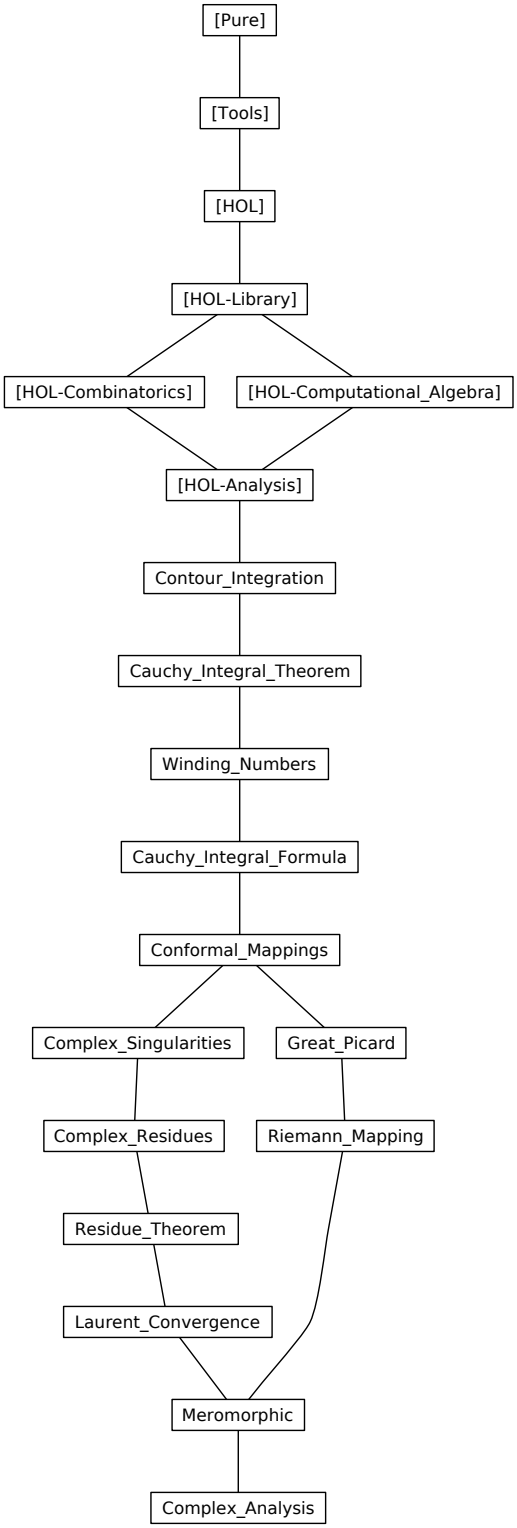
Complex Analysis

September 11, 2023

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
[HOL-Combinatorics]
[HOL-Computational_Algebra]
[HOL-Analysis]
Contour_Integration
Cauchy_Integral_Theorem
Winding_Numbers
Cauchy_Integral_Formula
Conformal_Mappings
Complex_Singularities
Great_Picard
Complex_Residues
Riemann_Mapping
Residue_Theorem
Laurent_Convergence
Meromorphic
Complex_Analysis

1 Contour integration

theory *Contour_Integration*
imports *HOL−Analysis.Analysis*
begin

lemma *lhopital_complex_simple*:
assumes (*f has_field_derivative f′*) (*at z*)
assumes (*g has_field_derivative g′*) (*at z*)
assumes *f z* = *0 g z* = *0 g′* ≠ *0 f′* / *g′* = *c*
shows ((λ*w*. *f w* / *g w*) ⟶ *c*) (*at z*)
proof −
have *eventually* (λ*w*. *w* ≠ *z*) (*at z*)
by (*auto simp*: *eventually_at_filter*)
hence *eventually* (λ*w*. ((*f w* − *f z*) / (*w* − *z*)) / ((*g w* − *g z*) / (*w* − *z*)) = *f w* / *g w*) (*at z*)
by *eventually_elim* (*simp add*: *assms field_split_simps*)
moreover have ((λ*w*. ((*f w* − *f z*) / (*w* − *z*)) / ((*g w* − *g z*) / (*w* − *z*))) ⟶ *f′* / *g′*) (*at z*)
by (*intro tendsto_divide has_field_derivativeD assms*)
ultimately have ((λ*w*. *f w* / *g w*) ⟶ *f′* / *g′*) (*at z*)
by (*blast intro*: *Lim_transform_eventually*)
with *assms* **show** *?thesis* **by** *simp*
qed

1.1 Definition

This definition is for complex numbers only, and does not generalise to line integrals in a vector field

definition *has_contour_integral* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ (*real* ⇒ *complex*) ⇒ *bool*
(**infixr** *has′_contour′_integral* *50*)
where (*f has_contour_integral i*) *g* ≡
((λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x within* {*0..1*}))
has_integral i) {*0..1*}

definition *contour_integrable_on*
(**infixr** *contour′_integrable′_on* *50*)
where *f contour_integrable_on g* ≡ ∃ *i*. (*f has_contour_integral i*) *g*

definition *contour_integral*
where *contour_integral g f* ≡ *SOME i*. (*f has_contour_integral i*) *g* ∨ ¬ *f contour_integrable_on g* ∧ *i*=*0*

lemma *not_integrable_contour_integral*: ¬ *f contour_integrable_on g* ⟹ *contour_integral g f* = *0*
unfolding *contour_integrable_on_def contour_integral_def* **by** *blast*

lemma *contour_integral_unique*: (*f has_contour_integral i*) *g* ⟹ *contour_integral*

g f = i
apply (*simp add*: *contour_integral_def has_contour_integral_def contour_integrable_on_def*)
using *has_integral_unique* **by** *blast*

lemma *has_contour_integral_eqpath*:
⟦(*f has_contour_integral y*) *p*; *f contour_integrable_on γ*;
contour_integral p f = contour_integral γ f⟧
⟹ (*f has_contour_integral y*) *γ*
using *contour_integrable_on_def contour_integral_unique* **by** *auto*

lemma *has_contour_integral_integral*:
f contour_integrable_on i ⟹ (*f has_contour_integral* (*contour_integral i f*))
i
by (*metis contour_integral_unique contour_integrable_on_def*)

lemma *has_contour_integral_unique*:
(*f has_contour_integral i*) *g* ⟹ (*f has_contour_integral j*) *g* ⟹ *i = j*
using *has_integral_unique*
by (*auto simp*: *has_contour_integral_def*)

lemma *has_contour_integral_integrable*: (*f has_contour_integral i*) *g* ⟹ *f contour_integrable_on g*
using *contour_integrable_on_def* **by** *blast*

Show that we can forget about the localized derivative.

lemma *has_integral_localized_vector_derivative*:
((λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x within* {*a..b*})) *has_integral i*) {*a..b*}
⟷
((λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x*)) *has_integral i*) {*a..b*}
proof −
have ∗: {*a..b*} − {*a*,*b*} = *interior* {*a..b*}
by (*simp add*: *atLeastAtMost_diff_ends*)
show *?thesis*
by (*rule has_integral_spike_eq* [*of* {*a*,*b*}]) (*auto simp*: *at_within_interior* [*of* _ {*a..b*}])
qed

lemma *integrable_on_localized_vector_derivative*:
(λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x within* {*a..b*})) *integrable_on* {*a..b*}
⟷
(λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x*)) *integrable_on* {*a..b*}
by (*simp add*: *integrable_on_def has_integral_localized_vector_derivative*)

lemma *has_contour_integral*:
(*f has_contour_integral i*) *g* ⟷
((λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *has_integral i*) {*0..1*}
by (*simp add*: *has_integral_localized_vector_derivative has_contour_integral_def*)

lemma *contour_integrable_on*:

f contour_integrable_on g ⟷
(λ*t*. *f*(*g t*) ∗ *vector_derivative g* (*at t*)) *integrable_on* {0..1}
by (*simp add*: *has_contour_integral integrable_on_def contour_integrable_on_def*)

lemma *has_contour_integral_mirror_iff*:
assumes *valid_path g*
shows (*f has_contour_integral I*) (−*g*) ⟷ ((λ*x*. −*f* (− *x*)) *has_contour_integral I*) *g*
proof −
from *assms* **have** *g piecewise_differentiable_on* {0..1}
by (*auto simp*: *valid_path_def piecewise_C1_imp_differentiable*)
then obtain *S* **where** *finite S* **and** *S*: ⋀*x*. *x* ∈ {0..1} − *S* ⟹ *g differentiable at x within* {0..1}
unfolding *piecewise_differentiable_on_def* **by** *blast*
have *S'*: *g differentiable at x* **if** *x* ∈ {0..1} − ({0, 1} ∪ *S*) **for** *x*
proof −
from *that* **have** *x* ∈ *interior* {0..1} **by** *auto*
with *S*[*of x*] *that* **show** *?thesis* **by** (*auto simp*: *at_within_interior*[*of* _ {0..1}])
qed

have (*f has_contour_integral I*) (−*g*) ⟷
((λ*x*. *f* (− *g x*) ∗ *vector_derivative* (−*g*) (*at x*)) *has_integral I*) {0..1}
by (*simp add*: *has_contour_integral*)
also have ... ⟷ ((λ*x*. −*f* (− *g x*) ∗ *vector_derivative g* (*at x*)) *has_integral I*) {0..1}
by (*intro has_integral_spike_finite_eq*[*of S* ∪ {0, 1}])
(*insert* ‹*finite S*› *S'*, *auto simp*: *o_def fun_Compl_def*)
also have ... ⟷ ((λ*x*. −*f* (−*x*)) *has_contour_integral I*) *g*
by (*simp add*: *has_contour_integral*)
finally show *?thesis* **.**
qed

lemma *contour_integral_on_mirror_iff*:
assumes *valid_path g*
shows *f contour_integrable_on* (−*g*) ⟷ (λ*x*. −*f* (− *x*)) *contour_integrable_on g*
by (*auto simp*: *contour_integrable_on_def has_contour_integral_mirror_iff assms*)

lemma *contour_integral_mirror*:
assumes *valid_path g*
shows *contour_integral* (−*g*) *f* = *contour_integral g* (λ*x*. −*f* (− *x*))
proof (*cases f contour_integrable_on* (−*g*))
case *True* **with** *contour_integral_unique assms* **show** *?thesis*
by (*auto simp*: *contour_integrable_on_def has_contour_integral_mirror_iff*)
next
case *False* **then show** *?thesis*
by (*simp add*: *assms contour_integral_on_mirror_iff not_integrable_contour_integral*)
qed

1.2 Reversing a path

lemma *has_contour_integral_reversepath*:
assumes *valid_path g* **and** *f*: (*f has_contour_integral i*) *g*
shows (*f has_contour_integral* (−*i*)) (*reversepath g*)
proof −
{ fix *S x*
assume *xs*: *g C1_differentiable_on* ({*0..1*} − *S*) *x* ∉ (−) *1* ‘ *S 0* ≤ *x x* ≤ *1*
have *vector_derivative* (λ*x*. *g* (*1* − *x*)) (*at x within* {*0..1*}) =
− *vector_derivative g* (*at* (*1* − *x*) *within* {*0..1*})
proof −
obtain *f′* **where** *f′*: (*g has_vector_derivative f′*) (*at* (*1* − *x*))
using *xs*
by (*force simp*: *has_vector_derivative_def C1_differentiable_on_def*)
have (*g* ∘ (λ*x*. *1* − *x*) *has_vector_derivative* −*1* ∗*R f′*) (*at x*)
by (*intro vector_diff_chain_within has_vector_derivative_at_within* [*OF f′*] *derivative_eq_intros* | *simp*)+
then have *mf′*: ((λ*x*. *g* (*1* − *x*)) *has_vector_derivative* −*f′*) (*at x*)
by (*simp add*: *o_def*)
show *?thesis*
using *xs*
by (*auto simp*: *vector_derivative_at_within_ivl* [*OF mf′*] *vector_derivative_at_within_ivl* [*OF f′*])
qed
} note ∗ = *this*
obtain *S* **where** *S*: *continuous_on* {*0..1*} *g finite S g C1_differentiable_on* {*0..1*} − *S*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def*)
have ((λ*x*. − (*f* (*g* (*1* − *x*)) ∗ *vector_derivative g* (*at* (*1* − *x*) *within* {*0..1*}))) *has_integral* −*i*)
{*0..1*}
using *has_integral_affinity01* [**where** *m*= −*1* **and** *c*=*1*, *OF f* [*unfolded has_contour_integral_def*]]
by (*simp add*: *has_integral_neg*)
then show *?thesis*
using *S*
unfolding *reversepath_def has_contour_integral_def*
by (*rule_tac S* = (λ*x*. *1* − *x*) ‘ *S* **in** *has_integral_spike_finite*) (*auto simp*: ∗)
qed

lemma *contour_integrable_reversepath*:
valid_path g ⟹ *f contour_integrable_on g* ⟹ *f contour_integrable_on* (*reversepath g*)
using *has_contour_integral_reversepath contour_integrable_on_def* **by** *blast*

lemma *contour_integrable_reversepath_eq*:
valid_path g ⟹ (*f contour_integrable_on* (*reversepath g*) ⟷ *f contour_integrable_on g*)
using *contour_integrable_reversepath valid_path_reversepath* **by** *fastforce*

lemma *contour_integral_reversepath*:
assumes *valid_path g*
shows *contour_integral* (*reversepath g*) *f* = − (*contour_integral g f*)
proof (*cases f contour_integrable_on g*)
case *True* **then show** *?thesis*
by (*simp add*: *assms contour_integral_unique has_contour_integral_integral has_contour_integral_reversepath*)
next
case *False* **then have** ¬ *f contour_integrable_on* (*reversepath g*)
by (*simp add*: *assms contour_integrable_reversepath_eq*)
with *False* **show** *?thesis* **by** (*simp add*: *not_integrable_contour_integral*)
qed

1.3 Joining two paths together

lemma *has_contour_integral_join*:
assumes (*f has_contour_integral i1*) *g1* (*f has_contour_integral i2*) *g2*
valid_path g1 valid_path g2
shows (*f has_contour_integral* (*i1* + *i2*)) (*g1* +++ *g2*)
proof −
obtain *s1 s2*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
and *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms*
by (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have *1*: ((λ*x*. *f* (*g1 x*) ∗ *vector_derivative g1* (*at x*)) *has_integral i1*) {*0..1*}
and *2*: ((λ*x*. *f* (*g2 x*) ∗ *vector_derivative g2* (*at x*)) *has_integral i2*) {*0..1*}
using *assms*
by (*auto simp*: *has_contour_integral*)
have *i1*: ((λ*x*. (*2*∗*f* (*g1* (*2*∗*x*))) ∗ *vector_derivative g1* (*at* (*2*∗*x*))) *has_integral i1*) {*0..1/2*}
and *i2*: ((λ*x*. (*2*∗*f* (*g2* (*2*∗*x* − *1*))) ∗ *vector_derivative g2* (*at* (*2*∗*x* − *1*))) *has_integral i2*) {*1/2..1*}
using *has_integral_affinity01* [*OF 1*, **where** *m*= *2* **and** *c*=*0*, *THEN has_integral_cmul* [**where** *c*=*2*]]
has_integral_affinity01 [*OF 2*, **where** *m*= *2* **and** *c*=−*1*, *THEN has_integral_cmul* [**where** *c*=*2*]]
by (*simp_all only*: *image_affinity_atLeastAtMost_div_diff*, *simp_all add*: *scaleR_conv_of_real mult_ac*)
have *g1*: *vector_derivative* (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z*) =
2 ∗$_R$ *vector_derivative g1* (*at* (*z*∗*2*))
if *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1* **for** *z*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
show *0* < |*z* − *1/2*|
using *that* **by** *auto*
have ((∗) *2 has_vector_derivative 2*) (*at z*)
by (*simp add*: *has_vector_derivative_def has_derivative_def bounded_linear_mult_left*)
moreover have (*g1 has_vector_derivative vector_derivative g1* (*at* (*z* ∗ *2*)))

```
(at (2 * z))
      using s1 that by (auto simp: algebra_simps vector_derivative_works)
    ultimately
    show ((λx. g1 (2 * x)) has_vector_derivative 2 *R vector_derivative g1 (at
(z * 2))) (at z)
      by (intro vector_diff_chain_at [simplified o_def])
  qed (use that in ‹simp_all add: dist_real_def abs_if split: if_split_asm›)

  have g2: vector_derivative (λx. if x*2 ≤ 1 then g1 (2*x) else g2 (2*x - 1))
(at z) =
          2 *R vector_derivative g2 (at (z*2 - 1))
        if 1 < z*2 z ≤ 1 z*2 - 1 ∉ s2 for z
  proof (rule vector_derivative_at [OF has_vector_derivative_transform_within])
    show 0 < |z - 1/2|
      using that by auto
    have ((λx. 2 * x - 1) has_vector_derivative 2) (at z)
    by (simp add: has_vector_derivative_def has_derivative_def bounded_linear_mult_left)
    moreover have (g2 has_vector_derivative vector_derivative g2 (at (z * 2 -
1))) (at (2 * z - 1))
      using s2 that by (auto simp: algebra_simps vector_derivative_works)
    ultimately
    show ((λx. g2 (2 * x - 1)) has_vector_derivative 2 *R vector_derivative g2
(at (z * 2 - 1))) (at z)
      by (intro vector_diff_chain_at [simplified o_def])
  qed (use that in ‹simp_all add: dist_real_def abs_if split: if_split_asm›)

  have ((λx. f ((g1 +++ g2) x) * vector_derivative (g1 +++ g2) (at x)) has_integral
i1) {0..1/2}
  proof (rule has_integral_spike_finite [OF _ _ i1])
    show finite (insert (1/2) ((*) 2 -` s1))
      using s1 by (force intro: finite_vimageI [where h = (*)2] inj_onI)
  qed (auto simp add: joinpaths_def scaleR_conv_of_real mult_ac g1)
  moreover have ((λx. f ((g1 +++ g2) x) * vector_derivative (g1 +++ g2) (at
x)) has_integral i2) {1/2..1}
  proof (rule has_integral_spike_finite [OF _ _ i2])
    show finite (insert (1/2) ((λx. 2 * x - 1) -` s2))
      using s2 by (force intro: finite_vimageI [where h = λx. 2*x-1] inj_onI)
  qed (auto simp add: joinpaths_def scaleR_conv_of_real mult_ac g2)
  ultimately
  show ?thesis
    by (simp add: has_contour_integral has_integral_combine [where c = 1/2])
qed

lemma contour_integrable_joinI:
  assumes f contour_integrable_on g1 f contour_integrable_on g2
          valid_path g1 valid_path g2
    shows f contour_integrable_on (g1 +++ g2)
  using assms
  by (meson has_contour_integral_join contour_integrable_on_def)
```

lemma *contour_integrable_joinD1*:
assumes *f contour_integrable_on* (*g1 +++ g2*) *valid_path g1*
shows *f contour_integrable_on g1*
proof −
obtain *s1*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have (λ*x*. *f* ((*g1 +++ g2*) (*x*/*2*)) ∗ *vector_derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable_on* {*0..1*}
using *assms integrable_affinity* [*of* _ *0 1/2 1/2 0*] *integrable_on_subcbox* [**where** *a*=*0* **and** *b*=*1/2*]
by (*fastforce simp*: *contour_integrable_on*)
then have ∗: (λ*x*. (*f* ((*g1 +++ g2*) (*x*/*2*))/*2*) ∗ *vector_derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable_on* {*0..1*}
by (*auto dest*: *integrable_cmul* [**where** *c*=*1/2*] *simp*: *scaleR_conv_of_real*)
have *g1*: *vector_derivative* (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at* (*z*/*2*)) =
2 ∗$_R$ *vector_derivative g1* (*at z*)
if *0* < *z z* < *1 z* ∉ *s1* **for** *z*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
show *0* < |(*z* − *1*)/*2*|
using *that* **by** *auto*
have §: ((λ*x*. *x* ∗ *2*) *has_vector_derivative 2*) (*at* (*z*/*2*))
using *s1* **by** (*auto simp*: *has_vector_derivative_def has_derivative_def bounded_linear_mult_left*)
have (*g1 has_vector_derivative vector_derivative g1* (*at z*)) (*at z*)
using *s1 that* **by** (*auto simp*: *vector_derivative_works*)
then show ((λ*x*. *g1* (*2* ∗ *x*)) *has_vector_derivative 2* ∗$_R$ *vector_derivative g1* (*at z*)) (*at* (*z*/*2*))
using *vector_diff_chain_at* [*OF* §] **by** (*auto simp*: *field_simps o_def*)
qed (*use that* **in** ‹*simp_all add*: *field_simps dist_real_def abs_if split*: *if_split_asm*›)
have *fin01*: *finite* ({*0*, *1*} ∪ *s1*)
by (*simp add*: *s1*)
show *?thesis*
unfolding *contour_integrable_on*
by (*intro integrable_spike_finite* [*OF fin01* _ ∗]) (*auto simp*: *joinpaths_def scaleR_conv_of_real g1*)
qed

lemma *contour_integrable_joinD2*:
assumes *f contour_integrable_on* (*g1 +++ g2*) *valid_path g2*
shows *f contour_integrable_on g2*
proof −
obtain *s2*
where *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)

have ($\lambda x.\ f\ ((g1\ +{+}{+}\ g2)\ (x/2\ +\ 1/2))\ *\ vector\_derivative\ (g1\ +{+}{+}\ g2)\ (at\ (x/2\ +\ 1/2))$) *integrable_on* $\{0..1\}$
using *assms integrable_affinity* [*of* _ $1/2$::*real* $1\ 1/2\ 1/2$]
integrable_on_subcbox [**where** $a{=}1/2$ **and** $b{=}1$]
by (*fastforce simp*: *contour_integrable_on image_affinity_atLeastAtMost_diff*)
then have $*$: ($\lambda x.\ (f\ ((g1\ +{+}{+}\ g2)\ (x/2\ +\ 1/2))/2)\ *\ vector\_derivative\ (g1\ +{+}{+}\ g2)\ (at\ (x/2\ +\ 1/2))$)
integrable_on $\{0..1\}$
by (*auto dest*: *integrable_cmul* [**where** $c{=}1/2$] *simp*: *scaleR_conv_of_real*)
have *g2*: *vector_derivative* ($\lambda x.$ *if* $x{*}2 \le 1$ *then g1* $(2{*}x)$ *else g2* $(2{*}x - 1)$) $(at\ (z/2{+}1/2)) =$
$2\ *_R$ *vector_derivative g2* $(at\ z)$
if $0 < z\ z < 1\ z \notin s2$ **for** z
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
show $0 < |z/2|$
using *that* **by** *auto*
have §: ($(\lambda x.\ x * 2 - 1)$ *has_vector_derivative* 2) $(at\ ((1\ +\ z)/2))$
using *s2* **by** (*auto simp*: *has_vector_derivative_def has_derivative_def bounded_linear_mult_left*)
have (*g2 has_vector_derivative vector_derivative g2* $(at\ z)$) $(at\ z)$
using *s2 that* **by** (*auto simp*: *vector_derivative_works*)
then show ($(\lambda x.\ g2\ (2{*}x - 1))$ *has_vector_derivative* $2\ *_R$ *vector_derivative g2* $(at\ z)$) $(at\ (z/2\ +\ 1/2))$
using *vector_diff_chain_at* [*OF* §] **by** (*auto simp*: *field_simps o_def*)
qed (*use that* **in** ‹*simp_all add*: *field_simps dist_real_def abs_if split*: *if_split_asm*›)
have *fin01*: *finite* $(\{0,\ 1\} \cup s2)$
by (*simp add*: *s2*)
show *?thesis*
unfolding *contour_integrable_on*
by (*intro integrable_spike_finite* [*OF fin01* _ $*$]) (*auto simp*: *joinpaths_def scaleR_conv_of_real g2*)
qed

lemma *contour_integrable_join* [*simp*]:
⟦*valid_path g1*; *valid_path g2*⟧
$\Longrightarrow$ *f contour_integrable_on* $(g1\ +{+}{+}\ g2) \longleftrightarrow$ *f contour_integrable_on g1* $\wedge$ *f contour_integrable_on g2*
using *contour_integrable_joinD1 contour_integrable_joinD2 contour_integrable_joinI*
by *blast*

lemma *contour_integral_join* [*simp*]:
⟦*f contour_integrable_on g1*; *f contour_integrable_on g2*; *valid_path g1*; *valid_path g2*⟧
$\Longrightarrow$ *contour_integral* $(g1\ +{+}{+}\ g2)\ f$ = *contour_integral g1 f* + *contour_integral g2 f*
by (*simp add*: *has_contour_integral_integral has_contour_integral_join contour_integral_unique*)

1.4 Shifting the starting point of a (closed) path

lemma *has_contour_integral_shiftpath*:
assumes *f*: (*f has_contour_integral i*) *g valid_path g*
and *a*: *a* ∈ {*0..1*}
shows (*f has_contour_integral i*) (*shiftpath a g*)
proof −
obtain *S*
where *S*: *finite S* **and** *g*: ∀ *x*∈{*0..1*} − *S*. *g differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have ∗: ((λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *has_integral i*) {*0..1*}
using *assms* **by** (*auto simp*: *has_contour_integral*)
then have *i*: *i* = *integral* {*a..1*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) +
integral {*0..a*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*))
apply (*rule has_integral_unique*)
apply (*subst add.commute*)
apply (*subst Henstock_Kurzweil_Integration.integral_combine*)
using *assms* ∗ *integral_unique* **by** *auto*

have *vd1*: *vector_derivative* (*shiftpath a g*) (*at x*) = *vector_derivative g* (*at* (*x* + *a*))
if *0* ≤ *x x* + *a* < *1 x* ∉ (λ*x*. *x* − *a*) ‘ *S* **for** *x*
unfolding *shiftpath_def*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
have ((λ*x*. *g* (*x* + *a*)) *has_vector_derivative vector_derivative g* (*at* (*a* + *x*))) (*at x*)
proof (*rule vector_diff_chain_at* [*of _ 1*, *simplified o_def scaleR_one*])
show ((λ*x*. *x* + *a*) *has_vector_derivative 1*) (*at x*)
by (*rule derivative_eq_intros* | *simp*)+
have *g differentiable at* (*x* + *a*)
using *g a that* **by** *force*
then show (*g has_vector_derivative vector_derivative g* (*at* (*a* + *x*))) (*at* (*x* + *a*))
by (*metis add.commute vector_derivative_works*)
qed
then
show ((λ*x*. *g* (*a* + *x*)) *has_vector_derivative vector_derivative g* (*at* (*x* + *a*))) (*at x*)
by (*auto simp*: *field_simps*)
show *0* < *dist* (*1* − *a*) *x*
using *that* **by** *auto*
qed (*use that* **in** ‹*auto simp*: *dist_real_def*›)

have *vd2*: *vector_derivative* (*shiftpath a g*) (*at x*) = *vector_derivative g* (*at* (*x* + *a* − *1*))
if *x* ≤ *1 1* < *x* + *a x* ∉ (λ*x*. *x* − *a* + *1*) ‘ *S* **for** *x*
unfolding *shiftpath_def*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
have ((λ*x*. *g* (*x* + *a* − *1*)) *has_vector_derivative vector_derivative g* (*at*

```
(a+x−1))) (at x)
    proof (rule vector_diff_chain_at [of _ 1, simplified o_def scaleR_one])
      show ((λx. x + a − 1) has_vector_derivative 1) (at x)
        by (rule derivative_eq_intros | simp)+
      have g differentiable at (x+a−1)
        using g a that by force
      then show (g has_vector_derivative vector_derivative g (at (a+x−1))) (at
(x + a − 1))
        by (metis add.commute vector_derivative_works)
    qed
   then show ((λx. g (a + x − 1)) has_vector_derivative vector_derivative g (at
(x + a − 1))) (at x)
      by (auto simp: field_simps)
    show 0 < dist (1 − a) x
      using that by auto
  qed (use that in ‹auto simp: dist_real_def›)

  have va1: (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on ({a..1})
    using ∗ a   by (fastforce intro: integrable_subinterval_real)
  have v0a: (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on ({0..a})
    using ∗ a by (force intro: integrable_subinterval_real)
  have finite ({1 − a} ∪ (λx. x − a) ‘ S)
    using S by blast
  then have ((λx. f (shiftpath a g x) ∗ vector_derivative (shiftpath a g) (at x))
      has_integral integral {a..1} (λx. f (g x) ∗ vector_derivative g (at x))) {0..1
− a}
    apply (rule has_integral_spike_finite
       [where f = λx. f(g(a+x)) ∗ vector_derivative g (at(a+x))])
    subgoal
      using a by (simp add: vd1) (force simp: shiftpath_def add.commute)
    subgoal
      using has_integral_affinity [where m=1 and c=a] integrable_integral [OF
va1]
      by (force simp add: add.commute)
    done
  moreover
  have finite ({1 − a} ∪ (λx. x − a + 1) ‘ S)
    using S by blast
  then have ((λx. f (shiftpath a g x) ∗ vector_derivative (shiftpath a g) (at x))
          has_integral  integral {0..a} (λx. f (g x) ∗ vector_derivative g (at x)))
{1 − a..1}
    apply (rule has_integral_spike_finite
       [where f = λx. f(g(a+x−1)) ∗ vector_derivative g (at(a+x−1))])
    subgoal
      using a by (simp add: vd2) (force simp: shiftpath_def add.commute)
    subgoal
      using has_integral_affinity [where m=1 and c=a−1, simplified, OF inte-
grable_integral [OF v0a]]
      by (force simp add: algebra_simps)
```

```
    done
  ultimately show ?thesis
    using a
    by (auto simp: i has_contour_integral intro: has_integral_combine [where c
= 1−a])
qed

lemma has_contour_integral_shiftpath_D:
  assumes (f has_contour_integral i) (shiftpath a g)
          valid_path g pathfinish g = pathstart g a ∈ {0..1}
    shows (f has_contour_integral i) g
proof −
  obtain S
    where S: finite S and g: ∀ x∈{0..1} − S. g differentiable at x
   using assms by (auto simp: valid_path_def piecewise_C1_differentiable_on_def
C1_differentiable_on_eq)
  { fix x
    assume x: 0 < x x < 1 x ∉ S
    then have gx: g differentiable at x
      using g by auto
    have §: shiftpath (1 − a) (shiftpath a g) differentiable at x
      using assms x
      by (intro differentiable_transform_within [OF gx, of min x (1−x)])
         (auto simp: dist_real_def shiftpath_shiftpath abs_if split: if_split_asm)
    have vector_derivative g (at x within {0..1}) =
         vector_derivative (shiftpath (1 − a) (shiftpath a g)) (at x within {0..1})
      apply (rule vector_derivative_at_within_ivl
                [OF has_vector_derivative_transform_within_open
                    [where f = (shiftpath (1 − a) (shiftpath a g)) and S =
{0<..<1}−S]])
      using S assms x §
      apply (auto simp: finite_imp_closed open_Diff shiftpath_shiftpath
                        at_within_interior [of _ {0..1}] vector_derivative_works
[symmetric])
      done
  } note vd = this
  have fi: (f has_contour_integral i) (shiftpath (1 − a) (shiftpath a g))
    using assms  by (auto intro!: has_contour_integral_shiftpath)
  show ?thesis
    unfolding has_contour_integral_def
   proof (rule has_integral_spike_finite [of {0,1} ∪ S, OF _ _  fi [unfolded
has_contour_integral_def]])
    show finite ({0, 1} ∪ S)
      by (simp add: S)
  qed (use S assms vd in ‹auto simp: shiftpath_shiftpath›)
qed

lemma has_contour_integral_shiftpath_eq:
  assumes valid_path g pathfinish g = pathstart g a ∈ {0..1}
```

shows (*f has_contour_integral i*) (*shiftpath a g*) ⟷ (*f has_contour_integral i*) *g*
using *assms has_contour_integral_shiftpath has_contour_integral_shiftpath_D*
by *blast*

lemma *contour_integrable_on_shiftpath_eq*:
assumes *valid_path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows *f contour_integrable_on* (*shiftpath a g*) ⟷ *f contour_integrable_on g*
using *assms contour_integrable_on_def has_contour_integral_shiftpath_eq* **by** *auto*

lemma *contour_integral_shiftpath*:
assumes *valid_path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows *contour_integral* (*shiftpath a g*) *f* = *contour_integral g f*
using *assms*
by (*simp add*: *contour_integral_def contour_integrable_on_def has_contour_integral_shiftpath_eq*)

1.5 More about straight-line paths

lemma *has_contour_integral_linepath*:
shows (*f has_contour_integral i*) (*linepath a b*) ⟷
((λ*x*. *f*(*linepath a b x*) ∗ (*b* − *a*)) *has_integral i*) {*0..1*}
by (*simp add*: *has_contour_integral*)

lemma *has_contour_integral_trivial* [*iff*]: (*f has_contour_integral 0*) (*linepath a a*)
by (*simp add*: *has_contour_integral_linepath*)

lemma *has_contour_integral_trivial_iff* [*simp*]: (*f has_contour_integral i*) (*linepath a a*) ⟷ *i=0*
using *has_contour_integral_unique* **by** *blast*

lemma *contour_integral_trivial* [*simp*]: *contour_integral* (*linepath a a*) *f* = *0*
using *has_contour_integral_trivial contour_integral_unique* **by** *blast*

1.6 Relation to subpath construction

lemma *has_contour_integral_subpath_refl* [*iff*]: (*f has_contour_integral 0*) (*subpath u u g*)
by (*simp add*: *has_contour_integral subpath_def*)

lemma *contour_integrable_subpath_refl* [*iff*]: *f contour_integrable_on* (*subpath u u g*)
using *has_contour_integral_subpath_refl contour_integrable_on_def* **by** *blast*

lemma *contour_integral_subpath_refl* [*simp*]: *contour_integral* (*subpath u u g*) *f* = *0*
by (*simp add*: *contour_integral_unique*)

lemma *has_contour_integral_subpath*:

```
  assumes f: f contour_integrable_on g and g: valid_path g
      and uv: u ∈ {0..1} v ∈ {0..1} u ≤ v
    shows (f has_contour_integral  integral {u..v} (λx. f(g x) * vector_derivative
g (at x)))
          (subpath u v g)
proof (cases v=u)
  case True
  then show ?thesis
   using f  by (simp add: contour_integrable_on_def subpath_def has_contour_integral)
next
  case False
  obtain S where S: ⋀x. x ∈ {0..1} - S ⟹ g differentiable at x and fs: finite
S
   using g unfolding piecewise_C1_differentiable_on_def C1_differentiable_on_eq
valid_path_def by blast
  have §: (λt. f (g t) * vector_derivative g (at t)) integrable_on {u..v}
    using contour_integrable_on f integrable_on_subinterval uv by fastforce
  then have *: ((λx. f (g ((v - u) * x + u)) * vector_derivative g (at ((v - u)
* x + u)))
         has_integral (1 / (v - u)) * integral {u..v} (λt. f (g t) * vector_derivative
g (at t)))
          {0..1}
    using uv False unfolding has_integral_integral
    apply simp
    apply (drule has_integral_affinity [where m=v-u and c=u, simplified])
   apply (simp_all add: image_affinity_atLeastAtMost_div_diff scaleR_conv_of_real)
    apply (simp add: divide_simps)
    done

  have vd: vector_derivative (λx. g ((v-u) * x + u)) (at x) = (v-u) *R vec-
tor_derivative g (at ((v-u) * x + u))
    if x ∈ {0..1}  x ∉ (λt. (v-u) *R t + u) -` S for x
  proof (rule vector_derivative_at [OF vector_diff_chain_at [simplified o_def]])
    show ((λx. (v - u) * x + u) has_vector_derivative v - u) (at x)
      by (intro derivative_eq_intros | simp)+
 qed (use S uv mult_left_le [of x v-u] that in ‹auto simp: vector_derivative_works›)

  have fin: finite ((λt. (v - u) *R t + u) -` S)
    using fs by (auto simp: inj_on_def False finite_vimageI)
  show ?thesis
    unfolding subpath_def has_contour_integral
    apply (rule has_integral_spike_finite [OF fin])
    using has_integral_cmul [OF *, where c = v-u] fs assms
    by (auto simp: False vd scaleR_conv_of_real)
qed

lemma contour_integrable_subpath:
  assumes f contour_integrable_on g valid_path g u ∈ {0..1} v ∈ {0..1}
    shows f contour_integrable_on (subpath u v g)
```

proof (*cases u v rule*: *linorder_class.le_cases*)
case *le*
then show *?thesis*
by (*metis contour_integrable_on_def has_contour_integral_subpath* [*OF assms*])
next
case *ge*
with *assms* **show** *?thesis*
by (*metis* (*no_types*, *lifting*) *contour_integrable_on_def contour_integrable_reversepath_eq has_contour_integral_subpath reversepath_subpath valid_path_subpath*)
qed

lemma *has_integral_contour_integral_subpath*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows (((λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*)))
has_integral contour_integral (*subpath u v g*) *f*) {*u..v*}
using *assms*
proof −
have (λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*)) *integrable_on* {*u..v*}
by (*metis* (*full_types*) *assms*(*1*) *assms*(*3*) *assms*(*4*) *atLeastAtMost_iff atLeastatMost_subset_iff contour_integrable_on integrable_on_subinterval*)
then have ((λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*)) *has_integral integral* {*u..v*} (λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*))) {*u..v*}
by *blast*
then show *?thesis*
by (*metis* (*full_types*) *assms contour_integral_unique has_contour_integral_subpath*)
qed

lemma *contour_integral_subcontour_integral*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows *contour_integral* (*subpath u v g*) *f* =
integral {*u..v*} (λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*))
using *assms has_contour_integral_subpath contour_integral_unique* **by** *blast*

lemma *contour_integral_subpath_combine_less*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
u<*v v*<*w*
shows *contour_integral* (*subpath u v g*) *f* + *contour_integral* (*subpath v w g*) *f* =
contour_integral (*subpath u w g*) *f*
proof −
have (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *integrable_on* {*u..w*}
using *integrable_on_subcbox* [**where** *a*=*u* **and** *b*=*w* **and** *S* = {*0..1*}] *assms*
by (*auto simp*: *contour_integrable_on*)
with *assms* **show** *?thesis*
by (*auto simp*: *contour_integral_subcontour_integral Henstock_Kurzweil_Integration.integral_combine*)
qed

lemma *contour_integral_subpath_combine*:

assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
shows *contour_integral* (*subpath u v g*) *f* + *contour_integral* (*subpath v w g*) *f* =
contour_integral (*subpath u w g*) *f*
proof (*cases u≠v* ∧ *v≠w* ∧ *u≠w*)
case *True*
have ∗: *subpath v u g* = *reversepath*(*subpath u v g*) ∧
subpath w u g = *reversepath*(*subpath u w g*) ∧
subpath w v g = *reversepath*(*subpath v w g*)
by (*auto simp*: *reversepath_subpath*)
have *u* < *v* ∧ *v* < *w* ∨
u < *w* ∧ *w* < *v* ∨
v < *u* ∧ *u* < *w* ∨
v < *w* ∧ *w* < *u* ∨
w < *u* ∧ *u* < *v* ∨
w < *v* ∧ *v* < *u*
using *True assms* **by** *linarith*
with *assms* **show** *?thesis*
using *contour_integral_subpath_combine_less* [*of f g u v w*]
contour_integral_subpath_combine_less [*of f g u w v*]
contour_integral_subpath_combine_less [*of f g v u w*]
contour_integral_subpath_combine_less [*of f g v w u*]
contour_integral_subpath_combine_less [*of f g w u v*]
contour_integral_subpath_combine_less [*of f g w v u*]
by (*elim disjE*) (*auto simp*: ∗ *contour_integral_reversepath contour_integrable_subpath valid_path_subpath algebra_simps*)
next
case *False*
with *assms* **show** *?thesis*
by (*metis add.right_neutral contour_integral_reversepath contour_integral_subpath_refl diff_0 eq_diff_eq add_0 reversepath_subpath valid_path_subpath*)
qed

lemma *contour_integral_integral*:
contour_integral g f = *integral* {*0..1*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*))
by (*simp add*: *contour_integral_def integral_def has_contour_integral contour_integrable_on*)

lemma *contour_integral_cong*:
assumes *g* = *g′* ⋀*x*. *x* ∈ *path_image g* ⟹ *f x* = *f′ x*
shows *contour_integral g f* = *contour_integral g′ f′*
unfolding *contour_integral_integral* **using** *assms*
by (*intro integral_cong*) (*auto simp*: *path_image_def*)

lemma *contour_integral_spike_finite_simple_path*:
assumes *finite A simple_path g g* = *g′* ⋀*x*. *x* ∈ *path_image g* − *A* ⟹ *f x* = *f′ x*
shows *contour_integral g f* = *contour_integral g′ f′*

unfolding *contour_integral_integral*
proof (*rule integral_spike*)
have *finite* ($g -` A \cap \{0<..<1\}$) **using** ‹*simple_path g*› ‹*finite A*›
by (*intro finite_vimage_IntI simple_path_inj_on*) *auto*
hence *finite* ($\{0, 1\} \cup g -` A \cap \{0<..<1\}$) **by** *auto*
thus *negligible* ($\{0, 1\} \cup g -` A \cap \{0<..<1\}$) **by** (*rule negligible_finite*)
next
fix *x* **assume** $x \in \{0..1\} - (\{0, 1\} \cup g -` A \cap \{0<..<1\})$
hence $g\ x \in$ *path_image g* $-$ *A* **by** (*auto simp*: *path_image_def*)
from *assms*(*4*)[*OF this*] **and** *assms*(*3*)
show $f'\ (g'\ x) * \mathit{vector\_derivative}\ g'\ (\mathit{at}\ x) = f\ (g\ x) * \mathit{vector\_derivative}\ g\ (\mathit{at}\ x)$ **by** *simp*
qed

Contour integral along a segment on the real axis

lemma *has_contour_integral_linepath_Reals_iff*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows (*f has_contour_integral I*) (*linepath a b*) ⟷
((λ*x*. *f* (*of_real x*)) *has_integral I*) {*Re a..Re b*}
proof −
from *assms* **have** [*simp*]: *of_real* (*Re a*) = *a of_real* (*Re b*) = *b*
by (*simp_all add*: *complex_eq_iff*)
from *assms* **have** *a* ≠ *b* **by** *auto*
have ((λ*x*. *f* (*of_real x*)) *has_integral I*) (*cbox* (*Re a*) (*Re b*)) ⟷
((λ*x*. *f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*)) *has_integral I* /$_R$ (*Re b* − *Re a*)) {*0..1*}
by (*subst has_integral_affinity_iff* [*of Re b* − *Re a* _ *Re a*, *symmetric*])
(*insert assms*, *simp_all add*: *field_simps scaleR_conv_of_real*)
also have (λ*x*. *f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*)) =
(λ*x*. (*f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*) ∗ (*b* − *a*)) /$_R$ (*Re b* − *Re a*))
using ‹*a* ≠ *b*› **by** (*auto simp*: *field_simps fun_eq_iff scaleR_conv_of_real*)
also have (... *has_integral I* /$_R$ (*Re b* − *Re a*)) {*0..1*} ⟷
((λ*x*. *f* (*linepath a b x*) ∗ (*b* − *a*)) *has_integral I*) {*0..1*} **using** *assms*
by (*subst has_integral_cmul_iff*) (*auto simp*: *linepath_def scaleR_conv_of_real algebra_simps*)
also have ... ⟷ (*f has_contour_integral I*) (*linepath a b*) **unfolding** *has_contour_integral_def*
by (*intro has_integral_cong*) (*simp add*: *vector_derivative_linepath_within*)
finally show *?thesis* **by** *simp*
qed

lemma *contour_integrable_linepath_Reals_iff*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows (*f contour_integrable_on linepath a b*) ⟷
(λ*x*. *f* (*of_real x*)) *integrable_on* {*Re a..Re b*}
using *has_contour_integral_linepath_Reals_iff*[*OF assms*, *of f*]
by (*auto simp*: *contour_integrable_on_def integrable_on_def*)

lemma *contour_integral_linepath_Reals_eq*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows *contour_integral* (*linepath a b*) *f* = *integral* {*Re a*..*Re b*} (λ*x*. *f* (*of_real x*))
proof (*cases f contour_integrable_on linepath a b*)
case *True*
thus *?thesis* **using** *has_contour_integral_linepath_Reals_iff*[*OF assms*, *of f*]
using *has_contour_integral_integral has_contour_integral_unique* **by** *blast*
next
case *False*
thus *?thesis* **using** *contour_integrable_linepath_Reals_iff*[*OF assms*, *of f*]
by (*simp add*: *not_integrable_contour_integral not_integrable_integral*)
qed

1.7 Cauchy's theorem where there's a primitive

lemma *contour_integral_primitive_lemma*:
fixes *f* :: *complex* ⇒ *complex* **and** *g* :: *real* ⇒ *complex*
assumes *a* ≤ *b*
and ⋀*x*. *x* ∈ *S* ⟹ (*f has_field_derivative f' x*) (*at x within S*)
and *g piecewise_differentiable_on* {*a*..*b*} ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *S*
shows ((λ*x*. *f'*(*g x*) ∗ *vector_derivative g* (*at x within* {*a*..*b*}))
has_integral (*f*(*g b*) − *f*(*g a*))) {*a*..*b*}
proof −
obtain *K* **where** *finite K* **and** *K*: ∀ *x*∈{*a*..*b*} − *K*. *g differentiable* (*at x within* {*a*..*b*}) **and** *cg*: *continuous_on* {*a*..*b*} *g*
using *assms* **by** (*auto simp*: *piecewise_differentiable_on_def*)
have *continuous_on* (*g* ‘ {*a*..*b*}) *f*
using *assms*
by (*metis field_differentiable_def field_differentiable_imp_continuous_at continuous_on_eq_continuous_within continuous_on_subset image_subset_iff*)
then have *cfg*: *continuous_on* {*a*..*b*} (λ*x*. *f* (*g x*))
by (*rule continuous_on_compose* [*OF cg*, *unfolded o_def*])
{ fix *x*::*real*
assume *a*: *a* < *x* **and** *b*: *x* < *b* **and** *xk*: *x* ∉ *K*
then have *g differentiable at x within* {*a*..*b*}
using *K* **by** (*simp add*: *differentiable_at_withinI*)
then have (*g has_vector_derivative vector_derivative g* (*at x within* {*a*..*b*})) (*at x within* {*a*..*b*})
by (*simp add*: *vector_derivative_works has_field_derivative_def scaleR_conv_of_real*)
then have *gdiff*: (*g has_derivative* (λ*u*. *u* ∗ *vector_derivative g* (*at x within* {*a*..*b*}))) (*at x within* {*a*..*b*})
by (*simp add*: *has_vector_derivative_def scaleR_conv_of_real*)
have (*f has_field_derivative* (*f'* (*g x*))) (*at* (*g x*) *within g* ‘ {*a*..*b*})
using *assms* **by** (*metis a atLeastAtMost_iff b DERIV_subset image_subset_iff less_eq_real_def*)
then have *fdiff*: (*f has_derivative* (∗) (*f'* (*g x*))) (*at* (*g x*) *within g* ‘ {*a*..*b*})

```
    by (simp add: has_field_derivative_def)
  have ((λx. f (g x)) has_vector_derivative f' (g x) * vector_derivative g (at x
within {a..b})) (at x within {a..b})
    using diff_chain_within [OF gdiff fdiff]
  by (simp add: has_vector_derivative_def scaleR_conv_of_real o_def mult_ac)
 } note * = this
 show ?thesis
  using assms cfg *
 by (force simp: at_within_Icc_at intro: fundamental_theorem_of_calculus_interior_strong
[OF ‹finite K›])
qed

lemma contour_integral_primitive:
 assumes ⋀x. x ∈ S ⟹ (f has_field_derivative f' x) (at x within S)
    and valid_path g path_image g ⊆ S
  shows (f' has_contour_integral (f(pathfinish g) − f(pathstart g))) g
 using assms
 apply (simp add: valid_path_def path_image_def pathfinish_def pathstart_def
has_contour_integral_def)
 apply (auto intro!: piecewise_C1_imp_differentiable contour_integral_primitive_lemma
[of 0 1 S])
 done

corollary Cauchy_theorem_primitive:
 assumes ⋀x. x ∈ S ⟹ (f has_field_derivative f' x) (at x within S)
    and valid_path g  path_image g ⊆ S pathfinish g = pathstart g
  shows (f' has_contour_integral 0) g
 using assms by (metis diff_self contour_integral_primitive)
```

Existence of path integral for continuous function

```
lemma contour_integrable_continuous_linepath:
 assumes continuous_on (closed_segment a b) f
 shows f contour_integrable_on (linepath a b)
proof −
 have continuous_on (closed_segment a b) (λx. f x * (b − a))
   by (rule continuous_intros | simp add: assms)+
 then have continuous_on {0..1} (λx. f (linepath a b x) * (b − a))
    by (metis (no_types, lifting) continuous_on_compose continuous_on_cong
continuous_on_linepath linepath_image_01 o_apply)
 then have (λx. f (linepath a b x) *
       vector_derivative (linepath a b)
        (at x within {0..1})) integrable_on
   {0..1}
    by (metis (no_types, lifting) continuous_on_cong integrable_continuous_real
vector_derivative_linepath_within)
 then show ?thesis
  by (simp add: contour_integrable_on_def has_contour_integral_def integrable_on_def
[symmetric])
qed
```

lemma *has_field_der_id*: $((\lambda x.\ x^2/2)$ *has_field_derivative* $x)$ $(at\ x)$
by (*rule has_derivative_imp_has_field_derivative*)
(*rule derivative_intros* | *simp*)+

lemma *contour_integral_id* [*simp*]: *contour_integral* (*linepath a b*) $(\lambda y.\ y) = (b\hat{}2 - a\hat{}2)/2$
using *contour_integral_primitive* [*of UNIV* $\lambda x.\ x\hat{}2/2$ $\lambda x.\ x$ *linepath a b*] *contour_integral_unique*
by (*simp add*: *has_field_der_id*)

lemma *contour_integrable_on_const* [*iff*]: $(\lambda x.\ c)$ *contour_integrable_on* (*linepath a b*)
by (*simp add*: *contour_integrable_continuous_linepath*)

lemma *contour_integrable_on_id* [*iff*]: $(\lambda x.\ x)$ *contour_integrable_on* (*linepath a b*)
by (*simp add*: *contour_integrable_continuous_linepath*)

