k1-def*)

have $k1 > 0$

by (*rule less-le-trans*[$OF - k1$]) (*use p n in* $\langle \text{auto simp: } q\text{-def } \mu\text{-def} \rangle$)

define $k1'$ **where** $k1' = \text{nat (ceiling } k1)$

have $\mu \geq 0$ **using** p

by (*auto simp:* μ -*def* q -*def*)

have $\neg(x < k1') \longleftrightarrow \text{real } x \geq k1$ **for** x

unfolding $k1'$ -*def* **by** *linarith*

hence *eq: UNIV -* $\{.. < k1'\} = \{x. x \geq k1\}$

by *auto*
 hence $\text{measure-pmf.prob (neg-binomial-pmf } n \text{ } p) \{n. n \geq k1\} =$
 $1 - \text{measure-pmf.prob (neg-binomial-pmf } n \text{ } p) \{..<k1\}$
 using $\text{measure-pmf.prob-compl[of } \{..<k1\} \text{ neg-binomial-pmf } n \text{ } p]$ by *simp*
 also have $\text{measure-pmf.prob (neg-binomial-pmf } n \text{ } p) \{..<k1\} =$
 $\text{measure-pmf.prob (binomial-pmf (} n + k1' - 1) \text{ } q) \{..<k1\}$
 unfolding *q-def* using *p* by (*intro prob-neg-binomial-pmf-lessThan*) *auto*
 also have $1 - \dots = \text{measure-pmf.prob (binomial-pmf (} n + k1' - 1) \text{ } q) \{n.$
 $n \geq k1\}$
 using $\text{measure-pmf.prob-compl[of } \{..<k1\} \text{ binomial-pmf (} n + k1' - 1) \text{ } q]$
eq by *simp*
 also have $\{x. \text{real } x \geq k1\} = \{x. x \geq \text{real (} n + k1' - 1) * q + (k1 - \text{real (} n$
 $+ k1' - 1) * q)\}$
 by *simp*
 also have $\text{measure-pmf.prob (binomial-pmf (} n + k1' - 1) \text{ } q) \dots \leq$
 $\text{exp } (-2 * (k1 - \text{real (} n + k1' - 1) * q)^2 / \text{real (} n + k1' - 1))$
proof (*rule binomial-distribution.prob-ge*)
 show *binomial-distribution q*
 by *unfold-locales (use q in auto)*
next
 show $n + k1' - 1 > 0$
 using $\langle k1 > 0 \rangle n$ *unfolding k1'-def* by *linarith*
next
 have $\text{real (} n + \text{nat } \lceil k1 \rceil - 1) \leq \text{real } n + k1$
 using $\langle k1 > 0 \rangle$ by *linarith*
 hence $\text{real (} n + k1' - 1) * q \leq (\text{real } n + k1) * q$
 unfolding *k1'-def* by (*intro mult-right-mono*) (*use p in <simp-all add:*
q-def>)
 also have $\dots \leq k1$
 using *k1 p* by (*simp add: q-def field-simps μ-def*)
 finally show $0 \leq k1 - \text{real (} n + k1' - 1) * q$
 by *simp*
qed
 also have $\{x. \text{real (} n + k1' - 1) * q + (k1 - \text{real (} n + k1' - 1) * q) \leq \text{real}$
 $x\} = \{x. \text{real } x \geq k1\}$
 by *simp*
 also have $\text{exp } (-2 * (k1 - \text{real (} n + k1' - 1) * q)^2 / \text{real (} n + k1' - 1)) \leq$
 $\text{exp } (-2 * (k1 - (n + k1) * q)^2 / (n + k1))$
proof –
 have $\text{real (} n + k1' - \text{Suc } 0) \leq \text{real } n + k1$
 unfolding *k1'-def* using $\langle k1 > 0 \rangle$ by *linarith*
 moreover have $(\text{real } n + k1) * q \leq k1$
 using *k1 p* by (*auto simp: q-def field-simps μ-def*)
 moreover have $1 < n + k1'$
 using $n \langle k1 > 0 \rangle$ *unfolding k1'-def* by *linarith*
 ultimately have $2 * (k1 - \text{real (} n + k1' - 1) * q)^2 / \text{real (} n + k1' - 1) \geq$
 $2 * (k1 - (n + k1) * q)^2 / (n + k1)$
 by (*intro frac-le mult-left-mono power-mono mult-nonneg-nonneg mult-right-mono*
diff-mono)

```

      (use q in simp-all)
    thus ?thesis
      by simp
  qed
  also have ... = exp (-2 * (p * k1 - q * n)2 / (k1 + n))
    by (simp add: q-def algebra-simps)
  also have -2 * (p * k1 - q * n)2 = -2 * p2 * (k1 - μ)2
    using p by (auto simp: field-simps μ-def)
  also have k1 - μ = k
    by (simp add: k1-def μ-def)
  also note k1-def
  also have μ + k + real n = real n / p + k
    using p by (simp add: μ-def q-def field-simps)
  also have -2 * p2 * k2 / (real n / p + k) = -2 * p3 * k2 / (p * k + n)
    using p by (simp add: field-simps power3-eq-cube power2-eq-square)
  finally show ?thesis by (simp add: add-ac)
  qed
  qed

```

lemma *prob-neg-binomial-pmf-le-bound*:

```

  fixes n :: nat and k :: real
  defines μ ≡ real n * q / p
  assumes k: k ≥ 0
  shows measure-pmf.prob (neg-binomial-pmf n p) {x. real x ≤ μ - k}
    ≤ exp (-2 * p3 * k2 / (n - p * k))

```

proof –

```

  consider n = 0 | p = 1 | k > μ | n > 0 p ≠ 1 k ≤ μ

```

```

    by force

```

```

  thus ?thesis

```

```

  proof cases

```

```

    assume [simp]: n = 0

```

```

    show ?thesis using k

```

```

      by (simp add: indicator-def μ-def)

```

```

  next

```

```

    assume [simp]: p = 1

```

```

    show ?thesis using k

```

```

      by (auto simp add: indicator-def μ-def q-def)

```

```

  next

```

```

    assume k > μ

```

```

    hence {x. real x ≤ μ - k} = {}

```

```

      by auto

```

```

    thus ?thesis by simp

```

```

  next

```

```

    assume n: n > 0 and p ≠ 1 and k ≤ μ

```

```

    from ⟨p ≠ 1⟩ and p have p: p ∈ {0 < .. < 1}

```

```

      by auto

```

```

    from p have q: q ∈ {0 < .. < 1}

```

```

      by (auto simp: q-def)

```

```

define f :: real ⇒ real where f = (λx. (p * x - q * n)2 / (x + n))
have f-mono: f x ≥ f y if x ≥ 0 y ≤ n * q / p x ≤ y for x y :: real
  using that(3)
proof (rule DERIV-nonpos-imp-nonincreasing)
  fix t assume t: t ≥ x t ≤ y
  have x > -n
    using n ⟨x ≥ 0⟩ by linarith
  hence (f has-field-derivative ((p * t - q * n) * (n * (1 + p) + p * t) / (n +
t) ^ 2)) (at t)
    unfolding f-def using t
  by (auto intro!: derivative-eq-intros simp: algebra-simps q-def power2-eq-square)
  moreover {
    have p * t ≤ p * y
      using p by (intro mult-left-mono t) auto
    also have p * y ≤ q * n
      using ⟨y ≤ n * q / p⟩ p by (simp add: field-simps)
    finally have p * t ≤ q * n .
  }
  hence (p * t - q * n) * (n * (1 + p) + p * t) / (n + t) ^ 2 ≤ 0
    using p ⟨x ≥ 0⟩ t
    by (intro mult-nonpos-nonneg divide-nonpos-nonneg add-nonneg-nonneg
mult-nonneg-nonneg) auto
  ultimately show ∃ y. (f has-real-derivative y) (at t) ∧ y ≤ 0
    by blast
qed

```

```

define k1 where k1 = μ - k
have k1: k1 ≤ real n * q / p
  using assms by (simp add: μ-def k1-def)
have k1 ≥ 0
  using k ⟨k ≤ μ⟩ by (simp add: μ-def k1-def)

```

```

define k1' where k1' = nat (floor k1)
have μ ≥ 0 using p
  by (auto simp: μ-def q-def)
have (x ≤ k1') ↔ real x ≤ k1 for x
  unfolding k1'-def not-less using ⟨k1 ≥ 0⟩ by linarith
hence eq: {n. n ≤ k1} = {..k1'}
  by auto
hence measure-pmf.prob (neg-binomial-pmf n p) {n. n ≤ k1} =
  measure-pmf.prob (neg-binomial-pmf n p) {..k1'}
  by simp
also have measure-pmf.prob (neg-binomial-pmf n p) {..k1'} =
  measure-pmf.prob (binomial-pmf (n + k1') q) {..k1'}
  unfolding q-def using p by (intro prob-neg-binomial-pmf-atMost) auto
also note eq [symmetric]
also have {x. real x ≤ k1} = {x. x ≤ real (n + k1') * q - (real (n + k1') *
q - real k1')}
  using eq by auto

```

```

also have measure-pmf.probab (binomial-pmf ( $n + k1'$ )  $q$ ) ... ≤
       $\exp(-2 * (\text{real } (n + k1') * q - \text{real } k1')^2 / \text{real } (n + k1'))$ 
proof (rule binomial-distribution.probab-le)
  show binomial-distribution q
    by unfold-locales (use q in auto)
next
  show  $n + k1' > 0$ 
    using  $\langle k1 \geq 0 \rangle n$  unfolding k1'-def by linarith
next
  have  $p * k1' \leq p * k1$ 
    using  $p \langle k1 \geq 0 \rangle$  by (intro mult-left-mono) (auto simp: k1'-def)
  also have ... ≤  $q * n$ 
    using k1 p by (simp add: field-simps)
  finally show  $0 \leq \text{real } (n + k1') * q - \text{real } k1'$ 
    by (simp add: algebra-simps q-def)
qed
also have  $\{x. \text{real } x \leq \text{real } (n + k1') * q - (\text{real } (n + k1') * q - k1')\} =$ 
 $\{..k1'\}$ 
  by auto
also have  $\text{real } (n + k1') * q - k1' = -(p * k1' - q * n)$ 
  by (simp add: q-def algebra-simps)
also have ...  $^2 = (p * k1' - q * n)^2$ 
  by algebra
also have  $-2 * (p * \text{real } k1' - q * \text{real } n)^2 / \text{real } (n + k1') = -2 * f (\text{real } k1')$ 
  by (simp add: f-def)
also have  $f (\text{real } k1') \geq f k1$ 
  by (rule f-mono) (use \langle k1 \geq 0 \rangle k1 in \langle auto simp: k1'-def \rangle)
hence  $\exp(-2 * f (\text{real } k1')) \leq \exp(-2 * f k1)$ 
  by simp
also have ... =  $\exp(-2 * (p * k1 - q * n)^2 / (k1 + n))$ 
  by (simp add: f-def)

also have  $-2 * (p * k1 - q * n)^2 = -2 * p^2 * (k1 - \mu)^2$ 
  using  $p$  by (auto simp: field-simps \mu-def)
also have  $(k1 - \mu)^2 = k^2$ 
  by (simp add: k1-def \mu-def)
also note k1-def
also have  $\mu - k + \text{real } n = \text{real } n / p - k$ 
  using  $p$  by (simp add: \mu-def q-def field-simps)
also have  $-2 * p^2 * k^2 / (\text{real } n / p - k) = -2 * p^3 * k^2 / (n - p * k)$ 
  using  $p$  by (simp add: field-simps power3-eq-cube power2-eq-square)
also have  $\{..k1'\} = \{x. \text{real } x \leq \mu - k\}$ 
  using eq by (simp add: k1-def)
finally show ?thesis .
qed
qed

```

Due to the function $\exp(-l/x)$ being concave for $x \geq \frac{l}{2}$, the above two

bounds can be combined into the following one for moderate values of k .
(cf. <https://math.stackexchange.com/questions/1565559>)

lemma *prob-neg-binomial-pmf-abs-ge-bound*:

fixes $n :: \text{nat}$ **and** $k :: \text{real}$

defines $\mu \equiv \text{real } n * q / p$

assumes $k \geq 0$ **and** $n\text{-ge}$: $n \geq p * k * (p^2 * k + 1)$

shows $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. |\text{real } x - \mu| \geq k\} \leq$
 $2 * \exp (-2 * p \wedge 3 * k \wedge 2 / n)$

proof (*cases* $k = 0$)

case *False*

with $\langle k \geq 0 \rangle$ **have** k : $k > 0$

by *auto*

define $l :: \text{real}$ **where** $l = 2 * p \wedge 3 * k \wedge 2$

have l : $l > 0$

using $p \ k$ **by** (*auto simp: l-def*)

define $f :: \text{real} \Rightarrow \text{real}$ **where** $f = (\lambda x. \exp (-l / x))$

define f' **where** $f' = (\lambda x. -l * \exp (-l / x) / x \wedge 2)$

have $f'\text{-mono}$: $f' \ x \leq f' \ y$ **if** $x \geq l / 2 \ x \leq y$ **for** $x \ y :: \text{real}$

using *that(2)*

proof (*rule DERIV-nonneg-imp-nondecreasing*)

fix t **assume** t : $x \leq t \ t \leq y$

have $t > 0$

using *that l t by auto*

have (f' *has-field-derivative* $(l * (2 * t - l) / (\exp (l / t) * t \wedge 4)))$ (*at* t)

unfolding $f'\text{-def}$ **using** t *that* $\langle t > 0 \rangle$

by (*auto intro!: derivative-eq-intros simp: field-simps exp-minus simp flip: power-Suc*)

moreover **have** $l * (2 * t - l) / (\exp (l / t) * t \wedge 4) \geq 0$

using *that t l by (intro divide-nonneg-pos mult-nonneg-nonneg) auto*

ultimately show $\exists y. (f'$ *has-real-derivative* $y)$ (*at* t) $\wedge 0 \leq y$ **by** *blast*

qed

have *convex*: *convex-on* $\{l/2..\}$ $(\lambda x. -f \ x)$ **unfolding** $f\text{-def}$

proof (*intro convex-on-realI[where $f' = f'$]*)

show $((\lambda x. - \exp (-l / x))$ *has-real-derivative* $f' \ x)$ (*at* x) **if** $x \in \{l/2..\}$ **for** x

using *that l*

by (*auto intro!: derivative-eq-intros simp: f'-def power2-eq-square algebra-simps*)

qed (*use l in <auto intro!: f'-mono>*)

have *eq*: $\{x. |\text{real } x - \mu| \geq k\} = \{x. \text{real } x \leq \mu - k\} \cup \{x. \text{real } x \geq \mu + k\}$

by *auto*

have $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. |\text{real } x - \mu| \geq k\} \leq$

$\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. \text{real } x \leq \mu - k\} +$

$\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. \text{real } x \geq \mu + k\}$

by (*subst eq, rule measure-Un-le*) *auto*

also have $\dots \leq \exp (-2 * p \wedge 3 * k^2 / (n - p * k)) + \exp (-2 * p \wedge 3 * k^2 /$
 $(n + p * k))$

unfolding $\mu\text{-def}$

```

  by (intro prob-neg-binomial-pmf-le-bound prob-neg-binomial-pmf-ge-bound add-mono
    ⟨k ≥ 0⟩)
  also have ... = 2 * (1/2 * f (n - p * k) + 1/2 * f (n + p * k))
    by (simp add: f-def l-def)
  also have 1/2 * f (n - p * k) + 1/2 * f (n + p * k) ≤ f (1/2 * (n - p * k)
    + 1/2 * (n + p * k))
  proof -
    let ?x = n - p * k and ?y = n + p * k
    have le1: l / 2 ≤ ?x using n-ge
      by (simp add: l-def power2-eq-square power3-eq-cube algebra-simps)
    also have ... ≤ ?y
      using p k by simp
    finally have le2: l / 2 ≤ ?y .
    have -f ((1 - 1 / 2) *R ?x + (1 / 2) *R ?y) ≤ (1 - 1 / 2) * -f ?x + 1 /
    2 * -f ?y
      using le1 le2 by (intro convex-onD[OF convex]) auto
    thus ?thesis by simp
  qed
  also have 1/2 * (n - p * k) + 1/2 * (n + p * k) = n
    by (simp add: algebra-simps)
  also have 2 * f n = 2 * exp (-l / n)
    by (simp add: f-def)
  finally show ?thesis
    by (simp add: l-def)
qed auto

end

end

```

theory *Stream-Space*

imports

Infinite-Product-Measure

HOL-Library.Stream

HOL-Library.Linear-Temporal-Logic-on-Streams

begin

lemma *stream-eq-Stream-iff*: $s = x \#\# t \longleftrightarrow (shd\ s = x \wedge stl\ s = t)$
 by (cases s) simp

lemma *Stream-snth*: $(x \#\# s) !! n = (case\ n\ of\ 0 \Rightarrow x \mid Suc\ n \Rightarrow s !! n)$
 by (cases n) simp-all

definition *to-stream* :: $(nat \Rightarrow 'a) \Rightarrow 'a\ stream$ **where**
to-stream X = smap X nats

lemma *to-stream-nat-case*: $to-stream\ (case-nat\ x\ X) = x \#\# to-stream\ X$
 unfolding *to-stream-def*

by (*subst siterate.ctr*) (*simp add: smap-siterate[symmetric] stream.map-comp comp-def*)

lemma *to-stream-in-streams*: $to\text{-}stream\ X \in streams\ S \longleftrightarrow (\forall n. X\ n \in S)$
by (*simp add: to-stream-def streams-iff-snth*)

definition *stream-space* :: 'a measure \Rightarrow 'a stream measure **where**
stream-space $M =$
distr ($\prod_M i \in UNIV. M$) (*vimage-algebra* (*streams* (*space* M))) *snth* ($\prod_M i \in UNIV. M$) *to-stream*

lemma *space-stream-space*: $space\ (stream\text{-}space\ M) = streams\ (space\ M)$
by (*simp add: stream-space-def*)

lemma *streams-stream-space[intro]*: $streams\ (space\ M) \in sets\ (stream\text{-}space\ M)$
using *sets.top[of stream-space M]* **by** (*simp add: space-stream-space*)

lemma *stream-space-Stream*:
 $x\ \#\#\ \omega \in space\ (stream\text{-}space\ M) \longleftrightarrow x \in space\ M \wedge \omega \in space\ (stream\text{-}space\ M)$
by (*simp add: space-stream-space streams-Stream*)

lemma *stream-space-eq-distr*: $stream\text{-}space\ M = distr\ (\prod_M i \in UNIV. M)\ (stream\text{-}space\ M)\ to-stream$
unfolding *stream-space-def* **by** (*rule distr-cong*) *auto*

lemma *sets-stream-space-cong[measurable-cong]*:
 $sets\ M = sets\ N \implies sets\ (stream\text{-}space\ M) = sets\ (stream\text{-}space\ N)$
using *sets-eq-imp-space-eq[of M N]* **by** (*simp add: stream-space-def vimage-algebra-def cong: sets-PiM-cong*)

lemma *measurable-snth-PiM*: $(\lambda\omega\ n. \omega\ !!\ n) \in measurable\ (stream\text{-}space\ M)\ (\prod_M i \in UNIV. M)$
by (*auto intro!: measurable-vimage-algebra1*)
simp: space-PiM streams-iff-sset sset-range image-subset-iff stream-space-def)

lemma *measurable-snth[measurable]*: $(\lambda\omega. \omega\ !!\ n) \in measurable\ (stream\text{-}space\ M)\ M$
using *measurable-snth-PiM measurable-component-singleton* **by** (*rule measurable-compose*) *simp*

lemma *measurable-shd[measurable]*: $shd \in measurable\ (stream\text{-}space\ M)\ M$
using *measurable-snth[of 0]* **by** *simp*

lemma *measurable-stream-space2*:
assumes *f-snth*: $\bigwedge n. (\lambda x. f\ x\ !!\ n) \in measurable\ N\ M$
shows $f \in measurable\ N\ (stream\text{-}space\ M)$
unfolding *stream-space-def measurable-distr-eq2*
proof (*rule measurable-vimage-algebra2*)

```

show  $f \in \text{space } N \rightarrow \text{streams } (\text{space } M)$ 
  using  $f\text{-snth}[THEN \text{measurable-space}]$  by  $(\text{auto simp add: streams-iff-sset sset-range})$ 
show  $(\lambda x. (!!)(f x)) \in \text{measurable } N (Pi_M UNIV (\lambda i. M))$ 
proof  $(\text{rule measurable-PiM-single}')$ 
  show  $(\lambda x. (!!)(f x)) \in \text{space } N \rightarrow UNIV \rightarrow_E \text{space } M$ 
  using  $f\text{-snth}[THEN \text{measurable-space}]$  by  $\text{auto}$ 
qed  $(\text{rule } f\text{-snth})$ 
qed

```

lemma $\text{measurable-stream-coinduct}[\text{consumes } 1, \text{case-names shd stl, coinduct set: measurable}]$:

```

assumes  $F f$ 
assumes  $h: \bigwedge f. F f \implies (\lambda x. \text{shd } (f x)) \in \text{measurable } N M$ 
assumes  $t: \bigwedge f. F f \implies F (\lambda x. \text{stl } (f x))$ 
shows  $f \in \text{measurable } N (\text{stream-space } M)$ 
proof  $(\text{rule measurable-stream-space2})$ 
  fix  $n$  show  $(\lambda x. f x !! n) \in \text{measurable } N M$ 
  using  $\langle F f \rangle$  by  $(\text{induction } n \text{ arbitrary: } f) (\text{auto intro: } h t)$ 
qed

```

lemma $\text{measurable-sdrop}[\text{measurable}]$: $\text{sdrop } n \in \text{measurable } (\text{stream-space } M) (\text{stream-space } M)$

by $(\text{rule measurable-stream-space2}) (\text{simp add: sdrop-snth})$

lemma $\text{measurable-stl}[\text{measurable}]$: $(\lambda \omega. \text{stl } \omega) \in \text{measurable } (\text{stream-space } M) (\text{stream-space } M)$

by $(\text{rule measurable-stream-space2}) (\text{simp del: snth.simps add: snth.simps[symmetric]})$

lemma $\text{measurable-to-stream}[\text{measurable}]$: $\text{to-stream} \in \text{measurable } (\prod_M i \in UNIV. M) (\text{stream-space } M)$

by $(\text{rule measurable-stream-space2}) (\text{simp add: to-stream-def})$

lemma $\text{measurable-Stream}[\text{measurable } (raw)]$:

```

assumes  $f[\text{measurable}]$ :  $f \in \text{measurable } N M$ 
assumes  $g[\text{measurable}]$ :  $g \in \text{measurable } N (\text{stream-space } M)$ 
shows  $(\lambda x. f x \#\#\ g x) \in \text{measurable } N (\text{stream-space } M)$ 
by  $(\text{rule measurable-stream-space2}) (\text{simp add: Stream-snth})$ 

```

lemma $\text{measurable-smap}[\text{measurable}]$:

```

assumes  $X[\text{measurable}]$ :  $X \in \text{measurable } N M$ 
shows  $\text{smap } X \in \text{measurable } (\text{stream-space } N) (\text{stream-space } M)$ 
by  $(\text{rule measurable-stream-space2}) \text{ simp}$ 

```

lemma $\text{measurable-stake}[\text{measurable}]$:

$\text{stake } i \in \text{measurable } (\text{stream-space } (\text{count-space } UNIV)) (\text{count-space } (UNIV :: 'a::\text{countable list set}))$

by $(\text{induct } i) \text{ auto}$

lemma $\text{measurable-shift}[\text{measurable}]$:

assumes $f: f \in \text{measurable } N \text{ (stream-space } M)$
assumes $[\text{measurable}]$: $g \in \text{measurable } N \text{ (stream-space } M)$
shows $(\lambda x. \text{stake } n \text{ (} f x) \text{ @- } g x) \in \text{measurable } N \text{ (stream-space } M)$
using f **by** $(\text{induction } n \text{ arbitrary: } f) \text{ simp-all}$

lemma *measurable-case-stream-replace* $[\text{measurable (raw)}]$:
 $(\lambda x. f x (\text{shd } (g x)) (\text{stl } (g x))) \in \text{measurable } M N \implies (\lambda x. \text{case-stream } (f x) (g x)) \in \text{measurable } M N$
unfolding *stream.case-eq-if* .

lemma *measurable-ev-at* $[\text{measurable}]$:
assumes $[\text{measurable}]$: *Measurable.pred (stream-space M) P*
shows *Measurable.pred (stream-space M) (ev-at P n)*
by $(\text{induction } n) \text{ auto}$

lemma *measurable-alw* $[\text{measurable}]$:
 $\text{Measurable.pred (stream-space } M) P \implies \text{Measurable.pred (stream-space } M) (\text{alw } P)$
unfolding *alw-def*
by $(\text{coinduction rule: measurable-gfp-coinduct}) (\text{auto simp: inf-continuous-def})$

lemma *measurable-ev* $[\text{measurable}]$:
 $\text{Measurable.pred (stream-space } M) P \implies \text{Measurable.pred (stream-space } M) (\text{ev } P)$
unfolding *ev-def*
by $(\text{coinduction rule: measurable-lfp-coinduct}) (\text{auto simp: sup-continuous-def})$

lemma *measurable-until*:
assumes $[\text{measurable}]$: *Measurable.pred (stream-space M) φ Measurable.pred (stream-space M) ψ*
shows *Measurable.pred (stream-space M) (φ until ψ)*
unfolding *UNTIL-def*
by $(\text{coinduction rule: measurable-gfp-coinduct}) (\text{simp-all add: inf-continuous-def fun-eq-iff})$

lemma *measurable-holds* $[\text{measurable}]$: $\text{Measurable.pred } M P \implies \text{Measurable.pred (stream-space } M) (\text{holds } P)$
unfolding *holds.simps[abs-def]*
by $(\text{rule measurable-compose[OF measurable-shd]}) \text{ simp}$

lemma *measurable-hld* $[\text{measurable}]$: **assumes** $[\text{measurable}]$: $t \in \text{sets } M$ **shows** *Measurable.pred (stream-space M) (HLD t)*
unfolding *HLD-def* **by** *measurable*

lemma *measurable-nxt* $[\text{measurable (raw)}]$:
 $\text{Measurable.pred (stream-space } M) P \implies \text{Measurable.pred (stream-space } M) (\text{nxt } P)$
unfolding *nxt.simps[abs-def]* **by** *simp*

lemma *measurable-suntil*[*measurable*]:

assumes [*measurable*]: *Measurable.pred (stream-space M) Q Measurable.pred (stream-space M) P*
shows *Measurable.pred (stream-space M) (Q suntil P)*
unfolding *suntil-def* **by** (*coinduction rule: measurable-lfp-coinduct*) (*auto simp: sup-continuous-def*)

lemma *measurable-szip*:

$(\lambda(\omega 1, \omega 2). \text{ szip } \omega 1 \ \omega 2) \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N)$
 $(\text{ stream-space } (M \otimes_M N))$
proof (*rule measurable-stream-space2*)
fix *n*
have $(\lambda x. (\text{ case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{ szip } \omega 1 \ \omega 2) !! n) = (\lambda(\omega 1, \omega 2). (\omega 1 !! n, \omega 2 !! n))$
by *auto*
also have $\dots \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N) (M \otimes_M N)$
by *measurable*
finally show $(\lambda x. (\text{ case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{ szip } \omega 1 \ \omega 2) !! n) \in \text{ measurable } (\text{ stream-space } M \otimes_M \text{ stream-space } N) (M \otimes_M N)$

qed

lemma (*in prob-space*) *prob-space-stream-space: prob-space (stream-space M)*

proof –

interpret *product-prob-space* $\lambda\cdot. M \text{ UNIV } ..$

show *?thesis*

by (*subst stream-space-eq-distr*) (*auto intro!: P.prob-space-distr*)

qed

lemma (*in prob-space*) *nn-integral-stream-space:*

assumes [*measurable*]: $f \in \text{ borel-measurable } (\text{ stream-space } M)$

shows $(\int^+ X. f \ X \ \partial \text{ stream-space } M) = (\int^+ x. (\int^+ X. f \ (x \ \#\# \ X) \ \partial \text{ stream-space } M) \ \partial M)$

proof –

interpret *S: sequence-space M ..*

interpret *P: pair-sigma-finite M $\Pi_M i::\text{nat} \in \text{UNIV}. M ..$*

have $(\int^+ X. f \ X \ \partial \text{ stream-space } M) = (\int^+ X. f \ (\text{ to-stream } X) \ \partial S.S)$

by (*subst stream-space-eq-distr*) (*simp add: nn-integral-distr*)

also have $\dots = (\int^+ X. f \ (\text{ to-stream } ((\lambda(s, \omega). \text{ case-nat } s \ \omega) \ X)) \ \partial(M \otimes_M S.S))$

by (*subst S.PiM-iter[symmetric]*) (*simp add: nn-integral-distr*)

also have $\dots = (\int^+ x. \int^+ X. f \ (\text{ to-stream } ((\lambda(s, \omega). \text{ case-nat } s \ \omega) \ (x, X))) \ \partial S.S \ \partial M)$

by (*subst S.nn-integral-fst*) *simp-all*

also have $\dots = (\int^+ x. \int^+ X. f \ (x \ \#\# \ \text{ to-stream } X) \ \partial S.S \ \partial M)$

by (*auto intro!: nn-integral-cong simp: to-stream-nat-case*)

also have $\dots = (\int^+ x. \int^+ X. f \ (x \ \#\# \ X) \ \partial \text{ stream-space } M \ \partial M)$

by (*subst stream-space-eq-distr*)

(*simp add: nn-integral-distr cong: nn-integral-cong*)
finally show ?thesis .
qed

lemma (in *prob-space*) *emeasure-stream-space*:
assumes $X[\text{measurable}]$: $X \in \text{sets } (\text{stream-space } M)$
shows $\text{emeasure } (\text{stream-space } M) X = (\int^+ t. \text{emeasure } (\text{stream-space } M) \{x \in \text{space } (\text{stream-space } M). t \#\# x \in X\} \partial M)$
proof –
have eq : $\bigwedge x \text{ xs}. \text{xs} \in \text{space } (\text{stream-space } M) \implies x \in \text{space } M \implies$
 $\text{indicator } X (x \#\# \text{xs}) = \text{indicator } \{x \in \text{space } (\text{stream-space } M). x \#\# \text{xs} \in X\} \text{xs}$
by (*auto split: split-indicator*)
show ?thesis
using *nn-integral-stream-space*[of *indicator X*]
apply (*auto intro!: nn-integral-cong*)
apply (*subst nn-integral-cong*)
apply (*rule eq*)
apply *simp-all*
done
qed

lemma (in *prob-space*) *prob-stream-space*:
assumes $P[\text{measurable}]$: $\{x \in \text{space } (\text{stream-space } M). P x\} \in \text{sets } (\text{stream-space } M)$
shows $\mathcal{P}(x \text{ in } \text{stream-space } M. P x) = (\int^+ t. \mathcal{P}(x \text{ in } \text{stream-space } M. P (t \#\# x)) \partial M)$
proof –
interpret S : *prob-space stream-space M*
by (*rule prob-space-stream-space*)
show ?thesis
unfolding $S.\text{emeasure-eq-measure}$ [*symmetric*]
by (*subst emeasure-stream-space*) (*auto simp: stream-space-Stream intro!: nn-integral-cong*)
qed

lemma (in *prob-space*) *AE-stream-space*:
assumes $[\text{measurable}]$: $\text{Measurable.pred } (\text{stream-space } M) P$
shows $(\text{AE } X \text{ in } \text{stream-space } M. P X) = (\text{AE } x \text{ in } M. \text{AE } X \text{ in } \text{stream-space } M. P (x \#\# X))$
proof –
interpret stream : *prob-space stream-space M*
by (*rule prob-space-stream-space*)

have eq : $\bigwedge x X. \text{indicator } \{x. \neg P x\} (x \#\# X) = \text{indicator } \{X. \neg P (x \#\# X)\} X$
by (*auto split: split-indicator*)
show ?thesis
apply (*subst AE-iff-nn-integral, simp*)
apply (*subst nn-integral-stream-space, simp*)

```

  apply (subst eq)
  apply (subst nn-integral-0-iff-AE, simp)
  apply (simp add: AE-iff-nn-integral[symmetric])
  done
qed

```

```

lemma (in prob-space) AE-stream-all:
  assumes [measurable]: Measurable.pred M P and P: AE x in M. P x
  shows AE x in stream-space M. stream-all P x
proof -
  { fix n have AE x in stream-space M. P (x !! n)
    proof (induct n)
      case 0 with P show ?case
        by (subst AE-stream-space) (auto elim!: eventually-mono)
      next
        case (Suc n) then show ?case
          by (subst AE-stream-space) auto
    qed }
  then show ?thesis
    unfolding stream-all-def by (simp add: AE-all-countable)
qed

```

```

lemma streams-sets:
  assumes X[measurable]: X ∈ sets M shows streams X ∈ sets (stream-space M)
proof -
  have streams X = {x ∈ space (stream-space M). x ∈ streams X}
  using streams-mono[OF - sets.sets-into-space[OF X]] by (auto simp: space-stream-space)
  also have ... = {x ∈ space (stream-space M). gfp (λp x. shd x ∈ X ∧ p (stl x))
x}
  apply (simp add: set-eq-iff streams-def streamsp-def)
  apply (intro allI conj-cong refl arg-cong2[where f=gfp] ext)
  apply (case-tac xa)
  apply auto
  done
  also have ... ∈ sets (stream-space M)
  apply (intro predE)
  apply (coinduction rule: measurable-gfp-coinduct)
  apply (auto simp: inf-continuous-def)
  done
  finally show ?thesis .
qed

```

```

lemma sets-stream-space-in-sets:
  assumes space: space N = streams (space M)
  assumes sets:  $\bigwedge i. (\lambda x. x !! i) \in \text{measurable } N M$ 
  shows sets (stream-space M)  $\subseteq$  sets N
  unfolding stream-space-def sets-distr
  by (auto intro!: sets-image-in-sets measurable-Sup2 measurable-vimage-algebra2
del: subsetI equalityI)

```

simp add: sets-PiM-eq-proj snth-in space sets cong: measurable-cong-sets)

lemma *sets-stream-space-eq: sets (stream-space M) = sets (SUP i∈UNIV. vimage-algebra (streams (space M)) (λs. s !! i) M)*
by (*auto intro!: sets-stream-space-in-sets sets-Sup-in-sets sets-image-in-sets measurable-Sup1 snth-in measurable-vimage-algebra1 del: subsetI simp: space-Sup-eq-UN space-stream-space*)

lemma *sets-restrict-stream-space:*
assumes *S[measurable]: S ∈ sets M*
shows *sets (restrict-space (stream-space M) (streams S)) = sets (stream-space (restrict-space M S))*
using *S[THEN sets.sets-into-space]*
apply (*subst restrict-space-eq-vimage-algebra*)
apply (*simp add: space-stream-space streams-mono2*)
apply (*subst vimage-algebra-cong[OF refl refl sets-stream-space-eq]*)
apply (*subst sets-stream-space-eq*)
apply (*subst sets-vimage-Sup-eq[where Y=streams (space M)]*)
apply *simp*
apply *auto []*
apply (*auto intro: streams-mono*) *[]*
apply *auto []*
apply (*simp add: image-image space-restrict-space*)
apply (*simp add: vimage-algebra-cong[OF refl refl restrict-space-eq-vimage-algebra]*)
apply (*subst (1 2) vimage-algebra-vimage-algebra-eq*)
apply (*auto simp: streams-mono snth-in*)
done

primrec *sstart :: 'a set ⇒ 'a list ⇒ 'a stream set where*
sstart S [] = streams S
| [simp del]: sstart S (x # xs) = (##) x ‘ sstart S xs

lemma *in-sstart[simp]: s ∈ sstart S (x # xs) ↔ shd s = x ∧ stl s ∈ sstart S xs*
by (*cases s*) (*auto simp: sstart.simps(2)*)

lemma *sstart-in-streams: xs ∈ lists S ⇒ sstart S xs ⊆ streams S*
by (*induction xs*) (*auto simp: sstart.simps(2)*)

lemma *sstart-eq: x ∈ streams S ⇒ x ∈ sstart S xs = (∀ i < length xs. x !! i = xs ! i)*
by (*induction xs arbitrary: x*) (*auto simp: nth-Cons streams-stl split: nat.splits*)

lemma *sstart-sets: sstart S xs ∈ sets (stream-space (count-space UNIV))*

proof (*induction xs*)

case (*Cons x xs*)

note *Cons[measurable]*

have *sstart S (x # xs) =*

{s∈space (stream-space (count-space UNIV)). shd s = x ∧ stl s ∈ sstart S xs}

by (*simp add: set-eq-iff space-stream-space*)

also have $\dots \in \text{sets } (\text{stream-space } (\text{count-space } \text{UNIV}))$
 by *measurable*
 finally show *?case* .
 qed (*simp add: streams-sets*)

lemma *sigma-sets-singletons*:
 assumes *countable S*
 shows *sigma-sets S (($\lambda s. \{s\}$)'S) = Pow S*
proof *safe*
 interpret *sigma-algebra S sigma-sets S (($\lambda s. \{s\}$)'S)*
 by (*rule sigma-algebra-sigma-sets auto*)
 fix *A* assume $A \subseteq S$
 with *assms* have $(\bigcup a \in A. \{a\}) \in \text{sigma-sets } S ((\lambda s. \{s\})'S)$
 by (*intro countable-UN'*) (*auto dest: countable-subset*)
 then show $A \in \text{sigma-sets } S ((\lambda s. \{s\})'S)$
 by *simp*
 qed (*auto dest: sigma-sets-into-sp[rotated]*)

lemma *sets-count-space-eq-sigma*:
 $\text{countable } S \implies \text{sets } (\text{count-space } S) = \text{sets } (\text{sigma } S ((\lambda s. \{s\})'S))$
 by (*subst sets-measure-of*) (*auto simp: sigma-sets-singletons*)

lemma *sets-stream-space-sstart*:
 assumes $S[\text{simp}]: \text{countable } S$
 shows $\text{sets } (\text{stream-space } (\text{count-space } S)) = \text{sets } (\text{sigma } (\text{streams } S) (\text{sstart } S' \text{lists } S \cup \{\{\}\}))$
proof
 have $[\text{simp}]: \text{sstart } S \text{ ' lists } S \subseteq \text{Pow } (\text{streams } S)$
 by (*simp add: image-subset-iff sstart-in-streams*)

let $?S = \text{sigma } (\text{streams } S) (\text{sstart } S \text{ ' lists } S \cup \{\{\}\})$
 { fix *i a* assume $a \in S$
 { fix *x* have $(x \text{ !! } i = a \wedge x \in \text{streams } S) \longleftrightarrow (\exists xs \in \text{lists } S. \text{length } xs = i \wedge x \in \text{sstart } S (xs \text{ @ } [a]))$
proof (*induction i arbitrary: x*)
 case (*Suc i*) from *this*[*of stl x*] show *?case*
 by (*simp add: length-Suc-conv Bex-def ex-simps[symmetric] del: ex-simps*)
 (*metis stream.collapse streams-Stream*)
 qed (*insert <a ∈ S>, auto intro: streams-stl in-streams*) }
 then have $(\lambda x. x \text{ !! } i) \text{ -' } \{a\} \cap \text{streams } S = (\bigcup xs \in \{xs \in \text{lists } S. \text{length } xs = i\}. \text{sstart } S (xs \text{ @ } [a]))$
 by (*auto simp add: set-eq-iff*)
 also have $\dots \in \text{sets } ?S$
 using $\langle a \in S \rangle$ by (*intro sets.countable-UN'*) (*auto intro!: sigma-sets.Basic image-eqI*)
 finally have $(\lambda x. x \text{ !! } i) \text{ -' } \{a\} \cap \text{streams } S \in \text{sets } ?S . \}$
 then show $\text{sets } (\text{stream-space } (\text{count-space } S)) \subseteq \text{sets } (\text{sigma } (\text{streams } S) (\text{sstart } S' \text{lists } S \cup \{\{\}\}))$
 by (*intro sets-stream-space-in-sets*) (*auto simp: measurable-count-space-eq-countable*)

snth-in)

have *sigma-sets* (*space* (*stream-space* (*count-space* *S*))) (*sstart* *S* ‘*lists* *S* \cup {*{}*})
 \subseteq *sets* (*stream-space* (*count-space* *S*))
proof (*safe intro!*: *sets.sigma-sets-subset*)
fix *xs* **assume** $\forall x \in \text{set } xs. x \in S$
then have *sstart* *S* *xs* = {*x* \in *space* (*stream-space* (*count-space* *S*)). $\forall i < \text{length } xs. x !! i = xs ! i$ }
by (*induction xs*)
(*auto simp: space-stream-space nth-Cons split: nat.split intro: in-streams streams-stl*)
also have $\dots \in \text{sets} (\text{stream-space } (\text{count-space } S))$
by *measurable*
finally show *sstart* *S* *xs* \in *sets* (*stream-space* (*count-space* *S*)) .
qed
then show *sets* (*sigma* (*streams* *S*) (*sstart* *S* ‘*lists* *S* \cup {*{}*})) \subseteq *sets* (*stream-space* (*count-space* *S*))
by (*simp add: space-stream-space*)
qed

lemma *Int-stable-sstart*: *Int-stable* (*sstart* *S* ‘*lists* *S* \cup {*{}*})

proof –

{ **fix** *xs ys* **assume** *xs* \in *lists* *S* *ys* \in *lists* *S*
then have *sstart* *S* *xs* \cap *sstart* *S* *ys* \in *sstart* *S* ‘*lists* *S* \cup {*{}*}
proof (*induction xs ys rule: list-induct2'*)
case ($_4$ *x xs y ys*)
show ?*case*
proof *cases*
assume *x = y*
then have *sstart* *S* (*x* $\#$ *xs*) \cap *sstart* *S* (*y* $\#$ *ys*) = ($\#\#$) *x* ‘ (*sstart* *S* *xs* \cap *sstart* *S* *ys*)
by (*auto simp: image-iff intro!: stream.collapse[symmetric]*)
also have $\dots \in \text{sstart } S \text{ 'lists } S \cup \{\{\}\}$
using $_4$ **by** (*auto simp: sstart.simps(2)[symmetric] del: in-listsD*)
finally show ?*case* .
qed *auto*
qed (*simp-all add: sstart-in-streams inf.absorb1 inf.absorb2 image-eqI* [where *x = []*]) }
then show ?*thesis*
by (*auto simp: Int-stable-def*)
qed

lemma *stream-space-eq-sstart*:

assumes *S* [*simp*]: *countable* *S*
assumes *P*: *prob-space* *M* *prob-space* *N*
assumes *ae*: *AE* *x* *in* *M*. *x* \in *streams* *S* *AE* *x* *in* *N*. *x* \in *streams* *S*
assumes *sets-M*: *sets* *M* = *sets* (*stream-space* (*count-space* *UNIV*))
assumes *sets-N*: *sets* *N* = *sets* (*stream-space* (*count-space* *UNIV*))
assumes *: $\bigwedge xs. xs \neq [] \implies xs \in \text{lists } S \implies \text{emeasure } M (\text{sstart } S \text{ } xs) =$

```

emeasure N (sstart S xs)
  shows M = N
proof (rule measure-eqI-restrict-generator[OF Int-stable-sstart])
  have [simp]: sstart S ‘ lists S  $\subseteq$  Pow (streams S)
    by (simp add: image-subset-iff sstart-in-streams)

  interpret M: prob-space M by fact

  show sstart S ‘ lists S  $\cup$  {[]}  $\subseteq$  Pow (streams S)
    by (auto dest: sstart-in-streams del: in-listsD)

  { fix M :: 'a stream measure assume M: sets M = sets (stream-space (count-space UNIV))
    have sets (restrict-space M (streams S)) = sigma-sets (streams S) (sstart S ‘ lists S  $\cup$  {[]})
      by (subst sets-restrict-space-cong[OF M])
        (simp add: sets-restrict-stream-space restrict-count-space sets-stream-space-sstart)
    }

  from this[OF sets-M] this[OF sets-N]
  show sets (restrict-space M (streams S)) = sigma-sets (streams S) (sstart S ‘ lists S  $\cup$  {[]})
    sets (restrict-space N (streams S)) = sigma-sets (streams S) (sstart S ‘ lists S  $\cup$  {[]})
    by auto
  show {streams S}  $\subseteq$  sstart S ‘ lists S  $\cup$  {[]}
     $\cup$  {streams S} = streams S  $\wedge$  s. s  $\in$  {streams S}  $\implies$  emeasure M s  $\neq$   $\infty$ 
    using M.emeasure-space-1 space-stream-space[of count-space S] sets-eq-imp-space-eq[OF sets-M]
    by (auto simp add: image-eqI[where x=[]])
  show sets M = sets N
    by (simp add: sets-M sets-N)
next
  fix X assume X  $\in$  sstart S ‘ lists S  $\cup$  {[]}
  then obtain xs where X = {}  $\vee$  (xs  $\in$  lists S  $\wedge$  X = sstart S xs)
    by auto
  moreover have emeasure M (streams S) = 1
    using ae by (intro prob-space.emeasure-eq-1-AE[OF P(1)]) (auto simp: sets-M streams-sets)
  moreover have emeasure N (streams S) = 1
    using ae by (intro prob-space.emeasure-eq-1-AE[OF P(2)]) (auto simp: sets-N streams-sets)
  ultimately show emeasure M X = emeasure N X
    using P[THEN prob-space.emeasure-space-1]
    by (cases xs = [] (auto simp: * space-stream-space del: in-listsD))
qed (auto simp: * ae sets-M del: in-listsD intro!: streams-sets)

lemma sets-sstart[measurable]: sstart  $\Omega$  xs  $\in$  sets (stream-space (count-space UNIV))
proof (induction xs)
  case (Cons x xs)

```



```

note this[measurable]
have sstart  $\Omega$  ( $x \# xs$ ) =  $\{\omega \in \text{space } (\text{stream-space } (\text{count-space } UNIV)). \omega \in$ 
sstart  $\Omega$  ( $x \# xs$ ) $\}$ 
  by (auto simp: space-stream-space)
also have  $\dots \in \text{sets } (\text{stream-space } (\text{count-space } UNIV))$ 
  unfolding in-sstart by measurable
finally show ?case .
qed (auto intro!: streams-sets)

```

```

primrec scylinder :: 'a set  $\Rightarrow$  'a set list  $\Rightarrow$  'a stream set
where

```

```

  scylinder  $S$  [] = streams  $S$ 
| scylinder  $S$  ( $A \# As$ ) =  $\{\omega \in \text{streams } S. \text{shd } \omega \in A \wedge \text{stl } \omega \in \text{scylinder } S As\}$ 

```

```

lemma scylinder-streams: scylinder  $S$   $xs \subseteq \text{streams } S$ 
  by (induction xs) auto

```

```

lemma sets-scylinder:  $(\forall x \in \text{set } xs. x \in \text{sets } S) \Longrightarrow \text{scylinder } (\text{space } S) xs \in \text{sets}$ 
(stream-space  $S$ )
  by (induction xs) (auto simp: space-stream-space[symmetric])

```

```

lemma stream-space-eq-scylinder:

```

```

  assumes  $P$ : prob-space  $M$  prob-space  $N$ 
  assumes Int-stable  $G$  and  $S$ : sets  $S = \text{sets } (\text{sigma } (\text{space } S) G)$ 
    and  $C$ : countable  $C$   $C \subseteq G \cup C = \text{space } S$  and  $G$ :  $G \subseteq \text{Pow } (\text{space } S)$ 
  assumes sets-M: sets  $M = \text{sets } (\text{stream-space } S)$ 
  assumes sets-N: sets  $N = \text{sets } (\text{stream-space } S)$ 
  assumes *:  $\bigwedge xs. xs \neq [] \Longrightarrow xs \in \text{lists } G \Longrightarrow \text{emeasure } M (\text{scylinder } (\text{space } S)$ 
 $xs) = \text{emeasure } N (\text{scylinder } (\text{space } S) xs)$ 
  shows  $M = N$ 
proof (rule measure-eqI-generator-eq)
  interpret  $M$ : prob-space  $M$  by fact
  interpret  $N$ : prob-space  $N$  by fact

```

```

  let  $?G = \text{scylinder } (\text{space } S) \text{ 'lists } G$ 
  show sc-Pow:  $?G \subseteq \text{Pow } (\text{streams } (\text{space } S))$ 
    using scylinder-streams by auto

```

```

  have sets (stream-space  $S$ ) = sets (sigma (streams (space  $S$ ))  $?G$ )
    (is  $?S = \text{sets } ?R$ )

```

```

proof (rule antisym)

```

```

  let  $?V = \lambda i. \text{vimage-algebra } (\text{streams } (\text{space } S)) (\lambda s. s !! i) S$ 
  show  $?S \subseteq \text{sets } ?R$ 

```

```

  unfolding sets-stream-space-eq

```

```

proof (safe intro!: sets-Sup-in-sets del: subsetI equalityI)

```

```

  fix  $i :: \text{nat}$ 

```

```

  show space ( $?V i$ ) = space  $?R$ 

```

```

    using scylinder-streams by (subst space-measure-of) auto
  { fix  $A$  assume  $A \in G$ 

```

then have *scylinder* (space S) (replicate i (space S) @ $[A]$) = $(\lambda s. s !! i)$
 $- 'A \cap \text{streams}$ (space S)
by (induction i) (auto simp add: streams-shd streams-stl cong: conj-cong)
also have *scylinder* (space S) (replicate i (space S) @ $[A]$) = $(\bigcup xs \in \{xs \in \text{lists } C. \text{length } xs = i\}. \text{scylinder}$ (space S) (xs @ $[A]$))
apply (induction i)
apply auto []
apply (simp add: length-Suc-conv set-eq-iff ex-simps(1,2)[symmetric] cong: conj-cong del: ex-simps(1,2))
apply rule
subgoal for $i x$
apply (cases x)
apply (subst (2) $C(3)$ [symmetric])
apply (simp del: ex-simps(1,2) add: ex-simps(1,2)[symmetric] ac-simps
Bex-def)
apply auto
done
done
finally have $(\lambda s. s !! i) - 'A \cap \text{streams}$ (space S) = $(\bigcup xs \in \{xs \in \text{lists } C. \text{length } xs = i\}. \text{scylinder}$ (space S) (xs @ $[A]$))
..
also have $\dots \in ?R$
using $C(2) \langle A \in G \rangle$
by (intro sets.countable-UN' countable-Collect countable-lists C)
(auto intro!: in-measure-of[OF sc-Pow] imageI)
finally have $(\lambda s. s !! i) - 'A \cap \text{streams}$ (space S) $\in ?R . \}$
then show sets $(?V i) \subseteq ?R$
apply (subst vimage-algebra-cong[OF refl refl S])
apply (subst vimage-algebra-sigma[OF G])
apply (simp add: streams-iff-snth) []
apply (subst sigma-le-sets)
apply auto
done
qed
have $G \subseteq \text{sets } S$
unfolding S **using** G **by** auto
with $C(2)$ **show** sets $?R \subseteq ?S$
unfolding sigma-le-sets[OF sc-Pow] **by** (auto intro!: sets-scylinder)
qed
then show sets $M = \text{sigma-sets}$ (streams (space S)) (scylinder (space S) ' lists
 G)
sets $N = \text{sigma-sets}$ (streams (space S)) (scylinder (space S) ' lists G)
unfolding sets- M sets- N **by** (simp-all add: sc-Pow)

show Int-stable $?G$
proof (rule Int-stableI-image)
fix xs ys **assume** $xs \in \text{lists } G$ $ys \in \text{lists } G$
then show $\exists zs \in \text{lists } G. \text{scylinder}$ (space S) $xs \cap \text{scylinder}$ (space S) $ys =$
scylinder (space S) zs

```

proof (induction xs arbitrary: ys)
  case Nil then show ?case
    by (auto simp add: Int-absorb1 scylinder-streams)
next
  case xs: (Cons x xs)
  show ?case
  proof (cases ys)
    case Nil with xs.hyps show ?thesis
      by (auto simp add: Int-absorb2 scylinder-streams intro!: bexI[of - x#xs])
  next
    case ys: (Cons y ys')
    with xs.IH[of ys'] xs.prem1 obtain zs where
      zs ∈ lists G and eq: scylinder (space S) xs ∩ scylinder (space S) ys' =
scylinder (space S) zs
      by auto
    show ?thesis
    proof (intro bexI[of - (x ∩ y)#zs])
      show x ∩ y # zs ∈ lists G
      using <zs∈lists G> <x∈G> <y∈lists G> ys <Int-stable G> [THEN Int-stableD,
of x y] by auto
      show scylinder (space S) (x # xs) ∩ scylinder (space S) ys = scylinder
(space S) (x ∩ y # zs)
      by (auto simp add: eq[symmetric] ys)
    qed
  qed
qed
qed

show range (λ::nat. streams (space S)) ⊆ scylinder (space S) ‘ lists G
  (⋃ i. streams (space S)) = streams (space S)
  emeasure M (streams (space S)) ≠ ∞
  by (auto intro!: image-eqI[of - - []])

fix X assume X ∈ scylinder (space S) ‘ lists G
then obtain xs where xs: xs ∈ lists G and eq: X = scylinder (space S) xs
  by auto
then show emeasure M X = emeasure N X
proof (cases xs = [])
  assume xs = [] then show ?thesis
    unfolding eq
    using sets-M [THEN sets-eq-imp-space-eq] sets-N [THEN sets-eq-imp-space-eq]
      M.emeasure-space-1 N.emeasure-space-1
    by (simp add: space-stream-space[symmetric])
  next
    assume xs ≠ [] with xs show ?thesis
      unfolding eq by (intro *)
  qed
qed