1.8 Arithmetical combining theorems

lemma *has_contour_integral_neg*:
$(f$ *has_contour_integral* $i)$ $g \Longrightarrow ((\lambda x.\ -(f\ x))$ *has_contour_integral* $(-i))$ g
by (*simp add*: *has_integral_neg has_contour_integral_def*)

lemma *has_contour_integral_add*:
$[\![(f1$ *has_contour_integral i1*$)$ g; $(f2$ *has_contour_integral i2*$)$ $g]\!]$
$\Longrightarrow ((\lambda x.\ f1\ x + f2\ x)$ *has_contour_integral* $(i1 + i2))$ g
by (*simp add*: *has_integral_add has_contour_integral_def algebra_simps*)

lemma *has_contour_integral_diff*:
$[\![(f1$ *has_contour_integral i1*$)$ g; $(f2$ *has_contour_integral i2*$)$ $g]\!]$
$\Longrightarrow ((\lambda x.\ f1\ x - f2\ x)$ *has_contour_integral* $(i1 - i2))$ g
by (*simp add*: *has_integral_diff has_contour_integral_def algebra_simps*)

lemma *has_contour_integral_lmul*:
$(f$ *has_contour_integral* $i)$ $g \Longrightarrow ((\lambda x.\ c * (f\ x))$ *has_contour_integral* $(c*i))$ g
by (*simp add*: *has_contour_integral_def algebra_simps has_integral_mult_right*)

lemma *has_contour_integral_rmul*:
$(f$ *has_contour_integral* $i)$ $g \Longrightarrow ((\lambda x.\ (f\ x) * c)$ *has_contour_integral* $(i*c))$ g
by (*simp add*: *mult.commute has_contour_integral_lmul*)

lemma *has_contour_integral_div*:
$(f$ *has_contour_integral* $i)$ $g \Longrightarrow ((\lambda x.\ f\ x/c)$ *has_contour_integral* $(i/c))$ g
by (*simp add*: *field_class.field_divide_inverse*) (*metis has_contour_integral_rmul*)

lemma *has_contour_integral_eq*:
$[\![(f$ *has_contour_integral* $y)$ p; $\bigwedge x.\ x \in$ *path_image* $p \Longrightarrow f\ x = g\ x]\!] \Longrightarrow (g$

has_contour_integral y) *p*
by (*metis* (*mono_tags*, *lifting*) *has_contour_integral_def has_integral_eq image_eqI path_image_def*)

lemma *has_contour_integral_bound_linepath*:
assumes (*f has_contour_integral i*) (*linepath a b*)
0 ≤ *B* **and** *B*: ⋀*x*. *x* ∈ *closed_segment a b* ⟹ *norm*(*f x*) ≤ *B*
shows *norm i* ≤ *B* ∗ *norm*(*b* − *a*)
proof −
have *norm i* ≤ (*B* ∗ *norm* (*b* − *a*)) ∗ *content* (*cbox 0* (*1*::*real*))
proof (*rule has_integral_bound*
[*of* _ λ*x*. *f* (*linepath a b x*) ∗ *vector_derivative* (*linepath a b*) (*at x within* {*0*..*1*})])
show *cmod* (*f* (*linepath a b x*) ∗ *vector_derivative* (*linepath a b*) (*at x within* {*0*..*1*}))
≤ *B* ∗ *cmod* (*b* − *a*)
if *x* ∈ *cbox 0 1* **for** *x*::*real*
using *that box_real*(*2*) *norm_mult*
by (*metis B linepath_in_path mult_right_mono norm_ge_zero vector_derivative_linepath_within*)
qed (*use assms has_contour_integral_def* **in** *auto*)
then show *?thesis*
by (*auto simp*: *content_real*)
qed

lemma *has_contour_integral_const_linepath*: ((λ*x*. *c*) *has_contour_integral c*∗(*b* − *a*))(*linepath a b*)
unfolding *has_contour_integral_linepath*
by (*metis content_real diff_0_right has_integral_const_real lambda_one of_real_1 scaleR_conv_of_real zero_le_one*)

lemma *has_contour_integral_0*: ((λ*x*. *0*) *has_contour_integral 0*) *g*
by (*simp add*: *has_contour_integral_def*)

lemma *has_contour_integral_is_0*:
(⋀*z*. *z* ∈ *path_image g* ⟹ *f z* = *0*) ⟹ (*f has_contour_integral 0*) *g*
by (*rule has_contour_integral_eq* [*OF has_contour_integral_0*]) *auto*

lemma *has_contour_integral_sum*:
⟦*finite s*; ⋀*a*. *a* ∈ *s* ⟹ (*f a has_contour_integral i a*) *p*⟧
⟹ ((λ*x*. *sum* (λ*a*. *f a x*) *s*) *has_contour_integral sum i s*) *p*
by (*induction s rule*: *finite_induct*) (*auto simp*: *has_contour_integral_0 has_contour_integral_add*)

1.9 Operations on path integrals

lemma *contour_integral_const_linepath* [*simp*]: *contour_integral* (*linepath a b*) (λ*x*. *c*) = *c*∗(*b* − *a*)
by (*rule contour_integral_unique* [*OF has_contour_integral_const_linepath*])

lemma *contour_integral_neg*: *contour_integral g* (λ*z*. −*f z*) = −*contour_integral*

g f
by (*simp add*: *contour_integral_integral*)

lemma *contour_integral_add*:
f1 contour_integrable_on g ⟹ *f2 contour_integrable_on g* ⟹ *contour_integral g* (λ*x*. *f1 x* + *f2 x*) =
contour_integral g f1 + *contour_integral g f2*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_add*)

lemma *contour_integral_diff*:
f1 contour_integrable_on g ⟹ *f2 contour_integrable_on g* ⟹ *contour_integral g* (λ*x*. *f1 x* − *f2 x*) =
contour_integral g f1 − *contour_integral g f2*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_diff*)

lemma *contour_integral_lmul*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *c* ∗ *f x*) = *c*∗*contour_integral g f*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_lmul*)

lemma *contour_integral_rmul*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *f x* ∗ *c*) = *contour_integral g f* ∗ *c*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_rmul*)

lemma *contour_integral_div*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *f x* / *c*) = *contour_integral g f* / *c*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_div*)

lemma *contour_integral_eq*:
(⋀*x*. *x* ∈ *path_image p* ⟹ *f x* = *g x*) ⟹ *contour_integral p f* = *contour_integral p g*
using *contour_integral_cong contour_integral_def* **by** *fastforce*

lemma *contour_integral_eq_0*:
(⋀*z*. *z* ∈ *path_image g* ⟹ *f z* = *0*) ⟹ *contour_integral g f* = *0*
by (*simp add*: *has_contour_integral_is_0 contour_integral_unique*)

lemma *contour_integral_bound_linepath*:
shows
⟦*f contour_integrable_on* (*linepath a b*);
0 ≤ *B*; ⋀*x*. *x* ∈ *closed_segment a b* ⟹ *norm*(*f x*) ≤ *B*⟧
⟹ *norm*(*contour_integral* (*linepath a b*) *f*) ≤ *B*∗*norm*(*b* − *a*)
by (*meson has_contour_integral_bound_linepath has_contour_integral_integral*)

lemma *contour_integral_0* [*simp*]: *contour_integral g* (λ*x*. *0*) = *0*
by (*simp add*: *contour_integral_unique has_contour_integral_0*)

lemma *contour_integral_sum*:
⟦*finite s*; ⋀*a*. *a* ∈ *s* ⟹ (*f a*) *contour_integrable_on p*⟧
⟹ *contour_integral p* (λ*x*. *sum* (λ*a*. *f a x*) *s*) = *sum* (λ*a*. *contour_integral p* (*f a*)) *s*
by (*auto simp*: *contour_integral_unique has_contour_integral_sum has_contour_integral_integral*)

lemma *contour_integrable_eq*:
⟦*f contour_integrable_on p*; ⋀*x*. *x* ∈ *path_image p* ⟹ *f x* = *g x*⟧ ⟹ *g contour_integrable_on p*
unfolding *contour_integrable_on_def*
by (*metis has_contour_integral_eq*)

1.10 Arithmetic theorems for path integrability

lemma *contour_integrable_neg*:
f contour_integrable_on g ⟹ (λ*x*. −(*f x*)) *contour_integrable_on g*
using *has_contour_integral_neg contour_integrable_on_def* **by** *blast*

lemma *contour_integrable_add*:
⟦*f1 contour_integrable_on g*; *f2 contour_integrable_on g*⟧ ⟹ (λ*x*. *f1 x* + *f2 x*) *contour_integrable_on g*
using *has_contour_integral_add contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_diff*:
⟦*f1 contour_integrable_on g*; *f2 contour_integrable_on g*⟧ ⟹ (λ*x*. *f1 x* − *f2 x*) *contour_integrable_on g*
using *has_contour_integral_diff contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_lmul*:
f contour_integrable_on g ⟹ (λ*x*. *c* ∗ *f x*) *contour_integrable_on g*
using *has_contour_integral_lmul contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_rmul*:
f contour_integrable_on g ⟹ (λ*x*. *f x* ∗ *c*) *contour_integrable_on g*
using *has_contour_integral_rmul contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_div*:
f contour_integrable_on g ⟹ (λ*x*. *f x* / *c*) *contour_integrable_on g*
using *has_contour_integral_div contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_sum*:
⟦*finite s*; ⋀*a*. *a* ∈ *s* ⟹ (*f a*) *contour_integrable_on p*⟧
⟹ (λ*x*. *sum* (λ*a*. *f a x*) *s*) *contour_integrable_on p*
unfolding *contour_integrable_on_def*

by (*metis has_contour_integral_sum*)

lemma *contour_integrable_neg_iff*:
(λ*x*. −*f x*) *contour_integrable_on g* ⟷ *f contour_integrable_on g*
using *contour_integrable_neg*[*of f g*] *contour_integrable_neg*[*of* λ*x*. −*f x g*] **by** *auto*

lemma *contour_integrable_lmul_iff*:
c ≠ *0* ⟹ (λ*x*. *c* ∗ *f x*) *contour_integrable_on g* ⟷ *f contour_integrable_on g*
using *contour_integrable_lmul*[*of f g c*] *contour_integrable_lmul*[*of* λ*x*. *c* ∗ *f x g inverse c*]
by (*auto simp*: *field_simps*)

lemma *contour_integrable_rmul_iff*:
c ≠ *0* ⟹ (λ*x*. *f x* ∗ *c*) *contour_integrable_on g* ⟷ *f contour_integrable_on g*
using *contour_integrable_rmul*[*of f g c*] *contour_integrable_rmul*[*of* λ*x*. *c* ∗ *f x g inverse c*]
by (*auto simp*: *field_simps*)

lemma *contour_integrable_div_iff*:
c ≠ *0* ⟹ (λ*x*. *f x* / *c*) *contour_integrable_on g* ⟷ *f contour_integrable_on g*
using *contour_integrable_rmul_iff*[*of inverse c*] **by** (*simp add*: *field_simps*)

1.11 Reversing a path integral

lemma *has_contour_integral_reverse_linepath*:
(*f has_contour_integral i*) (*linepath a b*)
⟹ (*f has_contour_integral* (−*i*)) (*linepath b a*)
using *has_contour_integral_reversepath valid_path_linepath* **by** *fastforce*

lemma *contour_integral_reverse_linepath*:
continuous_on (*closed_segment a b*) *f* ⟹ *contour_integral* (*linepath a b*) *f* = − (*contour_integral*(*linepath b a*) *f*)
using *contour_integral_reversepath* **by** *fastforce*

Splitting a path integral in a flat way.*)

lemma *has_contour_integral_split*:
assumes *f*: (*f has_contour_integral i*) (*linepath a c*) (*f has_contour_integral j*) (*linepath c b*)
and *k*: *0* ≤ *k k* ≤ *1*
and *c*: *c* − *a* = *k* ∗$_R$ (*b* − *a*)
shows (*f has_contour_integral* (*i* + *j*)) (*linepath a b*)
proof (*cases k* = *0* ∨ *k* = *1*)
case *True*
then show *?thesis*
using *assms* **by** *auto*

```
next
  case False
  then have k: 0 < k k < 1 complex_of_real k ≠ 1
    using assms by auto
  have c': c = k *R (b − a) + a
    by (metis diff_add_cancel c)
  have bc: (b − c) = (1 − k) *R (b − a)
    by (simp add: algebra_simps c')
  { assume *: ((λx. f ((1 − x) *R a + x *R c) * (c − a)) has_integral i) {0..1}
    have ⋀x. (x / k) *R a + ((k − x) / k) *R a = a
    using False by (simp add: field_split_simps flip: real_vector.scale_left_distrib)
    then have ⋀x. ((k − x) / k) *R a + (x / k) *R c = (1 − x) *R a + x *R b
      using False by (simp add: c' algebra_simps)
    then have ((λx. f ((1 − x) *R a + x *R b) * (b − a)) has_integral i) {0..k}
      using k has_integral_affinity01 [OF *, of inverse k 0]
      by (force dest: has_integral_cmul [where c = inverse k]
          simp add: divide_simps mult.commute [of _ k] image_affinity_atLeastAtMost
c)
  } note fi = this
  { assume *: ((λx. f ((1 − x) *R c + x *R b) * (b − c)) has_integral j) {0..1}
   have **: ⋀x. (((1 − x) / (1 − k)) *R c + ((x − k) / (1 − k)) *R b) = ((1 −
x) *R a + x *R b)
      using k unfolding c' scaleR_conv_of_real
      apply (simp add: divide_simps)
     apply (simp add: distrib_right distrib_left right_diff_distrib left_diff_distrib)
      done
    have ((λx. f ((1 − x) *R a + x *R b) * (b − a)) has_integral j) {k..1}
      using k has_integral_affinity01 [OF *, of inverse(1 − k) −(k/(1 − k))]
    apply (simp add: divide_simps mult.commute [of _ 1−k] image_affinity_atLeastAtMost
** bc)
       apply (auto dest: has_integral_cmul [where k = (1 − k) *R j and c =
inverse (1 − k)])
      done
  } note fj = this
  show ?thesis
    using f k unfolding has_contour_integral_linepath
    by (simp add: linepath_def has_integral_combine [OF _ _ fi fj])
qed

lemma continuous_on_closed_segment_transform:
  assumes f: continuous_on (closed_segment a b) f
      and k: 0 ≤ k k ≤ 1
      and c: c − a = k *R (b − a)
    shows continuous_on (closed_segment a c) f
proof −
  have c': c = (1 − k) *R a + k *R b
    using c by (simp add: algebra_simps)
  have closed_segment a c ⊆ closed_segment a b
    by (metis c' ends_in_segment(1) in_segment(1) k subset_closed_segment)
```

then show *continuous_on* (*closed_segment a c*) *f*
by (*rule continuous_on_subset* [*OF f*])
qed

lemma *contour_integral_split*:
assumes *f*: *continuous_on* (*closed_segment a b*) *f*
and *k*: *0* ≤ *k k* ≤ *1*
and *c*: *c* − *a* = *k* ∗*R* (*b* − *a*)
shows *contour_integral*(*linepath a b*) *f* = *contour_integral*(*linepath a c*) *f* + *contour_integral*(*linepath c b*) *f*
proof −
have *c′*: *c* = (*1* − *k*) ∗*R* *a* + *k* ∗*R* *b*
using *c* **by** (*simp add*: *algebra_simps*)
have *closed_segment a c* ⊆ *closed_segment a b*
by (*metis c′ ends_in_segment*(*1*) *in_segment*(*1*) *k subset_closed_segment*)
moreover have *closed_segment c b* ⊆ *closed_segment a b*
by (*metis c′ ends_in_segment*(*2*) *in_segment*(*1*) *k subset_closed_segment*)
ultimately
have ∗: *continuous_on* (*closed_segment a c*) *f continuous_on* (*closed_segment c b*) *f*
by (*auto intro*: *continuous_on_subset* [*OF f*])
show *?thesis*
by (*rule contour_integral_unique*) (*meson* ∗ *c contour_integrable_continuous_linepath has_contour_integral_integral has_contour_integral_split k*)
qed

lemma *contour_integral_split_linepath*:
assumes *f*: *continuous_on* (*closed_segment a b*) *f*
and *c*: *c* ∈ *closed_segment a b*
shows *contour_integral*(*linepath a b*) *f* = *contour_integral*(*linepath a c*) *f* + *contour_integral*(*linepath c b*) *f*
using *c* **by** (*auto simp*: *closed_segment_def algebra_simps intro*!: *contour_integral_split* [*OF f*])

1.12 Reversing the order in a double path integral

The condition is stronger than needed but it's often true in typical situations

lemma *fst_im_cbox* [*simp*]: *cbox c d* ≠ {} ⟹ (*fst* ‘ *cbox* (*a*,*c*) (*b*,*d*)) = *cbox a b*
by (*auto simp*: *cbox_Pair_eq*)

lemma *snd_im_cbox* [*simp*]: *cbox a b* ≠ {} ⟹ (*snd* ‘ *cbox* (*a*,*c*) (*b*,*d*)) = *cbox c d*
by (*auto simp*: *cbox_Pair_eq*)

proposition *contour_integral_swap*:
assumes *fcon*: *continuous_on* (*path_image g* × *path_image h*) (λ(*y1*,*y2*). *f y1 y2*)
and *vp*: *valid_path g valid_path h*
and *gvcon*: *continuous_on* {*0..1*} (λ*t*. *vector_derivative g* (*at t*))

```
    and hvcon: continuous_on {0..1} (λt. vector_derivative h (at t))
  shows contour_integral g (λw. contour_integral h (f w)) =
        contour_integral h (λz. contour_integral g (λw. f w z))
proof −
  have gcon: continuous_on {0..1} g and hcon: continuous_on {0..1} h
   using assms by (auto simp: valid_path_def piecewise_C1_differentiable_on_def)
  have fgh1: ⋀x. (λt. f (g x) (h t)) = (λ(y1,y2). f y1 y2) ∘ (λt. (g x, h t))
    by (rule ext) simp
  have fgh2: ⋀x. (λt. f (g t) (h x)) = (λ(y1,y2). f y1 y2) ∘ (λt. (g t, h x))
    by (rule ext) simp
  have fcon_im1: ⋀x. 0 ≤ x ⟹ x ≤ 1 ⟹ continuous_on ((λt. (g x, h t)) ‘
{0..1}) (λ(x, y). f x y)
    by (rule continuous_on_subset [OF fcon]) (auto simp: path_image_def)
  have fcon_im2: ⋀x. 0 ≤ x ⟹ x ≤ 1 ⟹ continuous_on ((λt. (g t, h x)) ‘
{0..1}) (λ(x, y). f x y)
    by (rule continuous_on_subset [OF fcon]) (auto simp: path_image_def)
  have continuous_on (cbox (0, 0) (1, 1::real)) ((λx. vector_derivative g (at x))
∘ fst)
       continuous_on (cbox (0, 0) (1::real, 1)) ((λx. vector_derivative h (at x)) ∘
snd)
    by (rule continuous_intros | simp add: gvcon hvcon)+
 then have gvcon′: continuous_on (cbox (0, 0) (1, 1::real)) (λz. vector_derivative
g (at (fst z)))
      and hvcon′: continuous_on (cbox (0, 0) (1::real, 1)) (λx. vector_derivative
h (at (snd x)))
    by auto
  have continuous_on (cbox (0, 0) (1, 1)) ((λ(y1, y2). f y1 y2) ∘ (λw. ((g ∘ fst)
w, (h ∘ snd) w)))
    apply (intro gcon hcon continuous_intros | simp)+
    apply (auto simp: path_image_def intro: continuous_on_subset [OF fcon])
    done
  then have fgh: continuous_on (cbox (0, 0) (1, 1)) (λx. f (g (fst x)) (h (snd
x)))
    by auto
  have integral {0..1} (λx. contour_integral h (f (g x)) ∗ vector_derivative g (at
x)) =
        integral {0..1} (λx. contour_integral h (λy. f (g x) y ∗ vector_derivative g
(at x)))
  proof (rule integral_cong [OF contour_integral_rmul [symmetric]])
    have ⋀x. x ∈ {0..1} ⟹
         continuous_on {0..1} (λxa. f (g x) (h xa))
    by (subst fgh1) (rule fcon_im1 hcon continuous_intros | simp)+
    then show ⋀x. x ∈ {0..1} ⟹ f (g x) contour_integrable_on h
      unfolding contour_integrable_on
      using continuous_on_mult hvcon integrable_continuous_real by blast
  qed
  also have ... = integral {0..1}
                  (λy. contour_integral g (λx. f x (h y) ∗ vector_derivative h (at
y)))
```

unfolding *contour_integral_integral*
apply (*subst integral_swap_continuous* [**where** *'a* = *real* **and** *'b* = *real*, *of 0 0 1 1*, *simplified*])
subgoal
by (*rule fgh gvcon' hvcon' continuous_intros* | *simp add*: *split_def*)+
subgoal
unfolding *integral_mult_left* [*symmetric*]
by (*simp only*: *mult_ac*)
done
also have ... = *contour_integral h* (λ*z*. *contour_integral g* (λ*w*. *f w z*))
unfolding *contour_integral_integral integral_mult_left* [*symmetric*]
by (*simp add*: *algebra_simps*)
finally show *?thesis*
by (*simp add*: *contour_integral_integral*)
qed

lemma *valid_path_negatepath*: *valid_path* γ ⟹ *valid_path* (*uminus* ∘ γ)
unfolding *o_def* **using** *piecewise_C1_differentiable_neg valid_path_def* **by** *blast*

lemma *has_contour_integral_negatepath*:
assumes γ: *valid_path* γ **and** *cint*: ((λ*z*. *f* (− *z*)) *has_contour_integral* − *i*) γ
shows (*f has_contour_integral i*) (*uminus* ∘ γ)
proof −
obtain *S* **where** *cont*: *continuous_on* {*0..1*} γ **and** *finite S* **and** *diff*: γ *C1_differentiable_on* {*0..1*} − *S*
using γ **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def*)
have ((λ*x*. − (*f* (− γ *x*) ∗ *vector_derivative* γ (*at x within* {*0..1*}))) *has_integral i*) {*0..1*}
using *cint* **by** (*auto simp*: *has_contour_integral_def dest*: *has_integral_neg*)
then
have ((λ*x*. *f* (− γ *x*) ∗ *vector_derivative* (*uminus* ∘ γ) (*at x within* {*0..1*})) *has_integral i*) {*0..1*}
proof (*rule rev_iffD1* [*OF* _ *has_integral_spike_eq*])
show *negligible S*
by (*simp add*: ‹*finite S*› *negligible_finite*)
show *f* (− γ *x*) ∗ *vector_derivative* (*uminus* ∘ γ) (*at x within* {*0..1*}) =
− (*f* (− γ *x*) ∗ *vector_derivative* γ (*at x within* {*0..1*}))
if *x* ∈ {*0..1*} − *S* **for** *x*
proof −
have *vector_derivative* (*uminus* ∘ γ) (*at x within cbox 0 1*) = − *vector_derivative* γ (*at x within cbox 0 1*)
proof (*rule vector_derivative_within_cbox*)
show (*uminus* ∘ γ *has_vector_derivative* − *vector_derivative* γ (*at x within cbox 0 1*)) (*at x within cbox 0 1*)
using *that* **unfolding** *o_def*
by (*metis C1_differentiable_on_eq UNIV_I diff differentiable_subset has_vector_derivative_minus subsetI that vector_derivative_works*)
qed (*use that* **in** *auto*)

```
    then show ?thesis
      by simp
  qed
 qed
 then show ?thesis by (simp add: has_contour_integral_def)
qed

lemma contour_integrable_negatepath:
 assumes γ: valid_path γ and pi: (λz. f (− z)) contour_integrable_on γ
 shows f contour_integrable_on (uminus ∘ γ)
 by (metis γ add.inverse_inverse contour_integrable_on_def has_contour_integral_negatepath
pi)

lemma C1_differentiable_polynomial_function:
 fixes p :: real ⇒ 'a::euclidean_space
 shows polynomial_function p ⟹ p C1_differentiable_on S
 by (metis continuous_on_polymonial_function C1_differentiable_on_def has_vector_derivative_polynomi

lemma valid_path_polynomial_function:
 fixes p :: real ⇒ 'a::euclidean_space
 shows polynomial_function p ⟹ valid_path p
by (force simp: valid_path_def piecewise_C1_differentiable_on_def continuous_on_polymonial_function
C1_differentiable_polynomial_function)

lemma valid_path_subpath_trivial [simp]:
   fixes g :: real ⇒ 'a::euclidean_space
   shows z ≠ g x ⟹ valid_path (subpath x x g)
 by (simp add: subpath_def valid_path_polynomial_function)
```

1.13 Partial circle path

```
definition part_circlepath :: [complex, real, real, real, real] ⇒ complex
 where part_circlepath z r s t ≡ λx. z + of_real r * exp (i * of_real (linepath s
t x))

lemma pathstart_part_circlepath [simp]:
    pathstart(part_circlepath z r s t) = z + r*exp(i * s)
by (metis part_circlepath_def pathstart_def pathstart_linepath)

lemma pathfinish_part_circlepath [simp]:
    pathfinish(part_circlepath z r s t) = z + r*exp(i*t)
by (metis part_circlepath_def pathfinish_def pathfinish_linepath)

lemma reversepath_part_circlepath[simp]:
   reversepath (part_circlepath z r s t) = part_circlepath z r t s
 unfolding part_circlepath_def reversepath_def linepath_def
 by (auto simp:algebra_simps)

lemma has_vector_derivative_part_circlepath [derivative_intros]:
```

```
    ((part_circlepath z r s t) has_vector_derivative
      (i * r * (of_real t - of_real s) * exp(i * linepath s t x)))
     (at x within X)
  unfolding part_circlepath_def linepath_def scaleR_conv_of_real
  by (rule has_vector_derivative_real_field derivative_eq_intros | simp)+

lemma differentiable_part_circlepath:
  part_circlepath c r a b differentiable at x within A
  using has_vector_derivative_part_circlepath[of c r a b x A] differentiableI_vector
by blast

lemma vector_derivative_part_circlepath:
    vector_derivative (part_circlepath z r s t) (at x) =
      i * r * (of_real t - of_real s) * exp(i * linepath s t x)
  using has_vector_derivative_part_circlepath vector_derivative_at by blast

lemma vector_derivative_part_circlepath01:
    ⟦0 ≤ x; x ≤ 1⟧
    ⟹ vector_derivative (part_circlepath z r s t) (at x within {0..1}) =
         i * r * (of_real t - of_real s) * exp(i * linepath s t x)
  using has_vector_derivative_part_circlepath
  by (auto simp: vector_derivative_at_within_ivl)

lemma valid_path_part_circlepath [simp]: valid_path (part_circlepath z r s t)
  unfolding valid_path_def
  by (auto simp: C1_differentiable_on_eq vector_derivative_works vector_derivative_part_circlepath
has_vector_derivative_part_circlepath
           intro!: C1_differentiable_imp_piecewise continuous_intros)

lemma path_part_circlepath [simp]: path (part_circlepath z r s t)
  by (simp add: valid_path_imp_path)

proposition path_image_part_circlepath:
  assumes s ≤ t
    shows path_image (part_circlepath z r s t) = {z + r * exp(i * of_real x) | x.
s ≤ x ∧ x ≤ t}
proof -
  { fix z::real
    assume 0 ≤ z z ≤ 1
    with ‹s ≤ t› have ∃x. (exp (i * linepath s t z) = exp (i * of_real x)) ∧ s ≤ x
∧ x ≤ t
      apply (rule_tac x=(1 - z) * s + z * t in exI)
      apply (simp add: linepath_def scaleR_conv_of_real algebra_simps)
      by (metis (no_types) affine_ineq mult.commute mult_left_mono)
  }
  moreover
  { fix z
    assume s ≤ z z ≤ t
    then have z + of_real r * exp (i * of_real z) ∈ (λx. z + of_real r * exp (i *
```

```
linepath s t x)) ‘ {0..1}
      apply (rule_tac x=(z − s)/(t − s) in image_eqI)
      apply (simp add: linepath_def scaleR_conv_of_real divide_simps exp_eq)
      apply (auto simp: field_split_simps)
      done
  }
  ultimately show ?thesis
    by (fastforce simp add: path_image_def part_circlepath_def)
qed

lemma path_image_part_circlepath′:
  path_image (part_circlepath z r s t) = (λx. z + r ∗ cis x) ‘ closed_segment s t
proof −
  have path_image (part_circlepath z r s t) =
         (λx. z + r ∗ exp(i ∗ of_real x)) ‘ linepath s t ‘ {0..1}
    by (simp add: image_image path_image_def part_circlepath_def)
  also have linepath s t ‘ {0..1} = closed_segment s t
    by (rule linepath_image_01)
  finally show ?thesis by (simp add: cis_conv_exp)
qed

lemma path_image_part_circlepath_subset:
    ⟦s ≤ t; 0 ≤ r⟧ ⟹ path_image(part_circlepath z r s t) ⊆ sphere z r
by (auto simp: path_image_part_circlepath sphere_def dist_norm algebra_simps
norm_mult)

lemma in_path_image_part_circlepath:
  assumes w ∈ path_image(part_circlepath z r s t) s ≤ t 0 ≤ r
    shows norm(w − z) = r
proof −
  have w ∈ {c. dist z c = r}
   by (metis (no_types) path_image_part_circlepath_subset sphere_def subset_eq
assms)
  thus ?thesis
    by (simp add: dist_norm norm_minus_commute)
qed

lemma path_image_part_circlepath_subset′:
  assumes r ≥ 0
  shows   path_image (part_circlepath z r s t) ⊆ sphere z r
proof (cases s ≤ t)
  case True
 thus ?thesis using path_image_part_circlepath_subset[of s t r z] assms by simp
next
  case False
  thus ?thesis using path_image_part_circlepath_subset[of t s r z] assms
   by (subst reversepath_part_circlepath [symmetric], subst path_image_reversepath)
simp_all
qed
```

lemma *part_circlepath_cnj*: *cnj* (*part_circlepath c r a b x*) = *part_circlepath* (*cnj c*) *r* (−*a*) (−*b*) *x*
by (*simp add*: *part_circlepath_def exp_cnj linepath_def algebra_simps*)

lemma *contour_integral_bound_part_circlepath*:
assumes *f contour_integrable_on part_circlepath c r a b*
assumes $B \geq 0$ $r \geq 0$ $\bigwedge x.\ x \in$ *path_image* (*part_circlepath c r a b*) $\Longrightarrow$ *norm* (*f x*) $\leq B$
shows *norm* (*contour_integral* (*part_circlepath c r a b*) *f*) $\leq B * r * |b - a|$
proof −
let *?I* = *integral* {*0..1*} (λ*x*. *f* (*part_circlepath c r a b x*) ∗ i ∗ *of_real* (*r* ∗ (*b* − *a*)) ∗
exp (i ∗ *linepath a b x*))
have *norm ?I* ≤ *integral* {*0..1*} (λ*x*::*real*. *B* ∗ *1* ∗ (*r* ∗ |*b* − *a*|) ∗ *1*)
proof (*rule integral_norm_bound_integral*, *goal_cases*)
case *1*
with *assms*(*1*) **show** *?case*
by (*simp add*: *contour_integrable_on vector_derivative_part_circlepath mult_ac*)
next
case (*3 x*)
with *assms*(*2*−) **show** *?case* **unfolding** *norm_mult norm_of_real abs_mult*
by (*intro mult_mono*) (*auto simp*: *path_image_def*)
qed *auto*
also have *?I* = *contour_integral* (*part_circlepath c r a b*) *f*
by (*simp add*: *contour_integral_integral vector_derivative_part_circlepath mult_ac*)
finally show *?thesis* **by** *simp*
qed

lemma *has_contour_integral_part_circlepath_iff*:
assumes *a* < *b*
shows (*f has_contour_integral I*) (*part_circlepath c r a b*) ⟷
((λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *has_integral I*) {*a..b*}
proof −
have (*f has_contour_integral I*) (*part_circlepath c r a b*) ⟷
((λ*x*. *f* (*part_circlepath c r a b x*) ∗ *vector_derivative* (*part_circlepath c r a b*)
(*at x within* {*0..1*})) *has_integral I*) {*0..1*}
unfolding *has_contour_integral_def* **..**
also have ... ⟷ ((λ*x*. *f* (*part_circlepath c r a b x*) ∗ *r* ∗ (*b* − *a*) ∗ i ∗
cis (*linepath a b x*)) *has_integral I*) {*0..1*}
by (*intro has_integral_cong*, *subst vector_derivative_part_circlepath01*)
(*simp_all add*: *cis_conv_exp*)
also have ... ⟷ ((λ*x*. *f* (*c* + *r* ∗ *exp* (i ∗ *linepath* (*of_real a*) (*of_real b*) *x*)) ∗
r ∗ i ∗ *exp* (i ∗ *linepath* (*of_real a*) (*of_real b*) *x*) ∗
vector_derivative (*linepath* (*of_real a*) (*of_real b*))
(*at x within* {*0..1*})) *has_integral I*) {*0..1*}

by (*intro has_integral_cong*, *subst vector_derivative_linepath_within*)
(*auto simp*: *part_circlepath_def cis_conv_exp of_real_linepath* [*symmetric*])
also have ... $\longleftrightarrow$ ((λ*z*. *f* (*c* + *r* ∗ *exp* (i ∗ *z*)) ∗ *r* ∗ i ∗ *exp* (i ∗ *z*)) *has_contour_integral* *I*)
(*linepath* (*of_real a*) (*of_real b*))
by (*simp add*: *has_contour_integral_def*)
also have ... $\longleftrightarrow$ ((λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *has_integral I*) {*a*..*b*}
using *assms*
by (*subst has_contour_integral_linepath_Reals_iff*) (*simp_all add*: *cis_conv_exp*)
finally show *?thesis* **.**
qed

lemma *contour_integrable_part_circlepath_iff*:
assumes *a* < *b*
shows *f contour_integrable_on* (*part_circlepath c r a b*) $\longleftrightarrow$
(λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *integrable_on* {*a*..*b*}
using *assms* **by** (*auto simp*: *contour_integrable_on_def integrable_on_def*
has_contour_integral_part_circlepath_iff)

lemma *contour_integral_part_circlepath_eq*:
assumes *a* < *b*
shows *contour_integral* (*part_circlepath c r a b*) *f* =
integral {*a*..*b*} (λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*)
proof (*cases f contour_integrable_on part_circlepath c r a b*)
case *True*
hence (λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *integrable_on* {*a*..*b*}
using *assms* **by** (*simp add*: *contour_integrable_part_circlepath_iff*)
with *True* **show** *?thesis*
using *has_contour_integral_part_circlepath_iff*[*OF assms*]
contour_integral_unique has_integral_integrable_integral **by** *blast*
next
case *False*
hence ¬(λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *integrable_on* {*a*..*b*}
using *assms* **by** (*simp add*: *contour_integrable_part_circlepath_iff*)
with *False* **show** *?thesis*
by (*simp add*: *not_integrable_contour_integral not_integrable_integral*)
qed

lemma *contour_integral_part_circlepath_reverse*:
contour_integral (*part_circlepath c r a b*) *f* = −*contour_integral* (*part_circlepath* *c r b a*) *f*
by (*metis contour_integral_reversepath reversepath_part_circlepath valid_path_part_circlepath*)

lemma *contour_integral_part_circlepath_reverse′*:
b < *a* ⟹ *contour_integral* (*part_circlepath c r a b*) *f* =
−*contour_integral* (*part_circlepath c r b a*) *f*
by (*rule contour_integral_part_circlepath_reverse*)

lemma *finite_bounded_log*: *finite* {*z*::*complex*. *norm z* ≤ *b* ∧ *exp z* = *w*}

```
proof (cases w = 0)
  case True then show ?thesis by auto
next
  case False
  have *: finite {x. cmod ((2 * real_of_int x * pi) * i) ≤ b + cmod (Ln w)}
  proof (simp add: norm_mult finite_int_iff_bounded_le)
    show ∃k. abs ` {x. 2 * |of_int x| * pi ≤ b + cmod (Ln w)} ⊆ {..k}
    apply (rule_tac x=⌊(b + cmod (Ln w)) / (2*pi)⌋ in exI)
    apply (auto simp: field_split_simps le_floor_iff)
      done
  qed
  have [simp]: ⋀P f. {z. P z ∧ (∃n. z = f n)} = f ` {n. P (f n)}
    by blast
  have finite {z. cmod z ≤ b ∧ exp z = exp (Ln w)}
   using norm_add_leD by (fastforce intro: finite_subset [OF _ *] simp: exp_eq)
  then show ?thesis
    using False by auto
qed

lemma finite_bounded_log2:
  fixes a::complex
    assumes a ≠ 0
    shows finite {z. norm z ≤ b ∧ exp(a*z) = w}
proof −
  have *: finite ((λz. z / a) ` {z. cmod z ≤ b * cmod a ∧ exp z = w})
    by (rule finite_imageI [OF finite_bounded_log])
  show ?thesis
    by (rule finite_subset [OF _ *]) (force simp: assms norm_mult)
qed

lemma has_contour_integral_bound_part_circlepath_strong:
  assumes fi: (f has_contour_integral i) (part_circlepath z r s t)
      and finite k and le: 0 ≤ B 0 < r s ≤ t
      and B: ⋀x. x ∈ path_image(part_circlepath z r s t) − k ⟹ norm(f x) ≤ B
    shows cmod i ≤ B * r * (t − s)
proof −
  consider s = t | s < t using ‹s ≤ t› by linarith
  then show ?thesis
  proof cases
    case 1 with fi [unfolded has_contour_integral]
    have i = 0  by (simp add: vector_derivative_part_circlepath)
    with assms show ?thesis by simp
  next
    case 2
    have [simp]: |r| = r using ‹r > 0› by linarith
    have [simp]: cmod (complex_of_real t − complex_of_real s) = t−s
      by (metis 2 abs_of_pos diff_gt_0_iff_gt norm_of_real of_real_diff)
    have finite (part_circlepath z r s t −` {y} ∩ {0..1}) if y ∈ k for y
    proof −
```

```
      let ?w = (y − z)/of_real r / exp(i ∗ of_real s)
      have fin: finite (of_real −‘ {z. cmod z ≤ 1 ∧ exp (i ∗ complex_of_real (t −
s) ∗ z) = ?w})
        using ‹s < t›
       by (intro finite_vimageI [OF finite_bounded_log2]) (auto simp: inj_of_real)
      show ?thesis
        unfolding part_circlepath_def linepath_def vimage_def
        using le
      by (intro finite_subset [OF _ fin]) (auto simp: algebra_simps scaleR_conv_of_real
exp_add exp_diff)
    qed
    then have fin01: finite ((part_circlepath z r s t) −‘ k ∩ {0..1})
      by (rule finite_finite_vimage_IntI [OF ‹finite k›])
    have ∗∗: ((λx. if (part_circlepath z r s t x) ∈ k then 0
                   else f(part_circlepath z r s t x) ∗
                        vector_derivative (part_circlepath z r s t) (at x)) has_integral
i) {0..1}
     by (rule has_integral_spike [OF negligible_finite [OF fin01]]) (use fi has_contour_integral
in auto)
      have ∗: ⋀x. ⟦0 ≤ x; x ≤ 1; part_circlepath z r s t x ∉ k⟧ ⟹ cmod (f
(part_circlepath z r s t x)) ≤ B
      by (auto intro!: B [unfolded path_image_def image_def])
    show ?thesis
      apply (rule has_integral_bound [where ’a=real, simplified, OF _ ∗∗, sim-
plified])
    using assms le ∗ 2 ‹r > 0› by (auto simp add: norm_mult vector_derivative_part_circlepath)
  qed
qed

corollary contour_integral_bound_part_circlepath_strong:
  assumes f contour_integrable_on part_circlepath z r s t
      and finite k and 0 ≤ B 0 < r s ≤ t
      and ⋀x. x ∈ path_image(part_circlepath z r s t) − k ⟹ norm(f x) ≤ B
    shows cmod (contour_integral (part_circlepath z r s t) f) ≤ B ∗ r ∗ (t − s)
 using assms has_contour_integral_bound_part_circlepath_strong has_contour_integral_integral
by blast

lemma has_contour_integral_bound_part_circlepath:
      ⟦(f has_contour_integral i) (part_circlepath z r s t);
        0 ≤ B; 0 < r; s ≤ t;
        ⋀x. x ∈ path_image(part_circlepath z r s t) ⟹ norm(f x) ≤ B⟧
       ⟹ norm i ≤ B∗r∗(t − s)
  by (auto intro: has_contour_integral_bound_part_circlepath_strong)

lemma contour_integrable_continuous_part_circlepath:
     continuous_on (path_image (part_circlepath z r s t)) f
      ⟹ f contour_integrable_on (part_circlepath z r s t)
  unfolding contour_integrable_on has_contour_integral_def vector_derivative_part_circlepath
path_image_def
```

by (*best intro*: *integrable_continuous_real path_part_circlepath* [*unfolded path_def*] *continuous_intros*
continuous_on_compose2 [**where** *g*=*f*, *OF* _ _ *order_refl*])

lemma *simple_path_part_circlepath*:
simple_path(*part_circlepath z r s t*) ⟷ (*r* ≠ *0* ∧ *s* ≠ *t* ∧ |*s* − *t*| ≤ *2*∗*pi*)
proof (*cases r* = *0* ∨ *s* = *t*)
case *True*
then show *?thesis*
unfolding *part_circlepath_def simple_path_def loop_free_def*
by (*rule disjE*) (*force intro*: *bexI* [**where** *x* = *1/4*] *bexI* [**where** *x* = *1/3*])+
next
case *False* **then have** *r* ≠ *0 s* ≠ *t* **by** *auto*
have ∗: ⋀*x y z s t*. i∗((*1* − *x*) ∗ *s* + *x* ∗ *t*) = i∗(((*1* − *y*) ∗ *s* + *y* ∗ *t*)) + *z* ⟷ i∗(*x* − *y*) ∗ (*t* − *s*) = *z*
by (*simp add*: *algebra_simps*)
have *abs01*: ⋀*x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1*
⟹ (*x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0* ⟷ |*x* − *y*| ∈ {*0*,*1*})
by *auto*
have ∗∗: ⋀*x y*. (∃ *n*. (*complex_of_real x* − *of_real y*) ∗ (*of_real t* − *of_real s*) = *2* ∗ (*of_int n* ∗ *of_real pi*)) ⟷
(∃ *n*. |*x* − *y*| ∗ (*t* − *s*) = *2* ∗ (*of_int n* ∗ *pi*))
by (*force simp*: *algebra_simps abs_if dest*: *arg_cong* [**where** *f*=*Re*] *arg_cong* [**where** *f*=*complex_of_real*]
intro: *exI* [**where** *x* = −*n* **for** *n*])
have *1*: |*s* − *t*| ≤ *2* ∗ *pi*
if ⋀*x*. *0* ≤ *x* ∧ *x* ≤ *1* ⟹ (∃ *n*. *x* ∗ (*t* − *s*) = *2* ∗ (*real_of_int n* ∗ *pi*)) ⟶ *x* = *0* ∨ *x* = *1*
proof (*rule ccontr*)
assume ¬ |*s* − *t*| ≤ *2* ∗ *pi*
then have ∗: ⋀*n*. *t* − *s* ≠ *of_int n* ∗ |*s* − *t*|
using *False that* [*of 2*∗*pi* / |*t* − *s*|]
by (*simp add*: *abs_minus_commute divide_simps*)
show *False*
using ∗ [*of 1*] ∗ [*of* −*1*] **by** *auto*
qed
have *2*: |*s* − *t*| = |*2* ∗ (*real_of_int n* ∗ *pi*) / *x*| **if** *x* ≠ *0 x* ∗ (*t* − *s*) = *2* ∗ (*real_of_int n* ∗ *pi*) **for** *x n*
proof −
have *t*−*s* = *2* ∗ (*real_of_int n* ∗ *pi*)/*x*
using *that* **by** (*simp add*: *field_simps*)
then show *?thesis* **by** (*metis abs_minus_commute*)
qed
have *abs_away*: ⋀*P*. (∀ *x*∈{*0*..*1*}. ∀ *y*∈{*0*..*1*}. *P* |*x* − *y*|) ⟷ (∀ *x*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ⟶ *P x*)
by *force*
show *?thesis* **using** *False*
apply (*simp add*: *simple_path_def loop_free_def*)

apply (*simp add*: *part_circlepath_def linepath_def exp_eq* ∗ ∗∗ *abs01 del*: *Set.insert_iff*)
apply (*subst abs_away*)
apply (*auto simp*: *1*)
apply (*rule ccontr*)
apply (*auto simp*: *2 field_split_simps abs_mult dest*: *of_int_leD*)
done
qed

lemma *arc_part_circlepath*:
assumes $r \neq 0$ $s \neq t$ $|s - t| < 2*pi$
shows *arc* (*part_circlepath z r s t*)
proof −
have ∗: $x = y$ **if** *eq*: i ∗ (*linepath s t x*) = i ∗ (*linepath s t y*) + *2* ∗ *of_int n* ∗ *complex_of_real pi* ∗ i
and *x*: $x \in \{0..1\}$ **and** *y*: $y \in \{0..1\}$ **for** *x y n*
proof (*rule ccontr*)
assume $x \neq y$
have (*linepath s t x*) = (*linepath s t y*) + *2* ∗ *of_int n* ∗ *complex_of_real pi*
by (*metis add_divide_eq_iff complex_i_not_zero mult.commute nonzero_mult_div_cancel_left eq*)
then have $s*y + t*x = s*x + (t*y + of\_int\ n * (pi * 2))$
by (*force simp*: *algebra_simps linepath_def dest*: *arg_cong* [**where** *f*=*Re*])
with ‹$x \neq y$› **have** *st*: $s-t = (of\_int\ n * (pi * 2)\ /\ (y-x))$
by (*force simp*: *field_simps*)
have $|real\_of\_int\ n| < |y - x|$
using *assms* ‹$x \neq y$› **by** (*simp add*: *st abs_mult field_simps*)
then show *False*
using *assms x y st* **by** (*auto dest*: *of_int_lessD*)
qed
then have *inj_on* (*part_circlepath z r s t*) {*0..1*}
using *assms* **by** (*force simp add*: *part_circlepath_def inj_on_def exp_eq*)
then show *?thesis*
by (*simp add*: *arc_def*)
qed

1.14 Special case of one complete circle

definition *circlepath* :: [*complex*, *real*, *real*] ⇒ *complex*
where *circlepath z r* ≡ *part_circlepath z r 0* (*2∗pi*)

lemma *circlepath*: *circlepath z r* = (λ*x*. *z* + *r* ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ *of_real x*))
by (*simp add*: *circlepath_def part_circlepath_def linepath_def algebra_simps*)

lemma *pathstart_circlepath* [*simp*]: *pathstart* (*circlepath z r*) = *z* + *r*
by (*simp add*: *circlepath_def*)

lemma *pathfinish_circlepath* [*simp*]: *pathfinish* (*circlepath z r*) = *z* + *r*

by (*simp add*: *circlepath_def*) (*metis exp_two_pi_i mult.commute*)

lemma *circlepath_minus*: *circlepath z* (−*r*) *x* = *circlepath z r* (*x* + *1/2*)
proof −
have *z* + *of_real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ (*x* + *1/2*)) =
z + *of_real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x* + *pi* ∗ i)
by (*simp add*: *divide_simps*) (*simp add*: *algebra_simps*)
also have ... = *z* − *r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x*)
by (*simp add*: *exp_add*)
finally show *?thesis*
by (*simp add*: *circlepath path_image_def sphere_def dist_norm*)
qed

lemma *circlepath_add1*: *circlepath z r* (*x*+*1*) = *circlepath z r x*
using *circlepath_minus* [*of z r x*+*1/2*] *circlepath_minus* [*of z* −*r x*]
by (*simp add*: *add.commute*)

lemma *circlepath_add_half*: *circlepath z r* (*x* + *1/2*) = *circlepath z r* (*x* − *1/2*)
using *circlepath_add1* [*of z r x*−*1/2*]
by (*simp add*: *add.commute*)

lemma *path_image_circlepath_minus_subset*:
path_image (*circlepath z* (−*r*)) ⊆ *path_image* (*circlepath z r*)
proof −
have ∃*x*∈{*0..1*}. *circlepath z r* (*y* + *1/2*) = *circlepath z r x*
if *0* ≤ *y y* ≤ *1* **for** *y*
proof (*cases y* ≤ *1/2*)
case *False*
with *that* **show** *?thesis*
by (*force simp*: *circlepath_add_half*)
qed (*use that* **in** *force*)
then show *?thesis*
by (*auto simp add*: *path_image_def image_def circlepath_minus*)
qed

lemma *path_image_circlepath_minus*: *path_image* (*circlepath z* (−*r*)) = *path_image* (*circlepath z r*)
using *path_image_circlepath_minus_subset* **by** *fastforce*

lemma *has_vector_derivative_circlepath* [*derivative_intros*]:
((*circlepath z r*) *has_vector_derivative* (*2* ∗ *pi* ∗ i ∗ *r* ∗ *exp* (*2* ∗ *of_real pi* ∗ i ∗ *x*)))
(*at x within X*)
unfolding *circlepath_def scaleR_conv_of_real*
by (*rule derivative_eq_intros*) (*simp add*: *algebra_simps*)

lemma *vector_derivative_circlepath*:
vector_derivative (*circlepath z r*) (*at x*) =
2 ∗ *pi* ∗ i ∗ *r* ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ *x*)

using *has_vector_derivative_circlepath vector_derivative_at* **by** *blast*

lemma *vector_derivative_circlepath01*:
  ⟦*0* ≤ *x*; *x* ≤ *1*⟧
  ⟹ *vector_derivative* (*circlepath z r*) (*at x within* {*0..1*}) =
    2 ∗ *pi* ∗ i ∗ *r* ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ *x*)
using *has_vector_derivative_circlepath*
by (*auto simp*: *vector_derivative_at_within_ivl*)

lemma *valid_path_circlepath* [*simp*]: *valid_path* (*circlepath z r*)
by (*simp add*: *circlepath_def*)

lemma *path_circlepath* [*simp*]: *path* (*circlepath z r*)
by (*simp add*: *valid_path_imp_path*)

lemma *path_image_circlepath_nonneg*:
assumes *0* ≤ *r* **shows** *path_image* (*circlepath z r*) = *sphere z r*
proof −
have ∗: *x* ∈ (λ*u*. *z* + (*cmod* (*x* − *z*)) ∗ *exp* (i ∗ (*of_real u* ∗ (*of_real pi* ∗ *2*)))) ‘ {*0..1*} **for** *x*
proof (*cases x* = *z*)
case *True* **then show** *?thesis* **by** *force*
next
case *False*
define *w* **where** *w* = *x* − *z*
then have *w* ≠ *0* **by** (*simp add*: *False*)
have ∗∗: ⋀*t*. ⟦*Re w* = *cos t* ∗ *cmod w*; *Im w* = *sin t* ∗ *cmod w*⟧ ⟹ *w* = *of_real* (*cmod w*) ∗ *exp* (i ∗ *t*)
using *cis_conv_exp complex_eq_iff* **by** *auto*
obtain *t* **where** *0* ≤ *t t* < *2∗pi Re*(*w*/*norm w*) = *cos t Im*(*w*/*norm w*) = *sin t*
apply (*rule sincos_total_2pi* [*of Re*(*w*/(*norm w*)) *Im*(*w*/(*norm w*))])
by (*auto simp add*: *divide_simps* ‹*w* ≠ *0*› *cmod_power2* [*symmetric*])
then
show *?thesis*
using *False* ∗∗ *w_def* ‹*w* ≠ *0*›
by (*rule_tac x*=*t* / (*2∗pi*) **in** *image_eqI*) (*auto simp add*: *field_simps*)
qed
show *?thesis*
unfolding *circlepath path_image_def sphere_def dist_norm*
by (*force simp*: *assms algebra_simps norm_mult norm_minus_commute intro*: ∗)
qed

lemma *path_image_circlepath* [*simp*]:
  path_image (*circlepath z r*) = *sphere z* |*r*|
using *path_image_circlepath_minus*
by (*force simp*: *path_image_circlepath_nonneg abs_if*)

lemma *has_contour_integral_bound_circlepath_strong*:

⟦(*f has_contour_integral i*) (*circlepath z r*);
finite k; *0* ≤ *B*; *0* < *r*;
⋀*x*. ⟦*norm*(*x* − *z*) = *r*; *x* ∉ *k*⟧ ⟹ *norm*(*f x*) ≤ *B*⟧
⟹ *norm i* ≤ *B*∗(*2*∗*pi*∗*r*)
unfolding *circlepath_def*
by (*auto simp*: *algebra_simps in_path_image_part_circlepath dest*!: *has_contour_integral_bound_part_circ*

lemma *has_contour_integral_bound_circlepath*:
⟦(*f has_contour_integral i*) (*circlepath z r*);
0 ≤ *B*; *0* < *r*; ⋀*x*. *norm*(*x* − *z*) = *r* ⟹ *norm*(*f x*) ≤ *B*⟧
⟹ *norm i* ≤ *B*∗(*2*∗*pi*∗*r*)
by (*auto intro*: *has_contour_integral_bound_circlepath_strong*)

lemma *contour_integrable_continuous_circlepath*:
continuous_on (*path_image* (*circlepath z r*)) *f*
⟹ *f contour_integrable_on* (*circlepath z r*)
by (*simp add*: *circlepath_def contour_integrable_continuous_part_circlepath*)

lemma *simple_path_circlepath*: *simple_path*(*circlepath z r*) ⟷ (*r* ≠ *0*)
by (*simp add*: *circlepath_def simple_path_part_circlepath*)

lemma *notin_path_image_circlepath* [*simp*]: *cmod* (*w* − *z*) < *r* ⟹ *w* ∉ *path_image* (*circlepath z r*)
by (*simp add*: *sphere_def dist_norm norm_minus_commute*)

lemma *contour_integral_circlepath*:
assumes *r* > *0*
shows *contour_integral* (*circlepath z r*) (λ*w*. *1* / (*w* − *z*)) = *2* ∗ *complex_of_real pi* ∗ i
proof (*rule contour_integral_unique*)
show ((λ*w*. *1* / (*w* − *z*)) *has_contour_integral 2* ∗ *complex_of_real pi* ∗ i) (*circlepath z r*)
unfolding *has_contour_integral_def* **using** *assms has_integral_const_real* [*of* _ *0 1*]
apply (*subst has_integral_cong*)
apply (*simp add*: *vector_derivative_circlepath01*)
apply (*force simp*: *circlepath*)
done
qed

1.15 Uniform convergence of path integral

Uniform convergence when the derivative of the path is bounded, and in particular for the special case of a circle.

proposition *contour_integral_uniform_limit*:
assumes *ev_fint*: *eventually* (λ*n*::′*a*. (*f n*) *contour_integrable_on* γ) *F*
and *ul_f*: *uniform_limit* (*path_image* γ) *f l F*
and *noleB*: ⋀*t*. *t* ∈ {*0*..*1*} ⟹ *norm* (*vector_derivative* γ (*at t*)) ≤ *B*
and γ: *valid_path* γ

```
    and [simp]: ¬ trivial_limit F
  shows l contour_integrable_on γ ((λn. contour_integral γ (f n)) ⟶ con-
tour_integral γ l) F
proof −
  have 0 ≤ B by (meson noleB [of 0] atLeastAtMost_iff norm_ge_zero order_refl
order_trans zero_le_one)
  { fix e::real
    assume 0 < e
    then have 0 < e / (|B| + 1) by simp
    then have ∀F n in F. ∀x∈path_image γ. cmod (f n x − l x) < e / (|B| + 1)
      using ul_f [unfolded uniform_limit_iff dist_norm] by auto
    with ev_fint
    obtain a where fga: ⋀x. x ∈ {0..1} ⟹ cmod (f a (γ x) − l (γ x)) < e /
(|B| + 1)
                and inta: (λt. f a (γ t) ∗ vector_derivative γ (at t)) integrable_on
{0..1}
      using eventually_happens [OF eventually_conj]
      by (fastforce simp: contour_integrable_on path_image_def)
    have Ble: B ∗ e / (|B| + 1) ≤ e
      using ‹0 ≤ B› ‹0 < e› by (simp add: field_split_simps)
    have ∃h. (∀x∈{0..1}. cmod (l (γ x) ∗ vector_derivative γ (at x) − h x) ≤ e)
∧ h integrable_on {0..1}
    proof (intro exI conjI ballI)
      show cmod (l (γ x) ∗ vector_derivative γ (at x) − f a (γ x) ∗ vector_derivative
γ (at x)) ≤ e
        if x ∈ {0..1} for x
        apply (rule order_trans [OF _ Ble])
        using noleB [OF that] fga [OF that] ‹0 ≤ B› ‹0 < e›
      apply (fastforce simp: mult_ac dest: mult_mono [OF less_imp_le] simp add:
norm_mult left_diff_distrib [symmetric] norm_minus_commute divide_simps)
        done
    qed (rule inta)
  }
  then show lintg: l contour_integrable_on γ
  unfolding contour_integrable_on by (metis (mono_tags, lifting)integrable_uniform_limit_real)
  { fix e::real
    define B′ where B′ = B + 1
    have B′: B′ > 0 B′ > B using ‹0 ≤ B› by (auto simp: B′_def)
    assume 0 < e
    then have ev_no′: ∀F n in F. ∀x∈path_image γ. 2 ∗ cmod (f n x − l x) < e
/ B′
      using ul_f [unfolded uniform_limit_iff dist_norm, rule_format, of e / B′/2]
B′
      by (simp add: field_simps)
    have ie: integral {0..1::real} (λx. e/2) < e using ‹0 < e› by simp
    have ∗: cmod (f x (γ t) ∗ vector_derivative γ (at t) − l (γ t) ∗ vector_derivative
γ (at t)) ≤ e/2
      if t: t∈{0..1} and leB′: 2 ∗ cmod (f x (γ t) − l (γ t)) < e / B′ for x t
    proof −
```

```
    have 2 * cmod (f x (γ t) - l (γ t)) * cmod (vector_derivative γ (at t)) ≤ e
* (B/ B')
      using mult_mono [OF less_imp_le [OF leB'] noleB] B' ‹0 < e› t by auto
    also have ... < e
      by (simp add: B' ‹0 < e› mult_imp_div_pos_less)
    finally have 2 * cmod (f x (γ t) - l (γ t)) * cmod (vector_derivative γ (at
t)) < e .
    then show ?thesis
      by (simp add: left_diff_distrib [symmetric] norm_mult)
  qed
   have le_e: ⋀x. ⟦∀u∈{0..1}. 2 * cmod (f x (γ u) - l (γ u)) < e / B'; f x
contour_integrable_on γ⟧
        ⟹ cmod (integral {0..1}
              (λu. f x (γ u) * vector_derivative γ (at u) - l (γ u) * vector_derivative
γ (at u))) < e
    apply (rule le_less_trans [OF integral_norm_bound_integral ie])
     apply (simp add: lintg integrable_diff contour_integrable_on [symmetric])
    apply (blast intro: *)+
    done
  have ∀F x in F. dist (contour_integral γ (f x)) (contour_integral γ l) < e
    apply (rule eventually_mono [OF eventually_conj [OF ev_no' ev_fint]])
     apply (simp add: dist_norm contour_integrable_on path_image_def con-
tour_integral_integral)
      apply (simp add: lintg integral_diff [symmetric] contour_integrable_on
[symmetric] le_e)
    done
 }
 then show ((λn. contour_integral γ (f n)) ⟶ contour_integral γ l) F
   by (rule tendstoI)
qed

corollary contour_integral_uniform_limit_circlepath:
 assumes ∀F n::'a in F. (f n) contour_integrable_on (circlepath z r)
     and uniform_limit (sphere z r) f l F
     and ¬ trivial_limit F 0 < r
   shows l contour_integrable_on (circlepath z r)
           ((λn. contour_integral (circlepath z r) (f n)) ⟶ contour_integral
(circlepath z r) l) F
 using assms by (auto simp: vector_derivative_circlepath norm_mult intro!: con-
tour_integral_uniform_limit)

end
```

2 Complex Path Integrals and Cauchy's Integral Theorem

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Cauchy_Integral_Theorem*

imports
HOL−Analysis.Analysis
Contour_Integration
begin

lemma *leibniz_rule_holomorphic*:
fixes *f*::*complex* ⇒ *'b*::*euclidean_space* ⇒ *complex*
assumes ⋀*x t*. *x* ∈ *U* ⟹ *t* ∈ *cbox a b* ⟹ ((λ*x*. *f x t*) *has_field_derivative fx x t*) (*at x within U*)
assumes ⋀*x*. *x* ∈ *U* ⟹ (*f x*) *integrable_on cbox a b*
assumes *continuous_on* (*U* × (*cbox a b*)) (λ(*x*, *t*). *fx x t*)
assumes *convex U*
shows (λ*x*. *integral* (*cbox a b*) (*f x*)) *holomorphic_on U*
using *leibniz_rule_field_differentiable*[*OF assms*(*1*−*3*) _ *assms*(*4*)]
by (*auto simp*: *holomorphic_on_def*)

lemma *Ln_measurable* [*measurable*]: *Ln* ∈ *measurable borel borel*
proof −
have ∗: *Ln* (−*of_real x*) = *of_real* (*ln x*) + i ∗ *pi* **if** *x* > *0* **for** *x*
using *that* **by** (*subst Ln_minus*) (*auto simp*: *Ln_of_real*)
have ∗∗: *Ln* (*of_real x*) = *of_real* (*ln* (−*x*)) + i ∗ *pi* **if** *x* < *0* **for** *x*
using ∗[*of* −*x*] *that* **by** *simp*
have *cont*: (λ*x*. *indicat_real* (− ℝ≤0) *x* ∗*R Ln x*) ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_on_indicator continuous_intros*) *auto*
have (λ*x*. *if x* ∈ ℝ≤0 *then ln* (−*Re x*) + i ∗ *pi else indicator* (−ℝ≤0) *x* ∗*R Ln x*) ∈ *borel* →*M borel*
(**is** *?f* ∈ _) **by** (*rule measurable_If_set*[*OF* _ *cont*]) *auto*
hence (λ*x*. *if x* = *0 then Ln 0 else ?f x*) ∈ *borel* →*M borel* **by** *measurable*
also have (λ*x*. *if x* = *0 then Ln 0 else ?f x*) = *Ln*
by (*auto simp*: *fun_eq_iff* ∗∗ *nonpos_Reals_def*)
finally show *?thesis* **.**
qed

lemma *powr_complex_measurable* [*measurable*]:
assumes [*measurable*]: *f* ∈ *measurable M borel g* ∈ *measurable M borel*
shows (λ*x*. *f x powr g x* :: *complex*) ∈ *measurable M borel*
using *assms* **by** (*simp add*: *powr_def*)

The special case of midpoints used in the main quadrisection

lemma *has_contour_integral_midpoint*:
assumes (*f has_contour_integral i*) (*linepath a* (*midpoint a b*))
(*f has_contour_integral j*) (*linepath* (*midpoint a b*) *b*)
shows (*f has_contour_integral* (*i* + *j*)) (*linepath a b*)
proof (*rule has_contour_integral_split*)
show *midpoint a b* − *a* = (*1*/*2*) ∗*R* (*b* − *a*)
using *assms* **by** (*auto simp*: *midpoint_def scaleR_conv_of_real*)
qed (*use assms* **in** *auto*)

lemma *contour_integral_midpoint*:

assumes *continuous_on* (*closed_segment a b*) *f*
shows *contour_integral* (*linepath a b*) *f* =
contour_integral (*linepath a* (*midpoint a b*)) *f* + *contour_integral* (*linepath* (*midpoint a b*) *b*) *f*
proof (*rule contour_integral_split*)
show *midpoint a b* − *a* = (*1*/*2*) ∗$_R$ (*b* − *a*)
using *assms* **by** (*auto simp*: *midpoint_def scaleR_conv_of_real*)
qed (*use assms* **in** *auto*)

A couple of special case lemmas that are useful below

lemma *triangle_linear_has_chain_integral*:
((λ*x*. *m*∗*x* + *d*) *has_contour_integral 0*) (*linepath a b* +++ *linepath b c* +++ *linepath c a*)
proof (*rule Cauchy_theorem_primitive*)
show ⋀*x*. *x* ∈ *UNIV* ⟹ ((λ*x*. *m* / *2* ∗ *x*^{2} + *d* ∗ *x*) *has_field_derivative m* ∗ *x* + *d*) (*at x*)
by (*auto intro*!: *derivative_eq_intros*)
qed *auto*

lemma *has_chain_integral_chain_integral3*:
assumes (*f has_contour_integral i*) (*linepath a b* +++ *linepath b c* +++ *linepath c d*)
(**is** (*f has_contour_integral i*) *?g*)
shows *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* + *contour_integral* (*linepath c d*) *f* = *i*
(**is** *?lhs* = _)
proof −
have *f contour_integrable_on ?g*
using *assms contour_integrable_on_def* **by** *blast*
then have *?lhs* = *contour_integral ?g f*
by (*simp add*: *valid_path_join has_contour_integral_integrable*)
then show *?thesis*
using *assms contour_integral_unique* **by** *blast*
qed

lemma *has_chain_integral_chain_integral4*:
assumes (*f has_contour_integral i*) (*linepath a b* +++ *linepath b c* +++ *linepath c d* +++ *linepath d e*)
(**is** (*f has_contour_integral i*) *?g*)
shows *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* + *contour_integral* (*linepath c d*) *f* + *contour_integral* (*linepath d e*) *f* = *i*
(**is** *?lhs* = _)
proof −
have *f contour_integrable_on ?g*
using *assms contour_integrable_on_def* **by** *blast*
then have *?lhs* = *contour_integral ?g f*
by (*simp add*: *valid_path_join has_contour_integral_integrable*)
then show *?thesis*
using *assms contour_integral_unique* **by** *blast*

qed

2.1 The key quadrisection step

lemma *norm_sum_half*:
assumes *norm*(*a* + *b*) ≥ *e*
shows *norm a* ≥ *e*/2 ∨ *norm b* ≥ *e*/2
proof −
have *e* ≤ *norm* (− *a* − *b*)
by (*simp add*: *add.commute assms norm_minus_commute*)
thus *?thesis*
using *norm_triangle_ineq4 order_trans* **by** *fastforce*
qed

lemma *norm_sum_lemma*:
assumes *e* ≤ *norm* (*a* + *b* + *c* + *d*)
shows *e* / 4 ≤ *norm a* ∨ *e* / 4 ≤ *norm b* ∨ *e* / 4 ≤ *norm c* ∨ *e* / 4 ≤ *norm d*
proof −
have *e* ≤ *norm* ((*a* + *b*) + (*c* + *d*)) **using** *assms*
by (*simp add*: *algebra_simps*)
then show *?thesis*
by (*auto dest*!: *norm_sum_half*)
qed

lemma *Cauchy_theorem_quadrisection*:
assumes *f*: *continuous_on* (*convex hull* {*a*,*b*,*c*}) *f*
and *dist*: *dist a b* ≤ *K dist b c* ≤ *K dist c a* ≤ *K*
and *e*: *e* ∗ *K*^2 ≤
norm (*contour_integral*(*linepath a b*) *f* + *contour_integral*(*linepath b c*) *f* + *contour_integral*(*linepath c a*) *f*)
shows ∃ *a*′ *b*′ *c*′.
a′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *b*′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *c*′ ∈ *convex hull* {*a*,*b*,*c*} ∧
dist a′ *b*′ ≤ *K*/2 ∧ *dist b*′ *c*′ ≤ *K*/2 ∧ *dist c*′ *a*′ ≤ *K*/2 ∧
e ∗ (*K*/2)^2 ≤ *norm*(*contour_integral*(*linepath a*′ *b*′) *f* + *contour_integral*(*linepath b*′ *c*′) *f* + *contour_integral*(*linepath c*′ *a*′) *f*)
(**is** ∃ *x y z*. *?Φ x y z*)
proof −
note *divide_le_eq_numeral1* [*simp del*]
define *a*′ **where** *a*′ = *midpoint b c*
define *b*′ **where** *b*′ = *midpoint c a*
define *c*′ **where** *c*′ = *midpoint a b*
have *fabc*: *continuous_on* (*closed_segment a b*) *f continuous_on* (*closed_segment b c*) *f continuous_on* (*closed_segment c a*) *f*
using *f continuous_on_subset segments_subset_convex_hull* **by** *metis*+
have *fcont*′: *continuous_on* (*closed_segment c*′ *b*′) *f*
continuous_on (*closed_segment a*′ *c*′) *f*
continuous_on (*closed_segment b*′ *a*′) *f*
unfolding *a*′_*def b*′_*def c*′_*def*

by (*rule continuous_on_subset* [*OF f*],
metis midpoints_in_convex_hull convex_hull_subset hull_subset insert_subset segment_convex_hull)+
define *pathint* **where** *pathint x y* ≡ *contour_integral*(*linepath x y*) *f* **for** *x y*
have ∗: *pathint a b* + *pathint b c* + *pathint c a* =
(*pathint a c'* + *pathint c' b'* + *pathint b' a*) +
(*pathint a' c'* + *pathint c' b* + *pathint b a'*) +
(*pathint a' c* + *pathint c b'* + *pathint b' a'*) +
(*pathint a' b'* + *pathint b' c'* + *pathint c' a'*)
unfolding *pathint_def*
by (*simp add*: *fcont'* *contour_integral_reverse_linepath*) (*simp add*: *a'_def b'_def c'_def contour_integral_midpoint fabc*)
have [*simp*]: ⋀*x y*. *cmod* (*x* ∗ *2* − *y* ∗ *2*) = *cmod* (*x* − *y*) ∗ *2*
by (*metis left_diff_distrib mult.commute norm_mult_numeral1*)
have [*simp*]: ⋀*x y*. *cmod* (*x* − *y*) = *cmod* (*y* − *x*)
by (*simp add*: *norm_minus_commute*)
consider *e* ∗ K^2 / *4* ≤ *cmod* (*pathint a c'* + *pathint c' b'* + *pathint b' a*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' c'* + *pathint c' b* + *pathint b a'*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' c* + *pathint c b'* + *pathint b' a'*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' b'* + *pathint b' c'* + *pathint c' a'*)
using *assms* **by** (*metis* ∗ *norm_sum_lemma pathint_def*)
then show *?thesis*
proof *cases*
case *1* **then have** *?*Φ *a c' b'*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *c'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *2* **then have** *?*Φ *a' c' b*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *a'_def c'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *3* **then have** *?*Φ *a' c b'*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *a'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *4* **then have** *?*Φ *a' b' c'*
using *assms* **unfolding** *pathint_def* [*symmetric*]

apply (*clarsimp simp*: *a'_def c'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
qed
qed

2.2 Cauchy's theorem for triangles

lemma *triangle_points_closer*:
fixes *a*::*complex*
shows ⟦*x* ∈ *convex hull* {*a*,*b*,*c*}; *y* ∈ *convex hull* {*a*,*b*,*c*}⟧
⟹ *norm*(*x* − *y*) ≤ *norm*(*a* − *b*) ∨
norm(*x* − *y*) ≤ *norm*(*b* − *c*) ∨
norm(*x* − *y*) ≤ *norm*(*c* − *a*)
using *simplex_extremal_le* [*of* {*a*,*b*,*c*}]
by (*auto simp*: *norm_minus_commute*)

lemma *holomorphic_point_small_triangle*:
assumes *x*: *x* ∈ *S*
and *f*: *continuous_on S f*
and *cd*: *f field_differentiable* (*at x within S*)
and *e*: *0* < *e*
shows ∃*k*>*0*. ∀ *a b c*. *dist a b* ≤ *k* ∧ *dist b c* ≤ *k* ∧ *dist c a* ≤ *k* ∧
x ∈ *convex hull* {*a*,*b*,*c*} ∧ *convex hull* {*a*,*b*,*c*} ⊆ *S*
⟶ *norm*(*contour_integral*(*linepath a b*) *f* + *contour_integral*(*linepath b c*) *f* +
contour_integral(*linepath c a*) *f*)
≤ *e*∗(*dist a b* + *dist b c* + *dist c a*)^2
(**is** ∃*k*>*0*. ∀ *a b c*. _ ⟶ *?normle a b c*)
proof −
have *le_of_3*: ⋀*a x y z*. ⟦*0* ≤ *x*∗*y*; *0* ≤ *x*∗*z*; *0* ≤ *y*∗*z*; *a* ≤ (*e*∗(*x* + *y* + *z*))∗*x* + (*e*∗(*x* + *y* + *z*))∗*y* + (*e*∗(*x* + *y* + *z*))∗*z*⟧
⟹ *a* ≤ *e*∗(*x* + *y* + *z*)^2
by (*simp add*: *algebra_simps power2_eq_square*)
have *disj_le*: ⟦*x* ≤ *a* ∨ *x* ≤ *b* ∨ *x* ≤ *c*; *0* ≤ *a*; *0* ≤ *b*; *0* ≤ *c*⟧ ⟹ *x* ≤ *a* + *b* + *c*
for *x*::*real* **and** *a b c*
by *linarith*
have *fabc*: *f contour_integrable_on linepath a b f contour_integrable_on linepath b c f contour_integrable_on linepath c a*
if *convex hull* {*a*, *b*, *c*} ⊆ *S* **for** *a b c*
using *segments_subset_convex_hull that*
by (*metis continuous_on_subset f contour_integrable_continuous_linepath*)+
note *path_bound* = *has_contour_integral_bound_linepath* [*simplified norm_minus_commute*, *OF has_contour_integral_integral*]
{ fix *f' a b c d*
assume *d*: *0* < *d*

and f': $\bigwedge y.\ [\![cmod\ (y - x) \leq d;\ y \in S]\!] \Longrightarrow cmod\ (f\ y - f\ x - f' * (y - x)) \leq e * cmod\ (y - x)$
and *le*: $cmod\ (a - b) \leq d\ cmod\ (b - c) \leq d\ cmod\ (c - a) \leq d$
and *xc*: $x \in convex\ hull\ \{a, b, c\}$
and *S*: $convex\ hull\ \{a, b, c\} \subseteq S$
have *pa*: $contour\_integral\ (linepath\ a\ b)\ f + contour\_integral\ (linepath\ b\ c)\ f + contour\_integral\ (linepath\ c\ a)\ f =$
$contour\_integral\ (linepath\ a\ b)\ (\lambda y.\ f\ y - f\ x - f' * (y - x)) +$
$contour\_integral\ (linepath\ b\ c)\ (\lambda y.\ f\ y - f\ x - f' * (y - x)) +$
$contour\_integral\ (linepath\ c\ a)\ (\lambda y.\ f\ y - f\ x - f' * (y - x))$
apply (*simp add*: *contour_integral_diff contour_integral_lmul contour_integrable_lmul contour_integrable_diff fabc* [*OF S*])
apply (*simp add*: *field_simps*)
done
{ fix *y*
assume *yc*: $y \in convex\ hull\ \{a,b,c\}$
have $cmod\ (f\ y - f\ x - f' * (y - x)) \leq e * norm(y - x)$
proof (*rule f'*)
show $cmod\ (y - x) \leq d$
by (*metis triangle_points_closer* [*OF xc yc*] *le norm_minus_commute order_trans*)
qed (*use S yc* **in** *blast*)
also have $\ldots \leq e * (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$
by (*simp add*: *yc e xc disj_le* [*OF triangle_points_closer*])
finally have $cmod\ (f\ y - f\ x - f' * (y - x)) \leq e * (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$ **.**
} note *cm_le* = *this*
have *?normle a b c*
unfolding *dist_norm pa*
using *f' xc S e*
apply (*intro le_of_3 norm_triangle_le add_mono path_bound*)
apply (*simp_all add*: *contour_integral_diff contour_integral_lmul contour_integrable_lmul contour_integrable_diff fabc*)
apply (*blast intro*: *cm_le elim*: *dest*: *segments_subset_convex_hull* [*THEN subsetD*])+
done
} note $*$ = *this*
show *?thesis*
using *cd e*
apply (*simp add*: *field_differentiable_def has_field_derivative_def has_derivative_within_alt approachable_lt_le2 Ball_def*)
apply (*clarify dest*!: *spec mp*)
using $*$ **unfolding** *dist_norm*
apply *blast*
done
qed

Hence the most basic theorem for a triangle.

locale *Chain* =

```
  fixes x0 At Follows
  assumes At0: At x0 0
      and AtSuc: ⋀x n. At x n ⟹ ∃x'. At x' (Suc n) ∧ Follows x' x
begin
  primrec f where
    f 0 = x0
  | f (Suc n) = (SOME x. At x (Suc n) ∧ Follows x (f n))

  lemma At: At (f n) n
  proof (induct n)
    case 0 show ?case
      by (simp add: At0)
  next
    case (Suc n) show ?case
      by (metis (no_types, lifting) AtSuc [OF Suc] f.simps(2) someI_ex)
  qed

  lemma Follows: Follows (f(Suc n)) (f n)
    by (metis (no_types, lifting) AtSuc [OF At [of n]] f.simps(2) someI_ex)

  declare f.simps(2) [simp del]
end

lemma Chain3:
  assumes At0: At x0 y0 z0 0
      and AtSuc: ⋀x y z n. At x y z n ⟹ ∃x' y' z'. At x' y' z' (Suc n) ∧ Follows
x' y' z' x y z
  obtains f g h where
    f 0 = x0 g 0 = y0 h 0 = z0
                    ⋀n. At (f n) (g n) (h n) n
                     ⋀n. Follows (f(Suc n)) (g(Suc n)) (h(Suc n)) (f n) (g n) (h n)
proof −
  interpret three: Chain (x0,y0,z0) λ(x,y,z). At x y z λ(x',y',z'). λ(x,y,z). Follows
x' y' z' x y z
  proof qed (use At0 AtSuc in auto)
  show ?thesis
  proof
    show ⋀n. Follows (fst (three.f (Suc n))) (fst (snd (three.f (Suc n))))
                     (snd (snd (three.f (Suc n)))) (fst (three.f n))
                    (fst (snd (three.f n))) (snd (snd (three.f n)))
        ⋀n. At (fst (three.f n)) (fst (snd (three.f n))) (snd (snd (three.f n))) n
      using three.At three.Follows
      by (simp_all add: split_beta')
  qed auto
qed


proposition Cauchy_theorem_triangle:
  assumes f holomorphic_on (convex hull {a,b,c})
```

shows (*f has_contour_integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
proof −
have *contf*: *continuous_on* (*convex hull* {*a*,*b*,*c*}) *f*
by (*metis assms holomorphic_on_imp_continuous_on*)
let *?pathint* = λ*x y*. *contour_integral*(*linepath x y*) *f*
{ fix *y*::*complex*
assume *fy*: (*f has_contour_integral y*) (*linepath a b +++ linepath b c +++ linepath c a*)
and *ynz*: *y* ≠ *0*
define *K* **where** *K* = *1* + *max* (*dist a b*) (*max* (*dist b c*) (*dist c a*))
define *e* **where** *e* = *norm y* / *K*^*2*
have *K1*: *K* ≥ *1* **by** (*simp add*: *K_def max.coboundedI1*)
then have *K*: *K* > *0* **by** *linarith*
have [*iff*]: *dist a b* ≤ *K dist b c* ≤ *K dist c a* ≤ *K*
by (*simp_all add*: *K_def*)
have *e*: *e* > *0*
unfolding *e_def* **using** *ynz K1* **by** *simp*
define *At* **where** *At x y z n* ⟷
convex hull {*x*,*y*,*z*} ⊆ *convex hull* {*a*,*b*,*c*} ∧
dist x y ≤ *K*/*2*^*n* ∧ *dist y z* ≤ *K*/*2*^*n* ∧ *dist z x* ≤ *K*/*2*^*n* ∧
norm(*?pathint x y* + *?pathint y z* + *?pathint z x*) ≥ *e*∗(*K*/*2*^*n*)^*2*
for *x y z n*
have *At0*: *At a b c 0*
using *fy*
by (*simp add*: *At_def e_def has_chain_integral_chain_integral3*)
{ fix *x y z n*
assume *At*: *At x y z n*
then have *contf′*: *continuous_on* (*convex hull* {*x*,*y*,*z*}) *f*
using *contf At_def continuous_on_subset* **by** *metis*
have ∃*x′ y′ z′*. *At x′ y′ z′* (*Suc n*) ∧ *convex hull* {*x′*,*y′*,*z′*} ⊆ *convex hull* {*x*,*y*,*z*}
using *At Cauchy_theorem_quadrisection* [*OF contf′*, *of K*/*2*^*n e*]
apply (*simp add*: *At_def algebra_simps*)
apply (*meson convex_hull_subset empty_subsetI insert_subset subsetCE*)
done
} note *AtSuc* = *this*
obtain *fa fb fc*
where *f0* [*simp*]: *fa 0* = *a fb 0* = *b fc 0* = *c*
and *cosb*: ⋀*n*. *convex hull* {*fa n*, *fb n*, *fc n*} ⊆ *convex hull* {*a*,*b*,*c*}
and *dist*: ⋀*n*. *dist* (*fa n*) (*fb n*) ≤ *K*/*2*^*n*
⋀*n*. *dist* (*fb n*) (*fc n*) ≤ *K*/*2*^*n*
⋀*n*. *dist* (*fc n*) (*fa n*) ≤ *K*/*2*^*n*
and *no*: ⋀*n*. *norm*(*?pathint* (*fa n*) (*fb n*) +
?pathint (*fb n*) (*fc n*) +
?pathint (*fc n*) (*fa n*)) ≥ *e* ∗ (*K*/*2*^*n*)^*2*
and *conv_le*: ⋀*n*. *convex hull* {*fa*(*Suc n*), *fb*(*Suc n*), *fc*(*Suc n*)} ⊆ *convex hull* {*fa n*, *fb n*, *fc n*}
by (*rule Chain3* [*of At*, *OF At0 AtSuc*]) (*auto simp*: *At_def*)

```
    obtain x where x: ⋀n. x ∈ convex hull {fa n, fb n, fc n}
    proof (rule bounded_closed_nest)
      show ⋀n. closed (convex hull {fa n, fb n, fc n})
        by (simp add: compact_imp_closed finite_imp_compact_convex_hull)
      show ⋀m n. m ≤ n ⟹ convex hull {fa n, fb n, fc n} ⊆ convex hull {fa m,
fb m, fc m}
        by (erule transitive_stepwise_le) (auto simp: conv_le)
    qed (fastforce intro: finite_imp_bounded_convex_hull)+
    then have xin: x ∈ convex hull {a,b,c}
      using assms f0 by blast
    then have fx: f field_differentiable at x within (convex hull {a,b,c})
      using assms holomorphic_on_def by blast
    { fix k n
      assume k: 0 < k
        and le:
          ⋀x' y' z'.
            ⟦dist x' y' ≤ k; dist y' z' ≤ k; dist z' x' ≤ k;
             x ∈ convex hull {x',y',z'};
             convex hull {x',y',z'} ⊆ convex hull {a,b,c}⟧
            ⟹
            cmod (?pathint x' y' + ?pathint y' z' + ?pathint z' x') * 10
                 ≤ e * (dist x' y' + dist y' z' + dist z' x')²
        and Kk: K / k < 2 ^ n
      have K / 2 ^ n < k using Kk k
        by (auto simp: field_simps)
      then have DD: dist (fa n) (fb n) ≤ k dist (fb n) (fc n) ≤ k dist (fc n) (fa n)
≤ k
        using dist [of n]  k
        by linarith+
      have dle: (dist (fa n) (fb n) + dist (fb n) (fc n) + dist (fc n) (fa n))²
            ≤ (3 * K / 2 ^ n)²
        using dist [of n] e K
        by (simp add: abs_le_square_iff [symmetric])
      have less10: ⋀x y::real. 0 < x ⟹ y ≤ 9*x ⟹ y < x*10
        by linarith
      have e * (dist (fa n) (fb n) + dist (fb n) (fc n) + dist (fc n) (fa n))² ≤ e *
(3 * K / 2 ^ n)²
        using ynz dle e mult_le_cancel_left_pos by blast
      also have ... <
          cmod (?pathint (fa n) (fb n) + ?pathint (fb n) (fc n) + ?pathint (fc n) (fa
n)) * 10
        using no [of n] e K
      by (simp add: e_def field_simps) (simp only: zero_less_norm_iff [symmetric])
      finally have False
        using le [OF DD x cosb] by auto
    } then
    have ?thesis
      using holomorphic_point_small_triangle [OF xin contf fx, of e/10] e
      apply clarsimp
```

apply (*rule_tac y1=K/k* **in** *exE* [*OF real_arch_pow*[*of 2*]], *force+*)
done
}
moreover have *f contour_integrable_on* (*linepath a b +++ linepath b c +++ linepath c a*)
by *simp* (*meson contf continuous_on_subset contour_integrable_continuous_linepath segments_subset_convex_hull*(*1*) *segments_subset_convex_hull*(*3*) *segments_subset_convex_hull*(*5*))
ultimately show *?thesis*
using *has_contour_integral_integral* **by** *fastforce*
qed

2.3 Version needing function holomorphic in interior only

lemma *Cauchy_theorem_flat_lemma*:
assumes *f*: *continuous_on* (*convex hull* {*a*,*b*,*c*}) *f*
and *c*: $c - a = k *_R (b - a)$
and *k*: $0 \le k$
shows *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* + *contour_integral* (*linepath c a*) *f* = *0*
proof −
have *fabc*: *continuous_on* (*closed_segment a b*) *f continuous_on* (*closed_segment b c*) *f continuous_on* (*closed_segment c a*) *f*
using *f continuous_on_subset segments_subset_convex_hull* **by** *metis+*
show *?thesis*
proof (*cases* $k \le 1$)
case *True* **show** *?thesis*
by (*simp add*: *contour_integral_split* [*OF fabc*(*1*) *k True c*] *contour_integral_reverse_linepath fabc*)
next
case *False*
show *?thesis*
proof (*subst contour_integral_split* [*symmetric*])
show $b - a = (1/k) *_R (c - a)$
using *False c* **by** *force*
show *contour_integral* (*linepath a c*) *f* + *contour_integral* (*linepath c a*) *f* = *0*
by (*simp add*: *contour_integral_reverse_linepath fabc*(*3*))
show *continuous_on* (*closed_segment a c*) *f*
by (*metis closed_segment_commute fabc*(*3*))
qed (*use False* **in** *auto*)
qed
qed

lemma *Cauchy_theorem_flat*:
assumes *f*: *continuous_on* (*convex hull* {*a*,*b*,*c*}) *f*
and *c*: $c - a = k *_R (b - a)$
shows *contour_integral* (*linepath a b*) *f* +
contour_integral (*linepath b c*) *f* +

```
          contour_integral (linepath c a) f = 0
proof (cases 0 ≤ k)
  case True with assms show ?thesis
    by (blast intro: Cauchy_theorem_flat_lemma)
next
  case False
  have continuous_on (closed_segment a b) f continuous_on (closed_segment b
c) f continuous_on (closed_segment c a) f
    using f continuous_on_subset segments_subset_convex_hull by metis+
  moreover have contour_integral (linepath b a) f + contour_integral (linepath
a c) f +
                contour_integral (linepath c b) f = 0
  proof (rule Cauchy_theorem_flat_lemma [of b a c f 1−k])
    show continuous_on (convex hull {b, a, c}) f
      by (simp add: f insert_commute)
    show c − b = (1 − k) *R (a − b)
      using c by (auto simp: algebra_simps)
  qed (use False in auto)
  ultimately show ?thesis
   by (metis (no_types, lifting) contour_integral_reverse_linepath eq_neg_iff_add_eq_0
minus_add_cancel)
qed


proposition Cauchy_theorem_triangle_interior:
  assumes contf: continuous_on (convex hull {a,b,c}) f
      and holf:  f holomorphic_on interior (convex hull {a,b,c})
    shows (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
proof −
  define pathint where pathint ≡ λx y. contour_integral(linepath x y) f
 have fabc: continuous_on (closed_segment a b) f continuous_on (closed_segment
b c) f continuous_on (closed_segment c a) f
    using contf continuous_on_subset segments_subset_convex_hull by metis+
  have bounded (f ‘ (convex hull {a,b,c}))
   by (simp add: compact_continuous_image compact_convex_hull compact_imp_bounded
contf)
  then obtain B where 0 < B and Bnf: ⋀x. x ∈ convex hull {a,b,c} ⟹ norm
(f x) ≤ B
     by (auto simp: dest!: bounded_pos [THEN iffD1])
  have bounded (convex hull {a,b,c})
    by (simp add: bounded_convex_hull)
  then obtain C where C: 0 < C and Cno: ⋀y. y ∈ convex hull {a,b,c} ⟹
norm y < C
    using bounded_pos_less by blast
  then have diff_2C: norm(x − y) ≤ 2∗C
        if x: x ∈ convex hull {a, b, c} and y: y ∈ convex hull {a, b, c} for x y
  proof −
    have cmod x ≤ C
```

```
    using x by (meson Cno not_le not_less_iff_gr_or_eq)
  hence cmod (x - y) ≤ C + C
    using y by (meson Cno add_mono_thms_linordered_field(4) less_eq_real_def
norm_triangle_ineq4 order_trans)
  thus cmod (x - y) ≤ 2 * C
    by (metis mult_2)
 qed
 have contf': continuous_on (convex hull {b,a,c}) f
  using contf by (simp add: insert_commute)
 { fix y::complex
  assume fy: (f has_contour_integral y) (linepath a b +++ linepath b c +++
linepath c a)
     and ynz: y ≠ 0
  have pi_eq_y: pathint a b + pathint b c + pathint c a= y
     unfolding pathint_def by (rule has_chain_integral_chain_integral3 [OF
fy])
  have ?thesis
  proof (cases c=a ∨ a=b ∨ b=c)
    case True then show ?thesis
      using Cauchy_theorem_flat [OF contf, of 0]
      using has_chain_integral_chain_integral3 [OF fy] ynz
      by (force simp: fabc contour_integral_reverse_linepath)
  next
    case False
    then have car3: card {a, b, c} = Suc (DIM(complex))
      by auto
    { assume interior(convex hull {a,b,c}) = {}
      then have collinear{a,b,c}
        using interior_convex_hull_eq_empty [OF car3]
        by (simp add: collinear_3_eq_affine_dependent)
      with False obtain d where c ≠ a a ≠ b b ≠ c c - b = d *R (a - b)
        by (auto simp: collinear_3 collinear_lemma)
      then have False
        using False Cauchy_theorem_flat [OF contf'] pi_eq_y ynz
          by (simp add: fabc add_eq_0_iff contour_integral_reverse_linepath
pathint_def)
    }
    then obtain d where d: d ∈ interior (convex hull {a, b, c})
      by blast
    { fix d1
      assume d1_pos: 0 < d1
        and d1: ⋀x x'. ⟦x∈convex hull {a, b, c}; x'∈convex hull {a, b, c}; cmod
(x' - x) < d1⟧
                    ⟹ cmod (f x' - f x) < cmod y / (24 * C)
      define e where e = min 1 (min (d1/(4*C)) ((norm y / 24 / C) / B))
      define shrink where shrink x = x - e *R (x - d) for x
      have e: 0 < e e ≤ 1 e ≤ d1 / (4 * C) e ≤ cmod y / 24 / C / B
        using d1_pos ‹C>0› ‹B>0› ynz by (simp_all add: e_def)
      have e_le_d1: e * (4 * C) ≤ d1
```

```
      using e ‹C>0› by (simp add: field_simps)
    have shrink a ∈ interior(convex hull {a,b,c})
         shrink b ∈ interior(convex hull {a,b,c})
         shrink c ∈ interior(convex hull {a,b,c})
   using d e by (auto simp: hull_inc mem_interior_convex_shrink shrink_def)
    then have fhp0: (f has_contour_integral 0)
            (linepath (shrink a) (shrink b) +++ linepath (shrink b) (shrink c)
+++ linepath (shrink c) (shrink a))
     by (simp add: Cauchy_theorem_triangle holomorphic_on_subset [OF holf]
hull_minimal)
     then have f_0_shrink: pathint (shrink a) (shrink b) + pathint (shrink b)
(shrink c) + pathint (shrink c) (shrink a) = 0
      by (simp add: has_chain_integral_chain_integral3 pathint_def)
    have fpi_abc: f contour_integrable_on linepath (shrink a) (shrink b)
                  f contour_integrable_on linepath (shrink b) (shrink c)
                  f contour_integrable_on linepath (shrink c) (shrink a)
   using fhp0 by (auto simp: valid_path_join dest: has_contour_integral_integrable)
    have cmod_shr: ⋀x y. cmod (shrink y − shrink x − (y − x)) = e ∗ cmod
(x − y)
      using e by (simp add: shrink_def real_vector.scale_right_diff_distrib
[symmetric])
    have sh_eq: ⋀a b d::complex. (b − e ∗R (b − d)) − (a − e ∗R (a − d)) −
(b − a) = e ∗R (a − b)
      by (simp add: algebra_simps)
    have cmod y / (24 ∗ C) ≤ cmod y / cmod (b − a) / 12
      using False ‹C>0› diff_2C [of b a] ynz
      by (auto simp: field_split_simps hull_inc)
    have less_C: x ∗ cmod u < C if u ∈ convex hull {a,b,c} 0 ≤ x x ≤ 1 for
x u
    proof (cases x=0)
      case False
      with that show ?thesis
        using Cno [of u] mult_left_le_one_le [of cmod u x] le_less_trans
norm_ge_zero by blast
    qed (simp add: ‹0<C›)
    { fix u v
      assume uv: u ∈ convex hull {a, b, c} v ∈ convex hull {a, b, c} u≠v
        and fpi_uv: f contour_integrable_on linepath (shrink u) (shrink v)
      have shr_uv: shrink u ∈ interior(convex hull {a,b,c})
                   shrink v ∈ interior(convex hull {a,b,c})
        using d e uv
        by (auto simp: hull_inc mem_interior_convex_shrink shrink_def)
        have cmod_fuv: ⋀x. 0≤x ⟹ x≤1 ⟹ cmod (f (linepath (shrink u)
(shrink v) x)) ≤ B
     using shr_uv by (blast intro: Bnf linepath_in_convex_hull interior_subset
[THEN subsetD])
      { fix x::real  assume x: 0≤x x≤1
        have |1 − x| ∗ cmod u < C |x| ∗ cmod v < C
          using uv x by (auto intro!: less_C)
```

```
          moreover have  |x| * cmod d < C |1 - x| * cmod d < C
            using x d interior_subset by (auto intro!: less_C)
          ultimately
          have cmod_less_4C: cmod ((1 - x) *R u - (1 - x) *R d) + cmod (x
*R v - x *R d) < (C+C) + (C+C)
        by (metis add_strict_mono le_less_trans norm_scaleR norm_triangle_ineq4)
          have ll: linepath (shrink u) (shrink v) x - linepath u v x = -e * ((1 -
x) *R (u - d) + x *R (v - d))
        by (simp add: linepath_def shrink_def algebra_simps scaleR_conv_of_real)
         have cmod_less_dt: cmod (linepath (shrink u) (shrink v) x - linepath u
v x) < d1
            unfolding ll norm_mult scaleR_diff_right
                using ‹e>0› cmod_less_4C by (force intro: norm_triangle_lt
less_le_trans [OF _ e_le_d1])
           have cmod (f (linepath (shrink u) (shrink v) x)) * cmod (shrink v -
shrink u - (v - u)) +
                      cmod (v - u) * cmod (f (linepath (shrink u) (shrink v) x) -
f (linepath u v x))
                      ≤ B * (cmod y / 24 / C / B * 2 * C) + 2 * C * (cmod y /
24 / C)
          proof (intro add_mono [OF mult_mono])
            show cmod (f (linepath (shrink u) (shrink v) x)) ≤ B
              using cmod_fuv x by blast
            have B * (12 * (e * cmod (u - v))) ≤ 24 * e * C * B
              using e ‹B>0› diff_2C [of u v] uv by (auto simp: field_simps)
            also have ... ≤ cmod y
              using ‹C>0› ‹B>0› e by (simp add: field_simps)
            finally show cmod (shrink v - shrink u - (v - u)) ≤ cmod y / 24 /
C / B * 2 * C
          using ‹0 < B› ‹0 < C› by (simp add: cmod_shr mult_ac divide_simps)
            have cmod (f (linepath (shrink u) (shrink v) x) - f (linepath u v x))
< cmod y / (24 * C)
              using x uv shr_uv cmod_less_dt
               by (auto simp: hull_inc intro: d1 interior_subset [THEN subsetD]
linepath_in_convex_hull)
            also have ... ≤ cmod y / cmod (v - u) / 12
              using False uv ‹C>0› diff_2C [of v u] ynz
              by (auto simp: field_split_simps hull_inc)
             finally have cmod (f (linepath (shrink u) (shrink v) x) - f (linepath
u v x)) ≤ cmod y / cmod (v - u) / 12
              by simp
            then show cmod (v - u) * cmod (f (linepath (shrink u) (shrink v) x)
- f (linepath u v x))
                      ≤ 2 * C * (cmod y / 24 / C)
              using uv C  by (simp add: field_simps)
          qed (use ‹0 < B› in auto)
          also have ... ≤ cmod y / 6
            by simp
          finally have cmod (f (linepath (shrink u) (shrink v) x)) * cmod (shrink
```

```
v − shrink u − (v − u)) +
                    cmod (v − u) ∗ cmod (f (linepath (shrink u) (shrink v) x) −
f (linepath u v x))
                    ≤ cmod y / 6 .
      } note cmod_diff_le = this
      have f_uv: continuous_on (closed_segment u v) f
     by (blast intro: uv continuous_on_subset [OF contf closed_segment_subset_convex_hull])
      have **: ⋀f' x' f x::complex. f'*x' − f*x = f' * (x' − x) + x * (f' − f)
        by (simp add: algebra_simps)
      have norm (pathint (shrink u) (shrink v) − pathint u v)
          ≤ (B*(norm y /24/C/B)*2*C + (2*C)*(norm y/24/C)) * content
(cbox 0 (1::real))
        apply (rule has_integral_bound
            [of _ λx. f(linepath (shrink u) (shrink v) x) * (shrink v − shrink
u) − f(linepath u v x)*(v − u)
              _ 0 1])
        using ynz ‹0 < B› ‹0 < C›
     apply (simp_all add: pathint_def has_integral_diff has_contour_integral_linepath
[symmetric] has_contour_integral_integral
       fpi_uv f_uv contour_integrable_continuous_linepath del: le_divide_eq_numeral1)
        apply (auto simp: ** norm_triangle_le norm_mult cmod_diff_le simp
del: le_divide_eq_numeral1)
        done
      also have ... ≤ norm y / 6
        by simp
      finally have norm (pathint (shrink u) (shrink v) − pathint u v) ≤ norm
y / 6 .
      } note ∗ = this
      have norm (pathint (shrink a) (shrink b) − pathint a b) ≤ norm y / 6
        using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
      moreover
      have norm (pathint (shrink b) (shrink c) − pathint b c) ≤ norm y / 6
        using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
      moreover
      have norm (pathint (shrink c) (shrink a) − pathint c a) ≤ norm y / 6
        using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
      ultimately
      have norm((pathint (shrink a) (shrink b) − pathint a b) +
            (pathint (shrink b) (shrink c) − pathint b c) + (pathint (shrink c)
(shrink a) − pathint c a))
          ≤ norm y / 6 + norm y / 6 + norm y / 6
        by (metis norm_triangle_le add_mono)
      also have ... = norm y / 2
        by simp
        finally have norm((pathint (shrink a) (shrink b) + pathint (shrink b)
(shrink c) + pathint (shrink c) (shrink a)) −
                    (pathint a b + pathint b c + pathint c a))
          ≤ norm y / 2
        by (simp add: algebra_simps)
```

```
        then
        have norm(pathint a b + pathint b c + pathint c a) ≤ norm y / 2
             by (simp add: f_0_shrink) (metis (mono_tags) add.commute minus_add_distrib norm_minus_cancel uminus_add_conv_diff)
        then have False
          using pi_eq_y ynz by auto
       }
       note * = this
       have uniformly_continuous_on (convex hull {a,b,c}) f
       by (simp add: contf compact_convex_hull compact_uniformly_continuous)
       moreover have norm y / (24 * C) > 0
        using ynz ‹C > 0› by auto
       ultimately obtain δ where δ > 0 and
        ∀x∈convex hull {a, b, c}. ∀x'∈convex hull {a, b, c}.
           dist x' x < δ ⟶ dist (f x') (f x) < cmod y / (24 * C)
        using ‹C > 0› ynz unfolding uniformly_continuous_on_def dist_norm
by blast
       hence False using *[of δ] by (auto simp: dist_norm)
       then show ?thesis ..
     qed
  }
  moreover have f contour_integrable_on (linepath a b +++ linepath b c +++
linepath c a)
    using fabc contour_integrable_continuous_linepath by auto
  ultimately show ?thesis
    using has_contour_integral_integral by fastforce
qed
```