```

lemma *stream-space-coinduct*:

fixes $R :: 'a \text{ stream measure} \Rightarrow 'a \text{ stream measure} \Rightarrow \text{bool}$

assumes $R \ A \ B$

assumes $R: \bigwedge A \ B. R \ A \ B \implies \exists K \in \text{space} \ (\text{prob-algebra } M).$

$\exists A' \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M). \exists B' \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M).$

$(\exists y \text{ in } K. R \ (A' \ y) \ (B' \ y) \vee A' \ y = B' \ y) \wedge$

$A = \text{do} \ \{ y \leftarrow K; \omega \leftarrow A' \ y; \text{return} \ (\text{stream-space } M) \ (y \ \#\# \ \omega) \} \wedge$

$B = \text{do} \ \{ y \leftarrow K; \omega \leftarrow B' \ y; \text{return} \ (\text{stream-space } M) \ (y \ \#\# \ \omega) \}$

shows $A = B$

proof (*rule stream-space-eq-cylinder*)

let $?step = \lambda K \ L. \text{do} \ \{ y \leftarrow K; \omega \leftarrow L \ y; \text{return} \ (\text{stream-space } M) \ (y \ \#\# \ \omega) \}$

{ fix $K \ A \ A' \ \text{assume}$ $K: K \in \text{space} \ (\text{prob-algebra } M)$

and $A'[\text{measurable}]: A' \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M)$ **and** $A\text{-eq}:$

$A = ?step \ K \ A'$

have $ps: \text{prob-space } A$

unfolding $A\text{-eq}$ **by** (*rule prob-space-bind'[OF K]*) *measurable*

have $sets \ A = sets \ (\text{stream-space } M)$

unfolding $A\text{-eq}$ **by** (*rule sets-bind'[OF K]*) *measurable*

note $ps \ \text{this}$ }

note $** = \text{this}$

{ fix $A \ B \ \text{assume}$ $R \ A \ B$

obtain $K \ A' \ B' \ \text{where}$ $K: K \in \text{space} \ (\text{prob-algebra } M)$

and $A': A' \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M) \ A = ?step \ K \ A'$

and $B': B' \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M) \ B = ?step \ K \ B'$

using $R[\text{OF } \langle R \ A \ B \rangle]$ **by** *blast*

have $\text{prob-space } A \ \text{prob-space } B \ sets \ A = sets \ (\text{stream-space } M) \ sets \ B = sets \ (\text{stream-space } M)$

using $**[\text{OF } K \ A'] \ **[\text{OF } K \ B']$ **by** *auto* }

note $R\text{-D} = \text{this}$

show $\text{prob-space } A \ \text{prob-space } B \ sets \ A = sets \ (\text{stream-space } M) \ sets \ B = sets \ (\text{stream-space } M)$

using $R\text{-D}[\text{OF } \langle R \ A \ B \rangle]$ **by** *auto*

show $\text{Int-stable} \ (sets \ M) \ sets \ M = sets \ (\text{sigma} \ (\text{space } M) \ (sets \ M)) \ \text{countable} \ \{\text{space } M\}$

$\{\text{space } M\} \subseteq sets \ M \cup \{\text{space } M\} = \text{space } M \ sets \ M \subseteq \text{Pow} \ (\text{space } M)$

using $sets.\text{space-closed}[\text{of } M]$ **by** (*auto simp: Int-stable-def*)

{ fix $A \ As \ L \ K \ \text{assume}$ $K[\text{measurable}]: K \in \text{space} \ (\text{prob-algebra } M)$ **and** $A: A \in sets \ M \ As \in lists \ (sets \ M)$

and $L[\text{measurable}]: L \in M \rightarrow_M \text{prob-algebra} \ (\text{stream-space } M)$

from $A \ \text{have}$ $[\text{measurable}]: \forall x \in set \ (A \ \# \ As). x \in sets \ M \ \forall x \in set \ As. x \in sets \ M$

by *auto*

have $[\text{simp}]: \text{space } K = \text{space } M \ sets \ K = sets \ M$

using K **by** (*auto simp: space-prob-algebra intro!: sets-eq-imp-space-eq*)

```

have [simp]:  $x \in \text{space } M \implies \text{sets } (L \ x) = \text{sets } (\text{stream-space } M)$  for  $x$ 
  using measurable-space[OF L] by (auto simp: space-prob-algebra)
note sets-scylinder[measurable]
have *:  $\text{indicator } (\text{scylinder } (\text{space } M) (A \ \# \ As)) (x \ \#\# \ \omega) =$ 
   $(\text{indicator } A \ x * \text{indicator } (\text{scylinder } (\text{space } M) \ As) \ \omega) :: \text{ennreal}$  for  $\omega \ x$ 
using scylinder-streams[of space M As]  $\langle A \in \text{sets } M \rangle$  [THEN sets.sets-into-space]
  by (auto split: split-indicator)
have emeasure (?step K L)  $(\text{scylinder } (\text{space } M) (A \ \# \ As)) = (\int^+ y. L \ y$ 
   $(\text{scylinder } (\text{space } M) \ As) * \text{indicator } A \ y \ \partial K)$ 
  apply (subst emeasure-bind-prob-algebra[OF K])
  apply measurable
  apply (rule nn-integral-cong)
  apply (subst emeasure-bind-prob-algebra[OF L [THEN measurable-space]])
  apply (simp-all add: ac-simps * nn-integral-cmult-indicator del: scylinder.
  simp)
  apply measurable
  done }
note emeasure-step = this

fix  $Xs$  assume  $Xs \in \text{lists } (\text{sets } M)$ 
from this  $\langle R \ A \ B \rangle$  show  $\text{emeasure } A \ (\text{scylinder } (\text{space } M) \ Xs) = \text{emeasure } B$ 
   $(\text{scylinder } (\text{space } M) \ Xs)$ 
proof (induction  $Xs$  arbitrary:  $A \ B$ )
  case (Cons  $X \ Xs$ )
  obtain  $K \ A' \ B'$  where  $K: K \in \text{space } (\text{prob-algebra } M)$ 
  and  $A'$ [measurable]:  $A' \in M \rightarrow_M \text{prob-algebra } (\text{stream-space } M)$  and  $A: A =$ 
   $?step \ K \ A'$ 
  and  $B'$ [measurable]:  $B' \in M \rightarrow_M \text{prob-algebra } (\text{stream-space } M)$  and  $B: B =$ 
   $?step \ K \ B'$ 
  and  $AE-R: AE \ x \ \text{in } K. R \ (A' \ x) \ (B' \ x) \ \vee \ A' \ x = B' \ x$ 
  using  $R$ [OF  $\langle R \ A \ B \rangle$ ] by blast

  show ?case
  unfolding  $A \ B$  emeasure-step[OF K Cons.hyps A'] emeasure-step[OF K
  Cons.hyps B']
  apply (rule nn-integral-cong-AE)
  using  $AE-R$  by eventually-elim (auto simp add: Cons.IH)
next
  case Nil
  note  $R-D$ [OF this]
from this(1,2) [THEN prob-space.emeasure-space-1] this(3,4) [THEN sets-eq-imp-space-eq]
  show ?case
  by (simp add: space-stream-space)
qed
qed
end

```

theory *Tree-Space*

imports *HOL-Analysis.Analysis HOL-Library.Tree*

begin

lemma *countable-lfp*:

assumes *step*: $\bigwedge Y. \text{countable } Y \implies \text{countable } (F Y)$

and *cont*: *Order-Continuity.sup-continuous* *F*

shows *countable* (*lfp F*)

by(*subst sup-continuous-lfp[OF cont]*)(*simp add: countable-funpow[OF step]*)

lemma *countable-lfp-apply*:

assumes *step*: $\bigwedge Y x. (\bigwedge x. \text{countable } (Y x)) \implies \text{countable } (F Y x)$

and *cont*: *Order-Continuity.sup-continuous* *F*

shows *countable* (*lfp F x*)

proof –

{ **fix** *n*

have $\bigwedge x. \text{countable } ((F \smallfrown n) \text{ bot } x)$

by(*induct n*)(*auto intro: step*) }

thus *?thesis using cont by*(*simp add: sup-continuous-lfp*)

qed

inductive-set *trees* :: 'a set \Rightarrow 'a tree set **for** *S* :: 'a set **where**

[*intro!*]: *Leaf* \in *trees S*

| $l \in \text{trees } S \implies r \in \text{trees } S \implies v \in S \implies \text{Node } l v r \in \text{trees } S$

lemma *Node-in-trees-iff[simp]*: $\text{Node } l v r \in \text{trees } S \longleftrightarrow (l \in \text{trees } S \wedge v \in S \wedge r \in \text{trees } S)$

by (*subst trees.simps*) *auto*

lemma *trees-sub-lfp*: $\text{trees } S \subseteq \text{lfp } (\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{\text{Node } l v r\}))))$

proof

have *mono*: *mono* $(\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{\text{Node } l v r\}))))$

by (*auto simp: mono-def*)

fix *t* **assume** $t \in \text{trees } S$ **then show** $t \in \text{lfp } (\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{\text{Node } l v r\}))))$

proof *induction*

case 1 **then show** *?case*

by (*subst lfp-unfold[OF mono]*) *auto*

next

case 2 **then show** *?case*

by (*subst lfp-unfold[OF mono]*) *auto*

qed

qed

lemma *countable-trees*: $\text{countable } A \implies \text{countable } (\text{trees } A)$

proof (*intro countable-subset[OF trees-sub-lfp] countable-lfp*

sup-continuous-sup sup-continuous-const sup-continuous-id)

show *sup-continuous* $(\lambda T. (\bigcup l \in T. \bigcup v \in A. \bigcup r \in T. \{(l, v, r)\}))$)

unfolding *sup-continuous-def*
proof (*intro allI impI equalityI subsetI, goal-cases*)
case (1 M t)
then obtain $i\ j :: \text{nat}$ **and** $l\ x\ r$ **where** $t = \text{Node } l\ x\ r$ $x \in A$ $l \in M$ $i\ r \in M\ j$
by *auto*
hence $l \in M$ ($\max\ i\ j$) $r \in M$ ($\max\ i\ j$)
using *incseqD[OF <incseq M>, of i max i j] incseqD[OF <incseq M>, of j max i j]* **by** *auto*
with $\langle t = \text{Node } l\ x\ r \rangle$ **and** $\langle x \in A \rangle$ **show** *?case* **by** *auto*
qed *auto*
qed *auto*

lemma *trees-UNIV[simp]*: $\text{trees } UNIV = UNIV$

proof –
have $t \in \text{trees } UNIV$ **for** $t :: 'a$ *tree*
by (*induction t*) (*auto intro: trees.intros(2)*)
then show *?thesis* **by** *auto*
qed

instance *tree* :: (*countable*) *countable*

proof
have *countable* ($UNIV :: 'a$ *tree set*)
by (*subst trees-UNIV[symmetric]*) (*intro countable-trees[OF countableI-type]*)
then show \exists *to-nat*:: $'a$ *tree* \Rightarrow *nat. inj to-nat*
by (*auto simp: countable-def*)
qed

lemma *map-in-trees[intro]*: $(\bigwedge x. x \in \text{set-tree } t \Rightarrow f\ x \in S) \Rightarrow \text{map-tree } f\ t \in \text{trees } S$

by (*induction t*) (*auto intro: trees.intros(2)*)

primrec *trees-cyl* :: $'a$ *set tree* \Rightarrow $'a$ *tree set* **where**

$\text{trees-cyl } \text{Leaf} = \{\text{Leaf}\}$
 $|\ \text{trees-cyl } (\text{Node } l\ v\ r) = (\bigcup l' \in \text{trees-cyl } l. (\bigcup v' \in v. (\bigcup r' \in \text{trees-cyl } r. \{\text{Node } l'\ v'\ r'\})))$

definition *tree-sigma* :: $'a$ *measure* \Rightarrow $'a$ *tree measure*

where

$\text{tree-sigma } M = \text{sigma } (\text{trees } (\text{space } M)) (\text{trees-cyl } ' \text{trees } (\text{sets } M))$

lemma *Node-in-trees-cyl*: $\text{Node } l'\ v'\ r' \in \text{trees-cyl } t \iff$

$(\exists l\ v\ r. t = \text{Node } l\ v\ r \wedge l' \in \text{trees-cyl } l \wedge r' \in \text{trees-cyl } r \wedge v' \in v)$

by (*cases t*) *auto*

lemma *trees-cyl-sub-trees*:

assumes $t \in \text{trees } A$ $A \subseteq \text{Pow } B$ **shows** $\text{trees-cyl } t \subseteq \text{trees } B$

using *assms(1)*

proof *induction*

case (2 $l\ v\ r$) **with** $\langle A \subseteq \text{Pow } B \rangle$ **show** *?case*

by (auto intro!: trees.intros(2))
qed auto

lemma trees-cyl-sets-in-space: trees-cyl ‘ trees (sets M) \subseteq Pow (trees (space M))
using trees-cyl-sub-trees[OF - sets.space-closed, of - M] by auto

lemma space-tree-sigma: space (tree-sigma M) = trees (space M)
unfolding tree-sigma-def by (rule space-measure-of-conv)

lemma sets-tree-sigma-eq: sets (tree-sigma M) = sigma-sets (trees (space M))
(trees-cyl ‘ trees (sets M))
unfolding tree-sigma-def by (rule sets-measure-of) (rule trees-cyl-sets-in-space)

lemma Leaf-in-space-tree-sigma [measurable, simp, intro]: Leaf \in space (tree-sigma M)
by (auto simp: space-tree-sigma)

lemma Leaf-in-tree-sigma [measurable, simp, intro]: {Leaf} \in sets (tree-sigma M)
unfolding sets-tree-sigma-eq
by (rule sigma-sets.Basic) (auto intro: trees.intros(2) image-eqI[where x=Leaf])

lemma trees-cyl-map-treeI: $t \in$ trees-cyl (map-tree ($\lambda x.$ A) t) **if** *: $t \in$ trees A
using * by induction auto

lemma trees-cyl-map-in-sets:
($\bigwedge x. x \in$ set-tree t $\implies f x \in$ sets M) \implies trees-cyl (map-tree f t) \in sets (tree-sigma M)
by (subst sets-tree-sigma-eq) auto

lemma Node-in-tree-sigma:
assumes L: $X \in$ sets (M \otimes_M (tree-sigma M \otimes_M tree-sigma M))
shows {Node l v r | l v r. (v, l, r) \in X} \in sets (tree-sigma M)

proof –

let ?E = $\lambda s::$ unit tree. trees-cyl (map-tree ($\lambda-$. space M) s)
have 1: countable (range ?E)
by (intro countable-image countableI-type)
have 2: trees-cyl ‘ trees (sets M) \subseteq Pow (space (tree-sigma M))
using trees-cyl-sets-in-space[of M] by (simp add: space-tree-sigma)
have 3: sets (tree-sigma M) = sigma-sets (space (tree-sigma M)) (trees-cyl ‘ trees (sets M))
unfolding sets-tree-sigma-eq by (simp add: space-tree-sigma)
have 4: ($\bigcup s.$?E s) = space (tree-sigma M)
proof (safe; clarsimp simp: space-tree-sigma)
fix t s **assume** $t \in$ trees-cyl (map-tree ($\lambda-$::unit. space M) s)
then show $t \in$ trees (space M)
by (induction s arbitrary: t) auto
next
fix t **assume** $t \in$ trees (space M)
then show $\exists t'. t \in$?E t'

by (intro exI[of - map-tree ($\lambda \cdot$. ()) t])
 (auto simp: tree.map-comp comp-def intro: trees-cyl-map-treeI)

qed
have 5: range ?E \subseteq trees-cyl ‘ trees (sets M) **by** auto
let ?P = {A \times B | A B. A \in trees-cyl ‘ trees (sets M) \wedge B \in trees-cyl ‘ trees (sets M)}
have P: sets (tree-sigma M \otimes_M tree-sigma M) = sets (sigma (space (tree-sigma M) \times space (tree-sigma M))) ?P
by (rule sets-pair-eq[OF 2 3 1 5 4 2 3 1 5 4])

have sets (M \otimes_M (tree-sigma M \otimes_M tree-sigma M)) =
 sets (sigma (space M \times space (tree-sigma M \otimes_M tree-sigma M))) {A \times BC | A BC. A \in sets M \wedge BC \in ?P}
proof (rule sets-pair-eq)
show sets M \subseteq Pow (space M) sets M = sigma-sets (space M) (sets M)
by (auto simp: sets.sigma-sets-eq sets.space-closed)
show countable {space M} {space M} \subseteq sets M \cup {space M} = space M
by auto
show ?P \subseteq Pow (space (tree-sigma M \otimes_M tree-sigma M))
using trees-cyl-sets-in-space[of M]
by (auto simp: space-pair-measure space-tree-sigma subset-eq)
then show sets (tree-sigma M \otimes_M tree-sigma M) =
 sigma-sets (space (tree-sigma M \otimes_M tree-sigma M)) ?P
by (subst P, subst sets-measure-of) (auto simp: space-tree-sigma space-pair-measure)
show countable (($\lambda(a, b).$ a \times b) ‘ (range ?E \times range ?E))
by (intro countable-image countable-SIGMA countableI-type)
show ($\lambda(a, b).$ a \times b) ‘ (range ?E \times range ?E) \subseteq ?P
by auto

qed (insert 4, auto simp: space-pair-measure space-tree-sigma set-eq-iff)
also have ... = sigma-sets (space M \times trees (space M) \times trees (space M))
 {A \times BC | A BC. A \in sets M \wedge BC \in {A \times B | A B.
 A \in trees-cyl ‘ trees (sets M) \wedge B \in trees-cyl ‘ trees (sets M)}}
 (is - = sigma-sets ?X ?Y) **using** sets.space-closed[of M] trees-cyl-sub-trees[of - sets M space M]
by (subst sets-measure-of)
 (auto simp: space-pair-measure space-tree-sigma)

also have ?Y = {A \times trees-cyl B \times trees-cyl C | A B C. A \in sets M \wedge
 B \in trees (sets M) \wedge C \in trees (sets M)} **by** blast

finally have X \in sigma-sets (space M \times trees (space M) \times trees (space M))
 {A \times trees-cyl B \times trees-cyl C | A B C. A \in sets M \wedge B \in trees (sets M) \wedge
 C \in trees (sets M) }
using assms **by** blast

then show ?thesis
proof induction
case (Basic A')
then obtain A B C **where** A' = A \times trees-cyl B \times trees-cyl C
and *: A \in sets M B \in trees (sets M) C \in trees (sets M)
by auto
then have {Node l v r | l v r. (v, l, r) \in A'} = trees-cyl (Node B A C)

```

    by auto
  then show ?case
  by (auto simp del: trees-cyl.simps simp: sets-tree-sigma-eq intro!: sigma-sets.Basic
*)
next
case Empty show ?case by auto
next
case (Compl A)
have {Node l v r | l v r. (v, l, r) ∈ space M × trees (space M) × trees (space
M) - A} =
  (space (tree-sigma M) - {Node l v r | l v r. (v, l, r) ∈ A}) - {Leaf}
  by (auto simp: space-tree-sigma elim: trees.cases)
also have ... ∈ sets (tree-sigma M)
  by (intro sets.Diff Compl) auto
finally show ?case .
next
case (Union I)
have *: {Node l v r | l v r. (v, l, r) ∈ ⋃ (I ' UNIV)} =
  (⋃ i. {Node l v r | l v r. (v, l, r) ∈ I i}) by auto
show ?case unfolding * using Union(2) by (intro sets.countable-UN) auto
qed
qed

```

lemma measurable-left[measurable]: $left \in tree\text{-sigma } M \rightarrow_M tree\text{-sigma } M$

proof (rule measurableI)

show $t \in space (tree\text{-sigma } M) \implies left t \in space (tree\text{-sigma } M)$ **for** t

by (cases t) (auto simp: space-tree-sigma)

fix A **assume** $A: A \in sets (tree\text{-sigma } M)$

from sets.sets-into-space[OF this]

have *: $left - ' A \cap space (tree\text{-sigma } M) =$

(if $Leaf \in A$ then $\{Leaf\}$ else $\{\}$) \cup

$\{Node\ a\ v\ r \mid a\ v\ r. (v, a, r) \in space\ M \times A \times space (tree\text{-sigma } M)\}$

by (auto simp: space-tree-sigma elim: trees.cases)

show $left - ' A \cap space (tree\text{-sigma } M) \in sets (tree\text{-sigma } M)$

unfolding * **using** A **by** (intro sets.Un Node-in-tree-sigma pair-measureI) auto

qed

lemma measurable-right[measurable]: $right \in tree\text{-sigma } M \rightarrow_M tree\text{-sigma } M$

proof (rule measurableI)

show $t \in space (tree\text{-sigma } M) \implies right t \in space (tree\text{-sigma } M)$ **for** t

by (cases t) (auto simp: space-tree-sigma)

fix A **assume** $A: A \in sets (tree\text{-sigma } M)$

from sets.sets-into-space[OF this]

have *: $right - ' A \cap space (tree\text{-sigma } M) =$

(if $Leaf \in A$ then $\{Leaf\}$ else $\{\}$) \cup

$\{Node\ l\ v\ a \mid l\ v\ a. (v, l, a) \in space\ M \times space (tree\text{-sigma } M) \times A\}$

by (auto simp: space-tree-sigma elim: trees.cases)

show $right - ' A \cap space (tree\text{-sigma } M) \in sets (tree\text{-sigma } M)$

unfolding * **using** A **by** (intro sets.Un Node-in-tree-sigma pair-measureI) auto

qed

lemma *measurable-value'*: $value \in restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\}) \rightarrow_M M$

proof (rule *measurableI*)

show $t \in space (restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\})) \implies value t \in space M$ **for** t

by (cases t) (auto simp: *space-restrict-space space-tree-sigma*)

fix A **assume** A : $A \in sets M$

from *sets.sets-into-space*[*OF this*]

have $value - ' A \cap space (restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\})) = \{Node\ l\ a\ r \mid l\ a\ r. (a, l, r) \in A \times space (tree\text{-}sigma M) \times space (tree\text{-}sigma M)\}$

by (auto simp: *space-tree-sigma space-restrict-space elim: trees.cases*)

also have $\dots \in sets (tree\text{-}sigma M)$

using A **by** (intro *sets.Un Node-in-tree-sigma pair-measureI*) auto

finally show $value - ' A \cap space (restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\})) \in sets (restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\}))$

by (auto simp: *sets-restrict-space-iff space-restrict-space*)

qed

lemma *measurable-value*[*measurable (raw)*]:

assumes $f \in X \rightarrow_M tree\text{-}sigma M$

and $\bigwedge x. x \in space X \implies f\ x \neq Leaf$

shows $(\lambda\omega. value (f\ \omega)) \in X \rightarrow_M M$

proof –

from *assms* **have** $f \in X \rightarrow_M restrict\text{-}space (tree\text{-}sigma M) (-\{Leaf\})$

by (intro *measurable-restrict-space2*) auto

from *this* **and** *measurable-value'* **show** ?thesis **by** (rule *measurable-compose*)

qed

lemma *measurable-Node* [*measurable*]:

$(\lambda(l,x,r). Node\ l\ x\ r) \in tree\text{-}sigma M \otimes_M M \otimes_M tree\text{-}sigma M \rightarrow_M tree\text{-}sigma M$

proof (rule *measurable-sigma-sets*)

show $sets (tree\text{-}sigma M) = sigma\text{-}sets (trees (space M)) (trees\text{-}cyl\ ' trees (sets M))$

by (simp add: *sets-tree-sigma-eq*)

show $trees\text{-}cyl\ ' trees (sets M) \subseteq Pow (trees (space M))$

by (rule *trees-cyl-sets-in-space*)

show $(\lambda(l, x, r). \langle l, x, r \rangle) \in space (tree\text{-}sigma M \otimes_M M \otimes_M tree\text{-}sigma M) \rightarrow trees (space M)$

by (auto simp: *space-pair-measure space-tree-sigma*)

fix A **assume** t : $A \in trees\text{-}cyl\ ' trees (sets M)$

then obtain t **where** t : $t \in trees (sets M)$ $A = trees\text{-}cyl\ t$ **by** auto

show $(\lambda(l, x, r). \langle l, x, r \rangle) - ' A \cap$

$space (tree\text{-}sigma M \otimes_M M \otimes_M tree\text{-}sigma M)$

$\in sets (tree\text{-}sigma M \otimes_M M \otimes_M tree\text{-}sigma M)$

```

proof (cases t)
  case Leaf
    have ( $\lambda(l, x, r). \langle l, x, r \rangle$ ) - ‘ {Leaf :: 'a tree} = {} by auto
    with Leaf show ?thesis using t by simp
  next
    case (Node l B r)
    hence ( $\lambda(l, x, r). \langle l, x, r \rangle$ ) - ‘  $A \cap \text{space} (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M) =$ 
      trees-cyl l  $\times$  B  $\times$  trees-cyl r
      using t and Node and trees-cyl-sub-trees[of - sets M space M]
      by (auto simp: space-pair-measure space-tree-sigma
        dest: sets.sets-into-space[of - M])
    thus ?thesis using t and Node
      by (auto intro!: pair-measureI simp: sets-tree-sigma-eq)
  qed
qed

```

```

lemma measurable-Node' [measurable (raw)]:
  assumes [measurable]:  $l \in B \rightarrow_M \text{tree-sigma } A$ 
  assumes [measurable]:  $x \in B \rightarrow_M A$ 
  assumes [measurable]:  $r \in B \rightarrow_M \text{tree-sigma } A$ 
  shows ( $\lambda y. \text{Node } (l y) (x y) (r y)$ )  $\in B \rightarrow_M \text{tree-sigma } A$ 
proof -
  have ( $\lambda y. \text{Node } (l y) (x y) (r y)$ ) = ( $\lambda(a,b,c). \text{Node } a b c$ )  $\circ$  ( $\lambda y. (l y, x y, r y)$ )
    by (simp add: o-def)
  also have ...  $\in B \rightarrow_M \text{tree-sigma } A$ 
    by (intro measurable-comp[OF - measurable-Node]) simp-all
  finally show ?thesis .
qed

```

```

lemma measurable-rec-tree[measurable (raw)]:
  assumes t:  $t \in B \rightarrow_M \text{tree-sigma } M$ 
  assumes l:  $l \in B \rightarrow_M A$ 
  assumes n: ( $\lambda(x, l, v, r, al, ar). n x l v r al ar$ )  $\in$ 
    ( $B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \otimes_M A \otimes_M A$ )  $\rightarrow_M A$  (is
    ?N  $\in$  ?M  $\rightarrow_M A$ )
  shows ( $\lambda x. \text{rec-tree } (l x) (n x) (t x)$ )  $\in B \rightarrow_M A$ 
proof (rule measurable-piecewise-restrict)
  let ?C =  $\lambda t. \lambda s::\text{unit tree}. t - \text{trees-cyl} (\text{map-tree } (\lambda-. \text{space } M) s)$ 
  show countable (range (?C t)) by (intro countable-image countableI-type)
  show space B  $\subseteq (\bigcup s. ?C t s)$ 
  proof (safe; clarsimp)
    fix x assume x:  $x \in \text{space } B$  have t x  $\in \text{trees} (\text{space } M)$ 
      using t[THEN measurable-space, OF x] by (simp add: space-tree-sigma)
    then show  $\exists xa::\text{unit tree}. t x \in \text{trees-cyl} (\text{map-tree } (\lambda-. \text{space } M) xa)$ 
      by (intro exI[of - map-tree (\lambda-. ()) (t x)])
      (simp add: tree.map-comp comp-def trees-cyl-map-treeI)
  qed
fix  $\Omega$  assume  $\Omega \in \text{range} (?C t)$ 

```

```

then obtain  $s :: \text{unit tree}$  where  $\Omega = ?C\ t\ s$  by auto
then show  $\Omega \cap \text{space } B \in \text{sets } B$ 
  by (safe intro!: measurable-sets[OF t] trees-cyl-map-in-sets)
show  $(\lambda x. \text{rec-tree } (l\ x)\ (n\ x)\ (t\ x)) \in \text{restrict-space } B\ \Omega \rightarrow_M\ A$ 
  unfolding  $\Omega$  using  $t$ 
proof (induction s arbitrary: t)
  case Leaf
  show ?case
  proof (rule measurable-cong[THEN iffD2])
    fix  $\omega$  assume  $\omega \in \text{space } (\text{restrict-space } B\ (?C\ t\ \text{Leaf}))$ 
    then show  $\text{rec-tree } (l\ \omega)\ (n\ \omega)\ (t\ \omega) = l\ \omega$ 
      by (auto simp: space-restrict-space)
    next
    show  $l \in \text{restrict-space } B\ (?C\ t\ \text{Leaf}) \rightarrow_M\ A$ 
      using  $l$  by (rule measurable-restrict-space1)
    qed
  next
  case (Node ls u rs)
  let  $?F = \lambda \omega. ?N\ (\omega, \text{left } (t\ \omega), \text{value } (t\ \omega), \text{right } (t\ \omega),$ 
     $\text{rec-tree } (l\ \omega)\ (n\ \omega)\ (\text{left } (t\ \omega)), \text{rec-tree } (l\ \omega)\ (n\ \omega)\ (\text{right } (t\ \omega)))$ 
  show ?case
  proof (rule measurable-cong[THEN iffD2])
    fix  $\omega$  assume  $\omega \in \text{space } (\text{restrict-space } B\ (?C\ t\ (\text{Node } ls\ u\ rs)))$ 
    then show  $\text{rec-tree } (l\ \omega)\ (n\ \omega)\ (t\ \omega) = ?F\ \omega$ 
      by (auto simp: space-restrict-space)
    next
    show  $?F \in (\text{restrict-space } B\ (?C\ t\ (\text{Node } ls\ u\ rs))) \rightarrow_M\ A$ 
      apply (intro measurable-compose[OF - n] measurable-Pair[rotated])
      subgoal
        apply (rule measurable-restrict-mono[OF Node(2)])
        apply (rule measurable-compose[OF Node(3) measurable-right])
        by auto
      subgoal
        apply (rule measurable-restrict-mono[OF Node(1)])
        apply (rule measurable-compose[OF Node(3) measurable-left])
        by auto
      subgoal
        by (rule measurable-restrict-space1)
          (rule measurable-compose[OF Node(3) measurable-right])
      subgoal
        apply (rule measurable-compose[OF - measurable-value'])
        apply (rule measurable-restrict-space3[OF Node(3)])
        by auto
      subgoal
        by (rule measurable-restrict-space1)
          (rule measurable-compose[OF Node(3) measurable-left])
        by (rule measurable-restrict-space1) auto
    qed
  qed

```

qed

lemma *measurable-case-tree* [*measurable (raw)*]:

assumes $t \in B \rightarrow_M \text{tree-sigma } M$

assumes $l \in B \rightarrow_M A$

assumes $(\lambda(x, l, v, r). n x l v r)$

$\in B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \rightarrow_M A$

shows $(\lambda x. \text{case-tree } (l x) (n x) (t x)) \in B \rightarrow_M (A :: 'a \text{ measure})$

proof –

define n' **where** $n' = (\lambda x l v r (-::'a) (-::'a). n x l v r)$

have $(\lambda x. \text{case-tree } (l x) (n x) (t x)) = (\lambda x. \text{rec-tree } (l x) (n' x) (t x))$

(is $- = (\lambda x. \text{rec-tree } - (?n' x) -)$ **)** **by** (*rule ext*) (*auto split: tree.splits simp: n'-def*)

also have $\dots \in B \rightarrow_M A$

proof (*rule measurable-rec-tree*)

have $(\lambda(x, l, v, r, al, ar). n' x l v r al ar) =$

$(\lambda(x, l, v, r). n x l v r) \circ (\lambda(x, l, v, r, al, ar). (x, l, v, r))$

by (*simp add: n'-def o-def case-prod-unfold*)

also have $\dots \in B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \otimes_M A$

$\otimes_M A \rightarrow_M A$

using *assms(3)* **by** *measurable*

finally show $(\lambda(x, l, v, r, al, ar). n' x l v r al ar) \in \dots$.

qed (*insert assms, simp-all*)

finally show *?thesis* .

qed

hide-const (**open**) *left*

hide-const (**open**) *right*

end

28 Conditional Expectation

theory *Conditional-Expectation*

imports *Probability-Measure*

begin

28.1 Restricting a measure to a sub-sigma-algebra

definition *subalgebra::'a measure \Rightarrow 'a measure \Rightarrow bool* **where**

subalgebra $M F = ((\text{space } F = \text{space } M) \wedge (\text{sets } F \subseteq \text{sets } M))$

lemma *sub-measure-space*:

assumes $i: \text{subalgebra } M F$

shows *measure-space* (*space* M) (*sets* F) (*emeasure* M)

proof –

have *sigma-algebra* (*space* M) (*sets* F)

by (*metis i measure-space measure-space-def subalgebra-def*)

moreover have *positive* (*sets* F) (*emeasure* M)

using *Sigma-Algebra.positive-def* **by** *auto*
moreover have *countably-additive (sets F) (emeasure M)*
by (*meson countably-additive-def emeasure-countably-additive i order-trans sub-*
algebra-def subsetCE)
ultimately show *?thesis unfolding measure-space-def* **by** *simp*
qed

definition *restr-to-subalg::'a measure \Rightarrow 'a measure \Rightarrow 'a measure* **where**
restr-to-subalg M F = measure-of (space M) (sets F) (emeasure M)

lemma *space-restr-to-subalg:*

space (restr-to-subalg M F) = space M

unfolding *restr-to-subalg-def* **by** (*simp add: space-measure-of-conv*)

lemma *sets-restr-to-subalg [measurable-cong]:*

assumes *subalgebra M F*

shows *sets (restr-to-subalg M F) = sets F*

unfolding *restr-to-subalg-def* **by** (*metis sets.sets-measure-of-eq assms subalgebra-def*)

lemma *emeasure-restr-to-subalg:*

assumes *subalgebra M F*

A \in sets F

shows *emeasure (restr-to-subalg M F) A = emeasure M A*

unfolding *restr-to-subalg-def*

by (*metis assms subalgebra-def emeasure-measure-of-conv sub-measure-space sets.sigma-sets-eq*)

lemma *null-sets-restr-to-subalg:*

assumes *subalgebra M F*

shows *null-sets (restr-to-subalg M F) = null-sets M \cap sets F*

proof

have *x \in null-sets M \cap sets F* **if** *x \in null-sets (restr-to-subalg M F)* **for** *x*

by (*metis that Int-iff assms emeasure-restr-to-subalg null-setsD1 null-setsD2*
null-setsI

sets-restr-to-subalg subalgebra-def subsetD)

then show *null-sets (restr-to-subalg M F) \subseteq null-sets M \cap sets F* **by** *auto*

next

have *x \in null-sets (restr-to-subalg M F)* **if** *x \in null-sets M \cap sets F* **for** *x*

by (*metis that Int-iff assms null-setsD1 null-setsI sets-restr-to-subalg emea-*
sure-restr-to-subalg[OF assms])

then show *null-sets M \cap sets F \subseteq null-sets (restr-to-subalg M F)* **by** *auto*

qed

lemma *AE-restr-to-subalg:*

assumes *subalgebra M F*

AE x in (restr-to-subalg M F). P x

shows *AE x in M. P x*

proof –

obtain *A* **where** *A: $\bigwedge x. x \in$ space (restr-to-subalg M F) – A \implies P x* *A \in*
null-sets (restr-to-subalg M F)

using *AE-E3*[*OF assms*(2)] **by** *auto*
then have $A \in \text{null-sets } M$ **using** *null-sets-restr-to-subalg*[*OF assms*(1)] **by** *auto*
moreover have $\bigwedge x. x \in \text{space } M - A \implies P x$
using *space-restr-to-subalg* *A*(1) **by** *fastforce*
ultimately show *?thesis*
unfolding *eventually-ae-filter* **by** *auto*
qed

lemma *AE-restr-to-subalg2*:
assumes *subalgebra* *M F*
 $AE x \text{ in } M. P x$ **and** [*measurable*]: $P \in \text{measurable } F$ (*count-space UNIV*)
shows $AE x \text{ in } (\text{restr-to-subalg } M F). P x$
proof –
define *U* **where** $U = \{x \in \text{space } M. \neg(P x)\}$
then have *: $U = \{x \in \text{space } F. \neg(P x)\}$ **using** *assms*(1) **by** (*simp add*:
subalgebra-def)
then have $U \in \text{sets } F$ **by** *simp*
then have $U \in \text{sets } M$ **using** *assms*(1) **by** (*meson subalgebra-def subsetD*)
then have $U \in \text{null-sets } M$ **unfolding** *U-def* **using** *assms*(2) **using** *AE-iff-measurable*
by *blast*
then have $U \in \text{null-sets } (\text{restr-to-subalg } M F)$ **using** *null-sets-restr-to-subalg*[*OF*
assms(1)] $\langle U \in \text{sets } F \rangle$ **by** *auto*
then show *?thesis* **using** * **by** (*metis* (*no-types*, *lifting*) *Collect-mono U-def*
eventually-ae-filter space-restr-to-subalg)
qed

lemma *prob-space-restr-to-subalg*:
assumes *subalgebra* *M F*
 $\text{prob-space } M$
shows $\text{prob-space } (\text{restr-to-subalg } M F)$
by (*metis* (*no-types*, *lifting*) *assms*(1) *assms*(2) *emeasure-restr-to-subalg prob-space.emeasure-space-1*
prob-spaceI
 $\text{sets.top space-restr-to-subalg subalgebra-def}$)

lemma *finite-measure-restr-to-subalg*:
assumes *subalgebra* *M F*
 $\text{finite-measure } M$
shows $\text{finite-measure } (\text{restr-to-subalg } M F)$
by (*metis* (*no-types*, *lifting*) *assms* *emeasure-restr-to-subalg finite-measure.finite-emeasure-space*
finite-measureI sets.top space-restr-to-subalg subalgebra-def infinity-enreal-def)

lemma *measurable-in-subalg*:
assumes *subalgebra* *M F*
 $f \in \text{measurable } F N$
shows $f \in \text{measurable } (\text{restr-to-subalg } M F) N$
by (*metis* *measurable-cong-sets assms*(2) *sets-restr-to-subalg*[*OF assms*(1)])

lemma *measurable-in-subalg'*:
assumes *subalgebra* *M F*

$f \in \text{measurable } (\text{restr-to-subalg } M F) N$
shows $f \in \text{measurable } F N$
by (*metis measurable-cong-sets* *assms*(2) *sets-restr-to-subalg*[*OF assms*(1)])

lemma *measurable-from-subalg*:
assumes *subalgebra* $M F$
 $f \in \text{measurable } F N$
shows $f \in \text{measurable } M N$
using *assms* **unfolding** *measurable-def* *subalgebra-def* **by** *auto*

The following is the direct transposition of *nn_integral_subalgebra* (from *Nonnegative_Lebesgue_Integration*) in the current notations, with the removal of the useless assumption $f \geq 0$.

lemma *nn-integral-subalgebra2*:
assumes *subalgebra* $M F$ **and** [*measurable*]: $f \in \text{borel-measurable } F$
shows $(\int^+ x. f x \partial(\text{restr-to-subalg } M F)) = (\int^+ x. f x \partial M)$
proof (*rule nn-integral-subalgebra*)
show $f \in \text{borel-measurable } (\text{restr-to-subalg } M F)$
by (*rule measurable-in-subalg*[*OF assms*(1)]) *simp*
show *sets* $(\text{restr-to-subalg } M F) \subseteq \text{sets } M$ **by** (*metis sets-restr-to-subalg*[*OF assms*(1)] *assms*(1) *subalgebra-def*)
fix A **assume** $A \in \text{sets } (\text{restr-to-subalg } M F)$
then show *emeasure* $(\text{restr-to-subalg } M F) A = \text{emeasure } M A$
by (*metis sets-restr-to-subalg*[*OF assms*(1)] *emeasure-restr-to-subalg*[*OF assms*(1)])
qed (*auto simp add: assms space-restr-to-subalg sets-restr-to-subalg*[*OF assms*(1)])

The following is the direct transposition of *integral_subalgebra* (from *Bochner_Integration*) in the current notations.

lemma *integral-subalgebra2*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
assumes *subalgebra* $M F$ **and**
 [*measurable*]: $f \in \text{borel-measurable } F$
shows $(\int x. f x \partial(\text{restr-to-subalg } M F)) = (\int x. f x \partial M)$
by (*rule integral-subalgebra*,
metis measurable-in-subalg[*OF assms*(1)] *assms*(2),
auto simp add: assms space-restr-to-subalg sets-restr-to-subalg emeasure-restr-to-subalg,
meson assms(1) *subalgebra-def subset-eq*)

lemma *integrable-from-subalg*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
assumes *subalgebra* $M F$
 $\text{integrable } (\text{restr-to-subalg } M F) f$
shows *integrable* $M f$
proof (*rule integrableI-bounded*)
have [*measurable*]: $f \in \text{borel-measurable } F$ **using** *assms* **by** *auto*
then show $f \in \text{borel-measurable } M$ **using** *assms*(1) *measurable-from-subalg* **by**
blast

have $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial M) = (\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial(\text{restr-to-subalg } M F))$
by $(\text{rule } \text{nn-integral-subalgebra2}[\text{symmetric}], \text{auto simp add: assms})$
also have $\dots < \infty$ **using** $\text{integrable-iff-bounded assms}$ **by** auto
finally show $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial M) < \infty$ **by** simp
qed

lemma $\text{integrable-in-subalg}$:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second-countable-topology}\}$
assumes $[\text{measurable}]$: $\text{subalgebra } M F$
 $f \in \text{borel-measurable } F$
 $\text{integrable } M f$
shows $\text{integrable}(\text{restr-to-subalg } M F) f$
proof $(\text{rule } \text{integrableI-bounded})$
show $f \in \text{borel-measurable}(\text{restr-to-subalg } M F)$ **using** $\text{assms}(2)$ $\text{assms}(1)$ **by**
 auto
have $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial(\text{restr-to-subalg } M F)) = (\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial M)$
by $(\text{rule } \text{nn-integral-subalgebra2}, \text{auto simp add: assms})$
also have $\dots < \infty$ **using** $\text{integrable-iff-bounded assms}$ **by** auto
finally show $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial(\text{restr-to-subalg } M F)) < \infty$ **by** simp
qed

28.2 Nonnegative conditional expectation

The conditional expectation of a function f , on a measure space M , with respect to a sub sigma algebra F , should be a function g which is F -measurable whose integral on any F -set coincides with the integral of f . Such a function is uniquely defined almost everywhere. The most direct construction is to use the measure $f dM$, restrict it to the sigma-algebra F , and apply the Radon-Nikodym theorem to write it as $g dM|_F$ for some F -measurable function g . Another classical construction for L^2 functions is done by orthogonal projection on F -measurable functions, and then extending by density to L^1 . The Radon-Nikodym point of view avoids the L^2 machinery, and works for all positive functions.

In this paragraph, we develop the definition and basic properties for nonnegative functions, as the basics of the general case. As in the definition of integrals, the nonnegative case is done with ennreal-valued functions, without any integrability assumption.

definition $\text{nn-cond-exp} :: 'a \text{ measure} \Rightarrow 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{ennreal}) \Rightarrow ('a \Rightarrow \text{ennreal})$

where

$\text{nn-cond-exp } M F f =$
 $(\text{if } f \in \text{borel-measurable } M \wedge \text{subalgebra } M F$
 $\text{then RN-deriv}(\text{restr-to-subalg } M F)(\text{restr-to-subalg}(\text{density } M f) F)$
 $\text{else } (\lambda \cdot. 0))$

lemma

shows *borel-measurable-nn-cond-exp* [*measurable*]: *nn-cond-exp* $M F f \in \text{borel-measurable } F$

and *borel-measurable-nn-cond-exp2* [*measurable*]: *nn-cond-exp* $M F f \in \text{borel-measurable } M$

by (*simp-all add: nn-cond-exp-def*)

(*metis borel-measurable-RN-deriv borel-measurable-subalgebra sets-restr-to-subalg space-restr-to-subalg subalgebra-def*)

The good setting for conditional expectations is the situation where the subalgebra F gives rise to a sigma-finite measure space. To see what goes wrong if it is not sigma-finite, think of \mathbb{R} with the trivial sigma-algebra $\{\emptyset, \mathbb{R}\}$. In this case, conditional expectations have to be constant functions, so they have integral 0 or ∞ . This means that a positive integrable function can have no meaningful conditional expectation.

locale *sigma-finite-subalgebra* =

fixes $M F :: 'a \text{ measure}$

assumes *subalg: subalgebra* $M F$

and *sigma-fin-subalg: sigma-finite-measure* (*restr-to-subalg* $M F$)

lemma *sigma-finite-subalgebra-is-sigma-finite*:

assumes *sigma-finite-subalgebra* $M F$

shows *sigma-finite-measure* M

proof

have *subalg: subalgebra* $M F$

and *sigma-fin-subalg: sigma-finite-measure* (*restr-to-subalg* $M F$)

using *assms unfolding sigma-finite-subalgebra-def* **by** *auto*

obtain A **where** $Ap: \text{countable } A \wedge A \subseteq \text{sets } (\text{restr-to-subalg } M F) \wedge \bigcup A = \text{space } (\text{restr-to-subalg } M F) \wedge (\forall a \in A. \text{emeasure } (\text{restr-to-subalg } M F) a \neq \infty)$

using *sigma-finite-measure.sigma-finite-countable*[*OF sigma-fin-subalg*] **by** *fastforce*

have $A \subseteq \text{sets } F$ **using** Ap *sets-restr-to-subalg*[*OF subalg*] **by** *fastforce*

then have $A \subseteq \text{sets } M$ **using** *subalg subalgebra-def* **by** *force*

moreover have $\bigcup A = \text{space } M$ **using** Ap *space-restr-to-subalg* **by** *simp*

moreover have $\forall a \in A. \text{emeasure } M a \neq \infty$ **by** (*metis subsetD emeasure-restr-to-subalg*[*OF subalg*] $\langle A \subseteq \text{sets } F \rangle Ap$)

ultimately show $\exists A. \text{countable } A \wedge A \subseteq \text{sets } M \wedge \bigcup A = \text{space } M \wedge (\forall a \in A. \text{emeasure } M a \neq \infty)$ **using** Ap **by** *auto*

qed

sublocale *sigma-finite-subalgebra* \subseteq *sigma-finite-measure*

using *sigma-finite-subalgebra-is-sigma-finite sigma-finite-subalgebra-axioms* **by** *blast*

Conditional expectations are very often used in probability spaces. This is a special case of the previous one, as we prove now.

locale *finite-measure-subalgebra* = *finite-measure* +

fixes $F :: 'a \text{ measure}$

assumes *subalg: subalgebra* $M F$

lemma *finite-measure-subalgebra-is-sigma-finite*:
assumes *finite-measure-subalgebra* $M F$
shows *sigma-finite-subalgebra* $M F$
proof –
interpret *finite-measure-subalgebra* $M F$ **using** *assms* **by** *simp*
have *finite-measure* (*restr-to-subalg* $M F$)
using *finite-measure-restr-to-subalg* *subalg* *finite-measure-space* *finite-measureI*
unfolding *infinity-enreal-def* **by** *blast*
then have *sigma-finite-measure* (*restr-to-subalg* $M F$)
unfolding *finite-measure-def* **by** *simp*
then show *sigma-finite-subalgebra* $M F$ **unfolding** *sigma-finite-subalgebra-def*
using *subalg* **by** *simp*
qed

sublocale *finite-measure-subalgebra* \subseteq *sigma-finite-subalgebra*
proof –
have *finite-measure* (*restr-to-subalg* $M F$)
using *finite-measure-restr-to-subalg* *subalg* *finite-measure-space* *finite-measureI*
unfolding *infinity-enreal-def* **by** *blast*
then have *sigma-finite-measure* (*restr-to-subalg* $M F$)
unfolding *finite-measure-def* **by** *simp*
then show *sigma-finite-subalgebra* $M F$ **unfolding** *sigma-finite-subalgebra-def*
using *subalg* **by** *simp*
qed

context *sigma-finite-subalgebra*
begin

The next lemma is arguably the most fundamental property of conditional expectation: when computing an expectation against an F -measurable function, it is equivalent to work with a function or with its F -conditional expectation.

This property (even for bounded test functions) characterizes conditional expectations, as the second lemma below shows. From this point on, we will only work with it, and forget completely about the definition using Radon-Nikodym derivatives.

lemma *nn-cond-exp-intg*:
assumes [*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$
shows $(\int^+ x. f x * \text{nn-cond-exp } M F g x \partial M) = (\int^+ x. f x * g x \partial M)$
proof –
have [*measurable*]: $f \in \text{borel-measurable } M$
by (*meson* *assms* *subalg* *borel-measurable-subalgebra* *subalgebra-def*)
have *ac*: *absolutely-continuous* (*restr-to-subalg* $M F$) (*restr-to-subalg* (*density* $M g$) F)
unfolding *absolutely-continuous-def*
proof –
have *null-sets* (*restr-to-subalg* $M F$) = *null-sets* $M \cap$ *sets* F **by** (*rule* *null-sets-restr-to-subalg*[*OF*

subalg])
moreover have *null-sets* $M \subseteq \text{null-sets } (\text{density } M g)$
by (*rule absolutely-continuousI-density[unfolded absolutely-continuous-def]*)
auto
ultimately have *null-sets* $(\text{restr-to-subalg } M F) \subseteq \text{null-sets } (\text{density } M g) \cap$
sets F **by** *auto*
moreover have *null-sets* $(\text{density } M g) \cap \text{sets } F = \text{null-sets } (\text{restr-to-subalg}$
 $(\text{density } M g) F)$
by (*rule null-sets-restr-to-subalg[symmetric]*) (*metis subalg sets-density space-density*
subalgebra-def)
ultimately show *null-sets* $(\text{restr-to-subalg } M F) \subseteq \text{null-sets } (\text{restr-to-subalg}$
 $(\text{density } M g) F)$ **by** *auto*
qed

have $(\int^+ x. f x * \text{nn-cond-exp } M F g x \partial M) = (\int^+ x. f x * \text{nn-cond-exp } M F$
 $g x \partial(\text{restr-to-subalg } M F))$
by (*rule nn-integral-subalgebra2[symmetric]*) (*simp-all add: assms subalg*)
also have $\dots = (\int^+ x. f x * \text{RN-deriv } (\text{restr-to-subalg } M F) (\text{restr-to-subalg}$
 $(\text{density } M g) F) x \partial(\text{restr-to-subalg } M F))$
unfolding *nn-cond-exp-def* **using** *assms subalg* **by** *simp*
also have $\dots = (\int^+ x. \text{RN-deriv } (\text{restr-to-subalg } M F) (\text{restr-to-subalg } (\text{density}$
 $M g) F) x * f x \partial(\text{restr-to-subalg } M F))$
by (*simp add: mult commute*)
also have $\dots = (\int^+ x. f x \partial(\text{restr-to-subalg } (\text{density } M g) F))$
proof (*rule sigma-finite-measure.RN-deriv-nn-integral[symmetric]*)
show *sets* $(\text{restr-to-subalg } (\text{density } M g) F) = \text{sets } (\text{restr-to-subalg } M F)$
by (*metis subalg restr-to-subalg-def sets.sets-measure-of-eq space-density sub-*
algebra-def)
qed (*auto simp add: assms measurable-restrict ac measurable-in-subalg subalg*
sigma-fin-subalg)
also have $\dots = (\int^+ x. f x \partial(\text{density } M g))$
by (*metis nn-integral-subalgebra2 subalg assms(1) sets-density space-density*
subalgebra-def)
also have $\dots = (\int^+ x. g x * f x \partial M)$
by (*rule nn-integral-density*) (*simp-all add: assms*)
also have $\dots = (\int^+ x. f x * g x \partial M)$
by (*simp add: mult commute*)
finally show *?thesis* **by** *simp*
qed

lemma *nn-cond-exp-charact*:

assumes $\bigwedge A. A \in \text{sets } F \implies (\int^+ x \in A. f x \partial M) = (\int^+ x \in A. g x \partial M)$ **and**
 $[\text{measurable}]: f \in \text{borel-measurable } M g \in \text{borel-measurable } F$

shows $A \in M. g x = \text{nn-cond-exp } M F f x$

proof –

let $?MF = \text{restr-to-subalg } M F$

{

fix A **assume** $A \in \text{sets } ?MF$

then have $[\text{measurable}]: A \in \text{sets } F$ **using** *sets-restr-to-subalg[OF subalg]* **by**

simp
have $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. g x \partial M)$
by (*simp add: nn-integral-subalgebra2 subalg*)
also have $\dots = (\int^+ x \in A. f x \partial M)$ **using** *assms(1)* **by** *simp*
also have $\dots = (\int^+ x. \text{indicator } A x * f x \partial M)$ **by** (*simp add: mult.commute*)
also have $\dots = (\int^+ x. \text{indicator } A x * \text{nn-cond-exp } M F f x \partial M)$
by (*rule nn-cond-exp-intg[symmetric]*) (*auto simp add: assms*)
also have $\dots = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial M)$ **by** (*simp add: mult.commute*)
also have $\dots = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$
by (*simp add: nn-integral-subalgebra2 subalg*)
finally have $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$ **by** *simp*
} note $*$ = *this*
have $AE x \text{ in } ?MF. g x = \text{nn-cond-exp } M F f x$
by (*rule sigma-finite-measure.density-unique2*)
(*auto simp add: assms subalg sigma-fin-subalg AE-restr-to-subalg2 **)
then show *?thesis* **using** *AE-restr-to-subalg[OF subalg]* **by** *simp*
qed