2.4 Version allowing finite number of exceptional points

```
proposition Cauchy_theorem_triangle_cofinite:
  assumes continuous_on (convex hull {a,b,c}) f
      and finite S
      and (⋀x. x ∈ interior(convex hull {a,b,c}) - S ⟹ f field_differentiable (at
x))
    shows (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
using assms
proof (induction card S arbitrary: a b c S rule: less_induct)
  case (less S a b c)
  show ?case
  proof (cases S={})
    case True with less show ?thesis
    by (fastforce simp: holomorphic_on_def field_differentiable_at_within Cauchy_theorem_triangle_interio
  next
    case False
    then obtain d S' where d: S = insert d S' d ∉ S'
      by (meson Set.set_insert all_not_in_conv)
    then show ?thesis
```

```
  proof (cases d ∈ convex hull {a,b,c})
    case False
   show (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
    proof (rule less.hyps)
     show ⋀x. x ∈ interior (convex hull {a, b, c}) − S' ⟹ f field_differentiable
at x
      using False d interior_subset by (auto intro!: less.prems)
  qed (use d less.prems in auto)
  next
    case True
    have ∗: convex hull {a, b, d} ⊆ convex hull {a, b, c}
      by (meson True hull_subset insert_subset convex_hull_subset)
     have abd: (f has_contour_integral 0) (linepath a b +++ linepath b d +++
linepath d a)
    proof (rule less.hyps)
     show ⋀x. x ∈ interior (convex hull {a, b, d}) − S' ⟹ f field_differentiable
at x
       using d not_in_interior_convex_hull_3
         by (clarsimp intro!: less.prems) (metis ∗ insert_absorb insert_subset
interior_mono)
    qed (use d continuous_on_subset [OF _ ∗] less.prems in auto)
    have ∗: convex hull {b, c, d} ⊆ convex hull {a, b, c}
      by (meson True hull_subset insert_subset convex_hull_subset)
     have bcd: (f has_contour_integral 0) (linepath b c +++ linepath c d +++
linepath d b)
    proof (rule less.hyps)
     show ⋀x. x ∈ interior (convex hull {b, c, d}) − S' ⟹ f field_differentiable
at x
       using d not_in_interior_convex_hull_3
         by (clarsimp intro!: less.prems) (metis ∗ insert_absorb insert_subset
interior_mono)
    qed (use d continuous_on_subset [OF _ ∗] less.prems in auto)
    have ∗: convex hull {c, a, d} ⊆ convex hull {a, b, c}
      by (meson True hull_subset insert_subset convex_hull_subset)
     have cad: (f has_contour_integral 0) (linepath c a +++ linepath a d +++
linepath d c)
    proof (rule less.hyps)
     show ⋀x. x ∈ interior (convex hull {c, a, d}) − S' ⟹ f field_differentiable
at x
       using d not_in_interior_convex_hull_3
         by (clarsimp intro!: less.prems) (metis ∗ insert_absorb insert_subset
interior_mono)
    qed (use d continuous_on_subset [OF _ ∗] less.prems in auto)
    have f contour_integrable_on linepath a b
     using less.prems abd contour_integrable_joinD1 contour_integrable_on_def
by blast
    moreover have f contour_integrable_on linepath b c
     using less.prems bcd contour_integrable_joinD1 contour_integrable_on_def
```

by *blast*
moreover have *f contour_integrable_on linepath c a*
using *less.prems cad contour_integrable_joinD1 contour_integrable_on_def*
by *blast*
ultimately have *fpi*: *f contour_integrable_on* (*linepath a b +++ linepath b c +++ linepath c a*)
by *auto*
{ fix *y*::*complex*
assume *fy*: (*f has_contour_integral y*) (*linepath a b +++ linepath b c +++ linepath c a*)
and *ynz*: $y \neq 0$
have *cont_ad*: *continuous_on* (*closed_segment a d*) *f*
by (*meson* ∗ *continuous_on_subset less.prems*(*1*) *segments_subset_convex_hull*(*3*))
have *cont_bd*: *continuous_on* (*closed_segment b d*) *f*
by (*meson True closed_segment_subset_convex_hull continuous_on_subset hull_subset insert_subset less.prems*(*1*))
have *cont_cd*: *continuous_on* (*closed_segment c d*) *f*
by (*meson* ∗ *continuous_on_subset less.prems*(*1*) *segments_subset_convex_hull*(*2*))
have *contour_integral* (*linepath a b*) *f* = − (*contour_integral* (*linepath b d*) *f* + (*contour_integral* (*linepath d a*) *f*))
contour_integral (*linepath b c*) *f* = − (*contour_integral* (*linepath c d*) *f* + (*contour_integral* (*linepath d b*) *f*))
contour_integral (*linepath c a*) *f* = − (*contour_integral* (*linepath a d*) *f* + *contour_integral* (*linepath d c*) *f*)
using *has_chain_integral_chain_integral3* [*OF abd*]
has_chain_integral_chain_integral3 [*OF bcd*]
has_chain_integral_chain_integral3 [*OF cad*]
by (*simp_all add*: *algebra_simps add_eq_0_iff*)
then have *?thesis*
using *cont_ad cont_bd cont_cd fy has_chain_integral_chain_integral3 contour_integral_reverse_linepath* **by** *fastforce*
}
then show *?thesis*
using *fpi contour_integrable_on_def* **by** *blast*
qed
qed
qed

2.5 Cauchy's theorem for an open starlike set

lemma *starlike_convex_subset*:
assumes *S*: $a \in S$ *closed_segment b c* $\subseteq S$ **and** *subs*: $\bigwedge x.\ x \in S \Longrightarrow$ *closed_segment a x* $\subseteq S$
shows *convex hull* {*a*,*b*,*c*} $\subseteq S$
proof −
have *convex hull* {*b*, *c*} $\subseteq S$
using *assms*(*2*) *segment_convex_hull* **by** *auto*
then have $\bigwedge u\ v\ d.\ [\![0 \leq u;\ 0 \leq v;\ u + v = 1;\ d \in$ *convex hull* $\{b,\ c\}]\!] \Longrightarrow u *_R a + v *_R d \in S$

```
    by (meson subs convexD convex_closed_segment ends_in_segment subsetCE)
  then show ?thesis
    by (auto simp add: convex_hull_insert [of {b,c} a])
qed

lemma triangle_contour_integrals_starlike_primitive:
  assumes contf: continuous_on S f
      and S: a ∈ S open S
      and x: x ∈ S
      and subs: ⋀y. y ∈ S ⟹ closed_segment a y ⊆ S
      and zer: ⋀b c. closed_segment b c ⊆ S
                ⟹ contour_integral (linepath a b) f + contour_integral (linepath
b c) f +
                   contour_integral (linepath c a) f = 0
    shows ((λx. contour_integral(linepath a x) f) has_field_derivative f x) (at x)
proof −
  let ?pathint = λx y. contour_integral(linepath x y) f
  { fix e y
    assume e: 0 < e and bxe: ball x e ⊆ S and close: cmod (y − x) < e
    have y: y ∈ S
      using bxe close  by (force simp: dist_norm norm_minus_commute)
    have cont_ayf: continuous_on (closed_segment a y) f
      using contf continuous_on_subset subs y by blast
    have xys: closed_segment x y ⊆ S
    by (metis bxe centre_in_ball close closed_segment_subset convex_ball dist_norm
dual_order.trans e mem_ball norm_minus_commute)
    have ?pathint a y − ?pathint a x = ?pathint x y
    using zer [OF xys]  contour_integral_reverse_linepath [OF cont_ayf]  add_eq_0_iff
by force
  } note [simp] = this
  { fix e::real
    assume e: 0 < e
    have cont_atx: continuous (at x) f
      using x S contf continuous_on_eq_continuous_at by blast
    then obtain d1 where d1: d1>0 and d1_less: ⋀y. cmod (y − x) < d1 ⟹
cmod (f y − f x) < e/2
      unfolding continuous_at Lim_at dist_norm  using e
      by (drule_tac x=e/2 in spec) force
    obtain d2 where d2: d2>0 ball x d2 ⊆ S using  ‹open S› x
      by (auto simp: open_contains_ball)
    have dpos: min d1 d2 > 0 using d1 d2 by simp
    { fix y
      assume yx: y ≠ x and close: cmod (y − x) < min d1 d2
      have y: y ∈ S
        using d2 close  by (force simp: dist_norm norm_minus_commute)
      have closed_segment x y ⊆ S
        using close d2  by (auto simp: dist_norm norm_minus_commute dest!:
segment_bound(1))
      then have fxy: f contour_integrable_on linepath x y
```

```
      by (metis contour_integrable_continuous_linepath continuous_on_subset
[OF contf])
    then obtain i where i: (f has_contour_integral i) (linepath x y)
      by (auto simp: contour_integrable_on_def)
     then have ((λw. f w − f x) has_contour_integral (i − f x ∗ (y − x)))
(linepath x y)
    by (rule has_contour_integral_diff [OF _ has_contour_integral_const_linepath])
    then have cmod (i − f x ∗ (y − x)) ≤ e / 2 ∗ cmod (y − x)
    proof (rule has_contour_integral_bound_linepath)
      show ⋀u. u ∈ closed_segment x y ⟹ cmod (f u − f x) ≤ e / 2
       by (meson close d1_less le_less_trans less_imp_le min.strict_boundedE
segment_bound1)
    qed (use e in simp)
    also have ... < e ∗ cmod (y − x)
      by (simp add: e yx)
    finally have cmod (?pathint x y − f x ∗ (y−x)) / cmod (y−x) < e
      using i yx  by (simp add: contour_integral_unique divide_less_eq)
   }
   then have ∃ d>0. ∀ y. y ≠ x ∧ cmod (y−x) < d ⟶ cmod (?pathint x y − f
x ∗ (y−x)) / cmod (y−x) < e
    using dpos by blast
 }
 then have (λy. (?pathint x y − f x ∗ (y − x)) /R cmod (y − x)) −x→ 0
  by (simp add: Lim_at dist_norm inverse_eq_divide)
 then have (λy. (1 / cmod (y − x)) ∗R (?pathint a y − (?pathint a x + f x ∗ (y
− x)))) −x→ 0
  using ‹open S› x
  by (force simp: dist_norm open_contains_ball inverse_eq_divide [symmetric]
eventually_at intro: Lim_transform [OF _ tendsto_eventually])
 then show ?thesis
  by (simp add: has_field_derivative_def has_derivative_at2 bounded_linear_mult_right)
qed


lemma holomorphic_starlike_primitive:
 fixes f :: complex ⇒ complex
 assumes contf: continuous_on S f
    and S: starlike S and os: open S
    and k: finite k
    and fcd: ⋀x. x ∈ S − k ⟹ f field_differentiable at x
  shows ∃ g. ∀ x ∈ S. (g has_field_derivative f x) (at x)
proof −
 obtain a where a: a∈S and a_cs: ⋀x. x∈S ⟹ closed_segment a x ⊆ S
  using S by (auto simp: starlike_def)
 { fix x b c
  assume x ∈ S closed_segment b c ⊆ S
  then have abcs: convex hull {a, b, c} ⊆ S
   by (simp add: a a_cs starlike_convex_subset)
  then have continuous_on (convex hull {a, b, c}) f
```

by (*simp add*: *continuous_on_subset* [*OF contf*])
then have (*f has_contour_integral 0*) (*linepath a b +++ linepath b c +++ linepath c a*)
using *abcs interior_subset* **by** (*force intro*: *fcd Cauchy_theorem_triangle_cofinite* [*OF _ k*])
} note *0 = this*
show *?thesis*
proof (*intro exI ballI*)
show $\bigwedge x.\ x \in S \Longrightarrow ((\lambda x.\ contour\_integral\ (linepath\ a\ x)\ f)\ has\_field\_derivative\ f\ x)\ (at\ x)$
using *0 a a_cs contf has_chain_integral_chain_integral3 os triangle_contour_integrals_starlike_primitiv*
by *force*
qed
qed

lemma *Cauchy_theorem_starlike*:
$[\![open\ S;\ starlike\ S;\ finite\ k;\ continuous\_on\ S\ f;$
$\bigwedge x.\ x \in S - k \Longrightarrow f\ field\_differentiable\ at\ x;$
$valid\_path\ g;\ path\_image\ g \subseteq S;\ pathfinish\ g = pathstart\ g]\!]$
$\Longrightarrow (f\ has\_contour\_integral\ 0)\ g$
by (*metis holomorphic_starlike_primitive Cauchy_theorem_primitive at_within_open*)

lemma *Cauchy_theorem_starlike_simple*:
$[\![open\ S;\ starlike\ S;\ f\ holomorphic\_on\ S;\ valid\_path\ g;\ path\_image\ g \subseteq S;\ pathfinish\ g = pathstart\ g]\!]$
$\Longrightarrow (f\ has\_contour\_integral\ 0)\ g$
using *Cauchy_theorem_starlike* [*OF _ _ finite.emptyI*]
by (*simp add*: *holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at*)

2.6 Cauchy's theorem for a convex set

For a convex set we can avoid assuming openness and boundary analyticity

lemma *triangle_contour_integrals_convex_primitive*:
assumes *contf*: *continuous_on S f*
and *S*: *a ∈ S convex S*
and *x*: *x ∈ S*
and *zer*: $\bigwedge b\ c.\ [\![b \in S;\ c \in S]\!] \Longrightarrow contour\_integral\ (linepath\ a\ b)\ f + contour\_integral\ (linepath\ b\ c)\ f + contour\_integral\ (linepath\ c\ a)\ f = 0$
shows $((\lambda x.\ contour\_integral(linepath\ a\ x)\ f)\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof −
let *?pathint* = $\lambda x\ y.\ contour\_integral(linepath\ x\ y)\ f$
{ fix *y*
assume *y*: *y ∈ S*
have *cont_ayf*: *continuous_on* (*closed_segment a y*) *f*
using *S y* **by** (*meson contf continuous_on_subset convex_contains_segment*)
have *xys*: *closed_segment x y ⊆ S*

```
      using convex_contains_segment S x y by auto
    have ?pathint a y - ?pathint a x = ?pathint x y
    using zer [OF x y] contour_integral_reverse_linepath [OF cont_ayf] add_eq_0_iff
by force
  } note [simp] = this
  { fix e::real
    assume e: 0 < e
    have cont_atx: continuous (at x within S) f
      using x S contf by (simp add: continuous_on_eq_continuous_within)
    then obtain d1 where d1: d1>0 and d1_less: ⋀y. ⟦y ∈ S; cmod (y - x) <
d1⟧ ⟹ cmod (f y - f x) < e/2
      unfolding continuous_within Lim_within dist_norm using e
      by (drule_tac x=e/2 in spec) force
    { fix y
      assume yx: y ≠ x and close: cmod (y - x) < d1 and y: y ∈ S
      have fxy: f contour_integrable_on linepath x y
        using convex_contains_segment S x y
      by (blast intro!: contour_integrable_continuous_linepath continuous_on_subset
[OF contf])
      then obtain i where i: (f has_contour_integral i) (linepath x y)
        by (auto simp: contour_integrable_on_def)
        then have ((λw. f w - f x) has_contour_integral (i - f x * (y - x)))
(linepath x y)
      by (rule has_contour_integral_diff [OF _ has_contour_integral_const_linepath])
      then have cmod (i - f x * (y - x)) ≤ e / 2 * cmod (y - x)
      proof (rule has_contour_integral_bound_linepath)
        show ⋀u. u ∈ closed_segment x y ⟹ cmod (f u - f x) ≤ e / 2
        by (meson assms(3) close convex_contains_segment d1_less le_less_trans
less_imp_le segment_bound1 subset_iff x y)
      qed (use e in simp)
      also have ... < e * cmod (y - x)
        by (simp add: e yx)
      finally have cmod (?pathint x y - f x * (y-x)) / cmod (y-x) < e
        using i yx by (simp add: contour_integral_unique divide_less_eq)
    }
    then have ∃d>0. ∀y∈S. y ≠ x ∧ cmod (y-x) < d ⟶ cmod (?pathint x y -
f x * (y-x)) / cmod (y-x) < e
      using d1 by blast
  }
  then have ((λy. (?pathint x y - f x * (y - x)) /R cmod (y - x)) ⟶⟶ 0) (at
x within S)
    by (simp add: Lim_within dist_norm inverse_eq_divide)
  then have ((λy. (1 / cmod (y - x)) *R (?pathint a y - (?pathint a x + f x *
(y - x)))) ⟶⟶ 0)
            (at x within S)
    using linordered_field_no_ub
   by (force simp: inverse_eq_divide [symmetric] eventually_at intro: Lim_transform
[OF _ tendsto_eventually])
  then show ?thesis
```

by (*simp add*: *has_field_derivative_def has_derivative_within bounded_linear_mult_right*)
qed

lemma *contour_integral_convex_primitive*:
assumes *convex S continuous_on S f*
$\bigwedge a\ b\ c.\ [\![a \in S;\ b \in S;\ c \in S]\!] \Longrightarrow (f\ has\_contour\_integral\ 0)\ (linepath\ a\ b\ +++\ linepath\ b\ c\ +++\ linepath\ c\ a)$
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof (*cases S={}*)
case *False*
with *assms that* **show** *?thesis*
by (*blast intro*: *triangle_contour_integrals_convex_primitive has_chain_integral_chain_integral3*)
qed *auto*

lemma *holomorphic_convex_primitive*:
fixes *f* :: *complex* ⇒ *complex*
assumes *convex S finite K* **and** *contf*: *continuous_on S f*
and *fd*: $\bigwedge x.\ x \in interior\ S - K \Longrightarrow f\ field\_differentiable\ at\ x$
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof (*rule contour_integral_convex_primitive* [*OF* ‹*convex S*› *contf Cauchy_theorem_triangle_cofinite*])
have ∗: *convex hull* {*a*, *b*, *c*} ⊆ *S* **if** *a* ∈ *S b* ∈ *S c* ∈ *S* **for** *a b c*
by (*simp add*: ‹*convex S*› *hull_minimal that*)
show *continuous_on* (*convex hull* {*a*, *b*, *c*}) *f* **if** *a* ∈ *S b* ∈ *S c* ∈ *S* **for** *a b c*
by (*meson* ∗ *contf continuous_on_subset that*)
show *f field_differentiable at x* **if** *a* ∈ *S b* ∈ *S c* ∈ *S x* ∈ *interior* (*convex hull* {*a*, *b*, *c*}) − *K* **for** *a b c x*
by (*metis* ∗ *DiffD1 DiffD2 DiffI fd interior_mono subsetCE that*)
qed (*use assms* **in** ‹*force*+›)

lemma *holomorphic_convex_primitive′*:
fixes *f* :: *complex* ⇒ *complex*
assumes *convex S* **and** *open S* **and** *f holomorphic_on S*
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof (*rule holomorphic_convex_primitive*)
fix *x* **assume** *x* ∈ *interior S* − {}
with *assms* **show** *f field_differentiable at x*
by (*auto intro*!: *holomorphic_on_imp_differentiable_at simp*: *interior_open*)
qed (*use assms* **in** ‹*auto intro*: *holomorphic_on_imp_continuous_on*›)

corollary *Cauchy_theorem_convex*:
$[\![continuous\_on\ S\ f;\ convex\ S;\ finite\ K;$
$\bigwedge x.\ x \in interior\ S - K \Longrightarrow f\ field\_differentiable\ at\ x;$
$valid\_path\ g;\ path\_image\ g \subseteq S;\ pathfinish\ g = pathstart\ g]\!]$
$\Longrightarrow (f\ has\_contour\_integral\ 0)\ g$
by (*metis holomorphic_convex_primitive Cauchy_theorem_primitive*)

corollary *Cauchy_theorem_convex_simple*:
assumes *holf*: *f holomorphic_on S*
and *convex S valid_path g path_image g* ⊆ *S pathfinish g* = *pathstart g*

shows (*f has_contour_integral 0*) *g*
proof −
have *f holomorphic_on interior S*
by (*meson holf holomorphic_on_subset interior_subset*)
with *Cauchy_theorem_convex* [**where** *K* = {}] **show** *?thesis*
using *assms*
by (*metis Diff_empty finite.emptyI holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at open_interior*)
qed

In particular for a disc

corollary *Cauchy_theorem_disc*:
⟦*finite K*; *continuous_on* (*cball a e*) *f*;
⋀*x*. *x* ∈ *ball a e* − *K* ⟹ *f field_differentiable at x*;
valid_path g; *path_image g* ⊆ *cball a e*;
pathfinish g = *pathstart g*⟧ ⟹ (*f has_contour_integral 0*) *g*
by (*auto intro*: *Cauchy_theorem_convex*)

corollary *Cauchy_theorem_disc_simple*:
⟦*f holomorphic_on* (*ball a e*); *valid_path g*; *path_image g* ⊆ *ball a e*;
pathfinish g = *pathstart g*⟧ ⟹ (*f has_contour_integral 0*) *g*
by (*simp add*: *Cauchy_theorem_convex_simple*)

2.7 Generalize integrability to local primitives

lemma *contour_integral_local_primitive_lemma*:
fixes *f* :: *complex*⇒*complex*
assumes *gpd*: *g piecewise_differentiable_on* {*a*..*b*}
and *dh*: ⋀*x*. *x* ∈ *S* ⟹ (*f has_field_derivative f* ′ *x*) (*at x within S*)
and *gs*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *S*
shows
(λ*x*. *f* ′ (*g x*) ∗ *vector_derivative g* (*at x within* {*a*..*b*})) *integrable_on* {*a*..*b*}
proof (*cases cbox a b* = {})
case *False*
then show *?thesis*
unfolding *integrable_on_def* **by** (*auto intro*: *assms contour_integral_primitive_lemma*)
qed *auto*

lemma *contour_integral_local_primitive_any*:
fixes *f* :: *complex* ⇒ *complex*
assumes *gpd*: *g piecewise_differentiable_on* {*a*..*b*}
and *dh*: ⋀*x*. *x* ∈ *S*
⟹ ∃ *d h*. 0 < *d* ∧
(∀ *y*. *norm*(*y* − *x*) < *d* ⟶ (*h has_field_derivative f y*) (*at y within S*))
and *gs*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *S*
shows (λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*)) *integrable_on* {*a*..*b*}
proof −
{ fix *x*

```
    assume x: a ≤ x x ≤ b
    obtain d h where d: 0 < d
              and h: (⋀y. norm(y − g x) < d ⟹ (h has_field_derivative f y) (at
y within S))
      using x gs dh by (metis atLeastAtMost_iff)
     have continuous_on {a..b} g using gpd piecewise_differentiable_on_def by
blast
     then obtain e where e: e>0 and lessd: ⋀x'. x' ∈ {a..b} ⟹ |x' − x| < e
⟹ cmod (g x' − g x) < d
      using x d by (fastforce simp: dist_norm continuous_on_iff)
    have ∃e>0. ∀u v. u ≤ x ∧ x ≤ v ∧ {u..v} ⊆ ball x e ∧ (u ≤ v ⟶ a ≤ u ∧
v ≤ b) ⟶
                        (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on {u..v}
    proof −
      have (λx. f (g x) ∗ vector_derivative g (at x within {u..v})) integrable_on
{u..v}
       if u ≤ x x ≤ v and ball: {u..v} ⊆ ball x e and auvb: u ≤ v ⟹ a ≤ u ∧ v
≤ b
       for u v
      proof (rule contour_integral_local_primitive_lemma)
       show g piecewise_differentiable_on {u..v}
       by (metis atLeastatMost_subset_iff gpd piecewise_differentiable_on_subset
auvb)
       show ⋀x. x ∈ g ‘ {u..v} ⟹ (h has_field_derivative f x) (at x within g ‘
{u..v})
       using that by (force simp: ball_def dist_norm intro: lessd gs DERIV_subset
[OF h])
      qed auto
      then show ?thesis
       using e integrable_on_localized_vector_derivative by blast
    qed
  } then
  show ?thesis
    by (force simp: intro!: integrable_on_little_subintervals [of a b, simplified])
qed

lemma contour_integral_local_primitive:
  fixes f :: complex ⇒ complex
  assumes g: valid_path g path_image g ⊆ S
      and dh: ⋀x. x ∈ S
              ⟹ ∃d h. 0 < d ∧
                       (∀y. norm(y − x) < d ⟶ (h has_field_derivative f y) (at y
within S))
    shows f contour_integrable_on g
proof −
  have (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on {0..1}
    using contour_integral_local_primitive_any [OF _ dh] g
    unfolding path_image_def valid_path_def
   by (metis (no_types, lifting) image_subset_iff piecewise_C1_imp_differentiable)
```

then show *?thesis*
using *contour_integrable_on* **by** *presburger*
qed

In particular if a function is holomorphic

lemma *contour_integrable_holomorphic*:
assumes *contf*: *continuous_on S f*
and *os*: *open S*
and *k*: *finite k*
and *g*: *valid_path g path_image g ⊆ S*
and *fcd*: *⋀x. x ∈ S − k ⟹ f field_differentiable at x*
shows *f contour_integrable_on g*
proof −
{ fix *z*
assume *z*: *z ∈ S*
obtain *d* **where** *d>0* **and** *d*: *ball z d ⊆ S* **using** ‹*open S*› *z*
by (*auto simp*: *open_contains_ball*)
then have *contfb*: *continuous_on* (*ball z d*) *f*
using *contf continuous_on_subset* **by** *blast*
obtain *h* **where** *∀ y∈ball z d. (h has_field_derivative f y) (at y within ball z d)*
by (*metis holomorphic_convex_primitive* [*OF convex_ball k contfb fcd*] *d interior_subset Diff_iff subsetD*)
then have *∀ y∈ball z d. (h has_field_derivative f y) (at y within S)*
by (*metis open_ball at_within_open d os subsetCE*)
then have *∃ h. (∀ y. cmod (y − z) < d ⟶ (h has_field_derivative f y) (at y within S))*
by (*force simp*: *dist_norm norm_minus_commute*)
then have *∃ d h. 0 < d ∧ (∀ y. cmod (y − z) < d ⟶ (h has_field_derivative f y) (at y within S))*
using ‹*0 < d*› **by** *blast*
}
then show *?thesis*
by (*rule contour_integral_local_primitive* [*OF g*])
qed

lemma *contour_integrable_holomorphic_simple*:
assumes *fh*: *f holomorphic_on S*
and *os*: *open S*
and *g*: *valid_path g path_image g ⊆ S*
shows *f contour_integrable_on g*
proof −
have *⋀x. x ∈ S ⟹ f field_differentiable at x*
using *fh holomorphic_on_imp_differentiable_at os* **by** *blast*
moreover have *continuous_on S f*
by (*simp add*: *fh holomorphic_on_imp_continuous_on*)
ultimately show *?thesis*
by (*metis Diff_empty contour_integrable_holomorphic finite.emptyI g os*)
qed

lemma *continuous_on_inversediff*:
fixes *z*:: *'a*::*real_normed_field* **shows** $z \notin S \Longrightarrow$ *continuous_on S* ($\lambda w.\ 1\ /\ (w - z)$)
by (*rule continuous_intros* | *force*)+

lemma *contour_integrable_inversediff*:
assumes *g*: *valid_path g*
and *notin*: $z \notin$ *path_image g*
shows ($\lambda w.\ 1\ /\ (w-z)$) *contour_integrable_on g*
proof (*rule contour_integrable_holomorphic_simple*)
show ($\lambda w.\ 1\ /\ (w-z)$) *holomorphic_on UNIV* $-$ $\{z\}$
by (*auto simp*: *holomorphic_on_open open_delete intro*!: *derivative_eq_intros*)
qed (*use assms* **in** *auto*)

Key fact that path integral is the same for a "nearby" path. This is the main lemma for the homotopy form of Cauchy's theorem and is also useful if we want "without loss of generality" to assume some nice properties of a path (e.g. smoothness). It can also be used to define the integrals of analytic functions over arbitrary continuous paths. This is just done for winding numbers now.

A technical definition to avoid duplication of similar proofs, for paths joined at the ends versus looping paths

definition *linked_paths* :: *bool* $\Rightarrow$ (*real* $\Rightarrow$ *'a*) $\Rightarrow$ (*real* $\Rightarrow$ *'a*::*topological_space*) $\Rightarrow$ *bool*
where *linked_paths atends g h* ==
(*if atends then pathstart h* = *pathstart g* $\wedge$ *pathfinish h* = *pathfinish g*
else pathfinish g = *pathstart g* $\wedge$ *pathfinish h* = *pathstart h*)

This formulation covers two cases: g and h share their start and end points; g and h both loop upon themselves.

lemma *contour_integral_nearby*:
assumes *os*: *open S* **and** *p*: *path p path_image p* $\subseteq S$
shows $\exists\, d.\ 0 < d \wedge$
($\forall\, g\ h.$ *valid_path g* $\wedge$ *valid_path h* $\wedge$
($\forall\, t \in \{0..1\}.$ *norm*($g\ t - p\ t$) $< d \wedge$ *norm*($h\ t - p\ t$) $< d$) $\wedge$
linked_paths atends g h
$\longrightarrow$ *path_image g* $\subseteq S \wedge$ *path_image h* $\subseteq S \wedge$
($\forall f.$ *f holomorphic_on S* $\longrightarrow$ *contour_integral h f* = *contour_integral g f*))
proof $-$
have $\forall z.\ \exists e.\ z \in$ *path_image p* $\longrightarrow 0 < e \wedge$ *ball z e* $\subseteq S$
using *open_contains_ball os p*(*2*) **by** *blast*
then obtain *ee* **where** *ee*: $\bigwedge z.\ z \in$ *path_image p* $\Longrightarrow 0 <$ *ee z* $\wedge$ *ball z* (*ee z*) $\subseteq S$
by *metis*
define *cover* **where** *cover* = ($\lambda z.$ *ball z* (*ee z*/*3*)) ' (*path_image p*)
have *compact* (*path_image p*)

```
    by (metis p(1) compact_path_image)
  moreover have path_image p ⊆ (⋃c∈path_image p. ball c (ee c / 3))
    using ee by auto
  ultimately have ∃D ⊆ cover. finite D ∧ path_image p ⊆ ⋃D
    by (simp add: compact_eq_Heine_Borel cover_def)
  then obtain D where D: D ⊆ cover finite D path_image p ⊆ ⋃D
    by blast
  then obtain k where k: k ⊆ {0..1} finite k and D_eq: D = ((λz. ball z (ee z
/ 3)) ∘ p) ‘ k
    unfolding cover_def path_image_def image_comp
    by (meson finite_subset_image)
  then have kne: k ≠ {}
    using D by auto
  have pi: ⋀i. i ∈ k ⟹ p i ∈ path_image p
    using k  by (auto simp: path_image_def)
  then have eepi: ⋀i. i ∈ k ⟹ 0 < ee((p i))
    by (metis ee)
  define e where e = Min((ee ∘ p) ‘ k)
  have fin_eep: finite ((ee ∘ p) ‘ k)
    using k  by blast
  have 0 < e
    using ee k  by (simp add: kne e_def Min_gr_iff [OF fin_eep] eepi)
  have uniformly_continuous_on {0..1} p
    using p  by (simp add: path_def compact_uniformly_continuous)
  then obtain d::real where d: d>0
        and de: ⋀x x'. |x' − x| < d ⟹ x∈{0..1} ⟹ x'∈{0..1} ⟹ cmod (p x'
− p x) < e/3
    unfolding uniformly_continuous_on_def dist_norm real_norm_def
    by (metis divide_pos_pos ‹0 < e› zero_less_numeral)
  then obtain N::nat where N: N>0 inverse N < d
    using real_arch_inverse [of d]   by auto
  show ?thesis
  proof (intro exI conjI allI; clarify?)
    show e/3 > 0
      using ‹0 < e› by simp
    fix g h
   assume g: valid_path g and ghp: ∀t∈{0..1}. cmod (g t − p t) < e / 3 ∧  cmod
(h t − p t) < e / 3
       and h: valid_path h
       and joins: linked_paths atends g h
    { fix t::real
      assume t: 0 ≤ t t ≤ 1
      then obtain u where u: u ∈ k and ptu: p t ∈ ball(p u) (ee(p u) / 3)
        using ‹path_image p ⊆ ⋃D› D_eq by (force simp: path_image_def)
      then have ele: e ≤ ee (p u) using fin_eep
        by (simp add: e_def)
      have cmod (g t − p t) < e / 3 cmod (h t − p t) < e / 3
        using ghp t by auto
      with ele have cmod (g t − p t) < ee (p u) / 3
```

```
                cmod (h t − p t) < ee (p u) / 3
      by linarith+
    then have g t ∈ ball(p u) (ee(p u))  h t ∈ ball(p u) (ee(p u))
      using norm_diff_triangle_ineq [of g t p t p t p u]
            norm_diff_triangle_ineq [of h t p t p t p u] ptu eepi u
      by (force simp: dist_norm ball_def norm_minus_commute)+
    then have g t ∈ S h t ∈ S using ee u k
      by (auto simp: path_image_def ball_def)
  }
  then have ghs: path_image g ⊆ S path_image h ⊆ S
    by (auto simp: path_image_def)
  moreover
  { fix f
    assume fhols: f holomorphic_on S
    then have fpa: f contour_integrable_on g  f contour_integrable_on h
    using g ghs h holomorphic_on_imp_continuous_on os contour_integrable_holomorphic_simple
      by blast+
    have contf: continuous_on S f
      by (simp add: fhols holomorphic_on_imp_continuous_on)
    { fix z
      assume z: z ∈ path_image p
      have f holomorphic_on ball z (ee z)
        using fhols ee z holomorphic_on_subset by blast
      then have ∃ff. (∀ w ∈ ball z (ee z). (ff has_field_derivative f w) (at w))
        using holomorphic_convex_primitive [of ball z (ee z) {} f, simplified]
          by (metis open_ball at_within_open holomorphic_on_def holomor-
phic_on_imp_continuous_on mem_ball)
    }
    then obtain ff where ff:
        ⋀z w. [[z ∈ path_image p; w ∈ ball z (ee z)]] ⟹ (ff z has_field_derivative
f w) (at w)
      by metis
    { fix n
      assume n: n ≤ N
    then have contour_integral(subpath 0 (n/N) h) f − contour_integral(subpath
0 (n/N) g) f =
                          contour_integral(linepath (g(n/N)) (h(n/N))) f − con-
tour_integral(linepath (g 0) (h 0)) f
      proof (induct n)
        case 0 show ?case by simp
      next
        case (Suc n)
        obtain t where t: t ∈ k and p (n/N) ∈ ball(p t) (ee(p t) / 3)
         using ‹path_image p ⊆ ⋃ D› [THEN subsetD, where c=p (n/N)] D_eq
N Suc.prems
          by (force simp: path_image_def)
        then have ptu: cmod (p t − p (n/N)) < ee (p t) / 3
          by (simp add: dist_norm)
        have e3le: e/3 ≤ ee (p t) / 3  using fin_eep t
```

```
          by (simp add: e_def)
        { fix x
          assume x: n/N ≤ x x ≤ (1 + n)/N
          then have nN01: 0 ≤ n/N (1 + n)/N ≤ 1
            using Suc.prems by auto
          then have x01: 0 ≤ x x ≤ 1
            using x by linarith+
          have cmod (p t − p x) < ee (p t) / 3 + e/3
          proof (rule norm_diff_triangle_less [OF ptu de])
            show |real n / real N − x| < d
              using x N by (auto simp: field_simps)
          qed (use x01 Suc.prems in auto)
          then have ptx: cmod (p t − p x) < 2∗ee (p t)/3
            using e3le eepi [OF t] by simp
          have cmod (p t − g x) < 2∗ee (p t)/3 + e/3
            using ghp x01
        by (force simp add: norm_minus_commute intro!: norm_diff_triangle_less
[OF ptx])
          also have ... ≤ ee (p t)
            using e3le eepi [OF t] by simp
          finally have gg: cmod (p t − g x) < ee (p t) .
          have cmod (p t − h x) < 2∗ee (p t)/3 + e/3
            using ghp x01
        by (force simp add: norm_minus_commute intro!: norm_diff_triangle_less
[OF ptx])
          also have ... ≤ ee (p t)
            using e3le eepi [OF t] by simp
          finally have cmod (p t − g x) < ee (p t) cmod (p t − h x) < ee (p t)
            using gg by auto
        } note ptgh_ee = this
          have closed_segment (g (n/N)) (h (n/N)) = path_image (linepath (h
(n/N)) (g (n/N)))
          by (simp add: closed_segment_commute)
        also have pi_hgn: ... ⊆ ball (p t) (ee (p t))
          using ptgh_ee [of n/N] Suc.prems
         by (auto simp: field_simps dist_norm dest: segment_furthest_le [where
y=p t])
        finally have gh_ns: closed_segment (g (n/N)) (h (n/N)) ⊆ S
          using ee pi t by blast
       have pi_ghn': path_image (linepath (g ((1 + n) / N)) (h ((1 + n) / N)))
⊆ ball (p t) (ee (p t))
          using ptgh_ee [of (1+n)/N] Suc.prems
         by (auto simp: field_simps dist_norm dest: segment_furthest_le [where
y=p t])
        then have gh_n's: closed_segment (g ((1 + n) / N)) (h ((1 + n) / N))
⊆ S
          using ‹N>0› Suc.prems ee pi t
          by (auto simp: Path_Connected.path_image_join field_simps)
        have pi_subset_ball:
```

```
            path_image (subpath (n/N) ((1+n) / N) g +++ linepath (g ((1+n)
/ N)) (h ((1+n) / N)) +++
                        subpath ((1+n) / N) (n/N) h +++ linepath (h (n/N)) (g
(n/N)))
             ⊆ ball (p t) (ee (p t))
        proof (intro subset_path_image_join pi_hgn pi_ghn')
         show path_image (subpath (n/N) ((1+n) / N) g) ⊆ ball (p t) (ee (p t))
              path_image (subpath ((1+n) / N) (n/N) h) ⊆ ball (p t) (ee (p t))
           using ‹N>0› Suc.prems
           by (auto simp: path_image_subpath dist_norm field_simps ptgh_ee)
        qed
        have pi0: (f has_contour_integral 0)
                  (subpath (n/ N) ((Suc n)/N) g +++ linepath(g ((Suc n) / N))
(h((Suc n) / N)) +++
                       subpath ((Suc n) / N) (n/N) h +++ linepath(h (n/N)) (g
(n/N)))
        proof (rule Cauchy_theorem_primitive)
          show ⋀x. x ∈ ball (p t) (ee (p t))
                   ⟹ (ff (p t) has_field_derivative f x) (at x within ball (p t) (ee
(p t)))
           by (metis ff open_ball at_within_open pi t)
         qed (use Suc.prems pi_subset_ball in ‹simp_all add: valid_path_subpath
g h›)
        have fpa1: f contour_integrable_on subpath (n/N) (real (Suc n) / real N)
g
          using Suc.prems by (simp add: contour_integrable_subpath g fpa)
        have fpa2: f contour_integrable_on linepath (g (real (Suc n) / real N)) (h
(real (Suc n) / real N))
          using gh_n's
              by (auto intro!: contour_integrable_continuous_linepath continu-
ous_on_subset [OF contf])
        have fpa3: f contour_integrable_on linepath (h (n/N)) (g (n/N))
          using gh_ns
       by (auto simp: closed_segment_commute intro!: contour_integrable_continuous_linepath
continuous_on_subset [OF contf])
        have eq0: contour_integral (subpath (n/N) ((Suc n) / real N) g) f +
                  contour_integral (linepath (g ((Suc n) / N)) (h ((Suc n) / N))) f
+
                   contour_integral (subpath ((Suc n) / N) (n/N) h) f +
                   contour_integral (linepath (h (n/N)) (g (n/N))) f = 0
          using contour_integral_unique [OF pi0] Suc.prems
          by (simp add: g h fpa valid_path_subpath contour_integrable_subpath
                    fpa1 fpa2 fpa3 algebra_simps del: of_nat_Suc)
        have *: ⋀hn he hn' gn gd gn' hgn ghn gh0 ghn'.
                 ⟦hn − gn = ghn − gh0;
                  gd + ghn' + he + hgn = (0::complex);
                   hn − he = hn'; gn + gd = gn'; hgn = −ghn⟧ ⟹ hn' − gn' =
ghn' − gh0
          by (auto simp: algebra_simps)
```

```
        have contour_integral (subpath 0 (n/N) h) f − contour_integral (subpath
((Suc n) / N) (n/N) h) f =
              contour_integral (subpath 0 (n/N) h) f + contour_integral (subpath
(n/N) ((Suc n) / N) h) f
          unfolding reversepath_subpath [symmetric, of ((Suc n) / N)]
            using Suc.prems by (simp add: h fpa contour_integral_reversepath
valid_path_subpath contour_integrable_subpath)
        also have ... = contour_integral (subpath 0 ((Suc n) / N) h) f
          using Suc.prems by (simp add: contour_integral_subpath_combine h
fpa)
        finally have pi0_eq:
            contour_integral (subpath 0 (n/N) h) f − contour_integral (subpath
((Suc n) / N) (n/N) h) f =
             contour_integral (subpath 0 ((Suc n) / N) h) f .
        show ?case
        proof (rule * [OF Suc.hyps eq0 pi0_eq])
          show contour_integral (subpath 0 (n/N) g) f +
               contour_integral (subpath (n/N) ((Suc n) / N) g) f =
               contour_integral (subpath 0 ((Suc n) / N) g) f
            using Suc.prems contour_integral_subpath_combine fpa(1) g by auto
            show contour_integral (linepath (h (n/N)) (g (n/N))) f = − con-
tour_integral (linepath (g (n/N)) (h (n/N))) f
            by (metis contour_integral_unique fpa3 has_contour_integral_integral
has_contour_integral_reverse_linepath)
        qed (use Suc.prems in auto)
      qed
      } note ind = this
      have contour_integral h f = contour_integral g f
        using ind [OF order_refl] N joins
      by (simp add: linked_paths_def pathstart_def pathfinish_def split: if_split_asm)
    }
    ultimately
    show path_image g ⊆ S ∧ path_image h ⊆ S ∧ (∀ f. f holomorphic_on S ⟶
contour_integral h f = contour_integral g f)
      by metis
  qed
qed


lemma
  assumes open S path p path_image p ⊆ S
    shows contour_integral_nearby_ends:
      ∃ d. 0 < d ∧
            (∀ g h. valid_path g ∧ valid_path h ∧
                 (∀ t ∈ {0..1}. norm(g t − p t) < d ∧ norm(h t − p t) < d) ∧
                 pathstart h = pathstart g ∧ pathfinish h = pathfinish g
                 ⟶ path_image g ⊆ S ∧
                    path_image h ⊆ S ∧
                    (∀ f. f holomorphic_on S
```

```
                    ⟶ contour_integral h f = contour_integral g f))
  and contour_integral_nearby_loops:
   ∃d. 0 < d ∧
        (∀g h. valid_path g ∧ valid_path h ∧
            (∀t ∈ {0..1}. norm(g t − p t) < d ∧ norm(h t − p t) < d) ∧
            pathfinish g = pathstart g ∧ pathfinish h = pathstart h
            ⟶ path_image g ⊆ S ∧
              path_image h ⊆ S ∧
              (∀f. f holomorphic_on S
                  ⟶ contour_integral h f = contour_integral g f))
 using contour_integral_nearby [OF assms, where atends=True]
 using contour_integral_nearby [OF assms, where atends=False]
 unfolding linked_paths_def by simp_all

lemma contour_integral_bound_exists:
assumes S: open S
   and g: valid_path g
   and pag: path_image g ⊆ S
 shows ∃L. 0 < L ∧
        (∀f B. f holomorphic_on S ∧ (∀z ∈ S. norm(f z) ≤ B)
          ⟶ norm(contour_integral g f) ≤ L*B)
proof −
 have path g using g
   by (simp add: valid_path_imp_path)
 then obtain d::real and p
   where d: 0 < d
     and p: polynomial_function p path_image p ⊆ S
     and pi: ⋀f. f holomorphic_on S ⟹ contour_integral g f = contour_integral
p f
   using contour_integral_nearby_ends [OF S ‹path g› pag]
  by (metis cancel_comm_monoid_add_class.diff_cancel g norm_zero path_approx_polynomial_function
valid_path_polynomial_function)
 then obtain p' where p': polynomial_function p'
   ⋀x. (p has_vector_derivative (p' x)) (at x)
   by (blast intro: has_vector_derivative_polynomial_function that)
 then have bounded(p' ‘ {0..1})
   using continuous_on_polymonial_function
   by (force simp: intro!: compact_imp_bounded compact_continuous_image)
 then obtain L where L: L>0 and nop': ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ norm (p' x)
≤ L
   by (force simp: bounded_pos)
 { fix f B
   assume f: f holomorphic_on S and B: ⋀z. z∈S ⟹ cmod (f z) ≤ B
   then have f contour_integrable_on p ∧ valid_path p
     using p S
   by (blast intro: valid_path_polynomial_function contour_integrable_holomorphic_simple
holomorphic_on_imp_continuous_on)
   moreover have cmod (vector_derivative p (at x)) * cmod (f (p x)) ≤ L * B
if 0 ≤ x x ≤ 1 for x
```

```
    proof (rule mult_mono)
      show cmod (vector_derivative p (at x)) ≤ L
        by (metis nop' p'(2) that vector_derivative_at)
      show cmod (f (p x)) ≤ B
        by (metis B atLeastAtMost_iff imageI p(2) path_defs(4) subset_eq that)
    qed (use ‹L>0› in auto)
    ultimately
    have cmod (integral {0..1} (λx. f (p x) * vector_derivative p (at x))) ≤ L * B
      by (intro order_trans [OF integral_norm_bound_integral])
         (auto simp: mult.commute norm_mult contour_integrable_on)
    then have cmod (contour_integral g f) ≤ L * B
      using contour_integral_integral f pi by presburger
  } then
  show ?thesis using ‹L > 0›
    by (intro exI[of _ L]) auto
qed
```

2.8 Homotopy forms of Cauchy's theorem

```
lemma Cauchy_theorem_homotopic:
    assumes hom: if atends then homotopic_paths S g h else homotopic_loops S g
h
        and open S and f: f holomorphic_on S
        and vpg: valid_path g and vph: valid_path h
    shows contour_integral g f = contour_integral h f
proof −
  have pathsf: linked_paths atends g h
    using hom  by (auto simp: linked_paths_def homotopic_paths_imp_pathstart
homotopic_paths_imp_pathfinish homotopic_loops_imp_loop)
  obtain k :: real × real ⇒ complex
    where contk: continuous_on ({0..1} × {0..1}) k
      and ks: k ‘ ({0..1} × {0..1}) ⊆ S
      and k [simp]: ∀x. k (0, x) = g x ∀x. k (1, x) = h x
      and ksf: ∀t∈{0..1}. linked_paths atends g (λx. k (t, x))
        using hom pathsf by (auto simp: linked_paths_def homotopic_paths_def
homotopic_loops_def homotopic_with_def split: if_split_asm)
  have ucontk: uniformly_continuous_on ({0..1} × {0..1}) k
    by (blast intro: compact_Times compact_uniformly_continuous [OF contk])
  { fix t::real assume t: t ∈ {0..1}
    have Pair t ‘ {0..1} ⊆ {0..1} × {0..1}
      using t by force
    then have pak: path (k ∘ (λu. (t, u)))
      unfolding path_def
      by (intro continuous_intros continuous_on_subset [OF contk])+
    have pik: path_image (k ∘ Pair t) ⊆ S
      using ks t by (auto simp: path_image_def)
    obtain e where e>0 and e:
        ⋀g h. ⟦valid_path g; valid_path h;
                ∀u∈{0..1}. cmod (g u − (k ∘ Pair t) u) < e ∧ cmod (h u − (k ∘
```

```
Pair t) u) < e;
                linked_paths atends g h⟧
               ⟹ contour_integral h f = contour_integral g f
     using contour_integral_nearby [OF ‹open S› pak pik, of atends] f by metis
   obtain d where d>0 and d:
       ⋀x x'. ⟦x ∈ {0..1} × {0..1}; x' ∈ {0..1} × {0..1}; norm (x'−x) < d⟧ ⟹
norm (k x' − k x) < e/4
   by (rule uniformly_continuous_onE [OF ucontk, of e/4]) (auto simp: dist_norm
‹e>0›)
   { fix t1 t2
     assume t1: 0 ≤ t1 t1 ≤ 1 and t2: 0 ≤ t2 t2 ≤ 1 and ltd: |t1 − t| < d |t2
− t| < d
      have no2: norm(g1 − kt) < e if norm(g1 − k1) < e/4 norm(k1 − kt) <
e/4 for g1 k1 kt :: complex
     proof (rule norm_triangle_half_l)
       show cmod (g1 − k1) < e/2 cmod (kt − k1) < e/2
           using ‹e > 0› that by (auto simp: norm_minus_commute intro: or-
der_less_trans)
     qed
     have ∃ d>0. ∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                      (∀ u∈{0..1}. cmod (g1 u − k (t1, u)) < d ∧ cmod (g2 u − k
(t2, u)) < d) ∧
                       linked_paths atends g1 g2 ⟶
                       contour_integral g2 f = contour_integral g1 f
       using t t1 t2 ltd ‹e > 0›
     by (rule_tac x=e/4 in exI) (auto intro!: e simp: d no2 simp del: less_divide_eq_numeral1)
   }
   then have ∃ e. 0 < e ∧
           (∀ t1 t2. t1 ∈ {0..1} ∧ t2 ∈ {0..1} ∧ |t1 − t| < e ∧ |t2 − t| < e
             ⟶ (∃ d. 0 < d ∧
                 (∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                   (∀ u ∈ {0..1}.
                    norm(g1 u − k((t1,u))) < d ∧ norm(g2 u − k((t2,u))) < d) ∧
                     linked_paths atends g1 g2
                     ⟶ contour_integral g2 f = contour_integral g1 f)))
     by (rule_tac x=d in exI) (simp add: ‹d > 0›)
 }
 then obtain ee where ee:
      ⋀t. t ∈ {0..1} ⟹ ee t > 0 ∧
        (∀ t1 t2. t1 ∈ {0..1} ⟶ t2 ∈ {0..1} ⟶ |t1 − t| < ee t ⟶ |t2 − t| <
ee t
          ⟶ (∃ d. 0 < d ∧
              (∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                (∀ u ∈ {0..1}.
                   norm(g1 u − k((t1,u))) < d ∧ norm(g2 u − k((t2,u))) < d) ∧
                   linked_paths atends g1 g2
                   ⟶ contour_integral g2 f = contour_integral g1 f)))
   by metis
 note ee_rule = ee [THEN conjunct2, rule_format, of 0 0 0]
```

define C **where** $C = (\lambda t.\ \mathit{ball}\ t\ (\mathit{ee}\ t\ /\ 3))\ `\ \{0..1\}$
obtain C' **where** C': $C' \subseteq C$ *finite* C' **and** $C'01$: $\{0..1\} \subseteq \bigcup C'$
proof (*rule compactE* [*OF compact_interval*])
show $\{0..1\} \subseteq \bigcup C$
using *ee* [*THEN conjunct1*] **by** (*auto simp*: *C_def dist_norm*)
qed (*use C_def* **in** *auto*)
define *kk* **where** $kk = \{t \in \{0..1\}.\ \mathit{ball}\ t\ (\mathit{ee}\ t\ /\ 3) \in C'\}$
have *kk01*: $kk \subseteq \{0..1\}$ **by** (*auto simp*: *kk_def*)
define e **where** $e = \mathit{Min}\ (\mathit{ee}\ `\ kk)$
have C'_eq: $C' = (\lambda t.\ \mathit{ball}\ t\ (\mathit{ee}\ t\ /\ 3))\ `\ kk$
using C' **by** (*auto simp*: *kk_def C_def*)
have *ee_pos*[*simp*]: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow \mathit{ee}\ t > 0$
by (*simp add*: *kk_def ee*)
moreover have *finite kk*
using ‹*finite* C'› *kk01* **by** (*force simp*: C'_eq *inj_on_def ball_eq_ball_iff dest*: *ee_pos finite_imageD*)
moreover have $kk \neq \{\}$ **using** ‹$\{0..1\} \subseteq \bigcup C'$› C'_eq **by** *force*
ultimately have $e > 0$
using *finite_less_Inf_iff* [*of ee ` kk 0*] *kk01* **by** (*force simp*: *e_def*)
then obtain N::*nat* **where** $N > 0$ **and** N: $1/N < e/3$
by (*meson divide_pos_pos nat_approx_posE zero_less_Suc zero_less_numeral*)
have *e_le_ee*: $\bigwedge i.\ i \in kk \Longrightarrow e \leq \mathit{ee}\ i$
using ‹*finite kk*› **by** (*simp add*: *e_def Min_le_iff* [*of ee ` kk*])
have *plus*: $\exists\, t \in kk.\ x \in \mathit{ball}\ t\ (\mathit{ee}\ t\ /\ 3)$ **if** $x \in \{0..1\}$ **for** x
using C' *subsetD* [*OF C'01 that*] **unfolding** C'_eq **by** *blast*
have [*OF order_refl*]:
$\exists\, d.\ 0 < d \wedge (\forall\, j.\ \mathit{valid\_path}\ j \wedge (\forall\, u \in \{0..1\}.\ \mathit{norm}(j\ u - k\ (n/N,\ u)) < d) \wedge \mathit{linked\_paths}\ \mathit{atends}\ g\ j \longrightarrow \mathit{contour\_integral}\ j\ f = \mathit{contour\_integral}\ g\ f)$
if $n \leq N$ **for** n
using *that*
proof (*induct n*)
case *0* **show** *?case*
using *ee_rule*
by *clarsimp* (*metis diff_self norm_eq_zero vpg*)
next
case (*Suc n*)
then have *N01*: $n/N \in \{0..1\}$ $(\mathit{Suc}\ n)/N \in \{0..1\}$ **by** *auto*
then obtain t **where** t: $t \in kk$ $n/N \in \mathit{ball}\ t\ (\mathit{ee}\ t\ /\ 3)$
using *plus* [*of n/N*] **by** *blast*
then have *nN_less*: $|n/N - t| < \mathit{ee}\ t$
by (*simp add*: *dist_norm del*: *less_divide_eq_numeral1*)
have $n'N\_less$: $|\mathit{real}\ (\mathit{Suc}\ n)\ /\ \mathit{real}\ N - t| < \mathit{ee}\ t$
using t N ‹$N > 0$› *e_le_ee* [*of t*]
by (*simp add*: *dist_norm add_divide_distrib abs_diff_less_iff del*: *less_divide_eq_numeral1*) (*simp add*: *field_simps*)
have *t01*: $t \in \{0..1\}$ **using** ‹$kk \subseteq \{0..1\}$› ‹$t \in kk$› **by** *blast*
obtain *d1* **where** $d1 > 0$ **and** *d1*:
$\bigwedge g1\ g2.\ [\![\mathit{valid\_path}\ g1;\ \mathit{valid\_path}\ g2;$

```
                ∀u∈{0..1}. cmod (g1 u − k (n/N, u)) < d1 ∧ cmod (g2 u − k
((Suc n) / N, u)) < d1;
                linked_paths atends g1 g2⟧
               ⟹ contour_integral g2 f = contour_integral g1 f
    using ee [THEN conjunct2, rule_format, OF t01 N01 nN_less n'N_less] by
fastforce
  have n ≤ N using Suc.prems by auto
  with Suc.hyps
  obtain d2 where d2 > 0
    and d2: ⋀j. ⟦valid_path j; ∀u∈{0..1}. cmod (j u − k (n/N, u)) < d2;
linked_paths atends g j⟧
              ⟹ contour_integral j f = contour_integral g f
    by auto
  have Pair (n/ N) ` {0..1} ⊆ {0..1} × {0..1}
    using N01 by auto
  then have continuous_on {0..1} (k ∘ (λu. (n/N, u)))
    by (intro continuous_intros continuous_on_subset [OF contk])
  then have pkn: path (λu. k (n/N, u))
    by (simp add: path_def)
  have min12: min d1 d2 > 0 by (simp add: ‹0 < d1› ‹0 < d2›)
  obtain p where polynomial_function p
    and psf: pathstart p = pathstart (λu. k (n/N, u))
          pathfinish p = pathfinish (λu. k (n/N, u))
    and pk_le: ⋀t. t∈{0..1} ⟹ cmod (p t − k (n/N, t)) < min d1 d2
    using path_approx_polynomial_function [OF pkn min12] by blast
  then have vpp: valid_path p using valid_path_polynomial_function by blast
  have lpa: linked_paths atends g p
    by (metis (mono_tags, lifting) N01(1) ksf linked_paths_def pathfinish_def
pathstart_def psf)
  show ?case
  proof (intro exI; safe)
    fix j
    assume valid_path j linked_paths atends g j
      and ∀u∈{0..1}. cmod (j u − k (real (Suc n) / real N, u)) < min d1 d2
    then have contour_integral j f = contour_integral p f
     using pk_le N01(1) ksf by (force intro!: vpp d1 simp add: linked_paths_def
psf)
    also have ... = contour_integral g f
      using pk_le by (force intro!: vpp d2 lpa)
    finally show contour_integral j f = contour_integral g f .
  qed (simp add: ‹0 < d1› ‹0 < d2›)
 qed
 then obtain d where 0 < d
                  ⋀j. valid_path j ∧ (∀u ∈ {0..1}. norm(j u − k (1,u)) < d) ∧
linked_paths atends g j
                 ⟹ contour_integral j f = contour_integral g f
  using ‹N>0› by auto
 then have linked_paths atends g h ⟹ contour_integral h f = contour_integral
g f
```

using ‹*N*>*0*› *vph* **by** *fastforce*
then show *?thesis*
by (*simp add*: *pathsf*)
qed

proposition *Cauchy_theorem_homotopic_paths*:
assumes *hom*: *homotopic_paths S g h*
and *open S* **and** *f*: *f holomorphic_on S*
and *vpg*: *valid_path g* **and** *vph*: *valid_path h*
shows *contour_integral g f* = *contour_integral h f*
using *Cauchy_theorem_homotopic* [*of True S g h*] *assms* **by** *simp*

proposition *Cauchy_theorem_homotopic_loops*:
assumes *hom*: *homotopic_loops S g h*
and *open S* **and** *f*: *f holomorphic_on S*
and *vpg*: *valid_path g* **and** *vph*: *valid_path h*
shows *contour_integral g f* = *contour_integral h f*
using *Cauchy_theorem_homotopic* [*of False S g h*] *assms* **by** *simp*

lemma *has_contour_integral_newpath*:
⟦(*f has_contour_integral y*) *h*; *f contour_integrable_on g*; *contour_integral g f* = *contour_integral h f*⟧
⟹ (*f has_contour_integral y*) *g*
using *has_contour_integral_integral contour_integral_unique* **by** *auto*

lemma *Cauchy_theorem_null_homotopic*:
⟦*f holomorphic_on S*; *open S*; *valid_path g*; *homotopic_loops S g* (*linepath a a*)⟧
⟹ (*f has_contour_integral 0*) *g*
by (*metis Cauchy_theorem_homotopic_loops contour_integrable_holomorphic_simple valid_path_linepath*
contour_integral_trivial has_contour_integral_integral homotopic_loops_imp_subset)

end

3 Winding numbers

theory *Winding_Numbers*
imports *Cauchy_Integral_Theorem*
begin

3.1 Definition

definition *winding_number_prop* :: [*real* ⇒ *complex*, *complex*, *real*, *real* ⇒ *complex*, *complex*] ⇒ *bool* **where**
winding_number_prop γ z e p n ≡
valid_path p ∧ *z* ∉ *path_image p* ∧
pathstart p = *pathstart γ* ∧
pathfinish p = *pathfinish γ* ∧

$(\forall\, t \in \{0..1\}.\ norm(\gamma\ t - p\ t) < e) \wedge$
$contour\_integral\ p\ (\lambda w.\ 1/(w - z)) = 2 * pi * \mathrm{i} * n$

definition *winding_number*:: $[real \Rightarrow complex, complex] \Rightarrow complex$ **where**
$winding\_number\ \gamma\ z \equiv SOME\ n.\ \forall\, e > 0.\ \exists\, p.\ winding\_number\_prop\ \gamma\ z\ e\ p\ n$

lemma *winding_number*:
assumes $path\ \gamma\ z \notin path\_image\ \gamma\ 0 < e$
shows $\exists\, p.\ winding\_number\_prop\ \gamma\ z\ e\ p\ (winding\_number\ \gamma\ z)$
proof −
have $path\_image\ \gamma \subseteq UNIV - \{z\}$
using *assms* **by** *blast*
then obtain *d*
where *d*: $d>0$
and *pi_eq*: $\bigwedge h1\ h2.\ valid\_path\ h1 \wedge valid\_path\ h2 \wedge$
$(\forall\, t{\in}\{0..1\}.\ cmod\ (h1\ t - \gamma\ t) < d \wedge cmod\ (h2\ t - \gamma\ t) < d) \wedge$
$pathstart\ h2 = pathstart\ h1 \wedge pathfinish\ h2 = pathfinish\ h1 \longrightarrow$
$path\_image\ h1 \subseteq UNIV - \{z\} \wedge path\_image\ h2 \subseteq UNIV - \{z\} \wedge$
$(\forall f.\ f\ holomorphic\_on\ UNIV - \{z\} \longrightarrow contour\_integral\ h2\ f = contour\_integral\ h1\ f)$
using *contour_integral_nearby_ends* [*of* $UNIV - \{z\}\ \gamma$] *assms* **by** (*auto simp*: *open_delete*)
then obtain *h* **where** *h*: $polynomial\_function\ h \wedge pathstart\ h = pathstart\ \gamma \wedge pathfinish\ h = pathfinish\ \gamma \wedge$
$(\forall\, t \in \{0..1\}.\ norm(h\ t - \gamma\ t) < d/2)$
using *path_approx_polynomial_function* [*OF* ‹*path* γ›, *of* $d/2$] d **by** (*metis half_gt_zero_iff*)
define *nn* **where** $nn = 1/(2{*}\ pi{*}\mathrm{i}) * contour\_integral\ h\ (\lambda w.\ 1/(w - z))$
have $\exists\, n.\ \forall\, e > 0.\ \exists\, p.\ winding\_number\_prop\ \gamma\ z\ e\ p\ n$
proof (*rule_tac* $x{=}nn$ **in** *exI*, *clarify*)
fix *e*::*real*
assume *e*: $e>0$
obtain *p* **where** *p*: $polynomial\_function\ p \wedge$
$pathstart\ p = pathstart\ \gamma \wedge pathfinish\ p = pathfinish\ \gamma \wedge (\forall\, t{\in}\{0..1\}.\ cmod\ (p\ t - \gamma\ t) < min\ e\ (d/2))$
using *path_approx_polynomial_function* [*OF* ‹*path* γ›, *of* $min\ e\ (d/2)$] d ‹$0<e$›
by (*metis min_less_iff_conj zero_less_divide_iff zero_less_numeral*)
have $(\lambda w.\ 1\ /\ (w - z))\ holomorphic\_on\ UNIV - \{z\}$
by (*auto simp*: *intro*!: *holomorphic_intros*)
then have $winding\_number\_prop\ \gamma\ z\ e\ p\ nn$
using *pi_eq* [*of h p*] *h p d*
by (*auto simp*: *valid_path_polynomial_function norm_minus_commute nn_def winding_number_prop_def*)
then show $\exists\, p.\ winding\_number\_prop\ \gamma\ z\ e\ p\ nn$
by *metis*
qed

```
  then show ?thesis
    unfolding winding_number_def by (rule someI2_ex) (blast intro: ‹0<e›)
qed

lemma winding_number_unique:
  assumes γ: path γ z ∉ path_image γ
      and pi: ⋀e. e>0 ⟹ ∃p. winding_number_prop γ z e p n
   shows winding_number γ z = n
proof −
  have path_image γ ⊆ UNIV − {z}
    using assms by blast
  then obtain e
    where e: e>0
      and pi_eq: ⋀h1 h2 f. ⟦valid_path h1; valid_path h2;
                (∀t∈{0..1}. cmod (h1 t − γ t) < e ∧ cmod (h2 t − γ t) < e);
                   pathstart h2 = pathstart h1; pathfinish h2 = pathfinish h1; f
holomorphic_on UNIV − {z}⟧ ⟹
                contour_integral h2 f = contour_integral h1 f
   using contour_integral_nearby_ends [of UNIV − {z} γ] assms  by (auto simp:
open_delete)
  obtain p where p: winding_number_prop γ z e p n
    using pi [OF e] by blast
  obtain q where q: winding_number_prop γ z e q (winding_number γ z)
    using winding_number [OF γ e] by blast
  have 2 * complex_of_real pi * i * n = contour_integral p (λw. 1 / (w − z))
    using p by (auto simp: winding_number_prop_def)
  also have ... = contour_integral q (λw. 1 / (w − z))
  proof (rule pi_eq)
    show (λw. 1 / (w − z)) holomorphic_on UNIV − {z}
      by (auto intro!: holomorphic_intros)
 qed (use p q in ‹auto simp: winding_number_prop_def norm_minus_commute›)
  also have ... = 2 * complex_of_real pi * i * winding_number γ z
    using q by (auto simp: winding_number_prop_def)
  finally have 2 * complex_of_real pi * i * n = 2 * complex_of_real pi * i *
winding_number γ z .
  then show ?thesis
    by simp
qed


lemma winding_number_unique_loop:
  assumes γ: path γ z ∉ path_image γ
      and loop: pathfinish γ = pathstart γ
      and pi:
        ⋀e. e>0 ⟹ ∃p. valid_path p ∧ z ∉ path_image p ∧
                       pathfinish p = pathstart p ∧
                       (∀t ∈ {0..1}. norm (γ t − p t) < e) ∧
                       contour_integral p (λw. 1/(w − z)) = 2 * pi * i * n
   shows winding_number γ z = n
```

proof −
have *path_image* $\gamma \subseteq$ *UNIV* − {*z*}
using *assms* **by** *blast*
then obtain *e*
where *e*: *e*>*0*
and *pi_eq*: $\bigwedge$*h1 h2 f*. ⟦*valid_path h1*; *valid_path h2*;
(∀ *t*∈{*0..1*}. *cmod* (*h1 t* − γ *t*) < *e* ∧ *cmod* (*h2 t* − γ *t*) < *e*);
pathfinish h1 = *pathstart h1*; *pathfinish h2* = *pathstart h2*; *f* *holomorphic_on UNIV* − {*z*}⟧ ⟹
contour_integral h2 f = *contour_integral h1 f*
using *contour_integral_nearby_loops* [*of UNIV* − {*z*} γ] *assms* **by** (*auto simp*: *open_delete*)
obtain *p* **where** *p*:
valid_path p ∧ *z* ∉ *path_image p* ∧ *pathfinish p* = *pathstart p* ∧
(∀ *t* ∈ {*0..1*}. *norm* (γ *t* − *p t*) < *e*) ∧
contour_integral p (λ*w*. *1*/(*w* − *z*)) = *2* ∗ *pi* ∗ i ∗ *n*
using *pi* [*OF e*] **by** *blast*
obtain *q* **where** *q*: *winding_number_prop* γ *z e q* (*winding_number* γ *z*)
using *winding_number* [*OF* γ *e*] **by** *blast*
have *2* ∗ *complex_of_real pi* ∗ i ∗ *n* = *contour_integral p* (λ*w*. *1* / (*w* − *z*))
using *p* **by** *auto*
also have ... = *contour_integral q* (λ*w*. *1* / (*w* − *z*))
proof (*rule pi_eq*)
show (λ*w*. *1* / (*w* − *z*)) *holomorphic_on UNIV* − {*z*}
by (*auto intro*!: *holomorphic_intros*)
qed (*use p q loop* **in** ‹*auto simp*: *winding_number_prop_def norm_minus_commute*›)
also have ... = *2* ∗ *complex_of_real pi* ∗ i ∗ *winding_number* γ *z*
using *q* **by** (*auto simp*: *winding_number_prop_def*)
finally have *2* ∗ *complex_of_real pi* ∗ i ∗ *n* = *2* ∗ *complex_of_real pi* ∗ i ∗ *winding_number* γ *z* **.**
then show *?thesis*
by *simp*
qed

proposition *winding_number_valid_path*:
assumes *valid_path* γ *z* ∉ *path_image* γ
shows *winding_number* γ *z* = *1*/(*2*∗*pi*∗i) ∗ *contour_integral* γ (λ*w*. *1*/(*w* − *z*))
by (*rule winding_number_unique*)
(*use assms* **in** ‹*auto simp*: *valid_path_imp_path winding_number_prop_def*›)

proposition *has_contour_integral_winding_number*:
assumes γ: *valid_path* γ *z* ∉ *path_image* γ
shows ((λ*w*. *1*/(*w* − *z*)) *has_contour_integral* (*2*∗*pi*∗i∗*winding_number* γ *z*)) γ
by (*simp add*: *winding_number_valid_path has_contour_integral_integral contour_integrable_inversediff assms*)

lemma *winding_number_trivial* [*simp*]: *z* ≠ *a* ⟹ *winding_number*(*linepath a a*) *z* = *0*

by (*simp add*: *winding_number_valid_path*)

lemma *winding_number_subpath_trivial* [*simp*]: $z \neq g\ x \Longrightarrow$ *winding_number* (*subpath x x g*) $z = 0$
by (*simp add*: *path_image_subpath winding_number_valid_path*)

lemma *winding_number_join*:
assumes $\gamma 1$: *path* $\gamma 1\ z \notin$ *path_image* $\gamma 1$
and $\gamma 2$: *path* $\gamma 2\ z \notin$ *path_image* $\gamma 2$
and *pathfinish* $\gamma 1 =$ *pathstart* $\gamma 2$
shows *winding_number*($\gamma 1$ +++ $\gamma 2$) $z =$ *winding_number* $\gamma 1\ z +$ *winding_number* $\gamma 2\ z$
proof (*rule winding_number_unique*)
show $\exists p$. *winding_number_prop* ($\gamma 1$ +++ $\gamma 2$) $z\ e\ p$
(*winding_number* $\gamma 1\ z +$ *winding_number* $\gamma 2\ z$) **if** $e > 0$ **for** e
proof −
obtain *p1* **where** *winding_number_prop* $\gamma 1\ z\ e$ *p1* (*winding_number* $\gamma 1\ z$)
using ‹$0 < e$› $\gamma 1$ *winding_number* **by** *blast*
moreover
obtain *p2* **where** *winding_number_prop* $\gamma 2\ z\ e$ *p2* (*winding_number* $\gamma 2\ z$)
using ‹$0 < e$› $\gamma 2$ *winding_number* **by** *blast*
ultimately
have *winding_number_prop* ($\gamma 1$+++$\gamma 2$) $z\ e$ (*p1*+++*p2*) (*winding_number* $\gamma 1\ z +$ *winding_number* $\gamma 2\ z$)
using *assms*
apply (*simp add*: *winding_number_prop_def not_in_path_image_join contour_integrable_inversediff algebra_simps*)
apply (*auto simp*: *joinpaths_def*)
done
then show *?thesis*
by *blast*
qed
qed (*use assms* **in** ‹*auto simp*: *not_in_path_image_join*›)

lemma *winding_number_reversepath*:
assumes *path* $\gamma\ z \notin$ *path_image* γ
shows *winding_number*(*reversepath* γ) $z = -$ (*winding_number* $\gamma\ z$)
proof (*rule winding_number_unique*)
show $\exists p$. *winding_number_prop* (*reversepath* γ) $z\ e\ p$ ($-$ *winding_number* $\gamma\ z$)
if $e > 0$ **for** e
proof −
obtain p **where** *winding_number_prop* $\gamma\ z\ e\ p$ (*winding_number* $\gamma\ z$)
using ‹$0 < e$› *assms winding_number* **by** *blast*
then have *winding_number_prop* (*reversepath* γ) $z\ e$ (*reversepath* p) ($-$ *winding_number* $\gamma\ z$)
using *assms* **unfolding** *winding_number_prop_def*
apply (*simp add*: *contour_integral_reversepath contour_integrable_inversediff valid_path_imp_reverse*)
apply (*auto simp*: *reversepath_def*)

```
    done
  then show ?thesis
    by blast
 qed
qed (use assms in auto)

lemma winding_number_shiftpath:
 assumes γ: path γ z ∉ path_image γ
    and pathfinish γ = pathstart γ a ∈ {0..1}
  shows winding_number(shiftpath a γ) z = winding_number γ z
proof (rule winding_number_unique_loop)
 show ∃p. valid_path p ∧ z ∉ path_image p ∧ pathfinish p = pathstart p ∧
          (∀t∈{0..1}. cmod (shiftpath a γ t − p t) < e) ∧
          contour_integral p (λw. 1 / (w − z)) =
          2 ∗ pi ∗ i ∗ winding_number γ z
  if e > 0 for e
 proof −
  obtain p where winding_number_prop γ z e p (winding_number γ z)
    using ‹0 < e› assms winding_number by blast
  then show ?thesis
    apply (rule_tac x=shiftpath a p in exI)
    using assms that
    apply (auto simp: winding_number_prop_def path_image_shiftpath pathfin-
ish_shiftpath pathstart_shiftpath contour_integral_shiftpath)
    apply (simp add: shiftpath_def)
    done
 qed
qed (use assms in ‹auto simp: path_shiftpath path_image_shiftpath pathfinish_shiftpath
pathstart_shiftpath›)

lemma winding_number_split_linepath:
 assumes c ∈ closed_segment a b z ∉ closed_segment a b
  shows winding_number(linepath a b) z = winding_number(linepath a c) z +
winding_number(linepath c b) z
proof −
 have z ∉ closed_segment a c z ∉ closed_segment c b
  using assms by (meson convex_contains_segment convex_segment ends_in_segment
subsetCE)+
 then show ?thesis
  using assms
   by (simp add: winding_number_valid_path contour_integral_split_linepath
[symmetric] continuous_on_inversediff field_simps)
qed

lemma winding_number_cong:
  (⋀t. ⟦0 ≤ t; t ≤ 1⟧ ⟹ p t = q t) ⟹ winding_number p z = winding_number
q z
  by (simp add: winding_number_def winding_number_prop_def pathstart_def
pathfinish_def)
```

lemma *winding_number_constI*:
assumes $c{\neq}z$ **and** g: $\bigwedge t.\ [\![0{\leq}t;\ t{\leq}1]\!] \Longrightarrow g\ t = c$
shows *winding_number g z = 0*
proof −
have *winding_number g z = winding_number (linepath c c) z*
using *g winding_number_cong* **by** *fastforce*
moreover have *winding_number (linepath c c) z = 0*
using ‹$c{\neq}z$› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *winding_number_offset*: *winding_number p z = winding_number* ($\lambda w.$ *p w − z) 0*
unfolding *winding_number_def*
proof (*intro ext arg_cong* [**where** *f = Eps*] *arg_cong* [**where** *f = All*] *imp_cong refl*, *safe*)
fix *n e g*
assume *0 < e* **and** *g*: *winding_number_prop p z e g n*
then show ∃ *r. winding_number_prop* ($\lambda w.$ *p w − z) 0 e r n*
by (*rule_tac x*=$\lambda t.$ *g t − z* **in** *exI*)
(*force simp*: *winding_number_prop_def contour_integral_integral valid_path_def path_defs*
vector_derivative_def has_vector_derivative_diff_const piecewise_C1_differentiable_diff C1_differentiable_imp_piecewise)
next
fix *n e g*
assume *0 < e* **and** *g*: *winding_number_prop* ($\lambda w.$ *p w − z) 0 e g n*
then have *winding_number_prop p z e* ($\lambda t.$ *g t + z) n*
apply (*simp add*: *winding_number_prop_def contour_integral_integral valid_path_def path_defs*
piecewise_C1_differentiable_add vector_derivative_def has_vector_derivative_add_const C1_differentiable_imp_piecewise)
apply (*force simp*: *algebra_simps*)
done
then show ∃ *r. winding_number_prop p z e r n*
by *metis*
qed

lemma *winding_number_negatepath*:
assumes γ: *valid_path* γ **and** *0*: *0* $\notin$ *path_image* γ
shows *winding_number*(*uminus* ∘ γ) *0 = winding_number* γ *0*
proof −
have (/) *1 contour_integrable_on* γ
using *0* γ *contour_integrable_inversediff* **by** *fastforce*
then have (($\lambda z.$ *1/z*) *has_contour_integral contour_integral* γ ((/) *1*)) γ
by (*rule has_contour_integral_integral*)
then have (($\lambda z.$ *1 / − z*) *has_contour_integral − contour_integral* γ ((/) *1*)) γ

using *has_contour_integral_neg* **by** *auto*
then have *contour_integral* (*uminus* ∘ γ) ((/) 1) =
contour_integral γ ((/) 1)
using γ **by** (*simp add*: *contour_integral_unique has_contour_integral_negatepath*)
then show *?thesis*
using *assms* **by** (*simp add*: *winding_number_valid_path valid_path_negatepath image_def path_defs*)
qed

A combined theorem deducing several things piecewise.

lemma *winding_number_join_pos_combined*:
⟦*valid_path γ1*; *z* ∉ *path_image γ1*; 0 < *Re*(*winding_number γ1 z*);
valid_path γ2; *z* ∉ *path_image γ2*; 0 < *Re*(*winding_number γ2 z*); *pathfinish γ1* = *pathstart γ2*⟧
⟹ *valid_path*(*γ1* +++ *γ2*) ∧ *z* ∉ *path_image*(*γ1* +++ *γ2*) ∧ 0 < *Re*(*winding_number*(*γ1* +++ *γ2*) *z*)
by (*simp add*: *valid_path_join path_image_join winding_number_join valid_path_imp_path*)

3.1.1 Useful sufficient conditions for the winding number to be positive

lemma *Re_winding_number*:
⟦*valid_path γ*; *z* ∉ *path_image γ*⟧
⟹ *Re*(*winding_number γ z*) = *Im*(*contour_integral γ* (λ*w*. 1/(*w* − *z*))) / (2∗*pi*)
by (*simp add*: *winding_number_valid_path field_simps Re_divide power2_eq_square*)

lemma *winding_number_pos_le*:
assumes *γ*: *valid_path γ z* ∉ *path_image γ*
and *ge*: ⋀*x*. ⟦0 < *x*; *x* < 1⟧ ⟹ 0 ≤ *Im* (*vector_derivative γ* (*at x*) ∗ *cnj*(*γ x* − *z*))
shows 0 ≤ *Re*(*winding_number γ z*)
proof −
have *ge0*: 0 ≤ *Im* (*vector_derivative γ* (*at x*) / (*γ x* − *z*)) **if** *x*: 0 < *x x* < 1 **for** *x*
using *ge* **by** (*simp add*: *Complex.Im_divide algebra_simps x*)
let *?vd* = λ*x*. 1 / (*γ x* − *z*) ∗ *vector_derivative γ* (*at x*)
let *?int* = λ*z*. *contour_integral γ* (λ*w*. 1 / (*w* − *z*))
have 0 ≤ *Im* (*?int z*)
proof (*rule has_integral_component_nonneg* [*of* i, *simplified*])
show ⋀*x*. *x* ∈ *cbox 0 1* ⟹ 0 ≤ *Im* (*if* 0 < *x* ∧ *x* < 1 *then ?vd x else* 0)
by (*force simp*: *ge0*)
have ((λ*a*. 1 / (*a* − *z*)) *has_contour_integral contour_integral γ* (λ*w*. 1 / (*w* − *z*))) *γ*
using *γ* **by** (*simp flip*: *add*: *contour_integrable_inversediff has_contour_integral_integral*)
then have *hi*: (*?vd has_integral ?int z*) (*cbox 0 1*)
using *has_contour_integral* **by** *auto*
show ((λ*x*. *if* 0 < *x* ∧ *x* < 1 *then ?vd x else* 0) *has_integral ?int z*) (*cbox 0 1*)
by (*rule has_integral_spike_interior* [*OF hi*]) *simp*

```
  qed
  then show ?thesis
    by (simp add: Re_winding_number [OF γ] field_simps)
qed

lemma winding_number_pos_lt_lemma:
  assumes γ: valid_path γ z ∉ path_image γ
      and e: 0 < e
      and ge: ⋀x. ⟦0 < x; x < 1⟧ ⟹ e ≤ Im (vector_derivative γ (at x) / (γ x
− z))
    shows 0 < Re(winding_number γ z)
proof −
  let ?vd = λx. 1 / (γ x − z) ∗ vector_derivative γ (at x)
  let ?int = λz. contour_integral γ (λw. 1 / (w − z))
  have e ≤ Im (contour_integral γ (λw. 1 / (w − z)))
  proof (rule has_integral_component_le [of i λx. i∗e i∗e {0..1}, simplified])
    have ((λa. 1 / (a − z)) has_contour_integral contour_integral γ (λw. 1 / (w
− z))) γ
      thm has_integral_component_le [of i λx. i∗e i∗e {0..1}, simplified]
    using γ by (simp add: contour_integrable_inversediff has_contour_integral_integral)
    then have hi: (?vd has_integral ?int z) (cbox 0 1)
      using has_contour_integral by auto
    show ((λx. if 0 < x ∧ x < 1 then ?vd x else i ∗ e) has_integral ?int z) {0..1}
      by (rule has_integral_spike_interior [OF hi, simplified box_real]) (use e in
simp)
    show ⋀x. 0 ≤ x ∧ x ≤ 1 ⟹ e ≤ Im (if 0 < x ∧ x < 1 then ?vd x else i ∗ e)
      by (simp add: ge)
  qed (use has_integral_const_real [of _ 0 1] in auto)
  with e show ?thesis
    by (simp add: Re_winding_number [OF γ] field_simps)
qed

lemma winding_number_pos_lt:
  assumes γ: valid_path γ z ∉ path_image γ
      and e: 0 < e
      and ge: ⋀x. ⟦0 < x; x < 1⟧ ⟹ e ≤ Im (vector_derivative γ (at x) ∗ cnj(γ
x − z))
    shows 0 < Re (winding_number γ z)
proof −
  have bm: bounded ((λw. w − z) ‘ (path_image γ))
   using bounded_translation [of _ −z] γ by (simp add: bounded_valid_path_image)
  then obtain B where B: B > 0 and Bno: ⋀x. x ∈ (λw. w − z) ‘ (path_image
γ) ⟹ norm x ≤ B
    using bounded_pos [THEN iffD1, OF bm] by blast
  { fix x::real  assume x: 0 < x x < 1
    then have B2: cmod (γ x − z)^2 ≤ B^2 using Bno [of γ x − z]
      by (simp add: path_image_def power2_eq_square mult_mono')
    with x have γ x ≠ z using γ
      using path_image_def by fastforce
```

then have e / $B^2 \leq e$ / $(cmod\ (\gamma\ x - z))^2$
using B $B2$ e **by** (*auto simp*: *divide_left_mono*)
also have ... $\leq$ $Im\ (vector\_derivative\ \gamma\ (at\ x) * cnj\ (\gamma\ x - z))$ / $(cmod\ (\gamma\ x - z))^2$
using *ge* [*OF* x] **by** (*auto simp*: *divide_right_mono*)
finally have e / $B^2 \leq Im\ (vector\_derivative\ \gamma\ (at\ x) * cnj\ (\gamma\ x - z))$ / $(cmod\ (\gamma\ x - z))^2$ **.**
then have e / $B^2 \leq Im\ (vector\_derivative\ \gamma\ (at\ x)$ / $(\gamma\ x - z))$
by (*simp add*: *complex_div_cnj* [*of* _ $\gamma\ x - z$ **for** x] *del*: *complex_cnj_diff times_complex.sel*)
} note $* = this$
show *?thesis*
using e B **by** (*simp add*: $*$ *winding_number_pos_lt_lemma* [*OF* γ, *of* $e/B\hat{}2$])
qed

3.2 The winding number is an integer

Proof from the book Complex Analysis by Lars V. Ahlfors, Chapter 4, section 2.1, Also on page 134 of Serge Lang's book with the name title, etc.

lemma *exp_fg*:
fixes z::*complex*
assumes g: (g *has_vector_derivative* g') (*at* x *within* s)
and f: (f *has_vector_derivative* (g' / ($g\ x - z$))) (*at* x *within* s)
and z: $g\ x \neq z$
shows ($(\lambda x.\ exp(-f\ x) * (g\ x - z))$ *has_vector_derivative* 0) (*at* x *within* s)
proof −
have $*$: ($exp \circ (\lambda x.\ (-\ f\ x))$ *has_vector_derivative* $-\ (g'$ / $(g\ x - z)) * exp\ (-\ f\ x))$ (*at* x *within* s)
using *assms* **unfolding** *has_vector_derivative_def scaleR_conv_of_real*
by (*auto intro*!: *derivative_eq_intros*)
show *?thesis*
using z **by** (*auto intro*!: *derivative_eq_intros* $*$ [*unfolded o_def*] g)
qed

lemma *winding_number_exp_integral*:
fixes z::*complex*
assumes γ: γ *piecewise_C1_differentiable_on* $\{a..b\}$
and *ab*: $a \leq b$
and z: $z \notin \gamma$ ' $\{a..b\}$
shows ($\lambda x.$ *vector_derivative* γ (*at* x) / $(\gamma\ x - z))$ *integrable_on* $\{a..b\}$
(**is** *?thesis1*)
exp $(-\ (integral\ \{a..b\}\ (\lambda x.\ vector\_derivative\ \gamma\ (at\ x)$ / $(\gamma\ x - z)))) * (\gamma\ b - z) = \gamma\ a - z$
(**is** *?thesis2*)
proof −
let *?D*$\gamma = \lambda x.\ vector\_derivative\ \gamma\ (at\ x)$
have [*simp*]: $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow \gamma\ x \neq z$
using z **by** *force*
have *con_g*: *continuous_on* $\{a..b\}$ γ

using γ **by** (*simp add*: *piecewise_C1_differentiable_on_def*)
obtain k **where** *fink*: *finite* k **and** *g_C1_diff*: γ *C1_differentiable_on* ($\{a..b\} - k$)
using γ **by** (*force simp*: *piecewise_C1_differentiable_on_def*)
have $\circ$: *open* ($\{a<..<b\} - k$)
using ‹*finite k*› **by** (*simp add*: *finite_imp_closed open_Diff*)
moreover have $\{a<..<b\} - k \subseteq \{a..b\} - k$
by *force*
ultimately have *g_diff_at*: $\bigwedge x.\ [\![x \notin k;\ x \in \{a<..<b\}]\!] \Longrightarrow \gamma$ *differentiable at* x
by (*metis Diff_iff differentiable_on_subset C1_diff_imp_diff* [*OF g_C1_diff*] *differentiable_on_def at_within_open*)
{ fix w
assume $w \neq z$
have *continuous_on* (*ball* w (*cmod* $(w - z)$)) ($\lambda w.\ 1\ /\ (w - z)$)
by (*auto simp*: *dist_norm intro*!: *continuous_intros*)
moreover have $\bigwedge x.$ *cmod* $(w - x) <$ *cmod* $(w - z) \Longrightarrow \exists f'.$ (($\lambda w.\ 1\ /\ (w - z)$) *has_field_derivative* f') (*at* x)
by (*auto simp*: *intro*!: *derivative_eq_intros*)
ultimately have $\exists h.\ \forall y.$ *norm*$(y - w) <$ *norm*$(w - z) \longrightarrow$ (h *has_field_derivative* $1/(y - z)$) (*at* y)
using *holomorphic_convex_primitive* [*of ball* w (*norm*$(w - z)$) {} $\lambda w.\ 1/(w - z)$]
by (*force simp*: *field_differentiable_def Ball_def dist_norm at_within_open_NO_MATCH norm_minus_commute*)
}
then obtain h **where** h: $\bigwedge w\ y.\ w \neq z \Longrightarrow$ *norm*$(y - w) <$ *norm*$(w - z) \Longrightarrow$ ($h\ w$ *has_field_derivative* $1/(y - z)$) (*at* y)
by *meson*
have *exy*: $\exists y.$ (($\lambda x.$ *inverse* $(\gamma\ x - z) *$ *?Dγ* x) *has_integral* y) $\{a..b\}$
unfolding *integrable_on_def* [*symmetric*]
proof (*rule contour_integral_local_primitive_any* [*OF piecewise_C1_imp_differentiable* [*OF* γ]])
show $\exists d\ h.\ 0 < d \wedge$
$(\forall y.$ *cmod* $(y - w) < d \longrightarrow$ (h *has_field_derivative inverse* $(y - z)$)(*at* y *within* $- \{z\}$))
if $w \in - \{z\}$ **for** w
using *that inverse_eq_divide has_field_derivative_at_within h*
by (*metis Compl_insert DiffD2 insertCI right_minus_eq zero_less_norm_iff*)
qed *simp*
have *vg_int*: ($\lambda x.$ *?Dγ* $x\ /\ (\gamma\ x - z)$) *integrable_on* $\{a..b\}$
unfolding *box_real* [*symmetric*] *divide_inverse_commute*
by (*auto intro*!: *exy integrable_subinterval simp add*: *integrable_on_def ab*)
with *ab* **show** *?thesis1*
by (*simp add*: *divide_inverse_commute integral_def integrable_on_def*)
{ fix t
assume t: $t \in \{a..b\}$
have *cball*: *continuous_on* (*ball* (γ t) (*dist* (γ t) z)) ($\lambda x.$ *inverse* $(x - z)$)
using z **by** (*auto intro*!: *continuous_intros simp*: *dist_norm*)

have *icd*: $\bigwedge x.\ cmod\ (\gamma\ t - x) < cmod\ (\gamma\ t - z) \Longrightarrow (\lambda w.\ inverse\ (w - z))$ *field_differentiable at x*
unfolding *field_differentiable_def* **by** (*force simp*: *intro*!: *derivative_eq_intros*)
obtain *h* **where** *h*: $\bigwedge x.\ cmod\ (\gamma\ t - x) < cmod\ (\gamma\ t - z) \Longrightarrow$
(*h has_field_derivative inverse* $(x - z)$) (*at x within* $\{y.\ cmod\ (\gamma\ t - y) < cmod\ (\gamma\ t - z)\}$)
using *holomorphic_convex_primitive* [**where** $f = \lambda w.\ inverse(w - z)$, *OF convex_ball finite.emptyI cball icd*]
by *simp* (*auto simp*: *ball_def dist_norm that*)
have *exp* $(-\ (integral\ \{a..t\}\ (\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z)))) * (\gamma\ t - z) = \gamma\ a - z$
proof (*rule has_derivative_zero_unique_strong_interval* [*of* $\{a,b\} \cup k\ a\ b$])
show *continuous_on* $\{a..b\}$ $(\lambda b.\ exp\ (-\ integral\ \{a..b\}\ (\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))) * (\gamma\ b - z))$
by (*auto intro*!: *continuous_intros con_g indefinite_integral_continuous_1* [*OF vg_int*])
show $((\lambda b.\ exp\ (-\ integral\ \{a..b\}\ (\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))) * (\gamma\ b - z))$ *has_derivative* $(\lambda h.\ 0))$
(*at x within* $\{a..b\}$)
if $x \in \{a..b\} - (\{a,\ b\} \cup k)$ **for** *x*
proof −
have *x*: $x \notin k\ a < x\ x < b$
using *that* **by** *auto*
then have $x \in interior\ (\{a..b\} - k)$
using *open_subset_interior* [*OF* ○] **by** *fastforce*
then have *con*: *isCont* $?D\gamma\ x$
using *g_C1_diff x* **by** (*auto simp*: *C1_differentiable_on_eq intro*: *continuous_on_interior*)
then have *con_vd*: *continuous* (*at x within* $\{a..b\}$) $(\lambda x.\ ?D\gamma\ x)$
by (*rule continuous_at_imp_continuous_within*)
have *gdx*: γ *differentiable at x*
using *x* **by** (*simp add*: *g_diff_at*)
then obtain *d* **where** *d*: (γ *has_derivative* $(\lambda x.\ x *_R d)$) (*at x*)
by (*auto simp add*: *differentiable_iff_scaleR*)
show $((\lambda c.\ exp\ (-\ integral\ \{a..c\}\ (\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))) * (\gamma\ c - z))$ *has_derivative* $(\lambda h.\ 0))$
(*at x within* $\{a..b\}$)
proof (*rule exp_fg* [*unfolded has_vector_derivative_def*, *simplified*])
show (γ *has_derivative* $(\lambda c.\ c *_R d)$) (*at x within* $\{a..b\}$)
using *d* **by** (*blast intro*: *has_derivative_at_withinI*)
have $((\lambda x.\ integral\ \{a..x\}\ (\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z)))$ *has_vector_derivative* $d\ /\ (\gamma\ x - z))$
(*at x within* $\{a..b\}$)
proof (*rule has_vector_derivative_eq_rhs* [*OF integral_has_vector_derivative_continuous_at* [**where** $S = \{\}$, *simplified*]])
show *continuous* (*at x within* $\{a..b\}$) $(\lambda x.\ vector\_derivative\ \gamma\ (at\ x)\ /\ (\gamma\ x - z))$
using *continuous_at_imp_continuous_at_within differentiable_imp_continuous_within gdx x*
by (*intro con_vd continuous_intros*) (*force*+)

```
          show vector_derivative γ (at x) / (γ x − z) = d / (γ x − z)
            using d vector_derivative_at
            by (simp add: vector_derivative_at has_vector_derivative_def)
        qed (use x vg_int in auto)
        then show ((λx. integral {a..x} (λx. ?Dγ x / (γ x − z))) has_derivative
(λc. c *R (d / (γ x − z))))
            (at x within {a..b})
          by (auto simp: has_vector_derivative_def)
      qed (use x in auto)
    qed
  qed (use fink t in auto)
 }
 with ab show ?thesis2
  by (simp add: divide_inverse_commute integral_def)
qed

lemma winding_number_exp_2pi:
   ⟦path p; z ∉ path_image p⟧
    ⟹ pathfinish p − z = exp (2 * pi * i * winding_number p z) * (pathstart p
− z)
using winding_number [of p z 1] unfolding valid_path_def path_image_def path-
start_def pathfinish_def winding_number_prop_def
 by (force dest: winding_number_exp_integral(2) [of _ 0 1 z] simp: field_simps
contour_integral_integral exp_minus)

lemma integer_winding_number_eq:
 assumes γ: path γ and z: z ∉ path_image γ
 shows winding_number γ z ∈ ℤ ⟷ pathfinish γ = pathstart γ
proof −
 obtain p where p: valid_path p z ∉ path_image p
                  pathstart p = pathstart γ pathfinish p = pathfinish γ
          and eq: contour_integral p (λw. 1 / (w − z)) = complex_of_real (2 *
pi) * i * winding_number γ z
  using winding_number [OF assms, of 1] unfolding winding_number_prop_def
by auto
 then have wneq: winding_number γ z = winding_number p z
    using eq winding_number_valid_path by force
 have iff: (winding_number γ z ∈ ℤ) ⟷ (exp (contour_integral p (λw. 1 / (w
− z))) = 1)
   using eq by (simp add: exp_eq_1 complex_is_Int_iff)
 have γ 0 ≠ z
   by (metis pathstart_def pathstart_in_path_image z)
 then have exp (contour_integral p (λw. 1 / (w − z))) = (γ 1 − z) / (γ 0 − z)
   using p winding_number_exp_integral(2) [of p 0 1 z]
   by (simp add: valid_path_def path_defs contour_integral_integral exp_minus
field_split_simps)
 then have winding_number p z ∈ ℤ ⟷ pathfinish p = pathstart p
   using p wneq iff by (auto simp: path_defs)
 then show ?thesis using p eq
```

by (*auto simp*: *winding_number_valid_path*)
qed

theorem *integer_winding_number*:
⟦*path* γ; *pathfinish* γ = *pathstart* γ; *z* ∉ *path_image* γ⟧ ⟹ *winding_number* γ *z* ∈ ℤ
by (*metis integer_winding_number_eq*)

If the winding number's magnitude is at least one, then the path must contain points in every direction.*) We can thus bound the winding number of a path that doesn't intersect a given ray.