lemma *nn-cond-exp-F-meas:*

assumes $f \in \text{borel-measurable } F$
shows $AE x \text{ in } M. f x = \text{nn-cond-exp } M F f x$
by (*rule nn-cond-exp-charact*) (*auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma *nn-cond-exp-prod:*

assumes [*measurable*]: $f \in \text{borel-measurable } F g \in \text{borel-measurable } M$
shows $AE x \text{ in } M. f x * \text{nn-cond-exp } M F g x = \text{nn-cond-exp } M F (\lambda x. f x * g x) x$
proof (*rule nn-cond-exp-charact*)
have [*measurable*]: $f \in \text{borel-measurable } M$ **by** (*rule measurable-from-subalg[OF subalg assms(1)]*)
show $(\lambda x. f x * g x) \in \text{borel-measurable } M$ **by** *measurable*

fix A **assume** $A \in \text{sets } F$

then have [*measurable*]: $(\lambda x. f x * \text{indicator } A x) \in \text{borel-measurable } F$ **by** *measurable*
have $\int^+ x \in A. (f x * g x) \partial M = \int^+ x. (f x * \text{indicator } A x) * g x \partial M$
by (*simp add: mult.commute mult.left-commute*)
also have $\dots = \int^+ x. (f x * \text{indicator } A x) * \text{nn-cond-exp } M F g x \partial M$
by (*rule nn-cond-exp-intg[symmetric]*) (*auto simp add: assms*)
also have $\dots = \int^+ x \in A. (f x * \text{nn-cond-exp } M F g x) \partial M$
by (*simp add: mult.commute mult.left-commute*)
finally show $\int^+ x \in A. (f x * g x) \partial M = \int^+ x \in A. (f x * \text{nn-cond-exp } M F g x) \partial M$
by *simp*
qed (*auto simp add: assms*)

lemma *nn-cond-exp-sum:*

assumes $[measurable]: f \in \text{borel-measurable } M \ g \in \text{borel-measurable } M$
shows $AE \ x \text{ in } M. \text{ nn-cond-exp } M \ F \ f \ x + \text{ nn-cond-exp } M \ F \ g \ x = \text{ nn-cond-exp } M \ F \ (\lambda x. f \ x + g \ x) \ x$
proof (rule *nn-cond-exp-charact*)
fix A **assume** $[measurable]: A \in \text{sets } F$
then have $A \in \text{sets } M$ **by** (*meson subalg subalgebra-def subsetD*)
have $\int^{+x \in A}. (\text{nn-cond-exp } M \ F \ f \ x + \text{ nn-cond-exp } M \ F \ g \ x) \partial M = (\int^{+x \in A}. \text{ nn-cond-exp } M \ F \ f \ x \ \partial M) + (\int^{+x \in A}. \text{ nn-cond-exp } M \ F \ g \ x \ \partial M)$
by (rule *nn-set-integral-add*) (*auto simp add: assms ⟨A ∈ sets M⟩*)
also have $\dots = (\int^{+x}. \text{ indicator } A \ x * \text{ nn-cond-exp } M \ F \ f \ x \ \partial M) + (\int^{+x}. \text{ indicator } A \ x * \text{ nn-cond-exp } M \ F \ g \ x \ \partial M)$
by (*metis (no-types, lifting) mult.commute nn-integral-cong*)
also have $\dots = (\int^{+x}. \text{ indicator } A \ x * f \ x \ \partial M) + (\int^{+x}. \text{ indicator } A \ x * g \ x \ \partial M)$
by (*simp add: nn-cond-exp-intg*)
also have $\dots = (\int^{+x \in A}. f \ x \ \partial M) + (\int^{+x \in A}. g \ x \ \partial M)$
by (*metis (no-types, lifting) mult.commute nn-integral-cong*)
also have $\dots = \int^{+x \in A}. (f \ x + g \ x) \ \partial M$
by (rule *nn-set-integral-add[symmetric]*) (*auto simp add: assms ⟨A ∈ sets M⟩*)
finally show $\int^{+x \in A}. (f \ x + g \ x) \ \partial M = \int^{+x \in A}. (\text{nn-cond-exp } M \ F \ f \ x + \text{ nn-cond-exp } M \ F \ g \ x) \ \partial M$
by *simp*
qed (*auto simp add: assms*)

lemma *nn-cond-exp-cong*:

assumes $AE \ x \text{ in } M. f \ x = g \ x$
and $[measurable]: f \in \text{borel-measurable } M \ g \in \text{borel-measurable } M$
shows $AE \ x \text{ in } M. \text{ nn-cond-exp } M \ F \ f \ x = \text{ nn-cond-exp } M \ F \ g \ x$
proof (rule *nn-cond-exp-charact*)
fix A **assume** $[measurable]: A \in \text{sets } F$
have $\int^{+x \in A}. \text{ nn-cond-exp } M \ F \ f \ x \ \partial M = \int^{+x}. \text{ indicator } A \ x * \text{ nn-cond-exp } M \ F \ f \ x \ \partial M$
by (*simp add: mult.commute*)
also have $\dots = \int^{+x}. \text{ indicator } A \ x * f \ x \ \partial M$ **by** (*simp add: nn-cond-exp-intg assms*)
also have $\dots = \int^{+x \in A}. f \ x \ \partial M$ **by** (*simp add: mult.commute*)
also have $\dots = \int^{+x \in A}. g \ x \ \partial M$ **by** (rule *nn-set-integral-cong[OF assms(1)]*)
finally show $\int^{+x \in A}. g \ x \ \partial M = \int^{+x \in A}. \text{ nn-cond-exp } M \ F \ f \ x \ \partial M$ **by** *simp*
qed (*auto simp add: assms*)

lemma *nn-cond-exp-mono*:

assumes $AE \ x \text{ in } M. f \ x \leq g \ x$
and $[measurable]: f \in \text{borel-measurable } M \ g \in \text{borel-measurable } M$
shows $AE \ x \text{ in } M. \text{ nn-cond-exp } M \ F \ f \ x \leq \text{ nn-cond-exp } M \ F \ g \ x$
proof –
define h **where** $h = (\lambda x. g \ x - f \ x)$
have $[measurable]: h \in \text{borel-measurable } M$ **unfolding** $h\text{-def}$ **by** *simp*
have $*$: $AE \ x \text{ in } M. g \ x = f \ x + h \ x$ **unfolding** $h\text{-def}$ **using** *assms(1)* **by** (*auto simp: ennreal-ineq-diff-add*)
have $AE \ x \text{ in } M. \text{ nn-cond-exp } M \ F \ g \ x = \text{ nn-cond-exp } M \ F \ (\lambda x. f \ x + h \ x) \ x$

by (rule *nn-cond-exp-cong*) (auto simp add: * *assms*)
moreover have $AE\ x\ in\ M. nn-cond-exp\ M\ F\ f\ x + nn-cond-exp\ M\ F\ h\ x =$
 $nn-cond-exp\ M\ F\ (\lambda x. f\ x + h\ x)\ x$
by (rule *nn-cond-exp-sum*) (auto simp add: *assms*)
ultimately have $AE\ x\ in\ M. nn-cond-exp\ M\ F\ g\ x = nn-cond-exp\ M\ F\ f\ x +$
 $nn-cond-exp\ M\ F\ h\ x$ **by** auto
then show *?thesis* **by** force
qed

lemma *nested-subalg-is-sigma-finite*:

assumes *subalgebra M G subalgebra G F*
shows *sigma-finite-subalgebra M G*
unfolding *sigma-finite-subalgebra-def*
proof (auto simp add: *assms*)
have $\exists A. countable\ A \wedge A \subseteq sets\ (restr-to-subalg\ M\ F) \wedge \bigcup A = space\ (restr-to-subalg\ M\ F) \wedge (\forall a \in A. emeasure\ (restr-to-subalg\ M\ F)\ a \neq \infty)$
using *sigma-fin-subalg sigma-finite-measure-def* **by** auto
then obtain A **where** $A:countable\ A \wedge A \subseteq sets\ (restr-to-subalg\ M\ F) \wedge \bigcup A = space\ (restr-to-subalg\ M\ F) \wedge (\forall a \in A. emeasure\ (restr-to-subalg\ M\ F)\ a \neq \infty)$
by auto
have $sets\ F \subseteq sets\ M$
by (*meson assms order-trans subalgebra-def*)
then have $countable\ A \wedge A \subseteq sets\ (restr-to-subalg\ M\ G) \wedge \bigcup A = space\ (restr-to-subalg\ M\ F) \wedge (\forall a \in A. emeasure\ (restr-to-subalg\ M\ G)\ a \neq \infty)$
by (*metis (no-types) A assms basic-trans-rules(31) emeasure-restr-to-subalg order-trans sets-restr-to-subalg subalgebra-def*)
then show *sigma-finite-measure (restr-to-subalg M G)*
by (*metis sigma-finite-measure.intro space-restr-to-subalg*)
qed

lemma *nn-cond-exp-nested-subalg*:

assumes *subalgebra M G subalgebra G F*
and [*measurable*]: $f \in borel-measurable\ M$
shows $AE\ x\ in\ M. nn-cond-exp\ M\ F\ f\ x = nn-cond-exp\ M\ F\ (nn-cond-exp\ M\ G\ f)\ x$
proof (rule *nn-cond-exp-charact*, auto)
interpret $G: sigma-finite-subalgebra\ M\ G$ **by** (rule *nested-subalg-is-sigma-finite[OF assms(1) assms(2)]*)
fix A **assume** [*measurable*]: $A \in sets\ F$
then have [*measurable*]: $A \in sets\ G$ **using** *assms(2)* **by** (*meson subsetD subalgebra-def*)

have $set-nn-integral\ M\ A\ (nn-cond-exp\ M\ G\ f) = (\int^+ x. indicator\ A\ x * nn-cond-exp\ M\ G\ f\ x\ \partial M)$

by (*metis (no-types) mult commute*)

also have $\dots = (\int^+ x. indicator\ A\ x * f\ x\ \partial M)$ **by** (rule *G.nn-cond-exp-intg*, auto simp add: *assms*)

also have $\dots = (\int^+ x. indicator\ A\ x * nn-cond-exp\ M\ F\ f\ x\ \partial M)$ **by** (rule *nn-cond-exp-intg[symmetric]*, auto simp add: *assms*)

also have ... = *set-nn-integral* $M A$ (*nn-cond-exp* $M F f$) **by** (*metis* (*no-types*) *mult.commute*)

finally show *set-nn-integral* $M A$ (*nn-cond-exp* $M G f$) = *set-nn-integral* $M A$ (*nn-cond-exp* $M F f$).

qed

end

28.3 Real conditional expectation

Once conditional expectations of positive functions are defined, the definition for real-valued functions follows readily, by taking the difference of positive and negative parts. One could also define a conditional expectation of vector-space valued functions, as in *Bochner_Integral*, but since the real-valued case is the most important, and quicker to formalize, I concentrate on it. (It is also essential for the case of the most general Pettis integral.)

definition *real-cond-exp* :: 'a *measure* \Rightarrow 'a *measure* \Rightarrow ('a \Rightarrow real) \Rightarrow ('a \Rightarrow real) **where**

real-cond-exp $M F f$ =
 $(\lambda x. \text{enn2real}(\text{nn-cond-exp } M F (\lambda x. \text{ennreal } (f x)) x) - \text{enn2real}(\text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-f x)) x))$

lemma

shows *borel-measurable-cond-exp* [*measurable*]: *real-cond-exp* $M F f \in$ *borel-measurable* F

and *borel-measurable-cond-exp2* [*measurable*]: *real-cond-exp* $M F f \in$ *borel-measurable* M

unfolding *real-cond-exp-def* **by** *auto*

context *sigma-finite-subalgebra*

begin

lemma *real-cond-exp-abs*:

assumes [*measurable*]: $f \in$ *borel-measurable* M

shows $\forall x \text{ in } M. \text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (\text{abs}(f x))) x$

proof –

define fp **where** $fp = (\lambda x. \text{ennreal } (f x))$

define fm **where** $fm = (\lambda x. \text{ennreal } (-f x))$

have [*measurable*]: $fp \in$ *borel-measurable* M $fm \in$ *borel-measurable* M **unfolding** *fp-def fm-def* **by** *auto*

have $eq: \bigwedge x. \text{ennreal } |f x| = fp x + fm x$ **unfolding** *fp-def fm-def* **by** (*simp add: abs-real-def ennreal-neg*)

{
fix x **assume** $H: \text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$

```

have |real-cond-exp M F f x| ≤ |enn2real(nn-cond-exp M F fp x)| + |enn2real(nn-cond-exp
M F fm x)|
  unfolding real-cond-exp-def fp-def fm-def by (auto intro: abs-triangle-ineq4
simp del: enn2real-nonneg)
  from ennreal-leI[OF this]
  have abs(real-cond-exp M F f x) ≤ nn-cond-exp M F fp x + nn-cond-exp M F
fm x
    by simp (metis add commute ennreal-enn2real le-iff-add not-le top-unique)
  also have ... = nn-cond-exp M F (λx. fp x + fm x) x using H by simp
  finally have abs(real-cond-exp M F f x) ≤ nn-cond-exp M F (λx. fp x + fm x)
x by simp
}
moreover have AE x in M. nn-cond-exp M F fp x + nn-cond-exp M F fm x =
nn-cond-exp M F (λx. fp x + fm x) x
  by (rule nn-cond-exp-sum) (auto simp add: fp-def fm-def)
ultimately have AE x in M. abs(real-cond-exp M F f x) ≤ nn-cond-exp M F
(λx. fp x + fm x) x
  by auto
then show ?thesis using eq by simp
qed

```

The next lemma shows that the conditional expectation is an F -measurable function whose average against an F -measurable function f coincides with the average of the original function against f . It is obtained as a consequence of the same property for the positive conditional expectation, taking the difference of the positive and the negative part. The proof is given first assuming $f \geq 0$ for simplicity, and then extended to the general case in the subsequent lemma. The idea of the proof is essentially trivial, but the implementation is slightly tedious as one should check all the integrability properties of the different parts, and go back and forth between positive integral and signed integrals, and between real-valued functions and ennreal-valued functions.

Once this lemma is available, we will use it to characterize the conditional expectation, and never come back to the original technical definition, as we did in the case of the nonnegative conditional expectation.

lemma *real-cond-exp-intg-fpos*:

```

assumes integrable M (λx. f x * g x) and f-pos[simp]: ∧x. f x ≥ 0 and
  [measurable]: f ∈ borel-measurable F g ∈ borel-measurable M
shows integrable M (λx. f x * real-cond-exp M F g x)
  (∫ x. f x * real-cond-exp M F g x ∂M) = (∫ x. f x * g x ∂M)

```

proof –

```

have [measurable]: f ∈ borel-measurable M by (rule measurable-from-subalg[OF
subalg assms(3)])
define gp where gp = (λx. ennreal (g x))
define gm where gm = (λx. ennreal (– g x))
have [measurable]: gp ∈ borel-measurable M gm ∈ borel-measurable M unfolding
gp-def gm-def by auto

```

```

define h where h = ( $\lambda x. \text{ennreal}(\text{abs}(g\ x))$ )
have hgpgm:  $\bigwedge x. h\ x = gp\ x + gm\ x$  unfolding gp-def gm-def h-def by (simp
add: abs-real-def ennreal-neg)
have [measurable]: h  $\in$  borel-measurable M unfolding h-def by simp
have pos[simp]:  $\bigwedge x. h\ x \geq 0 \ \bigwedge x. gp\ x \geq 0 \ \bigwedge x. gm\ x \geq 0$  unfolding h-def gp-def
gm-def by simp-all
have gp-real:  $\bigwedge x. \text{enn2real}(gp\ x) = \max\ (g\ x)\ 0$ 
unfolding gp-def by (simp add: max-def ennreal-neg)
have gm-real:  $\bigwedge x. \text{enn2real}(gm\ x) = \max\ (-g\ x)\ 0$ 
unfolding gm-def by (simp add: max-def ennreal-neg)

have ( $\int^+ x. \text{norm}(f\ x * \max\ (g\ x)\ 0)\ \partial M$ )  $\leq$  ( $\int^+ x. \text{norm}(f\ x * g\ x)\ \partial M$ )
by (simp add: nn-integral-mono)
also have ...  $< \infty$  using assms(1) by (simp add: integrable-iff-bounded)
finally have ( $\int^+ x. \text{norm}(f\ x * \max\ (g\ x)\ 0)\ \partial M$ )  $< \infty$  by simp
then have int1: integrable M ( $\lambda x. f\ x * \max\ (g\ x)\ 0$ ) by (simp add: inte-
grableI-bounded)

have ( $\int^+ x. \text{norm}(f\ x * \max\ (-g\ x)\ 0)\ \partial M$ )  $\leq$  ( $\int^+ x. \text{norm}(f\ x * g\ x)\ \partial M$ )
by (simp add: nn-integral-mono)
also have ...  $< \infty$  using assms(1) by (simp add: integrable-iff-bounded)
finally have ( $\int^+ x. \text{norm}(f\ x * \max\ (-g\ x)\ 0)\ \partial M$ )  $< \infty$  by simp
then have int2: integrable M ( $\lambda x. f\ x * \max\ (-g\ x)\ 0$ ) by (simp add: inte-
grableI-bounded)

have ( $\int^+ x. f\ x * \text{nn-cond-exp}\ M\ F\ h\ x\ \partial M$ ) = ( $\int^+ x. f\ x * h\ x\ \partial M$ )
by (rule nn-cond-exp-intg) auto
also have ... =  $\int^+ x. \text{ennreal}\ (f\ x * \max\ (g\ x)\ 0 + f\ x * \max\ (-g\ x)\ 0)\ \partial M$ 
unfolding h-def
by (intro nn-integral-cong)(auto simp: ennreal-mult[symmetric] abs-mult split:
split-max)
also have ...  $< \infty$ 
using Bochner-Integration.integrable-add[OF int1 int2, THEN integrableD(2)]
by (auto simp add: less-top)
finally have *: ( $\int^+ x. f\ x * \text{nn-cond-exp}\ M\ F\ h\ x\ \partial M$ )  $< \infty$  by simp

have ( $\int^+ x. \text{norm}(f\ x * \text{real-cond-exp}\ M\ F\ g\ x)\ \partial M$ ) = ( $\int^+ x. f\ x * \text{abs}(\text{real-cond-exp}\ M\ F\ g\ x)\ \partial M$ )
by (simp add: abs-mult)
also have ...  $\leq$  ( $\int^+ x. f\ x * \text{nn-cond-exp}\ M\ F\ h\ x\ \partial M$ )
proof (rule nn-integral-mono-AE)
  {
fix x assume *:  $\text{abs}(\text{real-cond-exp}\ M\ F\ g\ x) \leq \text{nn-cond-exp}\ M\ F\ h\ x$ 
have  $\text{ennreal}\ (f\ x * |\text{real-cond-exp}\ M\ F\ g\ x|) = f\ x * \text{ennreal}(|\text{real-cond-exp}\ M\ F\ g\ x|)$ 
by (simp add: ennreal-mult)
also have ...  $\leq f\ x * \text{nn-cond-exp}\ M\ F\ h\ x$ 
using * by (auto intro!: mult-left-mono)
finally have  $\text{ennreal}\ (f\ x * |\text{real-cond-exp}\ M\ F\ g\ x|) \leq f\ x * \text{nn-cond-exp}\ M$ 

```

```

F h x
  by simp
}
  then show AE x in M. ennreal (f x * |real-cond-exp M F g x|) ≤ f x *
nn-cond-exp M F h x
  using real-cond-exp-abs[OF assms(4)] h-def by auto
qed
  finally have **: (∫+x. norm(f x * real-cond-exp M F g x) ∂M) < ∞ using *
by auto
  show integrable M (λx. f x * real-cond-exp M F g x)
  using ** by (intro integrableI-bounded) auto

  have (∫+x. f x * nn-cond-exp M F gp x ∂M) ≤ (∫+x. f x * nn-cond-exp M F h
x ∂M)
  proof (rule nn-integral-mono-AE)
    have AE x in M. nn-cond-exp M F gp x ≤ nn-cond-exp M F h x
    by (rule nn-cond-exp-mono) (auto simp add: hpggm)
    then show AE x in M. f x * nn-cond-exp M F gp x ≤ f x * nn-cond-exp M F
h x
    by (auto simp: mult-left-mono)
  qed
  then have a: (∫+x. f x * nn-cond-exp M F gp x ∂M) < ∞
  using * by auto
  have ennreal(norm(f x * enn2real(nn-cond-exp M F gp x))) ≤ f x * nn-cond-exp
M F gp x for x
  by (auto simp add: ennreal-mult intro!: mult-left-mono)
  (metis enn2real-ennreal enn2real-nonneg le-cases le-ennreal-iff)
  then have (∫+x. norm(f x * enn2real(nn-cond-exp M F gp x)) ∂M) ≤ (∫+x. f
x * nn-cond-exp M F gp x ∂M)
  by (simp add: nn-integral-mono)
  then have (∫+x. norm(f x * enn2real(nn-cond-exp M F gp x)) ∂M) < ∞ using
a by auto
  then have gp-int: integrable M (λx. f x * enn2real(nn-cond-exp M F gp x)) by
(simp add: integrableI-bounded)
  have gp-fin: AE x in M. f x * nn-cond-exp M F gp x ≠ ∞
  apply (rule nn-integral-PInf-AE) using a by auto

  have (∫ x. f x * enn2real(nn-cond-exp M F gp x) ∂M) = enn2real (∫+x. f x *
enn2real(nn-cond-exp M F gp x) ∂M)
  by (rule integral-eq-nn-integral) auto
  also have ... = enn2real(∫+x. ennreal(f x * enn2real(gp x)) ∂M)
  proof -
    {
      fix x assume f x * nn-cond-exp M F gp x ≠ ∞
      then have ennreal (f x * enn2real (nn-cond-exp M F gp x)) = ennreal (f x)
* nn-cond-exp M F gp x
      by (auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!:
ennreal-mult-left-cong)
    }
  
```

}
then have $AE\ x\ in\ M.$ $ennreal\ (f\ x\ * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gp\ x)) =$
 $ennreal\ (f\ x) * nn\text{-}cond\text{-}exp\ M\ F\ gp\ x$
using $gp\text{-}fin$ **by** $auto$
then have $(\int^+ x. f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gp\ x)\ \partial M) = (\int^+ x. f\ x * \$
 $nn\text{-}cond\text{-}exp\ M\ F\ gp\ x\ \partial M)$
by $(rule\ nn\text{-}integral\text{-}cong\text{-}AE)$
also have $\dots = (\int^+ x. f\ x * \ gp\ x\ \partial M)$
by $(rule\ nn\text{-}cond\text{-}exp\text{-}intg)\ (auto\ simp\ add:\ gp\text{-}def)$
also have $\dots = (\int^+ x. \ ennreal\ (f\ x * \ enn2real\ (gp\ x))\ \partial M)$
by $(rule\ nn\text{-}integral\text{-}cong\text{-}AE)\ (auto\ simp:\ ennreal\text{-}mult\ gp\text{-}def)$
finally have $(\int^+ x. f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gp\ x)\ \partial M) = (\int^+ x.$
 $ennreal\ (f\ x * \ enn2real\ (gp\ x))\ \partial M)$ **by** $simp$
then show $?thesis$ **by** $simp$
qed
also have $\dots = (\int x. f\ x * \ enn2real\ (gp\ x)\ \partial M)$
by $(rule\ integral\text{-}eq\text{-}nn\text{-}integral[symmetric])\ (auto\ simp\ add:\ gp\text{-}def)$
finally have $gp\text{-}expr:$ $(\int x. f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gp\ x)\ \partial M) = (\int x.$
 $f\ x * \ enn2real\ (gp\ x)\ \partial M)$ **by** $simp$

have $(\int^+ x. f\ x * \ nn\text{-}cond\text{-}exp\ M\ F\ gm\ x\ \partial M) \leq (\int^+ x. f\ x * \ nn\text{-}cond\text{-}exp\ M\ F$
 $h\ x\ \partial M)$
proof $(rule\ nn\text{-}integral\text{-}mono\text{-}AE)$
have $AE\ x\ in\ M.$ $nn\text{-}cond\text{-}exp\ M\ F\ gm\ x \leq nn\text{-}cond\text{-}exp\ M\ F\ h\ x$
by $(rule\ nn\text{-}cond\text{-}exp\text{-}mono)\ (auto\ simp\ add:\ hgpgm)$
then show $AE\ x\ in\ M.$ $f\ x * \ nn\text{-}cond\text{-}exp\ M\ F\ gm\ x \leq f\ x * \ nn\text{-}cond\text{-}exp\ M\ F$
 $h\ x$
by $(auto\ simp:\ mult\text{-}left\text{-}mono)$
qed
then have $a:$ $(\int^+ x. f\ x * \ nn\text{-}cond\text{-}exp\ M\ F\ gm\ x\ \partial M) < \infty$
using $*$ **by** $auto$
have $\bigwedge x. \ ennreal\ (norm\ (f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x))) \leq f\ x * \ nn\text{-}cond\text{-}exp$
 $M\ F\ gm\ x$
by $(auto\ simp\ add:\ ennreal\text{-}mult\ intro!:\ mult\text{-}left\text{-}mono)$
 $(metis\ enn2real\text{-}ennreal\ enn2real\text{-}nonneg\ le\text{-}cases\ le\text{-}ennreal\text{-}iff)$
then have $(\int^+ x. \ norm\ (f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x))\ \partial M) \leq (\int^+ x.$
 $f\ x * \ nn\text{-}cond\text{-}exp\ M\ F\ gm\ x\ \partial M)$
by $(simp\ add:\ nn\text{-}integral\text{-}mono)$
then have $(\int^+ x. \ norm\ (f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x))\ \partial M) < \infty$ **using**
 a **by** $auto$
then have $gm\text{-}int:$ $integrable\ M\ (\lambda x. f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x))$ **by**
 $(simp\ add:\ integrableI\text{-}bounded)$
have $gm\text{-}fin:$ $AE\ x\ in\ M.$ $f\ x * \ nn\text{-}cond\text{-}exp\ M\ F\ gm\ x \neq \infty$
apply $(rule\ nn\text{-}integral\text{-}PInf\text{-}AE)$ **using** a **by** $auto$

have $(\int x. f\ x * \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x)\ \partial M) = \ enn2real\ (\int^+ x. f\ x$
 $* \ enn2real\ (nn\text{-}cond\text{-}exp\ M\ F\ gm\ x)\ \partial M)$
by $(rule\ integral\text{-}eq\text{-}nn\text{-}integral)\ auto$