lemma *winding_number_pos_meets*:
fixes *z*::*complex*
assumes γ: *valid_path* γ **and** *z*: *z* ∉ *path_image* γ **and** *1*: *Re* (*winding_number* γ *z*) ≥ *1*
and *w*: *w* ≠ *z*
shows ∃ *a*::*real*. *0* < *a* ∧ *z* + *of_real a* ∗ (*w* − *z*) ∈ *path_image* γ
proof −
have [*simp*]: ⋀*x*. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ γ *x* ≠ *z*
using *z* **by** (*auto simp*: *path_image_def*)
have [*simp*]: *z* ∉ γ ‘ {*0..1*}
using *path_image_def z* **by** *auto*
have *gpd*: γ *piecewise_C1_differentiable_on* {*0..1*}
using γ *valid_path_def* **by** *blast*
define *r* **where** *r* = (*w* − *z*) / (γ *0* − *z*)
have [*simp*]: *r* ≠ *0*
using *w z* **by** (*auto simp*: *r_def*)
have *cont*: *continuous_on* {*0..1*}
(λ*x*. *Im* (*integral* {*0..x*} (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*))))
by (*intro continuous_intros indefinite_integral_continuous_1 winding_number_exp_integral* [*OF gpd*]; *simp*)
have *Arg2pi r* ≤ *2*∗*pi*
by (*simp add*: *Arg2pi less_eq_real_def*)
also have ... ≤ *Im* (*integral* {*0..1*} (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*)))
using *1*
by (*simp add*: *winding_number_valid_path* [*OF* γ *z*] *contour_integral_integral Complex.Re_divide field_simps power2_eq_square*)
finally have *Arg2pi r* ≤ *Im* (*integral* {*0..1*} (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*))) **.**
then have ∃ *t*. *t* ∈ {*0..1*} ∧ *Im*(*integral* {*0..t*} (λ*x*. *vector_derivative* γ (*at x*)/(γ *x* − *z*))) = *Arg2pi r*
by (*simp add*: *Arg2pi_ge_0 cont IVT'*)
then obtain *t* **where** *t*: *t* ∈ {*0..1*}
and *eqArg*: *Im* (*integral* {*0..t*} (λ*x*. *vector_derivative* γ (*at x*)/(γ *x* − *z*))) = *Arg2pi r*
by *blast*
define *i* **where** *i* = *integral* {*0..t*} (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*))
have *gpdt*: γ *piecewise_C1_differentiable_on* {*0..t*}

```
  by (metis atLeastAtMost_iff atLeastatMost_subset_iff order_refl piecewise_C1_differentiable_on_subset
gpd t)
 have exp (− i) ∗ (γ t − z) = γ 0 − z
  unfolding i_def
 proof (rule winding_number_exp_integral [OF gpdt])
  show z ∉ γ ‘ {0..t}
   using t z unfolding path_image_def by force
 qed (use t in auto)
 then have ∗: γ t − z = exp i ∗ (γ 0 − z)
  by (simp add: exp_minus field_simps)
 then have (w − z) = r ∗ (γ 0 − z)
  by (simp add: r_def)
 moreover have z + exp (Re i) ∗ (exp (i ∗ Im i) ∗ (γ 0 − z)) = γ t
  using ∗ by (simp add: exp_eq_polar field_simps)
 moreover have Arg2pi r = Im i
  using eqArg by (simp add: i_def)
 ultimately have z + complex_of_real (exp (Re i)) ∗ (w − z) / complex_of_real
(cmod r) = γ t
  using Complex_Transcendental.Arg2pi_eq [of r] ‹r ≠ 0›
  by (metis mult.left_commute nonzero_mult_div_cancel_left norm_eq_zero
of_real_0 of_real_eq_iff times_divide_eq_left)
 with t show ?thesis
  by (rule_tac x=exp(Re i) / norm r in exI) (auto simp: path_image_def)
qed

lemma winding_number_big_meets:
 fixes z::complex
 assumes γ: valid_path γ and z: z ∉ path_image γ and |Re (winding_number
γ z)| ≥ 1
   and w: w ≠ z
 shows ∃ a::real. 0 < a ∧ z + of_real a ∗ (w − z) ∈ path_image γ
proof −
 { assume Re (winding_number γ z) ≤ − 1
  then have Re (winding_number (reversepath γ) z) ≥ 1
   by (simp add: γ valid_path_imp_path winding_number_reversepath z)
  moreover have valid_path (reversepath γ)
   using γ valid_path_imp_reverse by auto
  moreover have z ∉ path_image (reversepath γ)
   by (simp add: z)
  ultimately have ∃ a::real. 0 < a ∧ z + of_real a ∗ (w − z) ∈ path_image
(reversepath γ)
   using winding_number_pos_meets w by blast
  then have ?thesis
   by simp
 }
 then show ?thesis
  using assms
  by (simp add: abs_if winding_number_pos_meets split: if_split_asm)
qed
```

lemma *winding_number_less_1*:
fixes *z*::*complex*
shows
⟦*valid_path* γ; $z \notin$ *path_image* γ; $w \neq z$;
$\bigwedge$*a*::*real*. $0 < a \Longrightarrow z + of\_real\ a * (w - z) \notin$ *path_image* γ⟧
$\Longrightarrow$ *Re*(*winding_number* γ *z*) < *1*
by (*auto simp*: *not_less dest*: *winding_number_big_meets*)

One way of proving that WN=1 for a loop.

lemma *winding_number_eq_1*:
fixes *z*::*complex*
assumes γ: *valid_path* γ **and** *z*: $z \notin$ *path_image* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *0*: *0* < *Re*(*winding_number* γ *z*) **and** *2*: *Re*(*winding_number* γ *z*) < *2*
shows *winding_number* γ *z* = *1*
proof −
have *winding_number* γ *z* ∈ *Ints*
by (*simp add*: γ *integer_winding_number loop valid_path_imp_path z*)
then show *?thesis*
using *0 2* **by** (*auto simp*: *Ints_def*)
qed

3.3 Continuity of winding number and invariance on connected sets

theorem *continuous_at_winding_number*:
fixes *z*::*complex*
assumes γ: *path* γ **and** *z*: $z \notin$ *path_image* γ
shows *continuous* (*at z*) (*winding_number* γ)
proof −
obtain *e* **where** *e>0* **and** *cbg*: *cball z e* ⊆ − *path_image* γ
using *open_contains_cball* [*of* − *path_image* γ] *z*
by (*force simp*: *closed_def* [*symmetric*] *closed_path_image* [*OF* γ])
then have *ppag*: *path_image* γ ⊆ − *cball z* (*e*/*2*)
by (*force simp*: *cball_def dist_norm*)
have *oc*: *open* (− *cball z* (*e*/*2*))
by (*simp add*: *closed_def* [*symmetric*])
obtain *d* **where** *d>0* **and** *pi_eq*:
$\bigwedge$*h1 h2*. ⟦*valid_path h1*; *valid_path h2*;
(∀ *t*∈{*0*..*1*}. *cmod* (*h1 t* − γ *t*) < *d* ∧ *cmod* (*h2 t* − γ *t*) < *d*);
pathstart h2 = *pathstart h1*; *pathfinish h2* = *pathfinish h1*⟧
$\Longrightarrow$
path_image h1 ⊆ − *cball z* (*e*/*2*) ∧
path_image h2 ⊆ − *cball z* (*e*/*2*) ∧
(∀ *f*. *f holomorphic_on* − *cball z* (*e*/*2*) ⟶ *contour_integral h2 f* = *contour_integral h1 f*)
using *contour_integral_nearby_ends* [*OF oc* γ *ppag*] **by** *metis*
obtain *p* **where** *valid_path p z* ∉ *path_image p*

```
          and p: pathstart p = pathstart γ pathfinish p = pathfinish γ
          and pg: ⋀t. t∈{0..1} ⟹ cmod (γ t − p t) < min d e/2
          and pi: contour_integral p (λx. 1 / (x − z)) = complex_of_real (2 *
pi) * i * winding_number γ z
    using winding_number [OF γ z, of min d e/2] ‹d>0› ‹e>0› by (auto simp:
winding_number_prop_def)
  { fix w
    assume d2: cmod (w − z) < d/2 and e2: cmod (w − z) < e/2
    have wnotp: w ∉ path_image p
    proof (clarsimp simp add: path_image_def)
      show False if w: w = p x and 0 ≤ x x ≤ 1 for x
      proof −
        have cmod (γ x − p x) < min d e/2
          using pg that by auto
        then have cmod (z − γ x) < e
       by (metis e2 less_divide_eq_numeral1(1) min_less_iff_conj norm_minus_commute
norm_triangle_half_l w)
        then show ?thesis
          using cbg that by (auto simp add: path_image_def cball_def dist_norm
less_eq_real_def)
      qed
    qed
    have wnotg: w ∉ path_image γ
      using cbg e2 ‹e>0› by (force simp: dist_norm norm_minus_commute)
    { fix k::real
      assume k: k>0
      then obtain q where q: valid_path q w ∉ path_image q
                           pathstart q = pathstart γ ∧ pathfinish q = pathfinish γ
                  and qg: ⋀t. t ∈ {0..1} ⟹ cmod (γ t − q t) < min k (min d e)
/ 2
                  and qi: contour_integral q (λu. 1 / (u − w)) = complex_of_real
(2 * pi) * i * winding_number γ w
        using winding_number [OF γ wnotg, of min k (min d e) / 2] ‹d>0› ‹e>0›
k
        by (force simp: min_divide_distrib_right winding_number_prop_def)
      moreover have ⋀t. t ∈ {0..1} ⟹ cmod (q t − γ t) < d ∧ cmod (p t − γ
t) < d
        using pg qg ‹0 < d› by (fastforce simp add: norm_minus_commute)
      moreover have (λu. 1 / (u−w)) holomorphic_on − cball z (e/2)
        using e2 by (auto simp: dist_norm norm_minus_commute intro!: holo-
morphic_intros)
      ultimately have contour_integral p (λu. 1 / (u − w)) = contour_integral q
(λu. 1 / (u − w))
        by (metis p ‹valid_path p› pi_eq)
      then have contour_integral p (λx. 1 / (x − w)) = complex_of_real (2 * pi)
* i * winding_number γ w
        by (simp add: pi qi)
    } note pip = this
    have path p
```

```
    by (simp add: ‹valid_path p› valid_path_imp_path)
  moreover have ⋀e. e > 0 ⟹ winding_number_prop p w e p (winding_number
γ w)
    by (simp add: ‹valid_path p› pip winding_number_prop_def wnotp)
  ultimately have winding_number p w = winding_number γ w
    using winding_number_unique wnotp by blast
 } note wnwn = this
 obtain pe where pe>0 and cbp: cball z (3 / 4 * pe) ⊆ - path_image p
  using ‹valid_path p› ‹z ∉ path_image p› open_contains_cball [of - path_image
p]
  by (force simp: closed_def [symmetric] closed_path_image [OF valid_path_imp_path])
 obtain L
  where L>0
    and L: ⋀f B. ⟦f holomorphic_on - cball z (3 / 4 * pe);
                  ∀z ∈ - cball z (3 / 4 * pe). cmod (f z) ≤ B⟧ ⟹
                  cmod (contour_integral p f) ≤ L * B
  using contour_integral_bound_exists [of - cball z (3/4*pe) p] cbp ‹valid_path
p› by blast
 { fix e::real and w::complex
  assume e: 0 < e and w: cmod (w - z) < pe/4 cmod (w - z) < e * pe^2 / (8
* L)
  then have [simp]: w ∉ path_image p
    using cbp p(2) ‹0 < pe›
   by (force simp: dist_norm norm_minus_commute path_image_def cball_def)
   have [simp]: contour_integral p (λx. 1/(x - w)) - contour_integral p (λx.
1/(x - z)) =
                 contour_integral p (λx. 1/(x - w) - 1/(x - z))
  by (simp add: ‹valid_path p› ‹z ∉ path_image p› contour_integrable_inversediff
contour_integral_diff)
  { fix x
    assume pe: 3/4 * pe < cmod (z - x)
    have cmod (w - x) < pe/4 + cmod (z - x)
      by (meson add_less_cancel_right norm_diff_triangle_le order_refl or-
der_trans_rules(21) w(1))
    then have wx: cmod (w - x) < 4/3 * cmod (z - x) using pe by simp
    have cmod (z - x) ≤ cmod (z - w) + cmod (w - x)
      using norm_diff_triangle_le by blast
    also have ... < pe/4 + cmod (w - x)
      using w by (simp add: norm_minus_commute)
    finally have pe/2 < cmod (w - x)
      using pe by auto
    then have pe_less: (pe/2)^2 < cmod (w - x) ^ 2
      by (simp add: ‹0 < pe› less_eq_real_def power_strict_mono)
    then have pe2: pe^2 < 4 * cmod (w - x) ^ 2
      by (simp add: power_divide)
    have 8 * L * cmod (w - z) < e * pe^2
      using w ‹L>0› by (simp add: field_simps)
    also have ... < e * 4 * cmod (w - x) * cmod (w - x)
      using pe2 ‹e>0› by (simp add: power2_eq_square)
```

also have ... $<$ $e * 4 * cmod\ (w - x) * (4/3 * cmod\ (z - x))$
using ‹$0 < pe$› *pe_less e less_eq_real_def wx* **by** *fastforce*
finally have $L * cmod\ (w - z) < 2/3 * e * cmod\ (w - x) * cmod\ (z - x)$
by *simp*
also have ... $\leq e * cmod\ (w - x) * cmod\ (z - x)$
using *e* **by** *simp*
finally have *Lwz*: $L * cmod\ (w - z) < e * cmod\ (w - x) * cmod\ (z - x)$ **.**
have $L * cmod\ (1 / (x - w) - 1 / (x - z)) \leq e$
proof (*cases* $x{=}z \vee x{=}w$)
case *True*
with *pe* ‹*pe*>0› *w* ‹*L*>0›
show *?thesis*
by (*force simp*: *norm_minus_commute*)
next
case *False*
with *wx w*(2) ‹*L*>0› *pe pe2 Lwz*
show *?thesis*
by (*auto simp*: *divide_simps mult_less_0_iff norm_minus_commute norm_divide norm_mult power2_eq_square*)
qed
} note *L_cmod_le* = *this*
let *?f* = $(\lambda x.\ 1 / (x - w) - 1 / (x - z))$
have *cmod* (*contour_integral p ?f*) $\leq L * (e * pe^2 / L / 4 * (inverse\ (pe / 2))^2)$
proof (*rule L*)
show *?f holomorphic_on* − *cball z* (*3 / 4 * pe*)
using ‹*pe*>0› *w*
by (*force simp*: *dist_norm norm_minus_commute intro*!: *holomorphic_intros*)
show $\forall u \in -$ *cball z* (*3 / 4 * pe*). *cmod* (*?f u*) $\leq e * pe^2 / L / 4 * (inverse\ (pe / 2))^2$
using ‹*pe*>0› *w* ‹*L*>0›
by (*auto simp*: *cball_def dist_norm field_simps L_cmod_le simp del*: *less_divide_eq_numeral1 le_divide_eq_numeral1*)
qed
also have ... $<$ *2*e*
using ‹*L*>0› *e* **by** (*force simp*: *field_simps*)
finally have *cmod* (*winding_number p w* − *winding_number p z*) $<$ *e*
using *pi_ge_two e*
by (*force simp*: *winding_number_valid_path* ‹*valid_path p*› ‹$z \notin$ *path_image p*› *field_simps norm_divide norm_mult intro*: *less_le_trans*)
} note *cmod_wn_diff* = *this*
have *isCont* (*winding_number p*) *z*
proof (*clarsimp simp add*: *continuous_at_eps_delta*)
fix *e*::*real* **assume** *e*>0
show $\exists d{>}0.\ \forall x'.\ dist\ x'\ z < d \longrightarrow dist\ (winding\_number\ p\ x')\ (winding\_number\ p\ z) < e$
using ‹*pe*>0› ‹*L*>0› ‹*e*>0›
by (*rule_tac x*=*min* (*pe*/4) (*e*/2*pe^2/*L*/4*) **in** *exI*) (*simp add*: *dist_norm cmod_wn_diff*)

```
  qed
  then show ?thesis
    apply (rule continuous_transform_within [where d = min d e/2])
    apply (auto simp: ‹d>0› ‹e>0› dist_norm wnwn)
    done
qed

corollary continuous_on_winding_number:
    path γ ⟹ continuous_on (− path_image γ) (λw. winding_number γ w)
 by (simp add: continuous_at_imp_continuous_on continuous_at_winding_number)
```

3.4 The winding number is constant on a connected region

```
lemma winding_number_constant:
  assumes γ: path γ and loop: pathfinish γ = pathstart γ and cs: connected S
and sg: S ∩ path_image γ = {}
  shows winding_number γ constant_on S
proof −
  have ∗: 1 ≤ cmod (winding_number γ y − winding_number γ z)
      if ne: winding_number γ y ≠ winding_number γ z and y ∈ S z ∈ S for y z
  proof −
    have winding_number γ y ∈ ℤ  winding_number γ z ∈ ℤ
      using that integer_winding_number [OF γ loop] sg ‹y ∈ S› by auto
    with ne show ?thesis
      by (auto simp: Ints_def simp flip: of_int_diff)
  qed
  have cont: continuous_on S (λw. winding_number γ w)
    using continuous_on_winding_number [OF γ] sg
    by (meson continuous_on_subset disjoint_eq_subset_Compl)
  show ?thesis
    using ∗ zero_less_one
    by (blast intro: continuous_discrete_range_constant [OF cs cont])
qed

lemma winding_number_eq:
    ⟦path γ; pathfinish γ = pathstart γ; w ∈ S; z ∈ S; connected S; S ∩ path_image
γ = {}⟧
      ⟹ winding_number γ w = winding_number γ z
  using winding_number_constant by (metis constant_on_def)

lemma open_winding_number_levelsets:
  assumes γ: path γ and loop: pathfinish γ = pathstart γ
  shows open {z. z ∉ path_image γ ∧ winding_number γ z = k}
proof (clarsimp simp: open_dist)
  fix z assume z: z ∉ path_image γ and k: k = winding_number γ z
  have open (− path_image γ)
    by (simp add: closed_path_image γ open_Compl)
  then obtain e where e>0 ball z e ⊆ − path_image γ
    using open_contains_ball [of − path_image γ] z by blast
```

then show $\exists e{>}0.\ \forall y.\ dist\ y\ z < e \longrightarrow y \notin path\_image\ \gamma \wedge winding\_number\ \gamma\ y = winding\_number\ \gamma\ z$
using ‹*e*>0› **by** (*force simp*: *norm_minus_commute dist_norm intro*: *winding_number_eq* [*OF assms*, **where** $S = ball\ z\ e$])
qed

3.5 Winding number is zero "outside" a curve

proposition *winding_number_zero_in_outside*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ **and** z: $z \in outside\ (path\_image\ \gamma)$
shows *winding_number* $\gamma\ z = 0$
proof −
obtain *B*::*real* **where** $0 < B$ **and** *B*: *path_image* $\gamma \subseteq ball\ 0\ B$
using *bounded_subset_ballD* [*OF bounded_path_image* [*OF* γ]] **by** *auto*
obtain *w*::*complex* **where** *w*: $w \notin ball\ 0\ (B + 1)$
by (*metis abs_of_nonneg le_less less_irrefl mem_ball_0 norm_of_real*)
have − $ball\ 0\ (B + 1) \subseteq outside\ (path\_image\ \gamma)$
using *B subset_ball* **by** (*intro outside_subset_convex*) *auto*
then have *wout*: $w \in outside\ (path\_image\ \gamma)$
using *w* **by** *blast*
moreover have *winding_number* γ *constant_on outside* (*path_image* γ)
using *winding_number_constant* [*OF* γ *loop*, *of outside*(*path_image* γ)] *connected_outside*
by (*metis DIM_complex bounded_path_image dual_order.refl* γ *outside_no_overlap*)
ultimately have *winding_number* $\gamma\ z$ = *winding_number* $\gamma\ w$
by (*metis* (*no_types*, *opaque_lifting*) *constant_on_def z*)
also have ... = 0
proof −
have *wnot*: $w \notin path\_image\ \gamma$ **using** *wout* **by** (*simp add*: *outside_def*)
{ fix *e*::*real* **assume** 0<*e*
obtain *p* **where** *p*: *polynomial_function p pathstart p* = *pathstart* γ *pathfinish p* = *pathfinish* γ
and *pg1*: $(\bigwedge t.\ [\![0 \le t; t \le 1]\!] \Longrightarrow cmod\ (p\ t - \gamma\ t) < 1)$
and *pge*: $(\bigwedge t.\ [\![0 \le t; t \le 1]\!] \Longrightarrow cmod\ (p\ t - \gamma\ t) < e)$
using *path_approx_polynomial_function* [*OF* γ, *of min 1 e*] ‹*e*>0›
by (*metis atLeastAtMost_iff min_less_iff_conj zero_less_one*)
have $\exists p.\ valid\_path\ p \wedge w \notin path\_image\ p \wedge$
pathstart p = *pathstart* $\gamma \wedge$ *pathfinish p* = *pathfinish* $\gamma \wedge$
$(\forall t{\in}\{0..1\}.\ cmod\ (\gamma\ t - p\ t) < e) \wedge contour\_integral\ p\ (\lambda wa.\ 1\ /\ (wa - w)) = 0$
proof (*intro exI conjI*)
have $\bigwedge x.\ [\![0 \le x; x \le 1]\!] \Longrightarrow cmod\ (p\ x) < B + 1$
using *B* **unfolding** *image_subset_iff path_image_def*
by (*meson add_strict_mono atLeastAtMost_iff le_less_trans mem_ball_0 norm_triangle_sub pg1*)
then have *pip*: *path_image p* ⊆ *ball* 0 (*B* + 1)
by (*auto simp add*: *path_image_def dist_norm ball_def*)
then show $w \notin path\_image\ p$ **using** *w* **by** *blast*

```
      show vap: valid_path p
        by (simp add: p(1) valid_path_polynomial_function)
      show ∀t∈{0..1}. cmod (γ t − p t) < e
        by (metis atLeastAtMost_iff norm_minus_commute pge)
      show contour_integral p (λwa. 1 / (wa − w)) = 0
      proof (rule contour_integral_unique [OF Cauchy_theorem_convex_simple
[OF _ convex_ball [of 0 B+1]]])
        have ⋀z. cmod z < B + 1 ⟹ z ≠ w
          using mem_ball_0 w by blast
        then show (λz. 1 / (z − w)) holomorphic_on ball 0 (B + 1)
          by (intro holomorphic_intros; simp add: dist_norm)
      qed (use p vap pip loop in auto)
    qed (use p in auto)
  }
  then show ?thesis
  by (auto intro: winding_number_unique [OF γ] simp add: winding_number_prop_def
wnot)
 qed
 finally show ?thesis .
qed

corollary winding_number_zero_const: a ≠ z ⟹ winding_number (λt. a) z =
0
 by (rule winding_number_zero_in_outside)
    (auto simp: pathfinish_def pathstart_def path_polynomial_function)

corollary winding_number_zero_outside:
   ⟦path γ; convex s; pathfinish γ = pathstart γ; z ∉ s; path_image γ ⊆ s⟧ ⟹
winding_number γ z = 0
 by (meson convex_in_outside outside_mono subsetCE winding_number_zero_in_outside)

lemma winding_number_zero_at_infinity:
 assumes γ: path γ and loop: pathfinish γ = pathstart γ
   shows ∃B. ∀z. B ≤ norm z ⟶ winding_number γ z = 0
proof −
 obtain B::real where 0 < B and B: path_image γ ⊆ ball 0 B
   using bounded_subset_ballD [OF bounded_path_image [OF γ]] by auto
 have winding_number γ z = 0 if B + 1 ≤ cmod z for z
 proof (rule winding_number_zero_outside [OF γ convex_cball loop])
   show z ∉ cball 0 B
     using that by auto
   show path_image γ ⊆ cball 0 B
     using B order.trans by blast
 qed
 then show ?thesis
   by metis
qed

lemma bounded_winding_number_nz:
```

assumes *path g pathfinish g = pathstart g*
shows *bounded {z. winding_number g z ≠ 0}*
proof −
obtain *B* **where** ⋀*x*. *norm x ≥ B ⟹ winding_number g x = 0*
using *winding_number_zero_at_infinity*[*OF assms*] **by** *auto*
thus *?thesis*
unfolding *bounded_iff* **by** (*intro exI*[*of _ B + 1*]) *force*
qed

lemma *winding_number_zero_point*:
⟦*path γ*; *convex S*; *pathfinish γ = pathstart γ*; *open S*; *path_image γ ⊆ S*⟧
⟹ ∃ *z*. *z ∈ S ∧ winding_number γ z = 0*
using *outside_compact_in_open* [*of path_image γ S*] *path_image_nonempty winding_number_zero_in_outside*
by (*fastforce simp add*: *compact_path_image*)

If a path winds round a set, it winds rounds its inside.

lemma *winding_number_around_inside*:
assumes *γ*: *path γ* **and** *loop*: *pathfinish γ = pathstart γ*
and *cls*: *closed S* **and** *cos*: *connected S* **and** *S_disj*: *S ∩ path_image γ = {}*
and *z*: *z ∈ S* **and** *wn_nz*: *winding_number γ z ≠ 0* **and** *w*: *w ∈ S ∪ inside S*
shows *winding_number γ w = winding_number γ z*
proof −
have *ssb*: *S ⊆ inside*(*path_image γ*)
proof
fix *x* :: *complex*
assume *x ∈ S*
hence *x ∉ path_image γ*
by (*meson disjoint_iff_not_equal S_disj*)
thus *x ∈ inside* (*path_image γ*)
by (*metis Compl_iff S_disj UnE γ* ‹*x ∈ S*› *cos inside_outside loop winding_number_eq winding_number_zero_in_outside wn_nz z*)
qed
show *?thesis*
proof (*rule winding_number_eq* [*OF γ loop w*])
show *z ∈ S ∪ inside S*
using *z* **by** *blast*
show *connected* (*S ∪ inside S*)
by (*simp add*: *cls connected_with_inside cos*)
show (*S ∪ inside S*) *∩ path_image γ = {}*
unfolding *disjoint_iff Un_iff*
by (*metis ComplD UnI1 γ cls compact_path_image connected_path_image inside_Un_outside inside_inside_compact_connected ssb subsetD*)
qed
qed

Bounding a WN by 1/2 for a path and point in opposite halfspaces.

lemma *winding_number_subpath_continuous*:

assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ
shows *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number*$($*subpath* 0 x $\gamma)$ $z)$
proof (*rule continuous_on_eq*)
let *?f* $= \lambda x.$ *integral* $\{0..x\}$ $(\lambda t.$ *vector_derivative* γ $(at\ t)$ / $(\gamma\ t - z))$
show *continuous_on* $\{0..1\}$ $(\lambda x.\ 1$ / $(2 * pi *$ i$) *$ *?f* $x)$
proof (*intro indefinite_integral_continuous_1 winding_number_exp_integral continuous_intros*)
show γ *piecewise_C1_differentiable_on* $\{0..1\}$
using γ *valid_path_def* **by** *blast*
qed (*use path_image_def z* **in** *auto*)
show *1* / $(2 * pi *$ i$) *$ *?f* $x =$ *winding_number* $($*subpath* 0 x $\gamma)$ z
if x: $x \in \{0..1\}$ **for** x
proof −
have *1* / $(2*pi*$i$) *$ *?f* $x =$ *1* / $(2*pi*$i$) *$ *contour_integral* $($*subpath* 0 x $\gamma)$ $(\lambda w.\ 1/(w - z))$
using *assms x*
by (*simp add*: *contour_integral_subcontour_integral* [*OF contour_integrable_inversediff*])
also have $\ldots =$ *winding_number* $($*subpath* 0 x $\gamma)$ z
proof (*subst winding_number_valid_path*)
show $z \notin$ *path_image* $($*subpath* 0 x $\gamma)$
using *assms x atLeastAtMost_iff path_image_subpath_subset* **by** *force*
qed (*use assms x valid_path_subpath* **in** ‹*force*+›)
finally show *?thesis* **.**
qed
qed

lemma *winding_number_ivt_pos*:
assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ **and** $0 \le w\ w \le$ *Re*$($*winding_number* $\gamma\ z)$
shows $\exists\, t \in \{0..1\}.$ *Re*$($*winding_number*$($*subpath* 0 t $\gamma)$ $z) = w$
proof −
have *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number* $($*subpath* 0 x $\gamma)$ $z)$
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* $($*winding_number* $($*subpath* 0 0 $\gamma)$ $z) \le w\ w \le$ *Re* $($*winding_number* $($*subpath* 0 1 $\gamma)$ $z)$
using *assms* **by** (*auto simp*: *path_image_def image_def*)
ultimately show *?thesis*
using *ivt_increasing_component_on_1*[*of 0 1*, **where** *?k = 1*] **by** *force*
qed

lemma *winding_number_ivt_neg*:
assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ **and** *Re*$($*winding_number* $\gamma\ z) \le w\ w \le 0$
shows $\exists\, t \in \{0..1\}.$ *Re*$($*winding_number*$($*subpath* 0 t $\gamma)$ $z) = w$
proof −
have *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number* $($*subpath* 0 x $\gamma)$ $z)$
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* $($*winding_number* $($*subpath* 0 0 $\gamma)$ $z) \ge w\ w \ge$ *Re* $($*winding_number* $($*subpath* 0 1 $\gamma)$ $z)$

```
    using assms by (auto simp: path_image_def image_def)
  ultimately show ?thesis
    using ivt_decreasing_component_on_1[of 0 1, where ?k = 1] by force
qed

lemma winding_number_ivt_abs:
  assumes γ: valid_path γ and z: z ∉ path_image γ and 0 ≤ w w ≤ |Re(winding_number
γ z)|
      shows ∃ t ∈ {0..1}. |Re (winding_number (subpath 0 t γ) z)| = w
  using assms winding_number_ivt_pos [of γ z w] winding_number_ivt_neg [of
γ z −w]
  by force

lemma winding_number_lt_half_lemma:
  assumes γ: valid_path γ and z: z ∉ path_image γ and az: a · z ≤ b and pag:
path_image γ ⊆ {w. a · w > b}
    shows Re(winding_number γ z) < 1/2
proof −
  { assume Re(winding_number γ z) ≥ 1/2
    then obtain t::real where t: 0 ≤ t t ≤ 1 and sub12: Re (winding_number
(subpath 0 t γ) z) = 1/2
      using winding_number_ivt_pos [OF γ z, of 1/2] by auto
   have gt: γ t − z = − (of_real (exp (− (2 ∗ pi ∗ Im (winding_number (subpath
0 t γ) z)))) ∗ (γ 0 − z))
      using winding_number_exp_2pi [of subpath 0 t γ z]
      apply (simp add: t γ valid_path_imp_path)
       using closed_segment_eq_real_ivl path_image_def t z by (fastforce simp:
path_image_subpath Euler sub12)
    have b < a · γ 0
    proof −
      have γ 0 ∈ {c. b < a · c}
        by (metis (no_types) pag atLeastAtMost_iff image_subset_iff order_refl
path_image_def zero_le_one)
      thus ?thesis
        by blast
    qed
    moreover have b < a · γ t
     by (metis atLeastAtMost_iff image_eqI mem_Collect_eq pag path_image_def
subset_iff t)
    ultimately have 0 < a · (γ 0 − z) 0 < a · (γ t − z) using az
      by (simp add: inner_diff_right)+
    then have False
      by (simp add: gt inner_mult_right mult_less_0_iff)
  }
  then show ?thesis by force
qed

lemma winding_number_lt_half:
  assumes valid_path γ a · z ≤ b path_image γ ⊆ {w. a · w > b}
```

shows $|Re\ (winding\_number\ \gamma\ z)| < 1/2$
proof −
have $z \notin path\_image\ \gamma$ **using** *assms* **by** *auto*
with *assms* **have** $Re\ (winding\_number\ \gamma\ z) < 0 \Longrightarrow - Re\ (winding\_number\ \gamma\ z) < 1/2$
by (*metis complex_inner_1_right winding_number_lt_half_lemma* [*OF valid_path_imp_reverse*, *of* γ *z a b*]
winding_number_reversepath valid_path_imp_path inner_minus_left path_image_reversepath)
with *assms* **show** *?thesis*
using ‹$z \notin path\_image\ \gamma$› *winding_number_lt_half_lemma* **by** *fastforce*
qed

lemma *winding_number_le_half*:
assumes γ: *valid_path* γ **and** z: $z \notin path\_image\ \gamma$
and *anz*: $a \neq 0$ **and** *azb*: $a \cdot z \leq b$ **and** *pag*: $path\_image\ \gamma \subseteq \{w.\ a \cdot w \geq b\}$
shows $|Re\ (winding\_number\ \gamma\ z)| \leq 1/2$
proof −
{ assume *wnz_12*: $|Re\ (winding\_number\ \gamma\ z)| > 1/2$
have *isCont* (*winding_number* γ) z
by (*metis continuous_at_winding_number valid_path_imp_path* γ z)
then obtain d **where** $d{>}0$ **and** d: $\bigwedge x'.\ dist\ x'\ z < d \Longrightarrow dist\ (winding\_number\ \gamma\ x')\ (winding\_number\ \gamma\ z) < |Re(winding\_number\ \gamma\ z)| - 1/2$
using *continuous_at_eps_delta wnz_12 diff_gt_0_iff_gt* **by** *blast*
define z' **where** $z' = z - (d\ /\ (2 * cmod\ a)) *_R a$
have $a \cdot z * 6 \leq d * cmod\ a + b * 6$
by (*metis* ‹$0 < d$› *add_increasing azb less_eq_real_def mult_nonneg_nonneg mult_right_mono norm_ge_zero norm_numeral*)
with *anz* **have** $*$: $a \cdot z' \leq b - d\ /\ 3 * cmod\ a$
unfolding *z'_def inner_mult_right' divide_inverse*
by (*simp add*: *field_split_simps algebra_simps dot_square_norm power2_eq_square*)
have $cmod\ (winding\_number\ \gamma\ z' - winding\_number\ \gamma\ z) < |Re\ (winding\_number\ \gamma\ z)| - 1/2$
using d [*of z'*] *anz* ‹$d{>}0$› **by** (*simp add*: *dist_norm z'_def*)
then have $1/2 < |Re\ (winding\_number\ \gamma\ z)| - cmod\ (winding\_number\ \gamma\ z' - winding\_number\ \gamma\ z)$
by *simp*
then have $1/2 < |Re\ (winding\_number\ \gamma\ z)| - |Re\ (winding\_number\ \gamma\ z') - Re\ (winding\_number\ \gamma\ z)|$
using *abs_Re_le_cmod* [*of winding_number* γ z' − *winding_number* γ z]
by *simp*
then have *wnz_12'*: $|Re\ (winding\_number\ \gamma\ z')| > 1/2$
by *linarith*
moreover have $|Re\ (winding\_number\ \gamma\ z')| < 1/2$
proof (*rule winding_number_lt_half* [*OF* γ $*$])
show $path\_image\ \gamma \subseteq \{w.\ b - d\ /\ 3 * cmod\ a < a \cdot w\}$
using *azb* ‹$d{>}0$› *pag* **by** (*auto simp*: *add_strict_increasing anz field_split_simps dest*!: *subsetD*)
qed
ultimately have *False*

```
    by simp
  }
  then show ?thesis by force
qed

lemma winding_number_lt_half_linepath:
  assumes z ∉ closed_segment a b shows |Re (winding_number (linepath a b) z)|
< 1/2
proof −
  obtain u v where u · z ≤ v and uv: ⋀x. x ∈ closed_segment a b ⟹ inner u
x > v
   using separating_hyperplane_closed_point assms closed_segment convex_closed_segment
less_eq_real_def by metis
  moreover have path_image (linepath a b) ⊆ {w. v < u · w}
    using in_segment(1) uv by auto
  ultimately show ?thesis
    using winding_number_lt_half by auto
qed
```

Positivity of WN for a linepath.

```
lemma winding_number_linepath_pos_lt:
    assumes 0 < Im ((b − a) ∗ cnj (b − z))
      shows 0 < Re(winding_number(linepath a b) z)
proof −
  have z: z ∉ path_image (linepath a b)
    using assms
    by (simp add: closed_segment_def) (force simp: algebra_simps)
  show ?thesis
     by (intro winding_number_pos_lt [OF valid_path_linepath z assms]) (simp
add: linepath_def algebra_simps)
qed
```

3.6 More winding number properties

including the fact that it's +-1 inside a simple closed curve.

```
lemma winding_number_homotopic_paths:
    assumes homotopic_paths (−{z}) g h
      shows winding_number g z = winding_number h z
proof −
  have path g path h using homotopic_paths_imp_path [OF assms] by auto
  moreover have pag: z ∉ path_image g and pah: z ∉ path_image h
    using homotopic_paths_imp_subset [OF assms] by auto
  ultimately obtain d e where d > 0 e > 0
       and d: ⋀p. ⟦path p; pathstart p = pathstart g; pathfinish p = pathfinish g;
∀ t∈{0..1}. norm (p t − g t) < d⟧
           ⟹ homotopic_paths (−{z}) g p
       and e: ⋀q. ⟦path q; pathstart q = pathstart h; pathfinish q = pathfinish h;
∀ t∈{0..1}. norm (q t − h t) < e⟧
           ⟹ homotopic_paths (−{z}) h q
```

```
    using homotopic_nearby_paths [of g -{z}] homotopic_nearby_paths [of h
-{z}] by force
  obtain p where p:
      valid_path p z ∉ path_image p
      pathstart p = pathstart g pathfinish p = pathfinish g
      and gp_less:∀ t∈{0..1}. cmod (g t - p t) < d
      and pap: contour_integral p (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number g z
   using winding_number [OF ‹path g› pag ‹0 < d›] unfolding winding_number_prop_def
by blast
  obtain q where q:
      valid_path q z ∉ path_image q
      pathstart q = pathstart h pathfinish q = pathfinish h
      and hq_less: ∀ t∈{0..1}. cmod (h t - q t) < e
      and paq:  contour_integral q (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number h z
   using winding_number [OF ‹path h› pah ‹0 < e›] unfolding winding_number_prop_def
by blast
  have homotopic_paths (- {z}) g p
    by (simp add: d p valid_path_imp_path norm_minus_commute gp_less)
  moreover have homotopic_paths (- {z}) h q
    by (simp add: e q valid_path_imp_path norm_minus_commute hq_less)
  ultimately have homotopic_paths (- {z}) p q
    by (blast intro: homotopic_paths_trans homotopic_paths_sym assms)
  then have contour_integral p (λw. 1/(w - z)) = contour_integral q (λw. 1/(w
- z))
    by (rule Cauchy_theorem_homotopic_paths) (auto intro!: holomorphic_intros
simp: p q)
  then show ?thesis
    by (simp add: pap paq)
qed

lemma winding_number_homotopic_loops:
    assumes homotopic_loops (-{z}) g h
      shows winding_number g z = winding_number h z
proof -
  have path g path h using homotopic_loops_imp_path [OF assms] by auto
  moreover have pag: z ∉ path_image g and pah: z ∉ path_image h
    using homotopic_loops_imp_subset [OF assms] by auto
   moreover have gloop: pathfinish g = pathstart g and hloop: pathfinish h =
pathstart h
    using homotopic_loops_imp_loop [OF assms] by auto
  ultimately obtain d e where d > 0 e > 0
      and d: ⋀p. ⟦path p; pathfinish p = pathstart p; ∀ t∈{0..1}. norm (p t - g t)
< d⟧
            ⟹ homotopic_loops (-{z}) g p
      and e: ⋀q. ⟦path q; pathfinish q = pathstart q; ∀ t∈{0..1}. norm (q t - h t)
< e⟧
            ⟹ homotopic_loops (-{z}) h q
```

```
    using homotopic_nearby_loops [of g -{z}] homotopic_nearby_loops [of h
-{z}] by force
  obtain p where p:
      valid_path p z ∉ path_image p
      pathstart p = pathstart g pathfinish p = pathfinish g
      and gp_less:∀ t∈{0..1}. cmod (g t - p t) < d
      and pap: contour_integral p (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number g z
   using winding_number [OF ‹path g› pag ‹0 < d›] unfolding winding_number_prop_def
by blast
  obtain q where q:
      valid_path q z ∉ path_image q
      pathstart q = pathstart h pathfinish q = pathfinish h
      and hq_less: ∀ t∈{0..1}. cmod (h t - q t) < e
      and paq:  contour_integral q (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number h z
   using winding_number [OF ‹path h› pah ‹0 < e›] unfolding winding_number_prop_def
by blast
  have gp: homotopic_loops (- {z}) g p
   by (simp add: gloop d gp_less norm_minus_commute p valid_path_imp_path)
  have hq: homotopic_loops (- {z}) h q
   by (simp add: e hloop hq_less norm_minus_commute q valid_path_imp_path)
  have contour_integral p (λw. 1/(w - z)) = contour_integral q (λw. 1/(w - z))
  proof (rule Cauchy_theorem_homotopic_loops)
    show homotopic_loops (- {z}) p q
     by (blast intro: homotopic_loops_trans homotopic_loops_sym gp hq assms)
  qed (auto intro!: holomorphic_intros simp: p q)
  then show ?thesis
   by (simp add: pap paq)
qed

lemma winding_number_paths_linear_eq:
  ⟦path g; path h; pathstart h = pathstart g; pathfinish h = pathfinish g;
    ⋀t. t ∈ {0..1} ⟹ z ∉ closed_segment (g t) (h t)⟧
      ⟹ winding_number h z = winding_number g z
 by (blast intro: sym homotopic_paths_linear winding_number_homotopic_paths)

lemma winding_number_loops_linear_eq:
  ⟦path g; path h; pathfinish g = pathstart g; pathfinish h = pathstart h;
    ⋀t. t ∈ {0..1} ⟹ z ∉ closed_segment (g t) (h t)⟧
      ⟹ winding_number h z = winding_number g z
 by (blast intro: sym homotopic_loops_linear winding_number_homotopic_loops)

lemma winding_number_nearby_paths_eq:
    ⟦path g; path h; pathstart h = pathstart g; pathfinish h = pathfinish g;
     ⋀t. t ∈ {0..1} ⟹ norm(h t - g t) < norm(g t - z)⟧
     ⟹ winding_number h z = winding_number g z
 by (metis segment_bound(2) norm_minus_commute not_le winding_number_paths_linear_eq)
```

lemma *winding_number_nearby_loops_eq*:
⟦*path g*; *path h*; *pathfinish g* = *pathstart g*; *pathfinish h* = *pathstart h*;
⋀*t*. *t* ∈ {*0..1*} ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *z*)⟧
⟹ *winding_number h z* = *winding_number g z*
by (*metis segment_bound*(*2*) *norm_minus_commute not_le winding_number_loops_linear_eq*)

lemma *winding_number_subpath_combine*:
assumes *path g* **and** *notin*: *z* ∉ *path_image g* **and** *u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
shows *winding_number* (*subpath u v g*) *z* + *winding_number* (*subpath v w g*) *z* =
winding_number (*subpath u w g*) *z* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *winding_number* (*subpath u v g* +++ *subpath v w g*) *z*
using *assms*
by (*metis* (*no_types*, *lifting*) *path_image_subpath_subset path_subpath pathfinish_subpath pathstart_subpath subsetD winding_number_join*)
also have ... = *?rhs*
by (*meson assms homotopic_join_subpaths subset_Compl_singleton winding_number_homotopic_paths*)
finally show *?thesis* **.**
qed

Winding numbers of circular contours

proposition *winding_number_part_circlepath_pos_less*:
assumes *s* < *t* **and** *no*: *norm*(*w* − *z*) < *r*
shows *0* < *Re* (*winding_number*(*part_circlepath z r s t*) *w*)
proof −
have *0* < *r* **by** (*meson no norm_not_less_zero not_le order.strict_trans2*)
note *valid_path_part_circlepath*
moreover have *w* ∉ *path_image* (*part_circlepath z r s t*)
using *assms* **by** (*auto simp*: *path_image_def image_def part_circlepath_def norm_mult linepath_def*)
moreover have *0* < *r* ∗ (*t* − *s*) ∗ (*r* − *cmod* (*w* − *z*))
using *assms* **by** (*metis* ‹*0* < *r*› *diff_gt_0_iff_gt mult_pos_pos*)
ultimately show *?thesis*
apply (*rule winding_number_pos_lt* [**where** *e* = *r*∗(*t* − *s*)∗(*r* − *norm*(*w* − *z*))])
apply (*simp add*: *vector_derivative_part_circlepath right_diff_distrib* [*symmetric*] *mult_ac mult_le_cancel_left_pos assms* ‹*0*<*r*›)
using *Re_Im_le_cmod* [*of w*−*z linepath s t x* **for** *x*]
by (*simp add*: *exp_Euler cos_of_real sin_of_real part_circlepath_def algebra_simps cos_squared_eq* [*unfolded power2_eq_square*])
qed

lemma *winding_number_circlepath_centre*: *0* < *r* ⟹ *winding_number* (*circlepath z r*) *z* = *1*
apply (*rule winding_number_unique_loop*)

apply (*simp_all add*: *sphere_def valid_path_imp_path*)
apply (*rule_tac x*=*circlepath z r* **in** *exI*)
apply (*simp add*: *sphere_def contour_integral_circlepath*)
done

proposition *winding_number_circlepath*:
assumes *norm*(*w* − *z*) < *r* **shows** *winding_number*(*circlepath z r*) *w* = *1*
proof (*cases w* = *z*)
case *True* **then show** *?thesis*
using *assms winding_number_circlepath_centre* **by** *auto*
next
case *False*
have [*simp*]: *r* > *0*
using *assms le_less_trans norm_ge_zero* **by** *blast*
define *r′* **where** *r′* = *norm*(*w* − *z*)
have *r′* < *r*
by (*simp add*: *assms r′_def*)
have *disjo*: *cball z r′* ∩ *sphere z r* = {}
using ‹*r′* < *r*› **by** (*force simp*: *cball_def sphere_def*)
have *winding_number*(*circlepath z r*) *w* = *winding_number*(*circlepath z r*) *z*
proof (*rule winding_number_around_inside* [**where** *S* = *cball z r′*])
show *winding_number* (*circlepath z r*) *z* ≠ *0*
by (*simp add*: *winding_number_circlepath_centre*)
show *cball z r′* ∩ *path_image* (*circlepath z r*) = {}
by (*simp add*: *disjo less_eq_real_def*)
qed (*auto simp*: *r′_def dist_norm norm_minus_commute*)
also have ... = *1*
by (*simp add*: *winding_number_circlepath_centre*)
finally show *?thesis* **.**
qed

lemma *no_bounded_connected_component_imp_winding_number_zero*:
assumes *g*: *path g path_image g* ⊆ *S pathfinish g* = *pathstart g z* ∉ *S*
and *nb*: ⋀*z*. *bounded* (*connected_component_set* (− *S*) *z*) ⟹ *z* ∈ *S*
shows *winding_number g z* = *0*
proof −
have *z* ∈ *outside* (*path_image g*)
by (*metis nb* [*of z*] ‹*path_image g* ⊆ *S*› ‹*z* ∉ *S*› *subsetD mem_Collect_eq outside outside_mono*)
then show *?thesis*
by (*simp add*: *g winding_number_zero_in_outside*)
qed

lemma *no_bounded_path_component_imp_winding_number_zero*:
assumes *g*: *path g path_image g* ⊆ *S pathfinish g* = *pathstart g z* ∉ *S*
and *nb*: ⋀*z*. *bounded* (*path_component_set* (− *S*) *z*) ⟹ *z* ∈ *S*
shows *winding_number g z* = *0*
by (*simp add*: *bounded_subset nb path_component_subset_connected_component*

no_bounded_connected_component_imp_winding_number_zero [*OF g*])