also have $\dots = \text{enn2real}(\int^+ x. \text{ennreal}(f x * \text{enn2real}(g m x)) \partial M)$
proof –
{
 fix x **assume** $f x * \text{nn-cond-exp } M F g m x \neq \infty$
 then have $\text{ennreal}(f x * \text{enn2real}(\text{nn-cond-exp } M F g m x)) = \text{ennreal}(f x)$
 $* \text{nn-cond-exp } M F g m x$
 by (*auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!:*
ennreal-mult-left-cong)
}
 then have $AE\ x\ \text{in } M. \text{ennreal}(f x * \text{enn2real}(\text{nn-cond-exp } M F g m x)) =$
 $\text{ennreal}(f x) * \text{nn-cond-exp } M F g m x$
 using *gm-fin* **by** *auto*
 then have $(\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F g m x) \partial M) = (\int^+ x. f x$
 $* \text{nn-cond-exp } M F g m x \partial M)$
 by (*rule nn-integral-cong-AE*)
 also have $\dots = (\int^+ x. f x * g m x \partial M)$
 by (*rule nn-cond-exp-intg*) (*auto simp add: gm-def*)
 also have $\dots = (\int^+ x. \text{ennreal}(f x * \text{enn2real}(g m x)) \partial M)$
 by (*rule nn-integral-cong-AE*) (*auto simp: ennreal-mult gm-def*)
 finally have $(\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F g m x) \partial M) = (\int^+ x.$
 $\text{ennreal}(f x * \text{enn2real}(g m x)) \partial M)$ **by** *simp*
 then show *?thesis* **by** *simp*
qed
 also have $\dots = (\int x. f x * \text{enn2real}(g m x) \partial M)$
 by (*rule integral-eq-nn-integral[symmetric]*) (*auto simp add: gm-def*)
 finally have *gm-expr*: $(\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g m x) \partial M) = (\int$
 $x. f x * \text{enn2real}(g m x) \partial M)$ **by** *simp*

have $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * \text{enn2real}(\text{nn-cond-exp}$
 $M F g p x) - f x * \text{enn2real}(\text{nn-cond-exp } M F g m x) \partial M)$
 unfolding *real-cond-exp-def gp-def gm-def* **by** (*simp add: right-diff-distrib*)
 also have $\dots = (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g p x) \partial M) - (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g m x) \partial M)$
 by (*rule Bochner-Integration.integral-diff*) (*simp-all add: gp-int gm-int*)
 also have $\dots = (\int x. f x * \text{enn2real}(g p x) \partial M) - (\int x. f x * \text{enn2real}(g m x)$
 $\partial M)$
 using *gp-expr gm-expr* **by** *simp*
 also have $\dots = (\int x. f x * \max(g x) 0 \partial M) - (\int x. f x * \max(-g x) 0 \partial M)$
 using *gp-real gm-real* **by** *simp*
 also have $\dots = (\int x. f x * \max(g x) 0 - f x * \max(-g x) 0 \partial M)$
 by (*rule Bochner-Integration.integral-diff[symmetric]*) (*simp-all add: int1 int2*)
 also have $\dots = (\int x. f x * g x \partial M)$
 by (*metis (mono-tags, opaque-lifting) diff-0 diff-zero eq-iff max.cobounded2*
max-def minus-minus neg-le-0-iff-le right-diff-distrib)
 finally show $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$
 by *simp*
qed

lemma *real-cond-exp-intg*:

assumes *integrable* M $(\lambda x. f x * g x)$ **and**

[*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$

shows *integrable* M $(\lambda x. f x * \text{real-cond-exp } M F g x)$

$(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$

proof –

have [*measurable*]: $f \in \text{borel-measurable } M$ **by** (*rule measurable-from-subalg[OF subalg assms(2)]*)

define *fp* **where** $fp = (\lambda x. \max (f x) 0)$

define *fm* **where** $fm = (\lambda x. \max (-f x) 0)$

have [*measurable*]: $fp \in \text{borel-measurable } M$ $fm \in \text{borel-measurable } M$

unfolding *fp-def fm-def* **by** *simp-all*

have [*measurable*]: $fp \in \text{borel-measurable } F$ $fm \in \text{borel-measurable } F$

unfolding *fp-def fm-def* **by** *simp-all*

have $(\int^+ x. \text{norm}(fp x * g x) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: fp-def nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(fp x * g x) \partial M) < \infty$ **by** *simp*

then have *intp*: *integrable* M $(\lambda x. fp x * g x)$ **by** (*simp add: integrableI-bounded*)

moreover have $\bigwedge x. fp x \geq 0$ **unfolding** *fp-def* **by** *simp*

ultimately have *Rp*: *integrable* M $(\lambda x. fp x * \text{real-cond-exp } M F g x)$

$(\int x. fp x * \text{real-cond-exp } M F g x \partial M) = (\int x. fp x * g x \partial M)$

using *real-cond-exp-intg-fpos* **by** *auto*

have $(\int^+ x. \text{norm}(fm x * g x) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (*simp add: fm-def nn-integral-mono*)

also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)

finally have $(\int^+ x. \text{norm}(fm x * g x) \partial M) < \infty$ **by** *simp*

then have *intm*: *integrable* M $(\lambda x. fm x * g x)$ **by** (*simp add: integrableI-bounded*)

moreover have $\bigwedge x. fm x \geq 0$ **unfolding** *fm-def* **by** *simp*

ultimately have *Rm*: *integrable* M $(\lambda x. fm x * \text{real-cond-exp } M F g x)$

$(\int x. fm x * \text{real-cond-exp } M F g x \partial M) = (\int x. fm x * g x \partial M)$

using *real-cond-exp-intg-fpos* **by** *auto*

have *integrable* M $(\lambda x. fp x * \text{real-cond-exp } M F g x - fm x * \text{real-cond-exp } M F g x)$

using *Rp(1) Rm(1) integrable-diff* **by** *simp*

moreover have $*$: $\bigwedge x. f x * \text{real-cond-exp } M F g x = fp x * \text{real-cond-exp } M F g x - fm x * \text{real-cond-exp } M F g x$

unfolding *fp-def fm-def* **by** (*simp add: max-def*)

ultimately show *integrable* M $(\lambda x. f x * \text{real-cond-exp } M F g x)$

by *simp*

have $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. fp x * \text{real-cond-exp } M F g x - fm x * \text{real-cond-exp } M F g x \partial M)$

using $*$ **by** *simp*

also have $\dots = (\int x. fp x * \text{real-cond-exp } M F g x \partial M) - (\int x. fm x * \text{real-cond-exp } M F g x \partial M)$

using $Rp(1)$ $Rm(1)$ **by** *simp*
also have $\dots = (\int x. fp\ x * g\ x\ \partial M) - (\int x. fm\ x * g\ x\ \partial M)$
using $Rp(2)$ $Rm(2)$ **by** *simp*
also have $\dots = (\int x. fp\ x * g\ x - fm\ x * g\ x\ \partial M)$
using *intm intp* **by** *simp*
also have $\dots = (\int x. f\ x * g\ x\ \partial M)$
unfolding *fp-def fm-def* **by** (*metis (no-types, opaque-lifting) diff-0 diff-zero max commute*
max-def minus-minus mult commute neg-le-iff-le right-diff-distrib)
finally show $(\int x. f\ x * real\ cond\ exp\ M\ F\ g\ x\ \partial M) = (\int x. f\ x * g\ x\ \partial M)$ **by**
simp
qed

lemma *real-cond-exp-intA*:
assumes [*measurable*]: *integrable M f A ∈ sets F*
shows $(\int x \in A. f\ x\ \partial M) = (\int x \in A. real\ cond\ exp\ M\ F\ f\ x\ \partial M)$
proof –
have $A \in sets\ M$ **by** (*meson assms(2) subalg subalgebra-def subsetD*)
have *integrable M* ($\lambda x. indicator\ A\ x * f\ x$) **using** *integrable-mult-indicator[OF*
 $\langle A \in sets\ M \rangle$ *assms(1)] by auto*
then show *?thesis* **using** *real-cond-exp-intg(2)[where ?f = indicator A and ?g*
 $= f, symmetric]$
unfolding *set-lebesgue-integral-def* **by** *auto*
qed

lemma *real-cond-exp-int [intro]*:
assumes *integrable M f*
shows *integrable M* (*real-cond-exp M F f*) $(\int x. real\ cond\ exp\ M\ F\ f\ x\ \partial M) =$
 $(\int x. f\ x\ \partial M)$
using *real-cond-exp-intg[where ?f = $\lambda x. 1$ and ?g = f] assms by auto*

lemma *real-cond-exp-charact*:
assumes $\bigwedge A. A \in sets\ F \implies (\int x \in A. f\ x\ \partial M) = (\int x \in A. g\ x\ \partial M)$
and [*measurable*]: *integrable M f integrable M g*
 $g \in borel\ measurable\ F$
shows *AE x in M. real-cond-exp M F f x = g x*
proof –
let $?MF = restr\ to\ subalg\ M\ F$
have *AE x in ?MF. real-cond-exp M F f x = g x*
proof (*rule AE-symmetric[OF density-unique-real]*)
fix A **assume** $A \in sets\ ?MF$
then have [*measurable*]: $A \in sets\ F$ **using** *sets-restr-to-subalg[OF subalg] by*
simp
then have a [*measurable*]: $A \in sets\ M$ **by** (*meson subalg subalgebra-def subsetD*)
have $(\int x \in A. g\ x\ \partial ?MF) = (\int x \in A. g\ x\ \partial M)$
unfolding *set-lebesgue-integral-def* **by** (*simp add: integral-subalgebra2 subalg*)
also have $\dots = (\int x \in A. f\ x\ \partial M)$ **using** *assms(1) by simp*
also have $\dots = (\int x. indicator\ A\ x * f\ x\ \partial M)$ **by** (*simp add: mult commute*
set-lebesgue-integral-def)

also have ... = $(\int x. \text{indicator } A \ x * \text{real-cond-exp } M \ F \ f \ x \ \partial M)$
apply (rule *real-cond-exp-intg(2)[symmetric]*) **using** *integrable-mult-indicator[OF a assms(2)]* **by** (*auto simp add: assms*)
also have ... = $(\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial M)$ **by** (*simp add: mult.commute set-lebesgue-integral-def*)
also have ... = $(\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial \ ?MF)$
by (*simp add: integral-subalgebra2 subalg set-lebesgue-integral-def*)
finally show $(\int x \in A. g \ x \ \partial \ ?MF) = (\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial \ ?MF)$
by simp
next
have *integrable* M (*real-cond-exp* $M \ F \ f$) **by** (rule *real-cond-exp-int(1)[OF assms(2)]*)
then show *integrable* $?MF$ (*real-cond-exp* $M \ F \ f$) **by** (*metis borel-measurable-cond-exp integrable-in-subalg[OF subalg]*)
show *integrable* (*restr-to-subalg* $M \ F$) g **by** (*simp add: assms(3) integrable-in-subalg[OF subalg]*)
qed
then show *?thesis* **using** *AE-restr-to-subalg[OF subalg]* **by auto**
qed

lemma *real-cond-exp-F-meas* [*intro, simp*]:

assumes *integrable* $M \ f$
 $f \in \text{borel-measurable } F$
shows *AE* x *in* M . *real-cond-exp* $M \ F \ f \ x = f \ x$
by (rule *real-cond-exp-charact, auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma *real-cond-exp-mult*:

assumes [*measurable*]: $f \in \text{borel-measurable } F \ g \in \text{borel-measurable } M$ *integrable* M $(\lambda x. f \ x * g \ x)$
shows *AE* x *in* M . *real-cond-exp* $M \ F$ $(\lambda x. f \ x * g \ x) \ x = f \ x * \text{real-cond-exp } M \ F \ g \ x$
proof (rule *real-cond-exp-charact*)
fix A **assume** $A \in \text{sets } F$
then have [*measurable*]: $(\lambda x. f \ x * \text{indicator } A \ x) \in \text{borel-measurable } F$ **by** *measurable*
have [*measurable*]: $A \in \text{sets } M$ **using** *subalg* **by** (*meson* $\langle A \in \text{sets } F \rangle$ *subalgebra-def subsetD*)
have $\int x \in A. (f \ x * g \ x) \ \partial M = \int x. (f \ x * \text{indicator } A \ x) * g \ x \ \partial M$
by (*simp add: mult.commute mult.left-commute set-lebesgue-integral-def*)
also have ... = $\int x. (f \ x * \text{indicator } A \ x) * \text{real-cond-exp } M \ F \ g \ x \ \partial M$
apply (rule *real-cond-exp-intg(2)[symmetric]*, *auto simp add: assms*)
using *integrable-mult-indicator[OF* $\langle A \in \text{sets } M \rangle$ *assms(3)]* **by** (*simp add: mult.commute mult.left-commute*)
also have ... = $\int x \in A. (f \ x * \text{real-cond-exp } M \ F \ g \ x) \ \partial M$
by (*simp add: mult.commute mult.left-commute set-lebesgue-integral-def*)
finally show $\int x \in A. (f \ x * g \ x) \ \partial M = \int x \in A. (f \ x * \text{real-cond-exp } M \ F \ g \ x) \ \partial M$
by simp
qed (*auto simp add: real-cond-exp-intg(1) assms*)

lemma *real-cond-exp-add* [*intro*]:
assumes [*measurable*]: *integrable M f integrable M g*
shows $\text{AE } x \text{ in } M. \text{real-cond-exp } M F (\lambda x. f x + g x) x = \text{real-cond-exp } M F f x$
 $+ \text{real-cond-exp } M F g x$
proof (*rule real-cond-exp-charact*)
have *integrable M (real-cond-exp M F f) integrable M (real-cond-exp M F g)*
using *real-cond-exp-int(1) assms by auto*
then show *integrable M ($\lambda x. \text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x$)*
by auto

fix *A* **assume** [*measurable*]: $A \in \text{sets } F$
then have $A \in \text{sets } M$ **by** (*meson subalg subalgebra-def subsetD*)
have *intAf: integrable M ($\lambda x. \text{indicator } A x * f x$)*
using *integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ assms(1)] by auto*
have *intAg: integrable M ($\lambda x. \text{indicator } A x * g x$)*
using *integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ assms(2)] by auto*

have $\int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M = (\int x \in A. \text{real-cond-exp } M F f x \partial M) + (\int x \in A. \text{real-cond-exp } M F g x \partial M)$
apply (*rule set-integral-add, auto simp add: assms set-integrable-def*)
using *integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ real-cond-exp-int(1)[OF assms(1)]]*
integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ real-cond-exp-int(1)[OF
assms(2)]] by simp-all
also have $\dots = (\int x. \text{indicator } A x * \text{real-cond-exp } M F f x \partial M) + (\int x. \text{indicator } A x * \text{real-cond-exp } M F g x \partial M)$
unfolding *set-lebesgue-integral-def by auto*
also have $\dots = (\int x. \text{indicator } A x * f x \partial M) + (\int x. \text{indicator } A x * g x \partial M)$
using *real-cond-exp-intg(2) assms $\langle A \in \text{sets } F \rangle$ intAf intAg by auto*
also have $\dots = (\int x \in A. f x \partial M) + (\int x \in A. g x \partial M)$
unfolding *set-lebesgue-integral-def by auto*
also have $\dots = \int x \in A. (f x + g x) \partial M$
by (*rule set-integral-add(2)[symmetric]*) (*auto simp add: assms set-integrable-def $\langle A \in \text{sets } M \rangle$ intAf intAg*)
finally show $\int x \in A. (f x + g x) \partial M = \int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M$
by simp
qed (*auto simp add: assms*)

lemma *real-cond-exp-cong*:
assumes *ae: AE x in M. f x = g x and* [*measurable*]: $f \in \text{borel-measurable } M g \in \text{borel-measurable } M$
shows $\text{AE } x \text{ in } M. \text{real-cond-exp } M F f x = \text{real-cond-exp } M F g x$
proof –
have $\text{AE } x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (g x)) x$
apply (*rule nn-cond-exp-cong*) **using** *assms by auto*
moreover have $\text{AE } x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-g x)) x$

apply (rule *nn-cond-exp-cong*) **using** *assms* **by** *auto*
ultimately show *AE x in M. real-cond-exp M F f x = real-cond-exp M F g x*
unfolding *real-cond-exp-def* **by** *auto*
qed

lemma *real-cond-exp-cmult* [*intro, simp*]:
fixes *c::real*
assumes *integrable M f*
shows *AE x in M. real-cond-exp M F (λx. c * f x) x = c * real-cond-exp M F f x*
by (rule *real-cond-exp-mult*[**where** *?f = λx. c* **and** *?g = f*], *auto simp add: assms borel-measurable-integrable*)

lemma *real-cond-exp-cdiv* [*intro, simp*]:
fixes *c::real*
assumes *integrable M f*
shows *AE x in M. real-cond-exp M F (λx. f x / c) x = real-cond-exp M F f x / c*
using *real-cond-exp-cmult*[*of - 1/c, OF assms*] **by** (*auto simp add: field-split-simps*)

lemma *real-cond-exp-diff* [*intro, simp*]:
assumes [*measurable*]: *integrable M f integrable M g*
shows *AE x in M. real-cond-exp M F (λx. f x - g x) x = real-cond-exp M F f x - real-cond-exp M F g x*
proof –
have *AE x in M. real-cond-exp M F (λx. f x + (- g x)) x = real-cond-exp M F f x + real-cond-exp M F (λx. -g x) x*
using *real-cond-exp-add*[**where** *?f = f* **and** *?g = λx. - g x*] *assms* **by** *auto*
moreover have *AE x in M. real-cond-exp M F (λx. -g x) x = - real-cond-exp M F g x*
using *real-cond-exp-cmult*[**where** *?f = g* **and** *?c = -1*] *assms(2)* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-pos* [*intro*]:
assumes *AE x in M. f x ≥ 0* **and** [*measurable*]: *f ∈ borel-measurable M*
shows *AE x in M. real-cond-exp M F f x ≥ 0*
proof –
define *g* **where** *g = (λx. max (f x) 0)*
have *AE x in M. f x = g x* **using** *assms g-def* **by** *auto*
then have *: *AE x in M. real-cond-exp M F f x = real-cond-exp M F g x* **using** *real-cond-exp-cong g-def* **by** *auto*

have $\bigwedge x. g x \geq 0$ **unfolding** *g-def* **by** *simp*
then have $(\lambda x. \text{ennreal}(-g x)) = (\lambda x. 0)$
by (*simp add: ennreal-neg*)
moreover have *AE x in M. 0 = nn-cond-exp M F (λx. 0) x*
by (rule *nn-cond-exp-F-meas, auto*)
ultimately have *AE x in M. nn-cond-exp M F (λx. ennreal(-g x)) x = 0*

by *simp*
then have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x = enn2real(nn\ cond\ exp\ M\ F\ (\lambda x.\ ennreal\ (g\ x))\ x)$
unfolding *real-cond-exp-def* **by** *auto*
then have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x \geq 0$ **by** *auto*
then show *?thesis* **using** *** **by** *auto*
qed

lemma *real-cond-exp-mono*:

assumes $AE\ x\ in\ M.\ f\ x \leq g\ x$ **and** $[measurable]:\ integrable\ M\ f\ integrable\ M\ g$
shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ f\ x \leq real\ cond\ exp\ M\ F\ g\ x$
proof –
have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x - real\ cond\ exp\ M\ F\ f\ x = real\ cond\ exp\ M\ F\ (\lambda x.\ g\ x - f\ x)\ x$
by $(rule\ AE\ symmetric[OF\ real\ cond\ exp\ diff],\ auto\ simp\ add:\ assms)$
moreover have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ g\ x - f\ x)\ x \geq 0$
by $(rule\ real\ cond\ exp\ pos,\ auto\ simp\ add:\ assms(1))$
ultimately have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x - real\ cond\ exp\ M\ F\ f\ x \geq 0$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma $(in\ -)\ measurable\ P\ restriction\ [measurable\ (raw)]$:

assumes $[measurable]:\ Measurable.pred\ M\ P\ A \in sets\ M$
shows $\{x \in A.\ P\ x\} \in sets\ M$
proof –
have $A \subseteq space\ M$ **using** $sets.sets\ into\ space[OF\ assms(2)]$.
then have $\{x \in A.\ P\ x\} = A \cap \{x \in space\ M.\ P\ x\}$ **by** *blast*
then show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-gr-c*:

assumes $[measurable]:\ integrable\ M\ f$
and $AE:\ AE\ x\ in\ M.\ f\ x > c$
shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ f\ x > c$
proof –
define X **where** $X = \{x \in space\ M.\ real\ cond\ exp\ M\ F\ f\ x \leq c\}$
have $[measurable]:\ X \in sets\ F$
unfolding $X\ def$ **apply** *measurable* **by** $(metis\ sets.top\ subalg\ subalgebra\ def)$
then have $[measurable]:\ X \in sets\ M$ **using** *sets-restr-to-subalg subalg subalgebra-def* **by** *blast*
have $emeasure\ M\ X = 0$
proof $(rule\ ccontr)$
assume $\neg(emeasure\ M\ X) = 0$
have $emeasure\ (restr\ to\ subalg\ M\ F)\ X = emeasure\ M\ X$
by $(simp\ add:\ emeasure\ restr\ to\ subalg\ subalg)$
then have $emeasure\ (restr\ to\ subalg\ M\ F)\ X > 0$
using $\langle \neg(emeasure\ M\ X) = 0 \rangle\ gr\ zeroI$ **by** *auto*
then obtain A **where** $A \in sets\ (restr\ to\ subalg\ M\ F)\ A \subseteq X$ $emeasure$

```

(restr-to-subalg M F) A > 0 emeasure (restr-to-subalg M F) A < ∞
  using sigma-fin-subalg by (metis emeasure-notin-sets ennreal-0 infinity-ennreal-def
le-less-linear neq-top-trans
  not-gr-zero order-refl sigma-finite-measure.approx-PInf-emeasure-with-finite)
  then have [measurable]: A ∈ sets F using subalg sets-restr-to-subalg by blast
  then have [measurable]: A ∈ sets M using sets-restr-to-subalg subalg subalge-
bra-def by blast
  have Ic: set-integrable M A (λx. c)
  unfolding set-integrable-def
  using ⟨emeasure (restr-to-subalg M F) A < ∞⟩ emeasure-restr-to-subalg subalg
by fastforce
  have If: set-integrable M A f
  unfolding set-integrable-def
  by (rule integrable-mult-indicator, auto simp add: ⟨integrable M f⟩)
  have AE x in M. indicator A x * c = indicator A x * f x
  proof (rule integral-ineq-eq-0-then-AE)
  have (∫ x∈A. c ∂M) = (∫ x∈A. f x ∂M)
  proof (rule antisym)
  show (∫ x∈A. c ∂M) ≤ (∫ x∈A. f x ∂M)
  apply (rule set-integral-mono-AE) using Ic If assms(2) by auto
  have (∫ x∈A. f x ∂M) = (∫ x∈A. real-cond-exp M F f x ∂M)
  by (rule real-cond-exp-intA, auto simp add: ⟨integrable M f⟩)
  also have ... ≤ (∫ x∈A. c ∂M)
  apply (rule set-integral-mono)
  unfolding set-integrable-def
  apply (rule integrable-mult-indicator, simp, simp add: real-cond-exp-int(1)[OF
⟨integrable M f⟩])
  using Ic X-def ⟨A ⊆ X⟩ by (auto simp: set-integrable-def)
  finally show (∫ x∈A. f x ∂M) ≤ (∫ x∈A. c ∂M) by simp
qed
  then have measure M A * c = LINT x|M. indicat-real A x * f x
  by (auto simp: set-lebesgue-integral-def)
  then show LINT x|M. indicat-real A x * c = LINT x|M. indicat-real A x *
f x
  by auto
  show AE x in M. indicat-real A x * c ≤ indicat-real A x * f x
  using AE unfolding indicator-def by auto
  qed (use Ic If in ⟨auto simp: set-integrable-def⟩)
  then have AE x∈A in M. c = f x by auto
  then have AE x∈A in M. False using assms(2) by auto
  have A ∈ null-sets M unfolding ae-filter-def by (meson AE-iff-null-sets ⟨A
∈ sets M⟩ ⟨AE x∈A in M. False⟩)
  then show False using ⟨emeasure (restr-to-subalg M F) A > 0⟩
  by (simp add: emeasure-restr-to-subalg null-setsD1 subalg)
  qed
  then show ?thesis using AE-iff-null-sets[OF ⟨X ∈ sets M⟩] unfolding X-def
by auto
  qed

```

lemma *real-cond-exp-less-c*:

assumes *[measurable]: integrable M f*

and *AE x in M. f x < c*

shows *AE x in M. real-cond-exp M F f x < c*

proof –

have *AE x in M. real-cond-exp M F f x = -real-cond-exp M F (λx. -f x) x*

using *real-cond-exp-cmult[OF ⟨integrable M f⟩, of -1]* **by** *auto*

moreover have *AE x in M. real-cond-exp M F (λx. -f x) x > -c*

apply *(rule real-cond-exp-gr-c)* **using** *assms* **by** *auto*

ultimately show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-ge-c*:

assumes *[measurable]: integrable M f*

and *AE x in M. f x ≥ c*

shows *AE x in M. real-cond-exp M F f x ≥ c*

proof –

obtain *u::nat ⇒ real where u: ∧n. u n < c u → c*

using *approx-from-below-dense-linorder[of c-1 c]* **by** *auto*

have **: AE x in M. real-cond-exp M F f x > u n* **for** *n::nat*

apply *(rule real-cond-exp-gr-c)* **using** *assms ⟨u n < c⟩* **by** *auto*

have *AE x in M. ∀n. real-cond-exp M F f x > u n*

by *(subst AE-all-countable, auto simp add: *)*

moreover have *real-cond-exp M F f x ≥ c* **if** *∀n. real-cond-exp M F f x > u n*

for *x*

proof –

have *real-cond-exp M F f x ≥ u n* **for** *n* **using** *that less-imp-le* **by** *auto*

then show *?thesis* **using** *u(2) LIMSEQ-le-const2* **by** *metis*

qed

ultimately show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-le-c*:

assumes *[measurable]: integrable M f*

and *AE x in M. f x ≤ c*

shows *AE x in M. real-cond-exp M F f x ≤ c*

proof –

have *AE x in M. real-cond-exp M F f x = -real-cond-exp M F (λx. -f x) x*

using *real-cond-exp-cmult[OF ⟨integrable M f⟩, of -1]* **by** *auto*

moreover have *AE x in M. real-cond-exp M F (λx. -f x) x ≥ -c*

apply *(rule real-cond-exp-ge-c)* **using** *assms* **by** *auto*

ultimately show *?thesis* **by** *auto*

qed

lemma *real-cond-exp-mono-strict*:

assumes *AE x in M. f x < g x* **and** *[measurable]: integrable M f integrable M g*

shows *AE x in M. real-cond-exp M F f x < real-cond-exp M F g x*

proof –

have *AE x in M. real-cond-exp M F g x - real-cond-exp M F f x = real-cond-exp*

$M F (\lambda x. g x - f x) x$
 by (rule *AE-symmetric*[*OF real-cond-exp-diff*], *auto simp add: assms*)
 moreover have *AE x in M. real-cond-exp M F* $(\lambda x. g x - f x) x > 0$
 by (rule *real-cond-exp-gr-c*, *auto simp add: assms*)
 ultimately have *AE x in M. real-cond-exp M F* $g x - \text{real-cond-exp } M F f x > 0$ by *auto*
 then show *?thesis* by *auto*
 qed

lemma *real-cond-exp-nested-subalg* [*intro, simp*]:
 assumes *subalgebra M G subalgebra G F*
 and [*measurable*]: *integrable M f*
 shows *AE x in M. real-cond-exp M F* $(\text{real-cond-exp } M G f) x = \text{real-cond-exp } M F f x$
proof (rule *real-cond-exp-charact*)
 interpret *G: sigma-finite-subalgebra M G* by (rule *nested-subalg-is-sigma-finite*[*OF assms(1) assms(2)*])
 show *integrable M* $(\text{real-cond-exp } M G f)$ by (*auto simp add: assms G.real-cond-exp-int(1)*)

fix *A* assume [*measurable*]: $A \in \text{sets } F$
 then have [*measurable*]: $A \in \text{sets } G$ using *assms(2)* by (*meson subsetD subalgebra-def*)
 have *set-lebesgue-integral M A* $(\text{real-cond-exp } M G f) = \text{set-lebesgue-integral } M A f$
 by (rule *G.real-cond-exp-intA[symmetric]*, *auto simp add: assms(3)*)
 also have $\dots = \text{set-lebesgue-integral } M A (\text{real-cond-exp } M F f)$
 by (rule *real-cond-exp-intA*, *auto simp add: assms(3)*)
 finally show *set-lebesgue-integral M A* $(\text{real-cond-exp } M G f) = \text{set-lebesgue-integral } M A (\text{real-cond-exp } M F f)$ by *auto*
 qed (*auto simp add: assms real-cond-exp-int(1)*)

lemma *real-cond-exp-sum* [*intro, simp*]:
 fixes $f::'b \Rightarrow 'a \Rightarrow \text{real}$
 assumes [*measurable*]: $\bigwedge i. \text{integrable } M (f i)$
 shows *AE x in M. real-cond-exp M F* $(\lambda x. \sum i \in I. f i x) x = (\sum i \in I. \text{real-cond-exp } M F (f i) x)$
proof (rule *real-cond-exp-charact*)
 fix *A* assume [*measurable*]: $A \in \text{sets } F$
 then have *A-meas* [*measurable*]: $A \in \text{sets } M$ by (*meson subsetD subalg subalgebra-def*)
 have *: *integrable M* $(\lambda x. \text{indicator } A x * f i x)$ for *i*
 using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *assms(1)*] by *auto*
 have **: *integrable M* $(\lambda x. \text{indicator } A x * \text{real-cond-exp } M F (f i) x)$ for *i*
 using *integrable-mult-indicator*[*OF* $\langle A \in \text{sets } M \rangle$ *real-cond-exp-int(1)*][*OF assms(1)*]]
 by *auto*
 have *inti*: $(\int x. \text{indicator } A x * f i x \partial M) = (\int x. \text{indicator } A x * \text{real-cond-exp } M F (f i) x \partial M)$ for *i*
 by (rule *real-cond-exp-intg(2)[symmetric]*, *auto simp add: **)

```

have ( $\int x \in A. (\sum i \in I. f i x) \partial M$ ) = ( $\int x. (\sum i \in I. \text{indicator } A x * f i x) \partial M$ )
  by (simp add: sum-distrib-left set-lebesgue-integral-def)
also have ... = ( $\sum i \in I. (\int x. \text{indicator } A x * f i x \partial M)$ )
  by (rule Bochner-Integration.integral-sum, simp add: *)
also have ... = ( $\sum i \in I. (\int x. \text{indicator } A x * \text{real-cond-exp } M F (f i) x \partial M)$ )
  using inti by auto
also have ... = ( $\int x. (\sum i \in I. \text{indicator } A x * \text{real-cond-exp } M F (f i) x) \partial M$ )
  by (rule Bochner-Integration.integral-sum[symmetric], simp add: **)
also have ... = ( $\int x \in A. (\sum i \in I. \text{real-cond-exp } M F (f i) x) \partial M$ )
  by (simp add: sum-distrib-left set-lebesgue-integral-def)
finally show ( $\int x \in A. (\sum i \in I. f i x) \partial M$ ) = ( $\int x \in A. (\sum i \in I. \text{real-cond-exp } M F (f i) x) \partial M$ ) by auto
qed (auto simp add: assms real-cond-exp-int(1)[OF assms(1)])

```

Jensen’s inequality, describing the behavior of the integral under a convex function, admits a version for the conditional expectation, as follows.

theorem *real-cond-exp-jensens-inequality:*

```

fixes q :: real  $\Rightarrow$  real
assumes X: integrable M X AE x in M. X x  $\in$  I
assumes I: I = {a <..< b}  $\vee$  I = {a <..<}  $\vee$  I = {..< b}  $\vee$  I = UNIV
assumes q: integrable M ( $\lambda x. q (X x)$ ) convex-on I q q  $\in$  borel-measurable borel
shows AE x in M. real-cond-exp M F X x  $\in$  I
  AE x in M. q (real-cond-exp M F X x)  $\leq$  real-cond-exp M F ( $\lambda x. q (X x)$ ) x
proof –
  have open I using I by auto
  then have interior I = I by (simp add: interior-eq)
  have [measurable]: I  $\in$  sets borel using I by auto
  define phi where phi = ( $\lambda x. \text{Inf } ((\lambda t. (q x - q t) / (x - t)) ' (\{x <..<\} \cap I))$ )
  have **: q (X x)  $\geq$  q (real-cond-exp M F X x) + phi (real-cond-exp M F X x) * (X x - real-cond-exp M F X x)
  if X x  $\in$  I real-cond-exp M F X x  $\in$  I for x
  unfolding phi-def apply (rule convex-le-Inf-differential)
  using <convex-on I q> that <interior I = I> by auto

```

It is not clear that the function ϕ is measurable. We replace it by a version which is better behaved.

```

define psi where psi = ( $\lambda x. \text{phi } x * \text{indicator } I x$ )
have A: psi y = phi y if y  $\in$  I for y unfolding psi-def indicator-def using that by auto
have *: q (X x)  $\geq$  q (real-cond-exp M F X x) + psi (real-cond-exp M F X x) * (X x - real-cond-exp M F X x)
  if X x  $\in$  I real-cond-exp M F X x  $\in$  I for x
  unfolding A[OF <real-cond-exp M F X x  $\in$  I>] using ** that by auto

```

note *I*

```

moreover have AE x in M. real-cond-exp M F X x > a if I  $\subseteq$  {a <..<} for a
  apply (rule real-cond-exp-gr-c) using X that by auto
moreover have AE x in M. real-cond-exp M F X x < b if I  $\subseteq$  {..< b} for b
  apply (rule real-cond-exp-less-c) using X that by auto

```


ultimately show $AE\ x\ in\ M.$ $real\ cond\ exp\ M\ F\ X\ x\ \in\ I$
by $(elim\ disjE)$ $(auto\ simp:\ subset\ eq)$
then have $main\ ineq:$ $AE\ x\ in\ M.$ $q\ (X\ x)\ \geq\ q\ (real\ cond\ exp\ M\ F\ X\ x)\ +\ psi$
 $(real\ cond\ exp\ M\ F\ X\ x)\ * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x)$
using $*\ X(2)$ **by** $auto$

Then, one wants to take the conditional expectation of this inequality. On the left, one gets the conditional expectation of $q \circ X$. On the right, the last term vanishes, and one is left with q of the conditional expectation, as desired. Unfortunately, this argument only works if $\psi \cdot X$ and $q(E(X|F))$ are integrable, and there is no reason why this should be true. The trick is to multiply by a F -measurable function which is small enough to make everything integrable.

obtain $f::'a\ \Rightarrow\ real$ **where** $[measurable]:\ f\ \in\ borel\ measurable\ (restr\ to\ subalg\ M\ F)$
 $integrable\ (restr\ to\ subalg\ M\ F)\ f$
and $f:$ $\bigwedge x. f\ x\ >\ 0\ \bigwedge x. f\ x\ \leq\ 1$
using $sigma\ finite\ measure.obtain\ positive\ integrable\ function[OF\ sigma\ fin\ subalg]$
by $metis$
then have $[measurable]:\ f\ \in\ borel\ measurable\ F$ **by** $(simp\ add:\ subalg)$
then have $[measurable]:\ f\ \in\ borel\ measurable\ M$ **using** $measurable\ from\ subalg[OF\ subalg]$ **by** $blast$
define g **where** $g\ =\ (\lambda x. f\ x / (1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|))$
define G **where** $G\ =\ (\lambda x. g\ x * psi\ (real\ cond\ exp\ M\ F\ X\ x))$
have $g:$ $g\ x\ >\ 0\ g\ x\ \leq\ 1$ **for** x **unfolding** $G\ def\ g\ def$ **using** $f[of\ x]$ **by** $(auto\ simp\ add:\ abs\ mult)$
have $G:$ $|G\ x|\ \leq\ 1$ **for** x **unfolding** $G\ def\ g\ def$ **using** $f[of\ x]$
proof $(auto\ simp\ add:\ abs\ mult)$
have $f\ x * |psi\ (real\ cond\ exp\ M\ F\ X\ x)|\ \leq\ 1 * |psi\ (real\ cond\ exp\ M\ F\ X\ x)|$
apply $(rule\ mult\ mono)$ **using** $f[of\ x]$ **by** $auto$
also have $\dots\ \leq\ 1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|$ **by** $auto$
finally show $f\ x * |psi\ (real\ cond\ exp\ M\ F\ X\ x)|\ \leq\ 1 + |psi\ (real\ cond\ exp\ M\ F\ X\ x)| + |q\ (real\ cond\ exp\ M\ F\ X\ x)|$
by $simp$
qed
have $AE\ x\ in\ M.$ $g\ x * q\ (X\ x)\ \geq\ g\ x * (q\ (real\ cond\ exp\ M\ F\ X\ x)\ +\ psi$
 $(real\ cond\ exp\ M\ F\ X\ x)\ * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x))$
using $main\ ineq\ g$ **by** $(auto\ simp\ add:\ divide_simps)$
then have $main\ G:$ $AE\ x\ in\ M.$ $g\ x * q\ (X\ x)\ \geq\ g\ x * q\ (real\ cond\ exp\ M\ F\ X\ x)$
 $+ G\ x * (X\ x\ -\ real\ cond\ exp\ M\ F\ X\ x)$
unfolding $G\ def$ **by** $(auto\ simp\ add:\ algebra_simps)$

To proceed, we need to know that ψ is measurable.

have $phi\ mono:$ $phi\ x\ \leq\ phi\ y$ **if** $x\ \leq\ y$ $x\ \in\ I\ y\ \in\ I$ **for** $x\ y$
proof $(cases\ x\ <\ y)$
case $True$

```

have  $q x + \text{phi } x * (y-x) \leq q y$ 
  unfolding phi-def apply (rule convex-le-Inf-differential) using  $\langle \text{convex-on } I$ 
 $q \rangle$  that  $\langle \text{interior } I = I \rangle$  by auto
  then have  $\text{phi } x \leq (q x - q y) / (x - y)$ 
    using that  $\langle x < y \rangle$  by (auto simp add: field-split-simps algebra-simps)
  moreover have  $(q x - q y) / (x - y) \leq \text{phi } y$ 
  unfolding phi-def proof (rule cInf-greatest, auto)
    fix  $t$  assume  $t \in I y < t$ 
    have  $(q x - q y) / (x - y) \leq (q x - q t) / (x - t)$ 
      apply (rule convex-on-diff[OF q(2)]) using  $\langle y < t \rangle \langle x < y \rangle \langle t \in I \rangle \langle x \in$ 
 $I \rangle$  by auto
    also have  $\dots \leq (q y - q t) / (y - t)$ 
      apply (rule convex-on-diff[OF q(2)]) using  $\langle y < t \rangle \langle x < y \rangle \langle t \in I \rangle \langle x \in$ 
 $I \rangle$  by auto
    finally show  $(q x - q y) / (x - y) \leq (q y - q t) / (y - t)$  by simp
  next
  obtain  $e$  where  $0 < e$   $\text{ball } y e \subseteq I$  using  $\langle \text{open } I \rangle \langle y \in I \rangle$  openE by blast
  then have  $y + e/2 \in \{y<..\} \cap I$  by (auto simp: dist-real-def)
  then show  $\{y<..\} \cap I = \{\} \implies \text{False}$  by auto
qed
ultimately show  $\text{phi } x \leq \text{phi } y$  by auto
next
case False
  then have  $x = y$  using  $\langle x \leq y \rangle$  by auto
  then show ?thesis by auto
qed
have [measurable]:  $\text{psi} \in \text{borel-measurable borel}$ 
  by (rule borel-measurable-piecewise-mono[of  $\{I, -I\}$ ])
  (auto simp add: psi-def indicator-def phi-mono intro: mono-onI)
have [measurable]:  $q \in \text{borel-measurable borel}$  using  $q$  by simp

have [measurable]:  $X \in \text{borel-measurable } M$ 
   $\text{real-cond-exp } M F X \in \text{borel-measurable } F$ 
   $g \in \text{borel-measurable } F g \in \text{borel-measurable } M$ 
   $G \in \text{borel-measurable } F G \in \text{borel-measurable } M$ 
  using  $X$  measurable-from-subalg[OF subalg] unfolding  $G$ -def  $g$ -def by auto
have int1: integrable (restr-to-subalg  $M F$ ) ( $\lambda x. g x * q (\text{real-cond-exp } M F X x)$ )
  apply (rule Bochner-Integration.integrable-bound[of  $- f$ ], auto simp add: subalg
 $\langle \text{integrable } (\text{restr-to-subalg } M F) f \rangle$ )
  unfolding  $g$ -def by (auto simp add: field-split-simps abs-mult algebra-simps)
have int2: integrable  $M$  ( $\lambda x. G x * (X x - \text{real-cond-exp } M F X x)$ )
  apply (rule Bochner-Integration.integrable-bound[of  $- \lambda x. |X x| + |\text{real-cond-exp}$ 
 $M F X x|$ ])
  apply (auto intro!: Bochner-Integration.integrable-add integrable-abs real-cond-exp-int
 $\langle \text{integrable } M X \rangle$  AE-I2)
  using  $G$  unfolding abs-mult by (meson abs-ge-zero abs-triangle-ineq4 dual-order.trans
mult-left-le-one-le)
have int3: integrable  $M$  ( $\lambda x. g x * q (X x)$ )
  apply (rule Bochner-Integration.integrable-bound[of  $- \lambda x. q(X x)$ ], auto simp

```

add: q(1) abs-mult
using *g* **by** (*simp add: less-imp-le mult-left-le-one-le*)

Taking the conditional expectation of the main convexity inequality `main_G`, we get the following.

have *AE x in M. real-cond-exp M F* ($\lambda x. g x * q (X x)$) $x \geq$ *real-cond-exp M F* ($\lambda x. g x * q (real-cond-exp M F X x) + G x * (X x - real-cond-exp M F X x)$) x
apply (*rule real-cond-exp-mono[OF main-G]*)
apply (*rule Bochner-Integration.integrable-add[OF integrable-from-subalg[OF subalg int1]]*)
using *int2 int3* **by** *auto*

This reduces to the desired inequality thanks to the properties of conditional expectation, i.e., the conditional expectation of an F -measurable function is this function, and one can multiply an F -measurable function outside of conditional expectations. Since all these equalities only hold almost everywhere, we formulate them separately, and then combine all of them to simplify the above equation, again almost everywhere.

moreover have *AE x in M. real-cond-exp M F* ($\lambda x. g x * q (X x)$) $x = g x * real-cond-exp M F ($\lambda x. q (X x)$) x
by (*rule real-cond-exp-mult, auto simp add: int3*)
moreover have *AE x in M. real-cond-exp M F* ($\lambda x. g x * q (real-cond-exp M F X x) + G x * (X x - real-cond-exp M F X x)$) x
 $= real-cond-exp M F$ ($\lambda x. g x * q (real-cond-exp M F X x)$) $x + real-cond-exp M F$ ($\lambda x. G x * (X x - real-cond-exp M F X x)$) x
by (*rule real-cond-exp-add, auto simp add: integrable-from-subalg[OF subalg int1] int2*)
moreover have *AE x in M. real-cond-exp M F* ($\lambda x. g x * q (real-cond-exp M F X x)$) $x = g x * q (real-cond-exp M F X x)$
by (*rule real-cond-exp-F-meas, auto simp add: integrable-from-subalg[OF subalg int1]*)
moreover have *AE x in M. real-cond-exp M F* ($\lambda x. G x * (X x - real-cond-exp M F X x)$) $x = G x * real-cond-exp M F$ ($\lambda x. (X x - real-cond-exp M F X x)$) x
by (*rule real-cond-exp-mult, auto simp add: int2*)
moreover have *AE x in M. real-cond-exp M F* ($\lambda x. (X x - real-cond-exp M F X x)$) $x = real-cond-exp M F X x - real-cond-exp M F$ ($\lambda x. real-cond-exp M F X x$) x
by (*rule real-cond-exp-diff, auto intro!: real-cond-exp-int <integrable M X>*)
moreover have *AE x in M. real-cond-exp M F* ($\lambda x. real-cond-exp M F X x$) $x = real-cond-exp M F X x$
by (*rule real-cond-exp-F-meas, auto intro!: real-cond-exp-int <integrable M X>*)
ultimately have *AE x in M. g x * real-cond-exp M F* ($\lambda x. q (X x)$) $x \geq g x * q (real-cond-exp M F X x)$
by *auto*
then show *AE x in M. real-cond-exp M F* ($\lambda x. q (X x)$) $x \geq q (real-cond-exp M F X x)$
using *g(1)* **by** (*auto simp add: field-split-simps*)
qed$

Jensen’s inequality does not imply that $q(E(X|F))$ is integrable, as it only proves an upper bound for it. Indeed, this is not true in general, as the following counterexample shows:

on $[1, \infty)$ with Lebesgue measure, let F be the sigma-algebra generated by the intervals $[n, n + 1)$ for integer n . Let $q(x) = -\sqrt{x}$ for $x \geq 0$. Define X which is equal to $1/n$ over $[n, n + 1/n)$ and 2^{-n} on $[n + 1/n, n + 1)$. Then X is integrable as $\sum 1/n^2 < \infty$, and $q(X)$ is integrable as $\sum 1/n^{3/2} < \infty$. On the other hand, $E(X|F)$ is essentially equal to $1/n^2$ on $[n, n + 1)$ (we neglect the term 2^{-n} , we only put it there because X should take its values in $I = (0, \infty)$). Hence, $q(E(X|F))$ is equal to $-1/n$ on $[n, n + 1)$, hence it is not integrable.

However, this counterexample is essentially the only situation where this function is not integrable, as shown by the next lemma.

lemma *integrable-convex-cond-exp*:

fixes $q :: \text{real} \Rightarrow \text{real}$

assumes X : *integrable* M X *AE* x *in* M . X $x \in I$

assumes I : $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = \text{UNIV}$

assumes q : *integrable* M $(\lambda x. q (X x))$ *convex-on* I q $q \in \text{borel-measurable borel}$

assumes H : *emeasure* M $(\text{space } M) = \infty \implies 0 \in I$

shows *integrable* M $(\lambda x. q (\text{real-cond-exp } M F X x))$

proof –

have [*measurable*]: $(\lambda x. q (\text{real-cond-exp } M F X x)) \in \text{borel-measurable } M$

$q \in \text{borel-measurable borel}$

$X \in \text{borel-measurable } M$

using $X(1)$ $q(3)$ **by** *auto*

have *open* I **using** I **by** *auto*

then have *interior* $I = I$ **by** (*simp add: interior-eq*)

consider *emeasure* M $(\text{space } M) = 0 \mid \text{emeasure } M$ $(\text{space } M) > 0 \wedge \text{emeasure } M$ $(\text{space } M) < \infty \mid \text{emeasure } M$ $(\text{space } M) = \infty$

by (*metis infinity-ennreal-def not-gr-zero top.not-eq-extremum*)

then show *?thesis*

proof (*cases*)

case 1

show *?thesis* **by** (*subst integrable-cong-AE[of - - $\lambda x. 0$], auto intro: emeasure-0-AE[OF 1]*)

next

case 2

interpret *finite-measure* M **using** 2 **by** (*auto intro!: finite-measureI*)

have $I \neq \{\}$

using $\langle \text{AE } x \text{ in } M. X x \in I \rangle$ 2 *eventually-mono integral-less-AE-space* **by** *fastforce*

then obtain z **where** $z \in I$ **by** *auto*

define A **where** $A = \text{Inf } ((\lambda t. (q z - q t) / (z - t)) \text{ ‘ } (\{z <..\} \cap I))$

have $q y \geq q z + A * (y - z)$ **if** $y \in I$ **for** y **unfolding** A -*def* **apply** (*rule*)

convex-le-Inf-differential)

using $\langle z \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle q(2)$ **by** *auto*
then have $AE\ x\ \text{in } M. q(\text{real-cond-exp } M\ F\ X\ x) \geq q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)$
using *real-cond-exp-jensens-inequality(1)[OF X I q]* **by** *auto*
moreover have $AE\ x\ \text{in } M. q(\text{real-cond-exp } M\ F\ X\ x) \leq \text{real-cond-exp } M\ F\ X\ x$
 $(\lambda x. q(X\ x))\ x$
using *real-cond-exp-jensens-inequality(2)[OF X I q]* **by** *auto*
moreover have $|a| \leq |b| + |c|$ **if** $b \leq a \wedge a \leq c$ **for** $a\ b\ c::\text{real}$
using *that by auto*
ultimately have $*$: $AE\ x\ \text{in } M. |q(\text{real-cond-exp } M\ F\ X\ x) - \text{real-cond-exp } M\ F\ X\ x - z|$
 $\leq |q(\text{real-cond-exp } M\ F\ X\ x) - (\lambda x. q(X\ x))\ x| + |q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)|$
by *auto*

show *integrable* $M\ (\lambda x. q(\text{real-cond-exp } M\ F\ X\ x))$

apply (*rule Bochner-Integration.integrable-bound[of - $\lambda x. |q(\text{real-cond-exp } M\ F\ X\ x) - (\lambda x. q(X\ x))\ x| + |q\ z + A * (\text{real-cond-exp } M\ F\ X\ x - z)|$]*)

apply (*auto intro!: Bochner-Integration.integrable-add integrable-abs integrable-mult-right Bochner-Integration.integrable-diff real-cond-exp-int(1)*)

using $X(1)\ q(1)$ ***** **by** *auto*

next

case 3

then have $0 \in I$ **using** $H\ \text{finite-measure.finite-emeasure-space}$ **by** *auto*

have $q(0) = 0$

proof (*rule ccontr*)

assume $*$: $\neg(q(0) = 0)$

define e **where** $e = |q(0)| / 2$

then have $e > 0$ **using** $*$ **by** *auto*

have *continuous (at 0) q*

using $q(2)\ \langle 0 \in I \rangle \langle \text{open } I \rangle \langle \text{interior } I = I \rangle$ *continuous-on-interior convex-on-continuous* **by** *blast*

then obtain d **where** $d: d > 0 \wedge y. |y - 0| < d \implies |q\ y - q\ 0| < e$ **using** $\langle e > 0 \rangle$

by (*metis continuous-at-real-range real-norm-def*)

then have $*$: $|q(y)| > e$ **if** $|y| < d$ **for** y

proof -

have $|q\ 0| \leq |q\ 0 - q\ y| + |q\ y|$ **by** *auto*

also have $\dots < e + |q\ y|$ **using** $d(2)$ **that** **by** *force*

finally have $|q\ y| > |q\ 0| - e$ **by** *auto*

then show *?thesis unfolding e-def* **by** *simp*

qed

have *emeasure* $M\ \{x \in \text{space } M. |X\ x| < d\} \leq \text{emeasure } M\ (\{x \in \text{space } M. |q(X\ x)| < e\})$

by (*rule emeasure-mono, auto simp add: * $\langle e > 0 \rangle$ less-imp-le ennreal-mult''[symmetric]*)

also have $\dots \leq (1/e) * (\int^+ x. \text{ennreal}(|q(X\ x)|) * \text{indicator } (\text{space } M)\ x\ \partial M)$

by (*rule nn-integral-Markov-inequality, auto*)

also have $\dots = (1/e) * (\int^+ x. \text{ennreal}(|q(X\ x)|)\ \partial M)$ **by** *auto*

also have $\dots = (1/e) * \text{ennreal}(\int x. |q(X\ x)|\ \partial M)$

using *nn-integral-eq-integral*[*OF integrable-abs*[*OF q*(1)]] **by** *auto*
also have $\dots < \infty$
by (*simp add: ennreal-mult-less-top*)
finally have $A: \text{emeasure } M \{x \in \text{space } M. |X x| < d\} < \infty$ **by** *simp*

have $\{x \in \text{space } M. |X x| \geq d\} = \{x \in \text{space } M. 1 \leq \text{ennreal}(1/d) * |X x|\}$
 $\cap \text{space } M$
by (*auto simp add: <d>0> ennreal-mult''[symmetric]*)
then have $\text{emeasure } M \{x \in \text{space } M. |X x| \geq d\} = \text{emeasure } M (\{x \in \text{space } M. 1 \leq \text{ennreal}(1/d) * |X x|\})$
by *auto*
also have $\dots \leq (1/d) * (\int^+ x. \text{ennreal}(|X x|) * \text{indicator } (\text{space } M) x \partial M)$
by (*rule nn-integral-Markov-inequality, auto*)
also have $\dots = (1/d) * (\int^+ x. \text{ennreal}(|X x|) \partial M)$ **by** *auto*
also have $\dots = (1/d) * \text{ennreal}(\int x. |X x| \partial M)$
using *nn-integral-eq-integral*[*OF integrable-abs*[*OF X*(1)]] **by** *auto*
also have $\dots < \infty$
by (*simp add: ennreal-mult-less-top*)
finally have $B: \text{emeasure } M \{x \in \text{space } M. |X x| \geq d\} < \infty$ **by** *simp*

have $\text{space } M = \{x \in \text{space } M. |X x| < d\} \cup \{x \in \text{space } M. |X x| \geq d\}$ **by** *auto*
then have $\text{emeasure } M (\text{space } M) = \text{emeasure } M (\{x \in \text{space } M. |X x| < d\} \cup \{x \in \text{space } M. |X x| \geq d\})$
by *simp*
also have $\dots \leq \text{emeasure } M \{x \in \text{space } M. |X x| < d\} + \text{emeasure } M \{x \in \text{space } M. |X x| \geq d\}$
by (*auto intro!: emeasure-subadditive*)
also have $\dots < \infty$ **using** $A B$ **by** *auto*
finally show $\text{False using } \langle \text{emeasure } M (\text{space } M) = \infty \rangle$ **by** *auto*
qed

define A **where** $A = \text{Inf } ((\lambda t. (q \ 0 - q \ t) / (0 - t)) ' (\{0 < ..\} \cap I))$
have $q \ y \geq q \ 0 + A * (y - 0)$ **if** $y \in I$ **for** y **unfolding** $A\text{-def}$ **apply** (*rule convex-le-Inf-differential*)
using $\langle 0 \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle q(2)$ **by** *auto*
then have $q \ y \geq A * y$ **if** $y \in I$ **for** y **using** $\langle q \ 0 = 0 \rangle$ **that** **by** *auto*
then have $\text{AE } x \text{ in } M. q (\text{real-cond-exp } M \ F \ X \ x) \geq A * \text{real-cond-exp } M \ F \ X \ x$
using *real-cond-exp-jensens-inequality*(1)[*OF X I q*] **by** *auto*
moreover have $\text{AE } x \text{ in } M. q (\text{real-cond-exp } M \ F \ X \ x) \leq \text{real-cond-exp } M \ F \ X \ x$
 $(\lambda x. q (X \ x)) \ x$
using *real-cond-exp-jensens-inequality*(2)[*OF X I q*] **by** *auto*
moreover have $|a| \leq |b| + |c|$ **if** $b \leq a \wedge a \leq c$ **for** $a \ b \ c::\text{real}$
using *that* **by** *auto*
ultimately have $*$: $\text{AE } x \text{ in } M. |q (\text{real-cond-exp } M \ F \ X \ x)| \leq |\text{real-cond-exp } M \ F \ X \ x| + |A * \text{real-cond-exp } M \ F \ X \ x|$
by *auto*

```

show integrable M ( $\lambda x. q$  (real-cond-exp M F X x))
  apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |real-cond-exp M F$ 
( $\lambda x. q$  (X x)) x| + |A * real-cond-exp M F X x |])
    apply (auto intro!: Bochner-Integration.integrable-add integrable-abs inte-
grable-mult-right Bochner-Integration.integrable-diff real-cond-exp-int(1))
      using X(1) q(1) * by auto
  qed
qed

end

end