3.7 Winding number for a triangle

lemma *wn_triangle1*:
assumes *0* ∈ *interior*(*convex hull* {*a*,*b*,*c*})
shows ¬ (*Im*(*a*/*b*) ≤ *0* ∧ *0* ≤ *Im*(*b*/*c*))
proof −
{ assume *0*: *Im*(*a*/*b*) ≤ *0* *0* ≤ *Im*(*b*/*c*)
have *0* ∉ *interior* (*convex hull* {*a*,*b*,*c*})
proof (*cases a*=*0* ∨ *b*=*0* ∨ *c*=*0*)
case *True* **then show** *?thesis*
by (*auto simp*: *not_in_interior_convex_hull_3*)
next
case *False*
then have *b* ≠ *0* **by** *blast*
{ fix *x y*::*complex* **and** *u*::*real*
assume *eq_f'*: *Im x* ∗ *Re b* ≤ *Im b* ∗ *Re x Im y* ∗ *Re b* ≤ *Im b* ∗ *Re y* *0* ≤ *u u* ≤ *1*
then have ((*1* − *u*) ∗ *Im x*) ∗ *Re b* ≤ *Im b* ∗ ((*1* − *u*) ∗ *Re x*)
by (*simp add*: *mult_left_mono mult.assoc mult.left_commute* [*of Im b*])
then have ((*1* − *u*) ∗ *Im x* + *u* ∗ *Im y*) ∗ *Re b* ≤ *Im b* ∗ ((*1* − *u*) ∗ *Re x* + *u* ∗ *Re y*)
using *eq_f' ordered_comm_semiring_class.comm_mult_left_mono*
by (*fastforce simp add*: *algebra_simps*)
}
then have *convex* {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*auto simp*: *algebra_simps convex_alt*)
with *False 0* **have** *convex hull* {*a*,*b*,*c*} ≤ {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*simp add*: *subset_hull mult.commute Complex.Im_divide divide_simps complex_neq_0* [*symmetric*])
moreover have *0* ∉ *interior*({*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*})
proof
assume *0* ∈ *interior* {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
then obtain *e* **where** *e*>*0* **and** *e*: *ball 0 e* ⊆ {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*meson mem_interior*)
define *z* **where** *z* ≡ − *sgn* (*Im b*) ∗ (*e*/*3*) + *sgn* (*Re b*) ∗ (*e*/*3*) ∗ i
have *cmod z* = *cmod* (*e*/*3*) ∗ *cmod* (− *sgn* (*Im b*) + *sgn* (*Re b*) ∗ i)
unfolding *z_def norm_mult* [*symmetric*] **by** (*simp add*: *algebra_simps*)
also have ... < *e*
using ‹*e*>*0*› **by** (*auto simp*: *norm_mult intro*: *le_less_trans* [*OF norm_triangle_ineq4*])
finally have *z* ∈ *ball 0 e*
using ‹*e*>*0*› **by** *simp*
then have *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*
using *e* **by** *blast*
then have *b*: *0* < *Re b* *0* < *Im b* **and** *disj*: *e* ∗ *Re b* < − (*Im b* ∗ *e*) ∨ *e* ∗

```
Re b = - (Im b * e)
        using ‹e>0› ‹b ≠ 0›
      by (auto simp add: z_def dist_norm sgn_if less_eq_real_def mult_less_0_iff
complex.expand split: if_split_asm)
       show False — or just one smt line
        using disj
       proof
        assume e * Re b < - (Im b * e)
        with b ‹0 < e› less_trans [of _ 0] show False
       by (metis (no_types) mult_pos_pos neg_less_0_iff_less not_less_iff_gr_or_eq)
       next
        assume e * Re b = - (Im b * e)
        with b ‹0 < e› show False
          by (metis mult_pos_pos neg_less_0_iff_less not_less_iff_gr_or_eq)
       qed
     qed
     ultimately show ?thesis
      using interior_mono by blast
   qed
  } with assms show ?thesis by blast
qed

lemma wn_triangle2_0:
  assumes 0 ∈ interior(convex hull {a,b,c})
  shows
       0 < Im((b - a) * cnj (b)) ∧
        0 < Im((c - b) * cnj (c)) ∧
        0 < Im((a - c) * cnj (a))
        ∨
        Im((b - a) * cnj (b)) < 0 ∧
        0 < Im((b - c) * cnj (b)) ∧
        0 < Im((a - b) * cnj (a)) ∧
        0 < Im((c - a) * cnj (c))
proof -
  have [simp]: {b,c,a} = {a,b,c} {c,a,b} = {a,b,c} by auto
  show ?thesis
    using wn_triangle1 [OF assms] wn_triangle1 [of b c a] wn_triangle1 [of c a
b] assms
    by (auto simp: algebra_simps Im_complex_div_gt_0 Im_complex_div_lt_0
not_le not_less)
qed

lemma wn_triangle2:
  assumes z ∈ interior(convex hull {a,b,c})
   shows 0 < Im((b - a) * cnj (b - z)) ∧
         0 < Im((c - b) * cnj (c - z)) ∧
         0 < Im((a - c) * cnj (a - z))
         ∨
         Im((b - a) * cnj (b - z)) < 0 ∧
```

```
        0 < Im((b − c) * cnj (b − z)) ∧
        0 < Im((a − b) * cnj (a − z)) ∧
        0 < Im((c − a) * cnj (c − z))
proof −
  have 0: 0 ∈ interior(convex hull {a−z, b−z, c−z})
    using assms convex_hull_translation [of −z {a,b,c}]
          interior_translation [of −z]
    by (simp cong: image_cong_simp)
  show ?thesis using wn_triangle2_0 [OF 0]
    by simp
qed

lemma wn_triangle3:
  assumes z: z ∈ interior(convex hull {a,b,c})
      and 0 < Im((b−a) * cnj (b−z))
        0 < Im((c−b) * cnj (c−z))
        0 < Im((a−c) * cnj (a−z))
    shows winding_number (linepath a b +++ linepath b c +++ linepath c a) z
= 1
proof −
 have znot[simp]: z ∉ closed_segment a b z ∉ closed_segment b c z ∉ closed_segment
c a
    using z interior_of_triangle [of a b c]
    by (auto simp: closed_segment_def)
 have gt0: 0 < Re (winding_number (linepath a b +++ linepath b c +++ linepath
c a) z)
    using assms
   by (simp add: winding_number_linepath_pos_lt path_image_join winding_number_join_pos_combined)
  have lt2: Re (winding_number (linepath a b +++ linepath b c +++ linepath c
a) z) < 2
    using winding_number_lt_half_linepath [of _ a b]
    using winding_number_lt_half_linepath [of _ b c]
    using winding_number_lt_half_linepath [of _ c a] znot
    by (fastforce simp add: winding_number_join path_image_join)
  show ?thesis
    by (rule winding_number_eq_1) (simp_all add: path_image_join gt0 lt2)
qed

proposition winding_number_triangle:
  assumes z: z ∈ interior(convex hull {a,b,c})
    shows winding_number(linepath a b +++ linepath b c +++ linepath c a) z =
          (if 0 < Im((b − a) * cnj (b − z)) then 1 else −1)
proof −
  have [simp]: {a,c,b} = {a,b,c}  by auto
 have znot[simp]: z ∉ closed_segment a b z ∉ closed_segment b c z ∉ closed_segment
c a
    using z interior_of_triangle [of a b c]
    by (auto simp: closed_segment_def)
 then have [simp]: z ∉ closed_segment b a z ∉ closed_segment c b z ∉ closed_segment
```

a c
using *closed_segment_commute* **by** *blast+*
have ∗: *winding_number* (*linepath a b +++ linepath b c +++ linepath c a*) *z* =
winding_number (*reversepath* (*linepath a c +++ linepath c b +++ linepath b a*)) *z*
by (*simp add*: *reversepath_joinpaths winding_number_join not_in_path_image_join*)
show *?thesis*
apply (*rule disjE* [*OF wn_triangle2* [*OF z*]])
subgoal
by (*simp add*: *wn_triangle3 z*)
subgoal
by (*simp add*: *path_image_join winding_number_reversepath* ∗ *wn_triangle3 z*)
done
qed

3.8 Winding numbers for simple closed paths

lemma *winding_number_from_innerpath*:
assumes *simple_path c1* **and** *c1*: *pathstart c1* = *a pathfinish c1* = *b*
and *simple_path c2* **and** *c2*: *pathstart c2* = *a pathfinish c2* = *b*
and *simple_path c* **and** *c*: *pathstart c* = *a pathfinish c* = *b*
and *c1c2*: *path_image c1* ∩ *path_image c2* = {*a*,*b*}
and *c1c*: *path_image c1* ∩ *path_image c* = {*a*,*b*}
and *c2c*: *path_image c2* ∩ *path_image c* = {*a*,*b*}
and *ne_12*: *path_image c* ∩ *inside*(*path_image c1* ∪ *path_image c2*) ≠ {}
and *z*: *z* ∈ *inside*(*path_image c1* ∪ *path_image c*)
and *wn_d*: *winding_number* (*c1 +++ reversepath c*) *z* = *d*
and *a* ≠ *b d* ≠ *0*
obtains *z* ∈ *inside*(*path_image c1* ∪ *path_image c2*) *winding_number* (*c1 +++ reversepath c2*) *z* = *d*
proof −
obtain *0*: *inside*(*path_image c1* ∪ *path_image c*) ∩ *inside*(*path_image c2* ∪ *path_image c*) = {}
and *1*: *inside*(*path_image c1* ∪ *path_image c*) ∪ *inside*(*path_image c2* ∪ *path_image c*) ∪
(*path_image c* − {*a*,*b*}) = *inside*(*path_image c1* ∪ *path_image c2*)
by (*rule split_inside_simple_closed_curve*
[*OF* ‹*simple_path c1*› *c1* ‹*simple_path c2*› *c2* ‹*simple_path c*› *c* ‹*a* ≠ *b*› *c1c2 c1c c2c ne_12*])
have *znot*: *z* ∉ *path_image c z* ∉ *path_image c1 z* ∉ *path_image c2*
using *union_with_outside z 1* **by** *auto*
then have *zout*: *z* ∈ *outside* (*path_image c* ∪ *path_image c2*)
by (*metis 0 ComplI UnE disjoint_iff_not_equal sup.commute union_with_inside z*)
have *wn_cc2*: *winding_number* (*c +++ reversepath c2*) *z* = *0*
by (*rule winding_number_zero_in_outside*; *simp add*: *zout* ‹*simple_path c2*› *c c2* ‹*simple_path c*› *simple_path_imp_path path_image_join*)
show *?thesis*

proof
show $z \in$ *inside* (*path_image c1* $\cup$ *path_image c2*)
using *1 z* **by** *blast*
have *winding_number c1 z* $-$ *winding_number c z* $= d$
using *assms znot*
by (*metis wn_d winding_number_join simple_path_imp_path winding_number_reversepath add.commute path_image_reversepath path_reversepath pathstart_reversepath uminus_add_conv_diff*)
then show *winding_number* (*c1 +++ reversepath c2*) *z* $= d$
using *wn_cc2* **by** (*simp add*: *winding_number_join simple_path_imp_path assms znot winding_number_reversepath*)
qed
qed

lemma *simple_closed_path_wn1*:
fixes *a*::*complex* **and** *e*::*real*
assumes $0 < e$
and *sp_pl*: *simple_path*(*p +++ linepath* (*a* $-$ *e*) (*a* $+$ *e*)) (**is** *simple_path ?pae*)
and *psp*: *pathstart p* $= a + e$
and *pfp*: *pathfinish p* $= a - e$
and *disj*: *ball a e* $\cap$ *path_image p* $= \{\}$
obtains *z* **where** $z \in$ *inside* (*path_image ?pae*) *cmod* (*winding_number ?pae z*) $= 1$
proof $-$
have *arc p* **and** *arc_lp*: *arc* (*linepath* (*a* $-$ *e*) (*a* $+$ *e*))
and *pap*: *path_image p* $\cap$ *path_image* (*linepath* (*a* $-$ *e*) (*a* $+$ *e*)) $\subseteq$ {*pathstart p*, *a*$-$*e*}
using *simple_path_join_loop_eq* [*of linepath* (*a* $-$ *e*) (*a* $+$ *e*) *p*] *assms* **by** *auto*
have *mid_eq_a*: *midpoint* (*a* $-$ *e*) (*a* $+$ *e*) $= a$
by (*simp add*: *midpoint_def*)
with *assms* **have** *a* $\in$ *path_image ?pae*
by (*simp add*: *assms path_image_join*) (*metis midpoint_in_closed_segment*)
then have *a* $\in$ *frontier*(*inside* (*path_image ?pae*))
using *assms* **by** (*simp add*: *Jordan_inside_outside*)
with ‹$0 < e$› **obtain** *c* **where** *c*: $c \in$ *inside* (*path_image ?pae*)
and *dac*: *dist a c* $< e$
by (*auto simp*: *frontier_straddle*)
then have $c \notin$ *path_image ?pae*
using *inside_no_overlap* **by** *blast*
then have $c \notin$ *path_image p* $c \notin$ *closed_segment* (*a* $-$ *e*) (*a* $+$ *e*)
by (*simp_all add*: *assms path_image_join*)
with ‹$0 < e$› *dac* **have** $c \notin$ *affine hull* {*a* $-$ *e*, *a* $+$ *e*}
by (*simp add*: *segment_as_ball not_le*)
with ‹$0 < e$› **have** $*$: $\neg$ *collinear* {*a* $-$ *e*, *c*,*a* $+$ *e*}
using *collinear_3_affine_hull* [*of a*$-$*e a*+*e*] **by** (*auto simp*: *insert_commute*)
have *13*: $1/3 + 1/3 + 1/3 = (1$::*real*$)$ **by** *simp*
have $(1/3) *_R (a -$ *of_real e*$) + (1/3) *_R c + (1/3) *_R (a +$ *of_real e*$) \in$ *interior*(*convex hull* {*a* $-$ *e*, *c*, *a* $+$ *e*})

```
    using interior_convex_hull_3_minimal [OF * DIM_complex]
    by clarsimp (metis 13 zero_less_divide_1_iff zero_less_numeral)
  then obtain z where z: z ∈ interior(convex hull {a − e, c, a + e}) by force
  have [simp]: z ∉ closed_segment (a − e) c
    by (metis DIM_complex Diff_iff IntD2 inf_sup_absorb interior_of_triangle
z)
  have [simp]: z ∉ closed_segment (a + e) (a − e)
    by (metis DIM_complex DiffD2 Un_iff interior_of_triangle z)
  have [simp]: z ∉ closed_segment c (a + e)
   by (metis (no_types, lifting) DIM_complex DiffD2 Un_insert_right inf_sup_aci(5)
insertCI interior_of_triangle mk_disjoint_insert z)
  show thesis
  proof
   have norm (winding_number (linepath (a − e) c +++ linepath c (a + e) +++
linepath (a + e) (a − e)) z) = 1
     using winding_number_triangle [OF z] by simp
    have zin: z ∈ inside (path_image (linepath (a + e) (a − e)) ∪ path_image p)
     and zeq: winding_number (linepath (a + e) (a − e) +++ reversepath p) z =
              winding_number (linepath (a − e) c +++ linepath c (a + e) +++
linepath (a + e) (a − e)) z
    proof (rule winding_number_from_innerpath
       [of linepath (a + e) (a − e) a+e a−e p
         linepath (a + e) c +++ linepath c (a − e) z
           winding_number (linepath (a − e)  c +++ linepath  c (a + e) +++
linepath (a + e) (a − e)) z])
      show sp_aec: simple_path (linepath (a + e) c +++ linepath c (a − e))
      proof (rule arc_imp_simple_path [OF arc_join])
        show arc (linepath (a + e) c)
         by (metis ‹c ∉ path_image p› arc_linepath pathstart_in_path_image psp)
        show arc (linepath c (a − e))
         by (metis ‹c ∉ path_image p› arc_linepath pathfinish_in_path_image pfp)
        show path_image (linepath (a + e) c) ∩ path_image (linepath c (a − e))
⊆ {pathstart (linepath c (a − e))}
          by clarsimp (metis * IntI Int_closed_segment closed_segment_commute
singleton_iff)
      qed auto
      show simple_path p
        using ‹arc p› arc_simple_path by blast
      show sp_ae2: simple_path (linepath (a + e) (a − e))
        using ‹arc p› arc_distinct_ends pfp psp by fastforce
      show pa: pathfinish (linepath (a + e) (a − e)) = a − e
          pathstart (linepath (a + e) c +++ linepath c (a − e)) = a + e
          pathfinish (linepath (a + e) c +++ linepath c (a − e)) = a − e
          pathstart p = a + e pathfinish p = a − e
          pathstart (linepath (a + e) (a − e)) = a + e
        by (simp_all add: assms)
      show 1: path_image (linepath (a + e) (a − e)) ∩ path_image p = {a + e,
a − e}
      proof
```

show *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image p* ⊆ {*a* + *e*, *a* − *e*}
using *pap closed_segment_commute psp segment_convex_hull* **by** *fastforce*
show {*a* + *e*, *a* − *e*} ⊆ *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image p*
using *pap pathfinish_in_path_image pathstart_in_path_image pfp psp* **by** *fastforce*
qed
show *2*: *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image* (*linepath* (*a* + *e*) *c* +++ *linepath c* (*a* − *e*)) =
{*a* + *e*, *a* − *e*} (**is** *?lhs* = *?rhs*)
proof
have ¬ *collinear* {*c*, *a* + *e*, *a* − *e*}
using ∗ **by** (*simp add*: *insert_commute*)
then have *convex hull* {*a* + *e*, *a* − *e*} ∩ *convex hull* {*a* + *e*, *c*} = {*a* + *e*}
convex hull {*a* + *e*, *a* − *e*} ∩ *convex hull* {*c*, *a* − *e*} = {*a* − *e*}
by (*metis* (*full_types*) *Int_closed_segment insert_commute segment_convex_hull*)+
then show *?lhs* ⊆ *?rhs*
by (*metis Int_Un_distrib equalityD1 insert_is_Un path_image_join path_image_linepath path_join_eq path_linepath segment_convex_hull simple_path_def sp_aec*)
show *?rhs* ⊆ *?lhs*
using *segment_convex_hull* **by** (*simp add*: *path_image_join*)
qed
have *path_image p* ∩ *path_image* (*linepath* (*a* + *e*) *c*) ⊆ {*a* + *e*}
proof (*clarsimp simp*: *path_image_join*)
fix *x*
assume *x* ∈ *path_image p* **and** *x_ac*: *x* ∈ *closed_segment* (*a* + *e*) *c*
then have *dist x a* ≥ *e*
by (*metis IntI all_not_in_conv disj dist_commute mem_ball not_less*)
with *x_ac dac* ‹*e* > *0*› **show** *x* = *a* + *e*
by (*auto simp*: *norm_minus_commute dist_norm closed_segment_eq_open dest*: *open_segment_furthest_le* [**where** *y*=*a*])
qed
moreover
have *path_image p* ∩ *path_image* (*linepath c* (*a* − *e*)) ⊆ {*a* − *e*}
proof (*clarsimp simp*: *path_image_join*)
fix *x*
assume *x* ∈ *path_image p* **and** *x_ac*: *x* ∈ *closed_segment c* (*a* − *e*)
then have *dist x a* ≥ *e*
by (*metis IntI all_not_in_conv disj dist_commute mem_ball not_less*)
with *x_ac dac* ‹*e* > *0*› **show** *x* = *a* − *e*
by (*auto simp*: *norm_minus_commute dist_norm closed_segment_eq_open dest*: *open_segment_furthest_le* [**where** *y*=*a*])
qed
ultimately
have *path_image p* ∩ *path_image* (*linepath* (*a* + *e*) *c* +++ *linepath c* (*a* − *e*)) ⊆ {*a* + *e*, *a* − *e*}
by (*force simp*: *path_image_join*)

then show *3*: *path_image p ∩ path_image (linepath (a + e) c +++ linepath c (a − e)) = {a + e, a − e}*
using *1 2* **by** *blast*
show *4*: *path_image (linepath (a + e) c +++ linepath c (a − e)) ∩ inside (path_image (linepath (a + e) (a − e)) ∪ path_image p) ≠ {}*
apply (*clarsimp simp*: *path_image_join segment_convex_hull disjoint_iff_not_equal*)
by (*metis* (*no_types*, *opaque_lifting*) *UnI1 Un_commute c closed_segment_commute ends_in_segment*(*1*) *path_image_join path_image_linepath pathstart_linepath pfp segment_convex_hull*)
show *zin_inside*: *z ∈ inside (path_image (linepath (a + e) (a − e)) ∪ path_image (linepath (a + e) c +++ linepath c (a − e)))*
proof (*simp add*: *path_image_join*)
show *z ∈ inside (closed_segment (a + e) (a − e) ∪ (closed_segment (a + e) c ∪ closed_segment c (a − e)))*
by (*metis z inside_of_triangle DIM_complex Un_commute closed_segment_commute*)
qed
show *5*: *winding_number (linepath (a + e) (a − e) +++ reversepath (linepath (a + e) c +++ linepath c (a − e))) z = winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z*
by (*simp add*: *reversepath_joinpaths path_image_join winding_number_join*)
show *6*: *winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z ≠ 0*
by (*simp add*: *winding_number_triangle z*)
show *winding_number (linepath (a + e) (a − e) +++ reversepath p) z = winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z*
by (*metis 1 2 3 4 5 6 pa sp_aec sp_ae2* ‹*arc p*› ‹*simple_path p*› *arc_distinct_ends winding_number_from_innerpath zin_inside*)
qed (*use assms* ‹*e > 0*› **in** *auto*)
show *z_inside*: *z ∈ inside (path_image ?pae)*
using *zin* **by** (*simp add*: *assms path_image_join Un_commute closed_segment_commute*)
have *cmod (winding_number ?pae z) = cmod ((winding_number(reversepath ?pae) z))*
proof (*subst winding_number_reversepath*)
show *path ?pae*
using *simple_path_imp_path sp_pl* **by** *blast*
show *z ∉ path_image ?pae*
by (*metis IntI emptyE inside_no_overlap z_inside*)
qed (*simp add*: *inside_def*)
also have *... = cmod (winding_number(linepath (a + e) (a − e) +++ reversepath p) z)*
by (*simp add*: *pfp reversepath_joinpaths*)
also have *... = cmod (winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z)*
by (*simp add*: *zeq*)
also have *... = 1*

```
    using z by (simp add: interior_of_triangle winding_number_triangle)
   finally show cmod (winding_number ?pae z) = 1 .
  qed
qed

lemma simple_closed_path_wn2:
  fixes a::complex and d e::real
  assumes 0 < d 0 < e
    and sp_pl: simple_path(p +++ linepath (a - d) (a + e))
    and psp:   pathstart p = a + e
    and pfp:   pathfinish p = a - d
obtains z where z ∈ inside (path_image (p +++ linepath (a - d) (a + e)))
                cmod (winding_number (p +++ linepath (a - d) (a + e)) z) = 1
proof -
  have [simp]: a + of_real x ∈ closed_segment (a - α) (a - β) ⟷ x ∈
closed_segment (-α) (-β) for x α β::real
    using closed_segment_translation_eq [of a]
    by (metis (no_types, opaque_lifting) add_uminus_conv_diff of_real_minus
of_real_closed_segment)
  have [simp]: a - of_real x ∈ closed_segment (a + α) (a + β) ⟷ -x ∈
closed_segment α β for x α β::real
  by (metis closed_segment_translation_eq diff_conv_add_uminus of_real_closed_segment
of_real_minus)
  have arc p and arc_lp: arc (linepath (a - d) (a + e)) and path p
    and pap: path_image p ∩ closed_segment (a - d) (a + e) ⊆ {a+e, a-d}
      using simple_path_join_loop_eq [of linepath (a - d) (a + e) p] assms
arc_imp_path by auto
  have 0 ∈ closed_segment (-d) e
    using ‹0 < d› ‹0 < e› closed_segment_eq_real_ivl by auto
  then have a ∈ path_image (linepath (a - d) (a + e))
    using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
  then have a ∉ path_image p
    using ‹0 < d› ‹0 < e› pap by auto
  then obtain k where 0 < k and k: ball a k ∩ (path_image p) = {}
    using ‹0 < e› ‹path p› not_on_path_ball by blast
  define kde where kde ≡ (min k (min d e)) / 2
  have 0 < kde kde < k kde < d kde < e
    using ‹0 < k› ‹0 < d› ‹0 < e› by (auto simp: kde_def)
  let ?q = linepath (a + kde) (a + e) +++ p +++ linepath (a - d) (a - kde)
  have - kde ∈ closed_segment (-d) e
    using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
  then have a_diff_kde: a - kde ∈ closed_segment (a - d) (a + e)
    using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
  then have clsub2: closed_segment (a - d) (a - kde) ⊆ closed_segment (a -
d) (a + e)
```

```
    by (simp add: subset_closed_segment)
  then have path_image p ∩ closed_segment (a − d) (a − kde) ⊆ {a + e, a −
d}
    using pap by force
  moreover
  have a + e ∉ path_image p ∩ closed_segment (a − d) (a − kde)
   using ‹0 < kde› ‹kde < d› ‹0 < e› by (auto simp: closed_segment_eq_real_ivl)
  ultimately have sub_a_diff_d: path_image p ∩ closed_segment (a − d) (a −
kde) ⊆ {a − d}
    by blast
  have kde ∈ closed_segment (−d) e
    using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
  then have a_diff_kde: a + kde ∈ closed_segment (a − d) (a + e)
    using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
  then have clsub1: closed_segment (a + kde) (a + e) ⊆ closed_segment (a −
d) (a + e)
    by (simp add: subset_closed_segment)
  then have closed_segment (a + kde) (a + e) ∩ path_image p ⊆ {a + e, a −
d}
    using pap by force
  moreover
  have closed_segment (a + kde) (a + e) ∩ closed_segment (a − d) (a − kde) =
{}
  proof (clarsimp intro!: equals0I)
    fix y
    assume y1: y ∈ closed_segment (a + kde) (a + e)
       and y2: y ∈ closed_segment (a − d) (a − kde)
    obtain u::real where u: y = a + u and 0 < u
    proof −
      obtain ξ where ξ: y = (1 − ξ) *R (a + kde) + ξ *R (a + e) and 0 ≤ ξ ξ
≤ 1
        using y1 by (auto simp: in_segment)
      show thesis
      proof
        show y = a + ((1 − ξ)*kde + ξ*e)
          using ξ by (auto simp: scaleR_conv_of_real algebra_simps)
        have (1 − ξ)*kde + ξ*e ≥ 0
          using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < e› by auto
        moreover have (1 − ξ)*kde + ξ*e ≠ 0
       using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < e› by (auto simp: add_nonneg_eq_0_iff)
        ultimately show (1 − ξ)*kde + ξ*e > 0 by simp
      qed
    qed
    moreover
    obtain v::real where v: y = a + v and v ≤ 0
    proof −
      obtain ξ where ξ: y = (1 − ξ) *R (a − d) + ξ *R (a − kde) and 0 ≤ ξ ξ
```

```
≤ 1
      using y2 by (auto simp: in_segment)
    show thesis
    proof
      show y = a + (- ((1 - ξ)*d + ξ*kde))
        using ξ by (force simp: scaleR_conv_of_real algebra_simps)
      show - ((1 - ξ)*d + ξ*kde) ≤ 0
        using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < d›
     by (metis add.left_neutral add_nonneg_nonneg le_diff_eq less_eq_real_def
neg_le_0_iff_le split_mult_pos_le)

    qed
  qed
  ultimately show False
    by auto
 qed
 moreover have a - d ∉ closed_segment (a + kde) (a + e)
  using ‹0 < kde› ‹kde < d› ‹0 < e› by (auto simp: closed_segment_eq_real_ivl)
 ultimately have sub_a_plus_e:
  closed_segment (a + kde) (a + e) ∩ (path_image p ∪ closed_segment (a - d)
(a - kde)) ⊆ {a + e}
  by auto
 have kde ∈ closed_segment (-kde) e
  using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
 then have a_add_kde: a + kde ∈ closed_segment (a - kde) (a + e)
  using of_real_closed_segment [THEN iffD2]
  by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 have closed_segment (a - kde) (a + kde) ∩ closed_segment (a + kde) (a + e)
= {a + kde}
  by (metis a_add_kde Int_closed_segment)
 moreover
 have path_image p ∩ closed_segment (a - kde) (a + kde) = {}
 proof (rule equals0I, clarify)
  fix y assume y ∈ path_image p y ∈ closed_segment (a - kde) (a + kde)
  with equals0D [OF k, of y] ‹0 < kde› ‹kde < k› show False
   by (auto simp: dist_norm dest: dist_decreases_closed_segment [where c=a])
 qed
 moreover
 have - kde ∈ closed_segment (-d) kde
  using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
 then have a_diff_kde': a - kde ∈ closed_segment (a - d) (a + kde)
  using of_real_closed_segment [THEN iffD2]
  by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 then have closed_segment (a - d) (a - kde) ∩ closed_segment (a - kde) (a +
kde) = {a - kde}
  by (metis Int_closed_segment)
 ultimately
```

```
  have pa_subset_pm_kde: path_image ?q ∩ closed_segment (a − kde) (a + kde)
⊆ {a − kde, a + kde}
    by (auto simp: path_image_join assms)
  have ge_kde1: ∃ y. x = a + of_real y ∧ y ≥ kde if x: x ∈ closed_segment (a +
kde) (a + e) for x
  proof −
    obtain u where 0 ≤ u u ≤ 1 and u: x = (1 − u) *R (a + kde) + u *R (a
+ e)
      using x by (auto simp: in_segment)
    then have kde ≤ (1 − u) * kde + u * e
      using ‹kde < e› segment_bound_lemma by auto
    moreover have x = a + ((1 − u) * kde + u * e)
      by (fastforce simp: u algebra_simps scaleR_conv_of_real)
    ultimately
    show ?thesis by blast
  qed
  have ge_kde2: ∃ y. x = a + of_real y ∧ y ≤ −kde if x: x ∈ closed_segment (a
− d) (a − kde) for x
  proof −
    obtain u where 0 ≤ u u ≤ 1 and u: x = (1 − u) *R (a − d) + u *R (a −
kde)
      using x by (auto simp: in_segment)
    then have kde ≤ ((1−u)*d + u*kde)
      using ‹kde < d› segment_bound_lemma by auto
    moreover have x = a − ((1−u)*d + u*kde)
      by (fastforce simp: u algebra_simps scaleR_conv_of_real)
    ultimately show ?thesis
      by (metis add_uminus_conv_diff neg_le_iff_le of_real_minus)
  qed
  have notin_paq: x ∉ path_image ?q if dist a x < kde for x
  proof −
    have x ∉ path_image p
      using k ‹kde < k› that by force
    then show ?thesis
      using that assms by (auto simp: path_image_join dist_norm dest!: ge_kde1
ge_kde2)
  qed
  obtain z where zin: z ∈ inside (path_image (?q +++ linepath (a − kde) (a +
kde)))
          and z1: cmod (winding_number (?q +++ linepath (a − kde) (a + kde))
z) = 1
  proof (rule simple_closed_path_wn1 [of kde ?q a])
    show simple_path (?q +++ linepath (a − kde) (a + kde))
    proof (intro simple_path_join_loop conjI)
      show arc ?q
      proof (rule arc_join)
        show arc (linepath (a + kde) (a + e))
          using ‹kde < e› ‹arc p› by (force simp: pfp)
        show arc (p +++ linepath (a − d) (a − kde))
```

```
      using ‹kde < d› ‹kde < e› ‹arc p› sub_a_diff_d by (force simp: pfp intro:
arc_join)
    qed (auto simp: psp pfp path_image_join sub_a_plus_e)
    show arc (linepath (a − kde) (a + kde))
      using ‹0 < kde› by auto
  qed (use pa_subset_pm_kde in auto)
 qed (use ‹0 < kde› notin_paq in auto)
 have eq: path_image (?q +++ linepath (a − kde) (a + kde)) = path_image (p
+++ linepath (a − d) (a + e))
        (is ?lhs = ?rhs)
 proof
  show ?lhs ⊆ ?rhs
    using clsub1 clsub2 apply (auto simp: path_image_join assms)
    by (meson subsetCE subset_closed_segment)
  show ?rhs ⊆ ?lhs
    apply (simp add: path_image_join assms Un_ac)
     by (metis Un_closed_segment Un_assoc a_diff_kde a_diff_kde' le_supI2
subset_refl)
  qed
 show thesis
 proof
  show zzin: z ∈ inside (path_image (p +++ linepath (a − d) (a + e)))
    by (metis eq zin)
  then have znotin: z ∉ path_image p
     by (metis ComplD Un_iff inside_Un_outside path_image_join pathfin-
ish_linepath pathstart_reversepath pfp reversepath_linepath)
  have znotin_d_kde: z ∉ closed_segment (a − d) (a + kde)
   by (metis ComplD Un_iff Un_closed_segment a_diff_kde inside_Un_outside
path_image_join path_image_linepath pathstart_linepath pfp zzin)
  have znotin_d_e: z ∉ closed_segment (a − d) (a + e)
   by (metis IntI UnCI emptyE inside_no_overlap path_image_join path_image_linepath
pathstart_linepath pfp zzin)
  have znotin_kde_e: z ∉ closed_segment (a + kde) (a + e) and znotin_d_kde':
z ∉ closed_segment (a − d) (a − kde)
    using clsub1 clsub2 zzin
    by (metis (no_types, opaque_lifting) IntI Un_iff emptyE inside_no_overlap
path_image_join path_image_linepath pathstart_linepath pfp subsetD)+
  have winding_number (linepath (a − d) (a + e)) z =
        winding_number (linepath (a − d) (a + kde)) z + winding_number
(linepath (a + kde) (a + e)) z
  proof (rule winding_number_split_linepath)
    show a + complex_of_real kde ∈ closed_segment (a − d) (a + e)
      by (simp add: a_diff_kde)
    show z ∉ closed_segment (a − d) (a + e)
    by (metis ComplD Un_iff inside_Un_outside path_image_join path_image_linepath
pathstart_linepath pfp zzin)
  qed
  also have ... = winding_number (linepath (a + kde) (a + e)) z +
            (winding_number (linepath (a − d) (a − kde)) z + winding_number
```

(*linepath* (*a* − *kde*) (*a* + *kde*)) *z*)
by (*simp add*: *winding_number_split_linepath* [*of a*−*kde*, *symmetric*] *znotin_d_kde a_diff_kde'*)
finally have *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* =
winding_number p z + *winding_number* (*linepath* (*a* + *kde*) (*a* + *e*)) *z* +
(*winding_number* (*linepath* (*a* − *d*) (*a* − *kde*)) *z* +
winding_number (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*)
by (*metis* (*no_types*, *lifting*) *ComplD Un_iff* ‹*arc p*› *add.assoc arc_imp_path eq path_image_join path_join_path_ends path_linepath simple_path_imp_path sp_pl union_with_outside winding_number_join zin*)
also have ... = *winding_number* (*linepath* (*a* + *kde*) (*a* + *e*)) *z*
+ *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* − *kde*)) *z*
+ *winding_number* (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*
using ‹*path p*› *znotin assms*
by *simp* (*metis Un_iff Un_closed_segment a_diff_kde' path_image_linepath path_linepath pathstart_linepath winding_number_join znotin_d_kde*)
also have ... = *winding_number ?q z* + *winding_number* (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*
using ‹*path p*› *znotin assms* **by** (*simp add*: *path_image_join winding_number_join znotin_kde_e znotin_d_kde'*)
also have ... = *winding_number* (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)) *z*
by (*metis* (*mono_tags*, *lifting*) *ComplD UnCI* ‹*path p*› *eq inside_outside path_image_join path_join_eq path_linepath pathfinish_join pathfinish_linepath pathstart_join pathstart_linepath pfp psp winding_number_join zzin*)
finally have *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* =
winding_number (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)) *z* **.**
then show *cmod* (*winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z*) = *1*
by (*simp add*: *z1*)
qed
qed

lemma *simple_closed_path_wn3*:
fixes *p* :: *real* ⇒ *complex*
assumes *simple_path p* **and** *loop*: *pathfinish p* = *pathstart p*
obtains *z* **where** *z* ∈ *inside* (*path_image p*) *cmod* (*winding_number p z*) = *1*
proof −
have *ins*: *inside*(*path_image p*) ≠ {} *open*(*inside*(*path_image p*))
connected(*inside*(*path_image p*))
and *out*: *outside*(*path_image p*) ≠ {} *open*(*outside*(*path_image p*))
connected(*outside*(*path_image p*))
and *bo*: *bounded*(*inside*(*path_image p*)) ¬ *bounded*(*outside*(*path_image p*))
and *ins_out*: *inside*(*path_image p*) ∩ *outside*(*path_image p*) = {}
inside(*path_image p*) ∪ *outside*(*path_image p*) = − *path_image p*
and *fro*: *frontier*(*inside*(*path_image p*)) = *path_image p*
frontier(*outside*(*path_image p*)) = *path_image p*
using *Jordan_inside_outside* [*OF assms*] **by** *auto*
obtain *a* **where** *a*: *a* ∈ *inside*(*path_image p*)
using ‹*inside* (*path_image p*) ≠ {}› **by** *blast*

obtain *d*::*real* **where** $0 < d$ **and** *d_fro*: $a - d \in$ *frontier* (*inside* (*path_image p*))
and *d_int*: $\bigwedge\varepsilon.\ [\![0 \le \varepsilon; \varepsilon < d]\!] \Longrightarrow (a - \varepsilon) \in$ *inside* (*path_image p*)
using *ray_to_frontier* [*of inside* (*path_image p*) *a* −*1*] *bo ins a interior_eq*
by (*metis ab_group_add_class.ab_diff_conv_add_uminus of_real_def scale_minus_right zero_neq_neg_one*)
obtain *e*::*real* **where** $0 < e$ **and** *e_fro*: $a + e \in$ *frontier* (*inside* (*path_image p*))
and *e_int*: $\bigwedge\varepsilon.\ [\![0 \le \varepsilon; \varepsilon < e]\!] \Longrightarrow (a + \varepsilon) \in$ *inside* (*path_image p*)
using *ray_to_frontier* [*of inside* (*path_image p*) *a 1*] *bo ins a interior_eq*
by (*metis of_real_def zero_neq_one*)
obtain *t0* **where** $0 \le t0\ t0 \le 1$ **and** *pt*: $p\ t0 = a - d$
using *a d_fro fro* **by** (*auto simp*: *path_image_def*)
obtain *q* **where** *simple_path q* **and** *q_ends*: *pathstart q* $= a - d$ *pathfinish q* $= a - d$
and *q_eq_p*: *path_image q* = *path_image p*
and *wn_q_eq_wn_p*: $\bigwedge z.\ z \in$ *inside*(*path_image p*) $\Longrightarrow$ *winding_number q z* = *winding_number p z*
proof
show *simple_path* (*shiftpath t0 p*)
by (*simp add*: *pathstart_shiftpath pathfinish_shiftpath simple_path_shiftpath path_image_shiftpath* ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› *assms*)
show *pathstart* (*shiftpath t0 p*) = $a - d$
using *pt* **by** (*simp add*: ‹*t0* ≤ *1*› *pathstart_shiftpath*)
show *pathfinish* (*shiftpath t0 p*) = $a - d$
by (*simp add*: ‹*0* ≤ *t0*› *loop pathfinish_shiftpath pt*)
show *path_image* (*shiftpath t0 p*) = *path_image p*
by (*simp add*: ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› *loop path_image_shiftpath*)
show *winding_number* (*shiftpath t0 p*) *z* = *winding_number p z*
if $z \in$ *inside* (*path_image p*) **for** *z*
by (*metis ComplD Un_iff* ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› ‹*simple_path p*› *atLeastAtMost_iff inside_Un_outside loop simple_path_imp_path that winding_number_shiftpath*)
qed
have *ad_not_ae*: $a - d \neq a + e$
by (*metis* ‹*0* < *d*› ‹*0* < *e*› *add.left_inverse add_left_cancel add_uminus_conv_diff le_add_same_cancel2 less_eq_real_def not_less of_real_add of_real_def of_real_eq_0_iff pt*)
have *ad_ae_q*: $\{a - d, a + e\} \subseteq$ *path_image q*
using ‹*path_image q* = *path_image p*› *d_fro e_fro fro*(*1*) **by** *auto*
have *ada*: *open_segment* $(a - d)\ a \subseteq$ *inside* (*path_image p*)
proof (*clarsimp simp*: *in_segment*)
fix *u*::*real* **assume** $0 < u\ u < 1$
with *d_int* **have** $a - (1 - u) * d \in$ *inside* (*path_image p*)
by (*metis* ‹*0* < *d*› *add.commute diff_add_cancel left_diff_distrib' less_add_same_cancel2 less_eq_real_def mult.left_neutral zero_less_mult_iff*)
then show $(1 - u) *_R (a - d) + u *_R a \in$ *inside* (*path_image p*)
by (*simp add*: *diff_add_eq of_real_def real_vector.scale_right_diff_distrib*)
qed

```
  have aae: open_segment a (a + e) ⊆ inside (path_image p)
  proof (clarsimp simp: in_segment)
    fix u::real assume 0 < u u < 1
    with e_int have a + u * e ∈ inside (path_image p)
        by (meson ‹0 < e› less_eq_real_def mult_less_cancel_right2 not_less
zero_less_mult_iff)
    then show (1 - u) *R a + u *R (a + e) ∈ inside (path_image p)
     by (metis (mono_tags, lifting) add.assoc of_real_mult pth_6 scaleR_collapse
scaleR_conv_of_real)
  qed
  have complex_of_real (d * d + (e * e + d * (e + e))) ≠ 0
    using ad_not_ae
     by (metis ‹0 < d› ‹0 < e› add_strict_left_mono less_add_same_cancel1
not_sum_squares_lt_zero
      of_real_eq_0_iff zero_less_double_add_iff_zero_less_single_add zero_less_mult_iff)
  moreover have ∃u>0. u < 1 ∧ d = u * d + u * e
    using ‹0 < d› ‹0 < e›
   by (rule_tac x=d / (d+e) in exI) (simp add: divide_simps scaleR_conv_of_real
flip: distrib_left)
  ultimately have a_in_de: a ∈ open_segment (a - d) (a + e)
   using ad_not_ae by (simp add: in_segment algebra_simps scaleR_conv_of_real
flip: of_real_add of_real_mult)
  then have open_segment (a - d) (a + e) ⊆ inside (path_image p)
    using ada a aae Un_open_segment [of a a-d a+e] by blast
  then have path_image q ∩ open_segment (a - d) (a + e) = {}
    using inside_no_overlap by (fastforce simp: q_eq_p)
  with ad_ae_q have paq_Int_cs: path_image q ∩ closed_segment (a - d) (a +
e) = {a - d, a + e}
    by (simp add: closed_segment_eq_open)
  obtain t where 0 ≤ t t ≤ 1 and qt: q t = a + e
    using a e_fro fro ad_ae_q by (auto simp: path_defs)
  then have t ≠ 0
    by (metis ad_not_ae pathstart_def q_ends(1))
  then have t ≠ 1
    by (metis ad_not_ae pathfinish_def q_ends(2) qt)
  have q01: q 0 = a - d q 1 = a - d
    using q_ends by (auto simp: pathstart_def pathfinish_def)
  obtain z where zin: z ∈ inside (path_image (subpath t 0 q +++ linepath (a -
d) (a + e)))
            and z1: cmod (winding_number (subpath t 0 q +++ linepath (a - d)
(a + e)) z) = 1
  proof (rule simple_closed_path_wn2 [of d e subpath t 0 q a], simp_all add: q01)
    show simple_path (subpath t 0 q +++ linepath (a - d) (a + e))
    proof (rule simple_path_join_loop, simp_all add: qt q01)
      have inj_on q (closed_segment t 0)
        using ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 0› ‹t ≠ 1›
      by (fastforce simp: simple_path_def loop_free_def inj_on_def closed_segment_eq_real_ivl)
      then show arc (subpath t 0 q)
        using ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 0›
```

```
      by (simp add: arc_subpath_eq simple_path_imp_path)
    show arc (linepath (a - d) (a + e))
      by (simp add: ad_not_ae)
    show path_image (subpath t 0 q) ∩ closed_segment (a - d) (a + e) ⊆ {a +
e, a - d}
      using qt paq_Int_cs ‹simple_path q› ‹0 ≤ t› ‹t ≤ 1›
     by (force simp: dest: rev_subsetD [OF _ path_image_subpath_subset] intro:
simple_path_imp_path)
   qed
 qed (auto simp: ‹0 < d› ‹0 < e› qt)
 have pa01_Un: path_image (subpath 0 t q) ∪ path_image (subpath 1 t q) =
path_image q
   unfolding path_image_subpath
   using ‹0 ≤ t› ‹t ≤ 1› by (force simp: path_image_def image_iff)
 with paq_Int_cs have pa_01q:
     (path_image (subpath 0 t q) ∪ path_image (subpath 1 t q)) ∩ closed_segment
(a - d) (a + e) = {a - d, a + e}
   by metis
 have z_notin_ed: z ∉ closed_segment (a + e) (a - d)
   using zin q01 by (simp add: path_image_join closed_segment_commute in-
side_def)
 have z_notin_0t: z ∉ path_image (subpath 0 t q)
   by (metis (no_types, opaque_lifting) IntI Un_upper1 subsetD empty_iff in-
side_no_overlap path_image_join
    path_image_subpath_commute pathfinish_subpath pathstart_def pathstart_linepath
q_ends(1) qt subpath_trivial zin)
 have [simp]: - winding_number (subpath t 0 q) z = winding_number (subpath
0 t q) z
   by (metis ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› atLeastAtMost_iff zero_le_one
         path_image_subpath_commute path_subpath real_eq_0_iff_le_ge_0
       reversepath_subpath simple_path_imp_path winding_number_reversepath
z_notin_0t)
 obtain z_in_q: z ∈ inside(path_image q)
     and wn_q: winding_number (subpath 0 t q +++ subpath t 1 q) z = -
winding_number (subpath t 0 q +++ linepath (a - d) (a + e)) z
 proof (rule winding_number_from_innerpath
       [of subpath 0 t q a-d a+e subpath 1 t q linepath (a - d) (a + e)
         z - winding_number (subpath t 0 q +++ linepath (a - d) (a + e)) z],
      simp_all add: q01 qt pa01_Un reversepath_subpath)
   show simple_path (subpath 0 t q) simple_path (subpath 1 t q)
     by (simp_all add: ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 0› ‹t ≠ 1› sim-
ple_path_subpath)
   show simple_path (linepath (a - d) (a + e))
     using ad_not_ae by blast
   show path_image (subpath 0 t q) ∩ path_image (subpath 1 t q) = {a - d, a +
e}  (is ?lhs = ?rhs)
   proof
     show ?lhs ⊆ ?rhs
       using ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 1› q_ends qt q01
```

by (*force simp*: *pathfinish_def qt simple_path_def loop_free_def path_image_subpath*)
show *?rhs* ⊆ *?lhs*
using ‹*0* ≤ *t*› ‹*t* ≤ *1*› *q01 qt* **by** (*force simp*: *path_image_subpath*)
qed
show *path_image* (*subpath 0 t q*) ∩ *closed_segment* (*a* − *d*) (*a* + *e*) = {*a* − *d*, *a* + *e*} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs* **using** *paq_Int_cs pa01_Un* **by** *fastforce*
show *?rhs* ⊆ *?lhs* **using** ‹*0* ≤ *t*› ‹*t* ≤ *1*› *q01 qt* **by** (*force simp*: *path_image_subpath*)
qed
show *path_image* (*subpath 1 t q*) ∩ *closed_segment* (*a* − *d*) (*a* + *e*) = {*a* − *d*, *a* + *e*} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs* **by** (*auto simp*: *pa_01q* [*symmetric*])
show *?rhs* ⊆ *?lhs* **using** ‹*0* ≤ *t*› ‹*t* ≤ *1*› *q01 qt* **by** (*force simp*: *path_image_subpath*)
qed
show *closed_segment* (*a* − *d*) (*a* + *e*) ∩ *inside* (*path_image q*) ≠ {}
using *a a_in_de open_closed_segment pa01_Un q_eq_p* **by** *fastforce*
show *z* ∈ *inside* (*path_image* (*subpath 0 t q*) ∪ *closed_segment* (*a* − *d*) (*a* + *e*))
by (*metis path_image_join path_image_linepath path_image_subpath_commute pathfinish_subpath pathstart_linepath q01*(*1*) *zin*)
show *winding_number* (*subpath 0 t q* +++ *linepath* (*a* + *e*) (*a* − *d*)) *z* = − *winding_number* (*subpath t 0 q* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z*
using *z_notin_ed z_notin_0t* ‹*0* ≤ *t*› ‹*simple_path q*› ‹*t* ≤ *1*›
by (*simp add*: *simple_path_imp_path qt q01 path_image_subpath_commute closed_segment_commute winding_number_join winding_number_reversepath* [*symmetric*])
show − *d* ≠ *e*
using *ad_not_ae* **by** *auto*
show *winding_number* (*subpath t 0 q* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* ≠ *0*
using *z1* **by** *auto*
qed
show *?thesis*
proof
show *z* ∈ *inside* (*path_image p*)
using *q_eq_p z_in_q* **by** *auto*
then have [*simp*]: *z* ∉ *path_image q*
by (*metis disjoint_iff_not_equal inside_no_overlap q_eq_p*)
have [*simp*]: *z* ∉ *path_image* (*subpath 1 t q*)
using *inside_def pa01_Un z_in_q* **by** *fastforce*
have *winding_number*(*subpath 0 t q* +++ *subpath t 1 q*) *z* = *winding_number*(*subpath 0 1 q*) *z*
using *z_notin_0t* ‹*0* ≤ *t*› ‹*simple_path q*› ‹*t* ≤ *1*›
by (*simp add*: *simple_path_imp_path qt path_image_subpath_commute winding_number_join winding_number_subpath_combine*)
with *wn_q* **have** *winding_number* (*subpath t 0 q* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* = − *winding_number q z*
by *auto*
with *z1* **have** *cmod* (*winding_number q z*) = *1*

```
    by simp
  with z1 wn_q_eq_wn_p show cmod (winding_number p z) = 1
    using z1 wn_q_eq_wn_p  by (simp add: ‹z ∈ inside (path_image p)›)
  qed
qed

proposition simple_closed_path_winding_number_inside:
  assumes simple_path γ
  obtains ⋀z. z ∈ inside(path_image γ) ⟹ winding_number γ z = 1
        | ⋀z. z ∈ inside(path_image γ) ⟹ winding_number γ z = −1
proof (cases pathfinish γ = pathstart γ)
  case True
  have path γ
    by (simp add: assms simple_path_imp_path)
  then have const: winding_number γ constant_on inside(path_image γ)
  proof (rule winding_number_constant)
    show connected (inside(path_image γ))
      by (simp add: Jordan_inside_outside True assms)
  qed (use inside_no_overlap True in auto)
 obtain z where zin: z ∈ inside (path_image γ) and z1: cmod (winding_number
γ z) = 1
    using simple_closed_path_wn3 [of γ] True assms by blast
  have winding_number γ z ∈ ℤ
    using zin integer_winding_number [OF ‹path γ› True] inside_def by blast
  moreover have real_of_int x = − 1 ⟷ x = −1 for x
    by linarith
  ultimately consider winding_number γ z = 1 | winding_number γ z = −1
    using z1 by (auto simp: Ints_def abs_if split: if_split_asm)
  with that const zin show ?thesis
    unfolding constant_on_def by metis
next
  case False
  then show ?thesis
    using inside_simple_curve_imp_closed assms that(2) by blast
qed

lemma simple_closed_path_abs_winding_number_inside:
  assumes simple_path γ z ∈ inside(path_image γ)
    shows |Re (winding_number γ z)| = 1
 by (metis assms norm_minus_cancel norm_one one_complex.simps(1) real_norm_def
simple_closed_path_winding_number_inside uminus_complex.simps(1))

lemma simple_closed_path_norm_winding_number_inside:
  assumes simple_path γ z ∈ inside(path_image γ)
  shows norm (winding_number γ z) = 1
proof −
  have pathfinish γ = pathstart γ
    using assms inside_simple_curve_imp_closed by blast
  with assms integer_winding_number have winding_number γ z ∈ ℤ
```

```
    by (simp add: inside_def simple_path_def)
  then show ?thesis
   by (metis assms norm_minus_cancel norm_one simple_closed_path_winding_number_inside)
qed

lemma simple_closed_path_winding_number_cases:
  assumes simple_path γ pathfinish γ = pathstart γ z ∉ path_image γ
  shows winding_number γ z ∈ {−1,0,1}
proof −
  consider z ∈ inside (path_image γ) | z ∈ outside (path_image γ)
    by (metis ComplI UnE assms(3) inside_Un_outside)
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
      using assms simple_closed_path_winding_number_inside by auto
  next
    case 2
    then show ?thesis
      using assms simple_path_def winding_number_zero_in_outside by blast
  qed
qed

lemma simple_closed_path_winding_number_pos:
  ⟦simple_path γ; pathfinish γ = pathstart γ; z ∉ path_image γ; 0 < Re(winding_number
γ z)⟧
    ⟹ winding_number γ z = 1
using simple_closed_path_winding_number_cases
  by fastforce
```