```

```

theory Essential-Supremum
imports HOL-Analysis.Analysis
begin

```

```

lemma ae-filter-eq-bot-iff: ae-filter M = bot  $\longleftrightarrow$  emeasure M (space M) = 0
  by (simp add: AE-iff-measurable trivial-limit-def)

```

29 The essential supremum

In this paragraph, we define the essential supremum and give its basic properties. The essential supremum of a function is its maximum value if one is allowed to throw away a set of measure 0. It is convenient to define it to be infinity for non-measurable functions, as it allows for neater statements in general. This is a prerequisite to define the space L^∞ .

```

definition esssup::'a measure  $\Rightarrow$  ('a  $\Rightarrow$  'b::{second-countable-topology, dense-linorder,
linorder-topology, complete-linorder})  $\Rightarrow$  'b
  where esssup M f = (if f  $\in$  borel-measurable M then Limsup (ae-filter M) f else
top)

```

```

lemma esssup-non-measurable: f  $\notin$  M  $\rightarrow_M$  borel  $\implies$  esssup M f = top
  by (simp add: esssup-def)

```

```

lemma esssup-eq-AE:

```

```

  assumes f: f  $\in$  M  $\rightarrow_M$  borel shows esssup M f = Inf {z. AE x in M. f x  $\leq$  z}

```

```

  unfolding esssup-def if-P[OF f] Limsup-def

```

```

proof (intro antisym INF-greatest Inf-greatest; clarsimp)

```

```

  fix y assume AE x in M. f x  $\leq$  y

```

```

  then have ( $\lambda x. f x \leq y$ )  $\in$  {P. AE x in M. P x}

```

```

    by simp

```

```

  then show (INF P $\in$ {P. AE x in M. P x}. SUP x $\in$ Collect P. f x)  $\leq$  y

```

```

    by (rule INF-lower2) (auto intro: SUP-least)

```

```

next

```

```

  fix P assume P: AE x in M. P x

```

```

  show Inf {z. AE x in M. f x  $\leq$  z}  $\leq$  (SUP x $\in$ Collect P. f x)

```

proof (rule *Inf-lower*; *clarsimp*)
show *AE* x in M . $f x \leq (\text{SUP } x \in \text{Collect } P. f x)$
using P **by** (auto *elim*: *eventually-mono simp*: *SUP-upper*)
qed
qed

lemma *esssup-eq*: $f \in M \rightarrow_M \text{borel} \implies \text{esssup } M f = \text{Inf } \{z. \text{emeasure } M \{x \in \text{space } M. f x > z\} = 0\}$
by (auto *simp add*: *esssup-eq-AE not-less*[*symmetric*] *AE-iff-measurable*[*OF - refl*]
intro!: *arg-cong*[**where** $f = \text{Inf}$])

lemma *esssup-zero-measure*:
 $\text{emeasure } M \{x \in \text{space } M. f x > \text{esssup } M f\} = 0$
proof (*cases* $\text{esssup } M f = \text{top}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then have $f[\text{measurable}]$: $f \in M \rightarrow_M \text{borel}$ **unfolding** *esssup-def* **by** *meson*
have $\text{esssup } M f < \text{top}$ **using** *False* **by** (auto *simp*: *less-top*)
have $*$: $\{x \in \text{space } M. f x > z\} \in \text{null-sets } M$ **if** $z > \text{esssup } M f$ **for** z
proof –
have $\exists w. w < z \wedge \text{emeasure } M \{x \in \text{space } M. f x > w\} = 0$
using $\langle z \rangle \text{esssup } M f$ **by** (auto *simp*: *esssup-eq Inf-less-iff*)
then obtain w **where** $w < z$ $\text{emeasure } M \{x \in \text{space } M. f x > w\} = 0$ **by**
auto
then have a : $\{x \in \text{space } M. f x > w\} \in \text{null-sets } M$ **by** *auto*
have b : $\{x \in \text{space } M. f x > z\} \subseteq \{x \in \text{space } M. f x > w\}$ **using** $\langle w < z \rangle$ **by**
auto
show *?thesis* **using** *null-sets-subset*[*OF a - b*] **by** *simp*
qed
obtain $u::\text{nat} \Rightarrow 'b$ **where** $u: \bigwedge n. u n > \text{esssup } M f u \longrightarrow \text{esssup } M f$
using *approx-from-above-dense-linorder*[*OF* $\langle \text{esssup } M f < \text{top} \rangle$] **by** *auto*
have $\{x \in \text{space } M. f x > \text{esssup } M f\} = (\bigcup n. \{x \in \text{space } M. f x > u n\})$
using u **apply** *auto*
apply (*metis* (*mono-tags*, *lifting*) *order-tendsto-iff eventually-mono LIMSEQ-unique*)
using *less-imp-le less-le-trans* **by** *blast*
also have $\dots \in \text{null-sets } M$
using $*$ [*OF* $u(1)$] **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *esssup-AE*: *AE* x in M . $f x \leq \text{esssup } M f$
proof (*cases* $f \in M \rightarrow_M \text{borel}$)
case *True* **then show** *?thesis*
by (*intro* *AE-I*[*OF - esssup-zero-measure*[*of - f*]]) *auto*
qed (*simp add*: *esssup-non-measurable*)

lemma *esssup-pos-measure*:

$f \in \text{borel-measurable } M \implies z < \text{esssup } M f \implies \text{emeasure } M \{x \in \text{space } M. f x > z\} > 0$

using *Inf-less-iff mem-Collect-eq not-gr-zero* **by** (*force simp: esssup-eq*)

lemma *esssup-I* [*intro*]: $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq c \implies \text{esssup } M f \leq c$

unfolding *esssup-def* **by** (*simp add: Limsup-bounded*)

lemma *esssup-AE-mono*: $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq g x \implies \text{esssup } M f \leq \text{esssup } M g$

by (*auto simp: esssup-def Limsup-mono*)

lemma *esssup-mono*: $f \in \text{borel-measurable } M \implies (\bigwedge x. f x \leq g x) \implies \text{esssup } M f \leq \text{esssup } M g$

by (*rule esssup-AE-mono*) *auto*

lemma *esssup-AE-cong*:

$f \in \text{borel-measurable } M \implies g \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x = g x \implies \text{esssup } M f = \text{esssup } M g$

by (*auto simp: esssup-def intro!: Limsup-eq*)

lemma *esssup-const*: $\text{emeasure } M (\text{space } M) \neq 0 \implies \text{esssup } M (\lambda x. c) = c$

by (*simp add: esssup-def Limsup-const ae-filter-eq-bot-iff*)

lemma *esssup-cmult*: **assumes** $c > (0::\text{real})$ **shows** $\text{esssup } M (\lambda x. c * f x::\text{ereal}) = c * \text{esssup } M f$

proof –

have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \implies f \in M \rightarrow_M \text{borel}$

proof (*subst measurable-cong*)

fix ω **show** $f \omega = \text{ereal } (1/c) * (\text{ereal } c * f \omega)$

using $\langle 0 < c \rangle$ **by** (*cases f ω*) *auto*

qed *auto*

then have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \iff f \in M \rightarrow_M \text{borel}$

by(*safe intro!: borel-measurable-ereal-times borel-measurable-const*)

with $\langle 0 < c \rangle$ **show** *?thesis*

by (*cases ae-filter M = bot*)

(*auto simp: esssup-def bot-ereal-def top-ereal-def Limsup-ereal-mult-left*)

qed

lemma *esssup-add*:

$\text{esssup } M (\lambda x. f x + g x::\text{ereal}) \leq \text{esssup } M f + \text{esssup } M g$

proof (*cases f \in borel-measurable M \wedge g \in borel-measurable M*)

case *True*

then have [*measurable*]: $(\lambda x. f x + g x) \in \text{borel-measurable } M$ **by** *auto*

have $f x + g x \leq \text{esssup } M f + \text{esssup } M g$ **if** $f x \leq \text{esssup } M f$ $g x \leq \text{esssup } M g$ **for** x

using *that add-mono* **by** *auto*

then have $\text{AE } x \text{ in } M. f x + g x \leq \text{esssup } M f + \text{esssup } M g$

using *esssup-AE[of f M] esssup-AE[of g M]* **by** *auto*

```

then show ?thesis using esssup-I by auto
next
  case False
  then have esssup M f + esssup M g = ∞ unfolding esssup-def top-ereal-def
by auto
  then show ?thesis by auto
qed

```

lemma *esssup-zero-space*:

emeasure M (space M) = 0 \implies *f ∈ borel-measurable M* \implies *esssup M f = (−∞::ereal)*

by (*simp add: esssup-def ae-filter-eq-bot-iff[symmetric] bot-ereal-def*)

end

30 Stopping times

theory *Stopping-Time*

imports *HOL-Analysis.Analysis*

begin

30.1 Stopping Time

This is also called strong stopping time. Then stopping time is T with alternative is $T x < t$ measurable.

definition *stopping-time* :: (*t::linorder* \implies *'a measure*) \implies (*'a* \implies *'t*) \implies *bool*

where

stopping-time F T = ($\forall t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t)$)

lemma *stopping-time-cong*: ($\bigwedge t x. x \in \text{space } (F t) \implies T x = S x$) \implies *stopping-time F T = stopping-time F S*

unfolding *stopping-time-def* **by** (*intro arg-cong[where f=All] ext measurable-cong simp*)

lemma *stopping-timeD*: *stopping-time F T* \implies *Measurable.pred (F t) ($\lambda x. T x \leq t$)*

by (*auto simp: stopping-time-def*)

lemma *stopping-timeD2*: *stopping-time F T* \implies *Measurable.pred (F t) ($\lambda x. t < T x$)*

unfolding *not-le[symmetric]* **by** (*auto intro: stopping-timeD Measurable.pred-intros-logic*)

lemma *stopping-timeI[intro?]*: ($\bigwedge t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t)$) \implies *stopping-time F T*

by (*auto simp: stopping-time-def*)

lemma *measurable-stopping-time*:

fixes *T* :: *'a* \implies *'t::\{linorder-topology, second-countable-topology\}*

assumes T : *stopping-time* F T
and M : $\bigwedge t. \text{sets } (F\ t) \subseteq \text{sets } M \wedge t. \text{space } (F\ t) = \text{space } M$
shows $T \in M \rightarrow_M \text{borel}$
proof (*rule borel-measurableI-le*)
show $\{x \in \text{space } M. T\ x \leq t\} \in \text{sets } M$ **for** t
using *stopping-timeD[OF T] M* **by** (*auto simp: Measurable.pred-def*)
qed

lemma *stopping-time-const*: *stopping-time* F $(\lambda x. c)$
by (*auto simp: stopping-time-def*)

lemma *stopping-time-min*:
stopping-time F $T \implies \text{stopping-time } F\ S \implies \text{stopping-time } F$ $(\lambda x. \min (T\ x)$
 $(S\ x))$
by (*auto simp: stopping-time-def min-le-iff-disj intro!: pred-intros-logic*)

lemma *stopping-time-max*:
stopping-time F $T \implies \text{stopping-time } F\ S \implies \text{stopping-time } F$ $(\lambda x. \max (T\ x)$
 $(S\ x))$
by (*auto simp: stopping-time-def intro!: pred-intros-logic*)

31 Filtration

locale *filtration* =
fixes Ω :: $'a$ *set* **and** F :: $'t::\{\text{linorder-topology, second-countable-topology}\} \Rightarrow 'a$
measure
assumes *space-F*: $\bigwedge i. \text{space } (F\ i) = \Omega$
assumes *sets-F-mono*: $\bigwedge i\ j. i \leq j \implies \text{sets } (F\ i) \subseteq \text{sets } (F\ j)$
begin

31.1 σ -algebra of a Stopping Time

definition *pre-sigma* :: $('a \Rightarrow 't) \Rightarrow 'a$ *measure*
where
pre-sigma $T = \text{sigma } \Omega \ \{A. \forall t. \{\omega \in A. T\ \omega \leq t\} \in \text{sets } (F\ t)\}$

lemma *space-pre-sigma*: *space* (*pre-sigma* T) = Ω
unfolding *pre-sigma-def* **using** *sets.space-closed[of F -]*
by (*intro space-measure-of*) (*auto simp: space-F*)

lemma *measure-pre-sigma[simp]*: *emeasure* (*pre-sigma* T) = $(\lambda-. 0)$
by (*simp add: pre-sigma-def emeasure-sigma*)

lemma *sigma-algebra-pre-sigma*:
assumes T : *stopping-time* F T
shows *sigma-algebra* $\Omega \ \{A. \forall t. \{\omega \in A. T\ \omega \leq t\} \in \text{sets } (F\ t)\}$
unfolding *sigma-algebra-iff2*
proof (*intro sigma-algebra-iff2[THEN iffD2] conjI ballI allI impI CollectI*)
show $\{A. \forall t. \{\omega \in A. T\ \omega \leq t\} \in \text{sets } (F\ t)\} \subseteq \text{Pow } \Omega$

```

    using sets.space-closed[of F -] by (auto simp: space-F)
next
  fix A t assume A ∈ {A. ∀ t. {ω ∈ A. T ω ≤ t} ∈ sets (F t)}
  then have {ω ∈ space (F t). T ω ≤ t} - {ω ∈ A. T ω ≤ t} ∈ sets (F t)
    using T stopping-timeD[measurable] by auto
  also have {ω ∈ space (F t). T ω ≤ t} - {ω ∈ A. T ω ≤ t} = {ω ∈ Ω - A. T
ω ≤ t}
    by (auto simp: space-F)
  finally show {ω ∈ Ω - A. T ω ≤ t} ∈ sets (F t) .
next
  fix AA :: nat ⇒ 'a set and t assume range AA ⊆ {A. ∀ t. {ω ∈ A. T ω ≤ t}
∈ sets (F t)}
  then have (⋃ i. {ω ∈ AA i. T ω ≤ t}) ∈ sets (F t) for t
    by auto
  also have (⋃ i. {ω ∈ AA i. T ω ≤ t}) = {ω ∈ ⋃ (AA ' UNIV). T ω ≤ t}
    by auto
  finally show {ω ∈ ⋃ (AA ' UNIV). T ω ≤ t} ∈ sets (F t) .
qed auto

```

lemma sets-pre-sigma: *stopping-time* $F T \implies \text{sets (pre-sigma } T) = \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets (F } t)\}$
unfolding pre-sigma-def by (rule sigma-algebra.sets-measure-of-eq[OF sigma-algebra-pre-sigma])

lemma sets-pre-sigmaI: *stopping-time* $F T \implies (\bigwedge t. \{\omega \in A. T \omega \leq t\} \in \text{sets (F } t)) \implies A \in \text{sets (pre-sigma } T)$
unfolding sets-pre-sigma by auto

lemma pred-pre-sigmaI:
assumes T : *stopping-time* $F T$
shows $(\bigwedge t. \text{Measurable.pred (F } t) (\lambda \omega. P \omega \wedge T \omega \leq t)) \implies \text{Measurable.pred (pre-sigma } T) P$
unfolding pred-def space-F space-pre-sigma by (intro sets-pre-sigmaI[OF T]) simp

lemma sets-pre-sigmaD: *stopping-time* $F T \implies A \in \text{sets (pre-sigma } T) \implies \{\omega \in A. T \omega \leq t\} \in \text{sets (F } t)$
unfolding sets-pre-sigma by auto

lemma stopping-time-le-const: *stopping-time* $F T \implies s \leq t \implies \text{Measurable.pred (F } t) (\lambda \omega. T \omega \leq s)$
using stopping-timeD[of F T] sets-F-mono[of - t] by (auto simp: pred-def space-F)

lemma measurable-stopping-time-pre-sigma:
assumes T : *stopping-time* $F T$ **shows** $T \in \text{pre-sigma } T \rightarrow_M \text{ borel}$
proof (intro borel-measurableI-le sets-pre-sigmaI[OF T])
 fix t t'
 have $\{\omega \in \text{space (F (min t' t))}. T \omega \leq \text{min t' t}\} \in \text{sets (F (min t' t))}$
 using T **unfolding pred-def**[symmetric] by (rule stopping-timeD)
 also have $\dots \subseteq \text{sets (F } t)$

by (rule sets-F-mono) simp
 finally show $\{\omega \in \{x \in \text{space (pre-sigma } T\). T x \leq t'\}. T \omega \leq t\} \in \text{sets (F } t)$
 by (simp add: space-pre-sigma space-F)
 qed

lemma mono-pre-sigma:

assumes T : stopping-time $F T$ and S : stopping-time $F S$
 and le : $\bigwedge \omega. \omega \in \Omega \implies T \omega \leq S \omega$
 shows $\text{sets (pre-sigma } T) \subseteq \text{sets (pre-sigma } S)$
 unfolding sets-pre-sigma[OF S] sets-pre-sigma[OF T]
 proof safe
 interpret sigma-algebra $\Omega \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets (F } t)\}$
 using T by (rule sigma-algebra-pre-sigma)
 fix $A t$ assume $A: \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets (F } t)$
 then have $A \subseteq \Omega$
 using sets-into-space by auto
 from A have $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space (F } t). S \omega \leq t\} \in \text{sets (F } t)$
 using stopping-timeD[OF S] by (auto simp: pred-def)
 also have $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space (F } t). S \omega \leq t\} = \{\omega \in A. S \omega \leq t\}$
 using $\langle A \subseteq \Omega \rangle$ sets-into-space[of A] le by (auto simp: space-F intro: order-trans)
 finally show $\{\omega \in A. S \omega \leq t\} \in \text{sets (F } t)$
 by auto
 qed

lemma stopping-time-less-const:

assumes T : stopping-time $F T$ shows Measurable.pred (F t) $(\lambda \omega. T \omega < t)$
 proof –
 obtain $D :: 't \text{ set}$
 where D : countable $D \bigwedge X. \text{open } X \implies X \neq \{\} \implies \exists d \in D. d \in X$
 using countable-dense-setE by auto
 show ?thesis
 proof cases
 assume *: $\forall t' < t. \exists t''. t' < t'' \wedge t'' < t$
 { fix t' assume $t' < t$
 with * have $\{t' <..< t\} \neq \{\}$
 by fastforce
 with $D(2)$ [OF - this]
 have $\exists d \in D. t' < d \wedge d < t$
 by auto }
 note ** = this

 show ?thesis
 proof (rule measurable-cong[THEN iffD2])
 show $T \omega < t \longleftrightarrow (\exists r \in \{r \in D. r < t\}. T \omega \leq r)$ for ω
 by (auto dest: ** intro: less-imp-le)
 show Measurable.pred (F t) $(\lambda \omega. \exists r \in \{r \in D. r < t\}. T \omega \leq r)$
 by (intro measurable-pred-countable stopping-time-le-const[OF T] countable-Collect D) auto
 qed

```

next
  assume  $\neg (\forall t' < t. \exists t''. t' < t'' \wedge t'' < t)$ 
  then obtain  $t'$  where  $t': t' < t \wedge t''. t'' < t \implies t'' \leq t'$ 
    by (auto simp: not-less[symmetric])
  show ?thesis
  proof (rule measurable-cong[THEN iffD2])
    show  $T \omega < t \iff T \omega \leq t'$  for  $\omega$ 
      using  $t'$  by auto
    show Measurable.pred (F t) ( $\lambda \omega. T \omega \leq t'$ )
      using  $\langle t' < t \rangle$  by (intro stopping-time-le-const[OF T]) auto
  qed
qed
qed

```

```

lemma stopping-time-eq-const: stopping-time F T  $\implies$  Measurable.pred (F t) ( $\lambda \omega. T \omega = t$ )
  unfolding eq-iff using stopping-time-less-const[of T t]
  by (intro pred-intros-logic stopping-time-le-const) (auto simp: not-less[symmetric])
)

```

```

lemma stopping-time-less:
  assumes T: stopping-time F T and S: stopping-time F S
  shows Measurable.pred (pre-sigma T) ( $\lambda \omega. T \omega < S \omega$ )
  proof (rule pred-pre-sigmaI[OF T])
    fix t
    obtain D :: 't set
      where [simp]: countable D and semidense-D:  $\bigwedge x y. x < y \implies (\exists b \in D. x \leq b \wedge b < y)$ 
      using countable-separating-set-linorder2 by auto
    show Measurable.pred (F t) ( $\lambda \omega. T \omega < S \omega \wedge T \omega \leq t$ )
    proof (rule measurable-cong[THEN iffD2])
      let ?f =  $\lambda \omega. \text{if } T \omega = t \text{ then } \neg S \omega \leq t \text{ else } \exists s \in \{s \in D. s \leq t\}. T \omega \leq s \wedge \neg (S \omega \leq s)$ 
      { fix  $\omega$  assume  $T \omega \leq t \wedge T \omega \neq t \wedge T \omega < S \omega$ 
        then have  $T \omega < \min t (S \omega)$ 
          by auto
        then obtain  $r$  where  $r \in D \wedge T \omega \leq r \wedge r < \min t (S \omega)$ 
          by (metis semidense-D)
        then have  $\exists s \in \{s \in D. s \leq t\}. T \omega \leq s \wedge s < S \omega$ 
          by auto }
      then show  $(T \omega < S \omega \wedge T \omega \leq t) = ?f \omega$  for  $\omega$ 
        by (auto simp: not-le)
      show Measurable.pred (F t) ?f
        by (intro pred-intros-logic measurable-If measurable-pred-countable countable-Collect
          stopping-time-le-const predE stopping-time-eq-const T S)
          auto
    qed
  qed

```

end

lemma *stopping-time-SUP-enat*:

fixes $T :: \text{nat} \Rightarrow ('a \Rightarrow \text{enat})$

shows $(\bigwedge i. \text{stopping-time } F (T i)) \implies \text{stopping-time } F (SUP i. T i)$

unfolding *stopping-time-def SUP-apply SUP-le-iff* **by** (*auto intro!*: *pred-intros-countable*)

lemma *less-eSuc-iff*: $a < eSuc b \longleftrightarrow (a \leq b \wedge a \neq \infty)$

by (*cases a*) *auto*

lemma *stopping-time-Inf-enat*:

fixes $F :: \text{enat} \Rightarrow 'a \text{ measure}$

assumes F : *filtration* ΩF

assumes P : $\bigwedge i. \text{Measurable.pred } (F i) (P i)$

shows *stopping-time* $F (\lambda \omega. \text{Inf } \{i. P i \omega\})$

proof (*rule stopping-timeI, cases*)

fix $t :: \text{enat}$ **assume** $t = \infty$ **then show** $\text{Measurable.pred } (F t) (\lambda \omega. \text{Inf } \{i. P i \omega\} \leq t)$

by *auto*

next

fix $t :: \text{enat}$ **assume** $t \neq \infty$

moreover

{ **fix** $i \omega$ **assume** $\text{Inf } \{i. P i \omega\} \leq t$

with $\langle t \neq \infty \rangle$ **have** $(\exists i \leq t. P i \omega)$

unfolding *Inf-le-iff* **by** (*cases t*) (*auto elim!*: *allE[of - eSuc t] simp: less-eSuc-iff*)

}

ultimately have $*$: $\bigwedge \omega. \text{Inf } \{i. P i \omega\} \leq t \longleftrightarrow (\exists i \in \{..t\}. P i \omega)$

by (*auto intro!*: *Inf-lower2*)

show $\text{Measurable.pred } (F t) (\lambda \omega. \text{Inf } \{i. P i \omega\} \leq t)$

unfolding $*$ **using** *filtration.sets-F-mono[OF F, of - t] P*

by (*intro pred-intros-countable-bounded*) (*auto simp: pred-def filtration.space-F[OF F]*)

qed

lemma *stopping-time-Inf-nat*:

fixes $F :: \text{nat} \Rightarrow 'a \text{ measure}$

assumes F : *filtration* ΩF

assumes P : $\bigwedge i \omega. \omega \in \Omega \implies \exists n. P n \omega$

shows *stopping-time* $F (\lambda \omega. \text{Inf } \{i. P i \omega\})$

unfolding *stopping-time-def*

proof (*intro allI, subst measurable-cong*)

fix $t \omega$ **assume** $\omega \in \text{space } (F t)$

then have $\omega \in \Omega$

using *filtration.space-F[OF F]* **by** *auto*

from *wf[OF this]* **have** $((LEAST n. P n \omega) \leq t) = (\exists i \leq t. P i \omega)$

by (*rule LeastI2-wellorder-ex*) *auto*

then show $(\text{Inf } \{i. P i \omega\} \leq t) = (\exists i \in \{..t\}. P i \omega)$

```

    by (simp add: Inf-nat-def Bex-def)
next
fix t from P show Measurable.pred (F t) ( $\lambda w. \exists i \in \{..t\}. P i w$ )
  using filtration.sets-F-mono[OF F, of - t]
  by (intro pred-intros-countable-bounded) (auto simp: pred-def filtration.space-F[OF
F])
qed

end

```

```

theory Probability
imports
  Central-Limit-Theorem
  Discrete-Topology
  PMF-Impl
  Projective-Limit
  Random-Permutations
  SPMF
  Product-PMF
  Hoeffding
  Stream-Space
  Tree-Space
  Conditional-Expectation
  Essential-Supremum
  Stopping-Time
begin

end

```