3.9 Winding number for rectangular paths

```
proposition winding_number_rectpath:
  assumes z ∈ box a1 a3
  shows   winding_number (rectpath a1 a3) z = 1
proof −
  from assms have less: Re a1 < Re a3 Im a1 < Im a3
    by (auto simp: in_box_complex_iff)
  define a2 a4 where a2 = Complex (Re a3) (Im a1) and a4 = Complex (Re
a1) (Im a3)
  let ?l1 = linepath a1 a2 and ?l2 = linepath a2 a3
  and ?l3 = linepath a3 a4 and ?l4 = linepath a4 a1
  from assms and less have z ∉ path_image (rectpath a1 a3)
    by (auto simp: path_image_rectpath_cbox_minus_box)
  also have path_image (rectpath a1 a3) =
              path_image ?l1 ∪ path_image ?l2 ∪ path_image ?l3 ∪ path_image
?l4
    by (simp add: rectpath_def Let_def path_image_join Un_assoc a2_def a4_def)
  finally have z ∉ ... .
```

```
  moreover have ∀ l∈{?l1,?l2,?l3,?l4}. Re (winding_number l z) > 0
    unfolding ball_simps HOL.simp_thms a2_def a4_def
    by (intro conjI; (rule winding_number_linepath_pos_lt;
       (insert assms, auto simp: a2_def a4_def in_box_complex_iff mult_neg_neg))+)
  ultimately have Re (winding_number (rectpath a1 a3) z) > 0
     by (simp add: winding_number_join path_image_join rectpath_def Let_def
a2_def a4_def)
  thus winding_number (rectpath a1 a3) z = 1 using assms less
    by (intro simple_closed_path_winding_number_pos simple_path_rectpath)
       (auto simp: path_image_rectpath_cbox_minus_box)
qed

proposition winding_number_rectpath_outside:
  assumes Re a1 ≤ Re a3 Im a1 ≤ Im a3
  assumes z ∉ cbox a1 a3
  shows   winding_number (rectpath a1 a3) z = 0
  using assms by (intro winding_number_zero_outside[OF _ _ _ assms(3)]
                        path_image_rectpath_subset_cbox) simp_all
```

A per-function version for continuous logs, a kind of monodromy

```
proposition winding_number_compose_exp:
  assumes path p
  shows winding_number (exp ∘ p) 0 = (pathfinish p − pathstart p) / (2 ∗ of_real
pi ∗ ı)
proof −
  obtain e where 0 < e and e: ⋀t. t ∈ {0..1} ⟹ e ≤ norm(exp(p t))
  proof
    have closed (path_image (exp ∘ p))
       by (simp add: assms closed_path_image holomorphic_on_exp holomor-
phic_on_imp_continuous_on path_continuous_image)
    then show 0 < setdist {0} (path_image (exp ∘ p))
    by (metis exp_not_eq_zero imageE image_comp infdist_eq_setdist infdist_pos_not_in_closed
path_defs(4) path_image_nonempty)
  next
    fix t::real
    assume t ∈ {0..1}
    have setdist {0} (path_image (exp ∘ p)) ≤ dist 0 (exp (p t))
    proof (rule setdist_le_dist)
      show exp (p t) ∈ path_image (exp ∘ p)
        using ‹t ∈ {0..1}› path_image_def by fastforce+
    qed auto
    then show setdist {0} (path_image (exp ∘ p)) ≤ cmod (exp (p t))
      by simp
  qed
  have bounded (path_image p)
    by (simp add: assms bounded_path_image)
  then obtain B where 0 < B and B: path_image p ⊆ cball 0 B
    by (meson bounded_pos mem_cball_0 subsetI)
  let ?B = cball (0::complex) (B+1)
```

have *uniformly_continuous_on ?B exp*
using *holomorphic_on_exp holomorphic_on_imp_continuous_on*
by (*force intro*: *compact_uniformly_continuous*)
then obtain *d* **where** *d* > *0*
and *d*: ⋀*x x'*. ⟦*x*∈*?B*; *x'*∈*?B*; *dist x' x* < *d*⟧ ⟹ *norm* (*exp x'* − *exp x*) < *e*
using ‹*e* > *0*› **by** (*auto simp*: *uniformly_continuous_on_def dist_norm*)
then have *min 1 d* > *0*
by *force*
then obtain *g* **where** *pfg*: *polynomial_function g* **and** *g 0* = *p 0 g 1* = *p 1*
and *gless*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *norm*(*g t* − *p t*) < *min 1 d*
using *path_approx_polynomial_function* [*OF* ‹*path p*›] ‹*d* > *0*› ‹*0* < *e*›
unfolding *pathfinish_def pathstart_def* **by** *blast*
have *winding_number* (*exp* ∘ *p*) *0* = *winding_number* (*exp* ∘ *g*) *0*
proof (*rule winding_number_nearby_paths_eq* [*symmetric*])
show *path* (*exp* ∘ *p*) *path* (*exp* ∘ *g*)
by (*simp_all add*: *pfg assms holomorphic_on_exp holomorphic_on_imp_continuous_on path_continuous_image path_polynomial_function*)
next
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
with *gless* **have** *norm*(*g t* − *p t*) < *1*
using *min_less_iff_conj* **by** *blast*
moreover have *ptB*: *norm* (*p t*) ≤ *B*
using *B t* **by** (*force simp*: *path_image_def*)
ultimately have *cmod* (*g t*) ≤ *B* + *1*
by (*meson add_mono_thms_linordered_field*(*4*) *le_less_trans less_imp_le norm_triangle_sub*)
with *ptB gless t* **have** *cmod* ((*exp* ∘ *g*) *t* − (*exp* ∘ *p*) *t*) < *e*
by (*auto simp*: *dist_norm d*)
with *e t* **show** *cmod* ((*exp* ∘ *g*) *t* − (*exp* ∘ *p*) *t*) < *cmod* ((*exp* ∘ *p*) *t* − *0*)
by *fastforce*
qed (*use* ‹*g 0* = *p 0*› ‹*g 1* = *p 1*› **in** ‹*auto simp*: *pathfinish_def pathstart_def*›)
also have ... = *1* / (*of_real* (*2* ∗ *pi*) ∗ i) ∗ *contour_integral* (*exp* ∘ *g*) (λ*w*. *1* / (*w* − *0*))
proof (*rule winding_number_valid_path*)
have *continuous_on* (*path_image g*) (*deriv exp*)
by (*metis DERIV_exp DERIV_imp_deriv continuous_on_cong holomorphic_on_exp holomorphic_on_imp_continuous_on*)
then show *valid_path* (*exp* ∘ *g*)
by (*simp add*: *field_differentiable_within_exp pfg valid_path_compose valid_path_polynomial_function*)
show *0* ∉ *path_image* (*exp* ∘ *g*)
by (*auto simp*: *path_image_def*)
qed
also have ... = *1* / (*of_real* (*2* ∗ *pi*) ∗ i) ∗ *integral* {*0..1*} (λ*x*. *vector_derivative g* (*at x*))
proof (*simp add*: *contour_integral_integral*, *rule integral_cong*)
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
show *vector_derivative* (*exp* ∘ *g*) (*at t*) / *exp* (*g t*) = *vector_derivative g* (*at t*)

proof −
have (*exp* ∘ *g has_vector_derivative vector_derivative* (*exp* ∘ *g*) (*at t*)) (*at t*)
by (*meson DERIV_exp differentiable_def field_vector_diff_chain_at has_vector_derivative_def has_vector_derivative_polynomial_function pfg vector_derivative_works*)
moreover have (*exp* ∘ *g has_vector_derivative vector_derivative g* (*at t*) ∗ *exp* (*g t*)) (*at t*)
by (*metis DERIV_exp field_vector_diff_chain_at has_vector_derivative_polynomial_function pfg vector_derivative_at*)
ultimately show *?thesis*
by (*simp add*: *divide_simps*, *rule vector_derivative_unique_at*)
qed
qed
also have ... = (*pathfinish p* − *pathstart p*) / (*2* ∗ *of_real pi* ∗ i)
proof −
have ((λ*x*. *vector_derivative g* (*at x*)) *has_integral g 1* − *g 0*) {*0..1*}
by (*meson differentiable_at_polynomial_function fundamental_theorem_of_calculus has_vector_derivative_at_within pfg vector_derivative_works zero_le_one*)
then show *?thesis*
unfolding *pathfinish_def pathstart_def*
using ‹*g 0* = *p 0*› ‹*g 1* = *p 1*› **by** *auto*
qed
finally show *?thesis* .
qed

end

4 Cauchy's Integral Formula

theory *Cauchy_Integral_Formula*
imports *Winding_Numbers*
begin

4.1 Proof

lemma *Cauchy_integral_formula_weak*:
assumes *S*: *convex S* **and** *finite k* **and** *conf*: *continuous_on S f*
and *fcd*: (⋀*x*. *x* ∈ *interior S* − *k* ⟹ *f field_differentiable at x*)
and *z*: *z* ∈ *interior S* − *k* **and** *vpg*: *valid_path* γ
and *pasz*: *path_image* γ ⊆ *S* − {*z*} **and** *loop*: *pathfinish* γ = *pathstart* γ
shows ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral* (*2*∗*pi* ∗ i ∗ *winding_number* γ *z* ∗ *f z*)) γ
proof −
let *?fz* = λ*w*. (*f w* − *f z*)/(*w* − *z*)
obtain *f'* **where** *f'*: (*f has_field_derivative f'*) (*at z*)
using *fcd* [*OF z*] **by** (*auto simp*: *field_differentiable_def*)
have *pas*: *path_image* γ ⊆ *S* **and** *znotin*: *z* ∉ *path_image* γ **using** *pasz* **by**

```
blast+
  have c: continuous (at x within S) (λw. if w = z then f' else (f w − f z) / (w −
z)) if x ∈ S for x
  proof (cases x = z)
    case True then show ?thesis
    using LIM_equal [of z ?fz λw. if w = z then f' else ?fz w] has_field_derivativeD
[OF f']
      by (force simp add: continuous_within Lim_at_imp_Lim_at_within)
  next
    case False
    then have dxz: dist x z > 0 by auto
    have cf: continuous (at x within S) f
      using conf continuous_on_eq_continuous_within that by blast
    have continuous (at x within S) (λw. (f w − f z) / (w − z))
      by (rule cf continuous_intros | simp add: False)+
    then show ?thesis
    using continuous_transform_within [OF _ dxz that] by (force simp: dist_commute)
  qed
  have fink': finite (insert z k) using ‹finite k› by blast
  have ∗: ((λw. if w = z then f' else ?fz w) has_contour_integral 0) γ
  proof (rule Cauchy_theorem_convex [OF _ S fink' _ vpg pas loop])
    show (λw. if w = z then f' else ?fz w) field_differentiable at w
      if w ∈ interior S − insert z k for w
    proof (rule field_differentiable_transform_within)
      show (λw. ?fz w) field_differentiable at w
        using that by (intro derivative_intros fcd; simp)
    qed (use that in ‹auto simp add: dist_pos_lt dist_commute›)
  qed (use c in ‹force simp: continuous_on_eq_continuous_within›)
  show ?thesis
    apply (rule has_contour_integral_eq)
    using znotin has_contour_integral_add [OF has_contour_integral_lmul [OF
has_contour_integral_winding_number [OF vpg znotin], of f z] ∗]
    apply (auto simp: ac_simps divide_simps)
    done
qed

theorem Cauchy_integral_formula_convex_simple:
  assumes convex S and holf: f holomorphic_on S and z ∈ interior S valid_path
γ path_image γ ⊆ S − {z}
      pathfinish γ = pathstart γ
    shows ((λw. f w / (w − z)) has_contour_integral (2∗pi ∗ i ∗ winding_number
γ z ∗ f z)) γ
proof −
  have ⋀x. x ∈ interior S ⟹ f field_differentiable at x
    using holf at_within_interior holomorphic_onD interior_subset by fastforce
  then show ?thesis
    using assms
    by (intro Cauchy_integral_formula_weak [where k = {}]) (auto simp: holo-
morphic_on_imp_continuous_on)
```

qed

Hence the Cauchy formula for points inside a circle.

theorem *Cauchy_integral_circlepath*:
assumes *contf*: *continuous_on* (*cball z r*) *f* **and** *holf*: *f holomorphic_on* (*ball z r*) **and** *wz*: *norm*(*w* − *z*) < *r*
shows ((λ*u*. *f u*/(*u* − *w*)) *has_contour_integral* (2 ∗ *of_real pi* ∗ i ∗ *f w*)) (*circlepath z r*)
proof −
have *r* > 0
using *assms le_less_trans norm_ge_zero* **by** *blast*
have ((λ*u*. *f u* / (*u* − *w*)) *has_contour_integral* (2 ∗ *pi*) ∗ i ∗ *winding_number* (*circlepath z r*) *w* ∗ *f w*) (*circlepath z r*)
proof (*rule Cauchy_integral_formula_weak* [**where** *S* = *cball z r* **and** *k* = {}])
show ⋀*x*. *x* ∈ *interior* (*cball z r*) − {} ⟹ *f field_differentiable at x*
using *holf holomorphic_on_imp_differentiable_at* **by** *auto*
have *w* ∉ *sphere z r*
by *simp* (*metis dist_commute dist_norm not_le order_refl wz*)
then show *path_image* (*circlepath z r*) ⊆ *cball z r* − {*w*}
using ‹*r* > 0› **by** (*auto simp add*: *cball_def sphere_def*)
qed (*use wz* **in** ‹*simp_all add*: *dist_norm norm_minus_commute contf*›)
then show *?thesis*
by (*simp add*: *winding_number_circlepath assms*)
qed

corollary *Cauchy_integral_circlepath_simple*:
assumes *f holomorphic_on cball z r norm*(*w* − *z*) < *r*
shows ((λ*u*. *f u*/(*u* − *w*)) *has_contour_integral* (2 ∗ *of_real pi* ∗ i ∗ *f w*)) (*circlepath z r*)
using *assms* **by** (*force simp*: *holomorphic_on_imp_continuous_on holomorphic_on_subset Cauchy_integral_circlepath*)

4.2 General stepping result for derivative formulas

lemma *Cauchy_next_derivative*:
assumes *continuous_on* (*path_image* γ) *f* ′
and *leB*: ⋀*t*. *t* ∈ {0..1} ⟹ *norm* (*vector_derivative* γ (*at t*)) ≤ *B*
and *int*: ⋀*w*. *w* ∈ *S* − *path_image* γ ⟹ ((λ*u*. *f* ′ *u* / (*u* − *w*)^*k*) *has_contour_integral* *f w*) γ
and *k*: *k* ≠ 0
and *open S*
and γ: *valid_path* γ
and *w*: *w* ∈ *S* − *path_image* γ
shows (λ*u*. *f* ′ *u* / (*u* − *w*)^(*Suc k*)) *contour_integrable_on* γ
and (*f has_field_derivative* (*k* ∗ *contour_integral* γ (λ*u*. *f* ′ *u*/(*u* − *w*)^(*Suc k*)))) (*at w*) (**is** *?thes2*)

```
proof −
  have open (S − path_image γ) using ‹open S› closed_valid_path_image γ by
blast
  then obtain d where d>0 and d: ball w d ⊆ S − path_image γ using w
    using open_contains_ball by blast
  have [simp]: ⋀n. cmod (1 + of_nat n) = 1 + of_nat n
    by (metis norm_of_nat of_nat_Suc)
  have cint: ⋀x. ⟦x ≠ w; cmod (x − w) < d⟧
        ⟹ (λz. (f′ z / (z − x) ^ k − f′ z / (z − w) ^ k) / (x ∗ k − w ∗ k))
contour_integrable_on γ
    using int w d
   apply (intro contour_integrable_div contour_integrable_diff has_contour_integral_integrable)
    by (force simp: dist_norm norm_minus_commute)
  have 1: ∀F n in at w. (λx. f′ x ∗ (inverse (x − n) ^ k − inverse (x − w) ^ k)
/ (n − w) / of_nat k)
                    contour_integrable_on γ
    unfolding eventually_at
    apply (rule_tac x=d in exI)
    apply (simp add: ‹d > 0› dist_norm field_simps cint)
    done
  have bim_g: bounded (image f′ (path_image γ))
   by (simp add: compact_imp_bounded compact_continuous_image compact_valid_path_image
assms)
  then obtain C where C > 0 and C: ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ cmod (f′ (γ x))
≤ C
    by (force simp: bounded_pos path_image_def)
  have twom: ∀F n in at w.
            ∀x∈path_image γ.
             cmod ((inverse (x − n) ^ k − inverse (x − w) ^ k) / (n − w) / k −
inverse (x − w) ^ Suc k) < e
        if 0 < e for e
  proof −
    have ∗: cmod ((inverse (x − u) ^ k − inverse (x − w) ^ k) / ((u − w) ∗ k)
− inverse (x − w) ^ Suc k)  < e
          if x: x ∈ path_image γ and u ≠ w and uwd: cmod (u − w) < d/2
             and uw_less: cmod (u − w) < e ∗ (d/2) ^(k+2) / (1 + real k)
          for u x
    proof −
      define ff where [abs_def]:
        ff n w =
          (if n = 0 then inverse(x − w)^k
           else if n = 1 then k / (x − w)^(Suc k)
           else (k ∗ of_real(Suc k)) / (x − w)^(k + 2)) for n :: nat and w
      have km1: ⋀z::complex. z ≠ 0 ⟹ z ^ (k − Suc 0) = z ^ k / z
       by (simp add: field_simps) (metis Suc_pred ‹k ≠ 0› neq0_conv power_Suc)
      have ff1: (ff i has_field_derivative ff (Suc i) z) (at z within ball w (d/2))
            if z ∈ ball w (d/2) i ≤ 1 for i z
      proof −
        have z ∉ path_image γ
```

```
      using ‹x ∈ path_image γ› d that ball_divide_subset_numeral by blast
    then have xz[simp]: x ≠ z using ‹x ∈ path_image γ› by blast
    then have neq: x * x + z * z ≠ x * (z * 2)
      by (blast intro: dest!: sum_sqs_eq)
    with xz have ⋀v. v ≠ 0 ⟹ (x * x + z * z) * v ≠ (x * (z * 2) * v) by
auto
    then have neqq: ⋀v. v ≠ 0 ⟹ x * (x * v) + z * (z * v) ≠ x * (z * (2 *
v))
      by (simp add: algebra_simps)
    show ?thesis using ‹i ≤ 1›
      apply (simp add: ff_def dist_norm Nat.le_Suc_eq km1, safe)
     apply (rule derivative_eq_intros | simp add: km1 | simp add: field_simps
neq neqq)+
      done
  qed
  { fix a::real and b::real assume ab: a > 0 b > 0
    then have k * (1 + real k) * (1 / a) ≤ k * (1 + real k) * (4 / b) ⟷ b
≤ 4 * a
      by (subst mult_le_cancel_left_pos)
        (use ‹k ≠ 0› in ‹auto simp: divide_simps›)
    with ab have real k * (1 + real k) / a ≤ (real k * 4 + real k * real k * 4)
/ b ⟷ b ≤ 4 * a
      by (simp add: field_simps)
  } note canc = this
  have ff2: cmod (ff (Suc 1) v) ≤ real (k * (k + 1)) / (d/2) ^(k + 2)
        if v ∈ ball w (d/2) for v
  proof −
    have lessd: ⋀z. cmod (γ z − v) < d/2 ⟹ cmod (w − γ z) < d
      by (metis that norm_minus_commute norm_triangle_half_r dist_norm
mem_ball)
    have d/2 ≤ cmod (x − v) using d x that
      using lessd d x
    by (auto simp add: dist_norm path_image_def ball_def not_less [symmetric]
del: divide_const_simps)
    then have d ≤ cmod (x − v) * 2
      by (simp add: field_split_simps)
    then have dpow_le: d ^(k+2) ≤ (cmod (x − v) * 2) ^(k+2)
      using ‹0 < d› order_less_imp_le power_mono by blast
    have x ≠ v using that
      using ‹x ∈ path_image γ› ball_divide_subset_numeral d by fastforce
    then show ?thesis
  using ‹d > 0› apply (simp add: ff_def norm_mult norm_divide norm_power
dist_norm canc)
    using dpow_le apply (simp add: field_split_simps)
    done
  qed
  have ub: u ∈ ball w (d/2)
    using uwd by (simp add: dist_commute dist_norm)
  have cmod (inverse (x − u) ^k − (inverse (x − w) ^k + of_nat k * (u −
```

```
w) / ((x − w) * (x − w) ^ k)))
                ≤ (real k * 4 + real k * real k * 4) * (cmod (u − w) * cmod (u −
w)) / (d * (d * (d/2) ^ k))
      using complex_Taylor [OF _ ff1 ff2 _ ub, of w, simplified]
      by (simp add: ff_def ‹0 < d›)
    then have cmod (inverse (x − u) ^ k − (inverse (x − w) ^ k + of_nat k *
(u − w) / ((x − w) * (x − w) ^ k)))
                ≤ (cmod (u − w) * real k) * (1 + real k) * cmod (u − w) / (d/2)
^ (k+2)
      by (simp add: field_simps)
    then have cmod (inverse (x − u) ^ k − (inverse (x − w) ^ k + of_nat k *
(u − w) / ((x − w) * (x − w) ^ k)))
              / (cmod (u − w) * real k)
              ≤ (1 + real k) * cmod (u − w) / (d/2) ^ (k+2)
        using ‹k ≠ 0› ‹u ≠ w› by (simp add: mult_ac zero_less_mult_iff
pos_divide_le_eq)
    also have ... < e
      using uw_less ‹0 < d› by (simp add: mult_ac divide_simps)
    finally have e: cmod (inverse (x−u)^k − (inverse (x−w)^k + of_nat k *
(u−w) / ((x−w) * (x−w)^k)))
                    / cmod ((u − w) * real k)   <   e
      by (simp add: norm_mult)
    have x ≠ u
    using uwd ‹0 < d› x d by (force simp: dist_norm ball_def norm_minus_commute)
    show ?thesis
      apply (rule le_less_trans [OF _ e])
      using ‹k ≠ 0› ‹x ≠ u› ‹u ≠ w›
      apply (simp add: field_simps norm_divide [symmetric])
      done
  qed
  show ?thesis
    unfolding eventually_at
    apply (rule_tac x = min (d/2) ((e*(d/2)^(k + 2))/(Suc k)) in exI)
    apply (force simp: ‹d > 0› dist_norm that simp del: power_Suc intro: *)
    done
 qed
 have 2: uniform_limit (path_image γ) (λn x. f' x * (inverse (x − n) ^ k −
inverse (x − w) ^ k) / (n − w) / of_nat k) (λx. f' x / (x − w) ^ Suc k) (at w)
  unfolding uniform_limit_iff dist_norm
 proof clarify
  fix e::real
  assume 0 < e
  have *: cmod (f' (γ x) * (inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) /
((u − w) * k) −
                f' (γ x) / ((γ x − w) * (γ x − w) ^ k)) < e
        if ec: cmod ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u −
w) * k) −
                inverse (γ x − w) * inverse (γ x − w) ^ k) < e / C
          and x: 0 ≤ x x ≤ 1
```

```
          for u x
    proof (cases (f' (γ x)) = 0)
      case True then show ?thesis by (simp add: ‹0 < e›)
    next
      case False
      have cmod (f' (γ x)) * (inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) * k) −
                        f' (γ x) / ((γ x − w) * (γ x − w) ^ k)) =
           cmod (f' (γ x)) * ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) * k) −
                              inverse (γ x − w) * inverse (γ x − w) ^ k))
        by (simp add: field_simps)
      also have ... = cmod (f' (γ x)) *
                      cmod ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u −
w) * k) −
                              inverse (γ x − w) * inverse (γ x − w) ^ k)
        by (simp add: norm_mult)
      also have ... < cmod (f' (γ x)) * (e/C)
        using False mult_strict_left_mono [OF ec] by force
      also have ... ≤ e using C
      by (metis False ‹0 < e› frac_le less_eq_real_def mult.commute pos_le_divide_eq
x zero_less_norm_iff)
      finally show ?thesis .
    qed
    show ∀F n in at w.
           ∀x∈path_image γ.
            cmod (f' x * (inverse (x − n) ^ k − inverse (x − w) ^ k) / (n − w)
/ of_nat k − f' x / (x − w) ^ Suc k) < e
        using twom [OF divide_pos_pos [OF ‹0 < e› ‹C > 0›]]    unfolding
path_image_def
      by (force intro: * elim: eventually_mono)
  qed
  show (λu. f' u / (u − w) ^ (Suc k)) contour_integrable_on γ
    by (rule contour_integral_uniform_limit [OF 1 2 leB γ]) auto
  have *: (λn. contour_integral γ (λx. f' x * (inverse (x − n) ^ k − inverse (x −
w) ^ k) / (n − w) / k))
          −w→ contour_integral γ (λu. f' u / (u − w) ^ (Suc k))
    by (rule contour_integral_uniform_limit [OF 1 2 leB γ]) auto
  have **: contour_integral γ (λx. f' x * (inverse (x − u) ^ k − inverse (x − w)
^ k) / ((u − w) * k)) =
           (f u − f w) / (u − w) / k
    if dist u w < d for u
  proof −
    have u: u ∈ S − path_image γ
      by (metis subsetD d dist_commute mem_ball that)
    have §: ((λx. f' x * inverse (x − u) ^ k) has_contour_integral f u) γ
          ((λx. f' x * inverse (x − w) ^ k) has_contour_integral f w) γ
      using u w by (simp_all add: field_simps int)
    show ?thesis
```

```
    apply (rule contour_integral_unique)
   apply (simp add: diff_divide_distrib algebra_simps § has_contour_integral_diff
has_contour_integral_div)
    done
 qed
 show ?thes2
   apply (simp add: has_field_derivative_iff del: power_Suc)
   apply (rule Lim_transform_within [OF tendsto_mult_left [OF *] ‹0 < d› ])
   apply (simp add: ‹k ≠ 0› **)
   done
qed

lemma Cauchy_next_derivative_circlepath:
 assumes contf: continuous_on (path_image (circlepath z r)) f
     and int: ⋀w. w ∈ ball z r ⟹ ((λu. f u / (u − w)^k) has_contour_integral
g w) (circlepath z r)
     and k: k ≠ 0
     and w: w ∈ ball z r
   shows (λu. f u / (u − w)^(Suc k)) contour_integrable_on (circlepath z r)
         (is ?thes1)
    and (g has_field_derivative (k * contour_integral (circlepath z r) (λu. f u/(u
− w)^(Suc k)))) (at w)
         (is ?thes2)
proof −
 have r > 0 using w
   using ball_eq_empty by fastforce
 have wim: w ∈ ball z r − path_image (circlepath z r)
   using w by (auto simp: dist_norm)
 show ?thes1 ?thes2
  by (rule Cauchy_next_derivative [OF contf _ int k open_ball valid_path_circlepath
wim, where B = 2 * pi * |r|];
      auto simp: vector_derivative_circlepath norm_mult)+
qed
```

In particular, the first derivative formula.

```
lemma Cauchy_derivative_integral_circlepath:
 assumes contf: continuous_on (cball z r) f
     and holf: f holomorphic_on ball z r
     and w: w ∈ ball z r
   shows (λu. f u/(u − w)^2) contour_integrable_on (circlepath z r)
         (is ?thes1)
    and (f has_field_derivative (1 / (2 * of_real pi * i) * contour_integral(circlepath
z r) (λu. f u / (u − w)^2))) (at w)
         (is ?thes2)
proof −
 have [simp]: r ≥ 0 using w
   using ball_eq_empty by fastforce
 have f: continuous_on (path_image (circlepath z r)) f
   by (rule continuous_on_subset [OF contf]) (force simp: cball_def sphere_def)
```

have *int*: $\bigwedge w.\ dist\ z\ w < r \Longrightarrow$
((λ*u*. *f u* / (*u* − *w*)) *has_contour_integral* (λ*x*. 2 ∗ *of_real pi* ∗ i ∗ *f x*) *w*) (*circlepath z r*)
by (*rule Cauchy_integral_circlepath* [*OF contf holf*]) (*simp add*: *dist_norm norm_minus_commute*)
show *?thes1*
apply (*simp add*: *power2_eq_square*)
apply (*rule Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1*, *simplified*])
apply (*blast intro*: *int*)
done
have ((λ*x*. 2 ∗ *of_real pi* ∗ i ∗ *f x*) *has_field_derivative contour_integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)^2)) (*at w*)
apply (*simp add*: *power2_eq_square*)
apply (*rule Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1* **and** *g* = λ*x*. 2 ∗ *of_real pi* ∗ i ∗ *f x*, *simplified*])
apply (*blast intro*: *int*)
done
then have *fder*: (*f has_field_derivative contour_integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)^2) / (2 ∗ *of_real pi* ∗ i)) (*at w*)
by (*rule DERIV_cdivide* [**where** *f* = λ*x*. 2 ∗ *of_real pi* ∗ i ∗ *f x* **and** *c* = 2 ∗ *of_real pi* ∗ i, *simplified*])
show *?thes2*
by *simp* (*rule fder*)
qed

4.3 Existence of all higher derivatives

proposition *derivative_is_holomorphic*:
assumes *open S*
and *fder*: $\bigwedge z.\ z \in S \Longrightarrow$ (*f has_field_derivative f' z*) (*at z*)
shows *f' holomorphic_on S*
proof −
have ∗: ∃*h*. (*f' has_field_derivative h*) (*at z*) **if** *z* ∈ *S* **for** *z*
proof −
obtain *r* **where** *r* > *0* **and** *r*: *cball z r* ⊆ *S*
using *open_contains_cball* ‹*z* ∈ *S*› ‹*open S*› **by** *blast*
then have *holf_cball*: *f holomorphic_on cball z r*
unfolding *holomorphic_on_def*
using *field_differentiable_at_within field_differentiable_def fder* **by** *fastforce*
then have *continuous_on* (*path_image* (*circlepath z r*)) *f*
using ‹*r* > *0*› **by** (*force elim*: *holomorphic_on_subset* [*THEN holomorphic_on_imp_continuous_on*])
then have *contfpi*: *continuous_on* (*path_image* (*circlepath z r*)) (λ*x*. *1*/(2 ∗ *of_real pi*∗i) ∗ *f x*)
by (*auto intro*: *continuous_intros*)+
have *contf_cball*: *continuous_on* (*cball z r*) *f* **using** *holf_cball*
by (*simp add*: *holomorphic_on_imp_continuous_on holomorphic_on_subset*)
have *holf_ball*: *f holomorphic_on ball z r* **using** *holf_cball*

```
    using ball_subset_cball holomorphic_on_subset by blast
  { fix w  assume w: w ∈ ball z r
    have intf: (λu. f u / (u − w)²) contour_integrable_on circlepath z r
      by (blast intro: w Cauchy_derivative_integral_circlepath [OF contf_cball
holf_ball])
    have fder': (f has_field_derivative 1 / (2 * of_real pi * i) * contour_integral
(circlepath z r) (λu. f u / (u − w)²))
                (at w)
      by (blast intro: w Cauchy_derivative_integral_circlepath [OF contf_cball
holf_ball])
    have f'_eq: f' w = contour_integral (circlepath z r) (λu. f u / (u − w)²) /
(2 * of_real pi * i)
     using fder' ball_subset_cball r w by (force intro: DERIV_unique [OF fder])
    have ((λu. f u / (u − w)² / (2 * of_real pi * i)) has_contour_integral
          contour_integral (circlepath z r) (λu. f u / (u − w)²) / (2 * of_real
pi * i))
          (circlepath z r)
     by (rule has_contour_integral_div [OF has_contour_integral_integral [OF
intf]])
    then have ((λu. f u / (2 * of_real pi * i * (u − w)²)) has_contour_integral
          contour_integral (circlepath z r) (λu. f u / (u − w)²) / (2 * of_real
pi * i))
          (circlepath z r)
     by (simp add: algebra_simps)
    then have ((λu. f u / (2 * of_real pi * i * (u − w)²)) has_contour_integral
f' w) (circlepath z r)
     by (simp add: f'_eq)
  } note * = this
  show ?thesis
    using Cauchy_next_derivative_circlepath [OF contfpi, of 2 f'] ‹0 < r› *
    using centre_in_ball mem_ball by force
 qed
 show ?thesis
   by (simp add: holomorphic_on_open [OF ‹open S›] *)
qed

lemma holomorphic_deriv [holomorphic_intros]:
   ⟦f holomorphic_on S; open S⟧ ⟹ (deriv f) holomorphic_on S
by (metis DERIV_deriv_iff_field_differentiable at_within_open derivative_is_holomorphic
holomorphic_on_def)

lemma analytic_deriv [analytic_intros]: f analytic_on S ⟹ (deriv f) analytic_on
S
 using analytic_on_holomorphic holomorphic_deriv by auto

lemma holomorphic_higher_deriv [holomorphic_intros]: ⟦f holomorphic_on S; open
S⟧ ⟹ (deriv ^^ n) f holomorphic_on S
 by (induction n) (auto simp: holomorphic_deriv)
```

lemma *analytic_higher_deriv* [*analytic_intros*]: *f analytic_on S* $\Longrightarrow$ (*deriv* ^^ *n*) *f analytic_on S*
unfolding *analytic_on_def* **using** *holomorphic_higher_deriv* **by** *blast*

lemma *has_field_derivative_higher_deriv*:
$[\![$*f holomorphic_on S*; *open S*; *x* $\in$ *S*$]\!]$
$\Longrightarrow$ ((*deriv* ^^ *n*) *f has_field_derivative* (*deriv* ^^ (*Suc n*)) *f x*) (*at x*)
by (*metis* (*no_types*, *opaque_lifting*) *DERIV_deriv_iff_field_differentiable at_within_open comp_apply*
funpow.simps(*2*) *holomorphic_higher_deriv holomorphic_on_def*)

lemma *higher_deriv_cmult*:
assumes *f holomorphic_on A x* $\in$ *A open A*
shows (*deriv* ^^ *j*) (*λx. c* $*$ *f x*) *x* = *c* $*$ (*deriv* ^^ *j*) *f x*
using *assms*
proof (*induction j arbitrary*: *f x*)
case (*Suc j f x*)
have *deriv* ((*deriv* ^^ *j*) (*λx. c* $*$ *f x*)) *x* = *deriv* (*λx. c* $*$ (*deriv* ^^ *j*) *f x*) *x*
using *eventually_nhds_in_open*[*of A x*] *assms*(*2*,*3*) *Suc.prems*
by (*intro deriv_cong_ev refl*) (*auto elim*!: *eventually_mono simp*: *Suc.IH*)
also have ... = *c* $*$ *deriv* ((*deriv* ^^ *j*) *f*) *x* **using** *Suc.prems assms*(*2*,*3*)
by (*intro deriv_cmult holomorphic_on_imp_differentiable_at holomorphic_higher_deriv*) *auto*
finally show *?case* **by** *simp*
qed *simp_all*

lemma *valid_path_compose_holomorphic*:
assumes *valid_path g* **and** *holo:f holomorphic_on S* **and** *open S path_image g* $\subseteq$ *S*
shows *valid_path* (*f* $\circ$ *g*)
by (*meson assms holomorphic_deriv holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at*
holomorphic_on_subset subsetD valid_path_compose)

4.4 Morera's theorem

lemma *Morera_local_triangle_ball*:
assumes $\bigwedge$*z*. *z* $\in$ *S*
$\Longrightarrow$ $\exists$ *e a*. *0* < *e* $\wedge$ *z* $\in$ *ball a e* $\wedge$ *continuous_on* (*ball a e*) *f* $\wedge$
($\forall$ *b c*. *closed_segment b c* $\subseteq$ *ball a e*
$\longrightarrow$ *contour_integral* (*linepath a b*) *f* +
contour_integral (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* = *0*)
shows *f analytic_on S*
proof −
{ **fix** *z* **assume** *z* $\in$ *S*
with *assms* **obtain** *e a* **where**
0 < *e* **and** *z*: *z* $\in$ *ball a e* **and** *contf*: *continuous_on* (*ball a e*) *f*
and *0*: $\bigwedge$*b c*. *closed_segment b c* $\subseteq$ *ball a e*

```
                ⟹ contour_integral (linepath a b) f +
                   contour_integral (linepath b c) f +
                   contour_integral (linepath c a) f = 0
    by blast
  have az: dist a z < e using mem_ball z by blast
  have ∃ e>0. f holomorphic_on ball z e
  proof (intro exI conjI)
    show f holomorphic_on ball z (e − dist a z)
    proof (rule holomorphic_on_subset)
      show ball z (e − dist a z) ⊆ ball a e
        by (simp add: dist_commute ball_subset_ball_iff)
      have sub_ball: ⋀y. dist a y < e ⟹ closed_segment a y ⊆ ball a e
        by (meson ‹0 < e› centre_in_ball convex_ball convex_contains_segment
mem_ball)
      show f holomorphic_on ball a e
      using triangle_contour_integrals_starlike_primitive [OF contf_ open_ball,
of a]
          derivative_is_holomorphic[OF open_ball]
        by (force simp add: 0 ‹0 < e› sub_ball)
    qed
  qed (simp add: az)
 }
 then show ?thesis
  by (simp add: analytic_on_def)
qed

lemma Morera_local_triangle:
 assumes ⋀z. z ∈ S
       ⟹ ∃ t. open t ∧ z ∈ t ∧ continuous_on t f ∧
            (∀ a b c. convex hull {a,b,c} ⊆ t
                  ⟶ contour_integral (linepath a b) f +
                    contour_integral (linepath b c) f +
                    contour_integral (linepath c a) f = 0)
 shows f analytic_on S
proof −
 { fix z assume z ∈ S
  with assms obtain t where
       open t and z: z ∈ t and contf: continuous_on t f
    and 0: ⋀a b c. convex hull {a,b,c} ⊆ t
                ⟹ contour_integral (linepath a b) f +
                   contour_integral (linepath b c) f +
                   contour_integral (linepath c a) f = 0
    by force
  then obtain e where e>0 and e: ball z e ⊆ t
    using open_contains_ball by blast
  have [simp]: continuous_on (ball z e) f using contf
    using continuous_on_subset e by blast
  have eq0: ⋀b c. closed_segment b c ⊆ ball z e ⟹
                  contour_integral (linepath z b) f +
```

```
                    contour_integral (linepath b c) f +
                    contour_integral (linepath c z) f = 0
      by (meson 0 z ‹0 < e› centre_in_ball closed_segment_subset convex_ball
dual_order.trans e starlike_convex_subset)
    have ∃e a. 0 < e ∧ z ∈ ball a e ∧ continuous_on (ball a e) f ∧
              (∀b c. closed_segment b c ⊆ ball a e ⟶
                   contour_integral (linepath a b) f + contour_integral (linepath b
c) f + contour_integral (linepath c a) f = 0)
      using ‹e > 0› eq0 by force
  }
  then show ?thesis
    by (simp add: Morera_local_triangle_ball)
qed

proposition Morera_triangle:
    ⟦continuous_on S f; open S;
      ⋀a b c. convex hull {a,b,c} ⊆ S
            ⟶ contour_integral (linepath a b) f +
              contour_integral (linepath b c) f +
              contour_integral (linepath c a) f = 0⟧
    ⟹ f analytic_on S
  using Morera_local_triangle by blast
```

4.5 Combining theorems for higher derivatives including Leibniz rule

```
lemma higher_deriv_linear [simp]:
    (deriv ^^ n) (λw. c*w) = (λz. if n = 0 then c*z else if n = 1 then c else 0)
  by (induction n) auto

lemma higher_deriv_const [simp]: (deriv ^^ n) (λw. c) = (λw. if n=0 then c else
0)
  by (induction n) auto

lemma higher_deriv_ident [simp]:
    (deriv ^^ n) (λw. w) z = (if n = 0 then z else if n = 1 then 1 else 0)
proof (induction n)
  case (Suc n)
  then show ?case by (metis higher_deriv_linear lambda_one)
qed auto

lemma higher_deriv_id [simp]:
    (deriv ^^ n) id z = (if n = 0 then z else if n = 1 then 1 else 0)
  by (simp add: id_def)

lemma has_complex_derivative_funpow_1:
    ⟦(f has_field_derivative 1) (at z); f z = z⟧ ⟹ (f^^n has_field_derivative 1)
(at z)
proof (induction n)
```

```
  case 0
  then show ?case
    by (simp add: id_def)
next
  case (Suc n)
  then show ?case
    by (metis DERIV_chain funpow_Suc_right mult.right_neutral)
qed

lemma higher_deriv_uminus:
  assumes f holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. -(f w)) z = - ((deriv ^^ n) f z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have *: ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have ⋀x. x ∈ S ⟹ - (deriv ^^ n) f x = (deriv ^^ n) (λw. - f w) x
    by (auto simp add: Suc)
  then have ((deriv ^^ n) (λw. - f w) has_field_derivative - deriv ((deriv ^^ n)
f) z) (at z)
    using  has_field_derivative_transform_within_open [of λw. -((deriv ^^ n) f
w)]
    using * DERIV_minus Suc.prems ‹open S› by blast
  then show ?case
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_add:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. f w + g w) z = (deriv ^^ n) f z + (deriv ^^ n) g z
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have *: ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
          ((deriv ^^ n) g has_field_derivative deriv ((deriv ^^ n) g) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have ⋀x. x ∈ S ⟹ (deriv ^^ n) f x + (deriv ^^ n) g x = (deriv ^^ n) (λw. f
w + g w) x
    by (auto simp add: Suc)
  then have ((deriv ^^ n) (λw. f w + g w) has_field_derivative
        deriv ((deriv ^^ n) f) z + deriv ((deriv ^^ n) g) z) (at z)
    using  has_field_derivative_transform_within_open [of λw. (deriv ^^ n) f w
+ (deriv ^^ n) g w]
```

```
    using * Deriv.field_differentiable_add Suc.prems ‹open S› by blast
  then show ?case
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_diff:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S z ∈ S
    shows (deriv ^^ n) (λw. f w - g w) z = (deriv ^^ n) f z - (deriv ^^ n) g z
  unfolding diff_conv_add_uminus higher_deriv_add
   using assms higher_deriv_add higher_deriv_uminus holomorphic_on_minus
by presburger

lemma Suc_choose: Suc n choose k = (n choose k) + (if k = 0 then 0 else (n
choose (k - 1)))
  by (cases k) simp_all

lemma higher_deriv_mult:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. f w * g w) z =
          (∑ i = 0..n. of_nat (n choose i) * (deriv ^^ i) f z * (deriv ^^ (n - i)) g
z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have *: ⋀n. ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
          ⋀n. ((deriv ^^ n) g has_field_derivative deriv ((deriv ^^ n) g) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have sumeq: (∑ i = 0..n.
                of_nat (n choose i) * (deriv ((deriv ^^ i) f) z * (deriv ^^ (n - i)) g
z + deriv ((deriv ^^ (n - i)) g) z * (deriv ^^ i) f z)) =
            g z * deriv ((deriv ^^ n) f) z + (∑ i = 0..n. (deriv ^^ i) f z * (of_nat
(Suc n choose i) * (deriv ^^ (Suc n - i)) g z))
    apply (simp add: Suc_choose algebra_simps sum.distrib)
    apply (subst (4) sum_Suc_reindex)
    apply (auto simp: algebra_simps Suc_diff_le intro: sum.cong)
    done
  have ((deriv ^^ n) (λw. f w * g w) has_field_derivative
          (∑ i = 0..Suc n. (Suc n choose i) * (deriv ^^ i) f z * (deriv ^^ (Suc n -
i)) g z))
        (at z)
    apply (rule has_field_derivative_transform_within_open
          [of λw. (∑ i = 0..n. of_nat (n choose i) * (deriv ^^ i) f w * (deriv ^^ (n
- i)) g w) _ _ S])
      apply (simp add: algebra_simps)
      apply (rule derivative_eq_intros | simp)+
```

```
      apply (auto intro: DERIV_mult * ‹open S› Suc.prems Suc.IH [symmetric])
    by (metis (no_types, lifting) mult.commute sum.cong sumeq)
  then show ?case
    unfolding funpow.simps o_apply
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_transform_within_open:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
      and fg: ⋀w. w ∈ S ⟹ f w = g w
    shows (deriv ^^ i) f z = (deriv ^^ i) g z
using z
by (induction i arbitrary: z)
  (auto simp: fg intro: complex_derivative_transform_within_open holomorphic_higher_deriv
assms)

lemma higher_deriv_compose_linear:
  fixes z::complex
  assumes f: f holomorphic_on T and S: open S and T: open T and z: z ∈ S
      and fg: ⋀w. w ∈ S ⟹ u * w ∈ T
    shows (deriv ^^ n) (λw. f (u * w)) z = u^n * (deriv ^^ n) f (u * z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have holo0: f holomorphic_on (*) u ‘ S
    by (meson fg f holomorphic_on_subset image_subset_iff)
  have holo2: (deriv ^^ n) f holomorphic_on (*) u ‘ S
   by (meson f fg holomorphic_higher_deriv holomorphic_on_subset image_subset_iff
T)
  have holo3: (λz. u ^ n * (deriv ^^ n) f (u * z)) holomorphic_on S
    by (intro holo2 holomorphic_on_compose [where g=(deriv ^^ n) f, unfolded
o_def] holomorphic_intros)
  have (*) u holomorphic_on S f holomorphic_on (*) u ‘ S
    by (rule holo0 holomorphic_intros)+
  then have holo1: (λw. f (u * w)) holomorphic_on S
    by (rule holomorphic_on_compose [where g=f, unfolded o_def])
  have deriv ((deriv ^^ n) (λw. f (u * w))) z = deriv (λz. u^n * (deriv ^^ n) f
(u*z)) z
 proof (rule complex_derivative_transform_within_open [OF _ holo3 S Suc.prems])
    show (deriv ^^ n) (λw. f (u * w)) holomorphic_on S
      by (rule holomorphic_higher_deriv [OF holo1 S])
  qed (simp add: Suc.IH)
  also have ... = u^n * deriv (λz. (deriv ^^ n) f (u * z)) z
  proof −
    have (deriv ^^ n) f analytic_on T
      by (simp add: analytic_on_open f holomorphic_higher_deriv T)
```

```
    then have (λw. (deriv ^^ n) f (u * w)) analytic_on S
    proof −
      have (deriv ^^ n) f ∘ (*) u holomorphic_on S
        by (simp add: holo2 holomorphic_on_compose)
      then show ?thesis
        by (simp add: S analytic_on_open o_def)
    qed
    then show ?thesis
     by (intro deriv_cmult analytic_on_imp_differentiable_at [OF _ Suc.prems])
  qed
  also have ... = u * u ^ n * deriv ((deriv ^^ n) f) (u * z)
  proof −
    have (deriv ^^ n) f field_differentiable at (u * z)
     using Suc.prems T f fg holomorphic_higher_deriv holomorphic_on_imp_differentiable_at
by blast
    then show ?thesis
      by (simp add: deriv_compose_linear)
  qed
  finally show ?case
    by simp
qed

lemma higher_deriv_add_at:
  assumes f analytic_on {z} g analytic_on {z}
    shows (deriv ^^ n) (λw. f w + g w) z = (deriv ^^ n) f z + (deriv ^^ n) g z
  using analytic_at_two assms higher_deriv_add by blast

lemma higher_deriv_diff_at:
  assumes f analytic_on {z} g analytic_on {z}
    shows (deriv ^^ n) (λw. f w − g w) z = (deriv ^^ n) f z − (deriv ^^ n) g z
  using analytic_at_two assms higher_deriv_diff by blast

lemma higher_deriv_uminus_at:
   f analytic_on {z} ⟹ (deriv ^^ n) (λw. −(f w)) z = − ((deriv ^^ n) f z)
  using higher_deriv_uminus by (auto simp: analytic_at)

lemma higher_deriv_mult_at:
  assumes f analytic_on {z} g analytic_on {z}
    shows (deriv ^^ n) (λw. f w * g w) z =
          (∑ i = 0..n. of_nat (n choose i) * (deriv ^^ i) f z * (deriv ^^ (n − i)) g
z)
  using analytic_at_two assms higher_deriv_mult by blast
```

Nonexistence of isolated singularities and a stronger integral formula.

```
proposition no_isolated_singularity:
  fixes z::complex
   assumes f: continuous_on S f and holf: f holomorphic_on (S − K) and S:
open S and K: finite K
    shows f holomorphic_on S
```

```
proof −
  { fix z
    assume z ∈ S and cdf: ⋀x. x ∈ S − K ⟹ f field_differentiable at x
    have f field_differentiable at z
    proof (cases z ∈ K)
      case False then show ?thesis by (blast intro: cdf ‹z ∈ S›)
    next
      case True
      with finite_set_avoid [OF K, of z]
      obtain d where d>0 and d: ⋀x. ⟦x∈K; x ≠ z⟧ ⟹ d ≤ dist z x
        by blast
      obtain e where e>0 and e: ball z e ⊆ S
        using  S ‹z ∈ S› by (force simp: open_contains_ball)
      have fde: continuous_on (ball z (min d e)) f
        by (metis Int_iff ball_min_Int continuous_on_subset e f subsetI)
      have cont: {a,b,c} ⊆ ball z (min d e) ⟹ continuous_on (convex hull {a, b,
c}) f for a b c
        by (simp add: hull_minimal continuous_on_subset [OF fde])
      have fd: ⟦{a,b,c} ⊆ ball z (min d e); x ∈ interior (convex hull {a, b, c}) −
K⟧
            ⟹ f field_differentiable at x for a b c x
        by (metis cdf Diff_iff Int_iff ball_min_Int subsetD convex_ball e inte-
rior_mono interior_subset subset_hull)
      obtain g where ⋀w. w ∈ ball z (min d e) ⟹ (g has_field_derivative f w)
(at w within ball z (min d e))
        apply (rule contour_integral_convex_primitive
                    [OF convex_ball fde Cauchy_theorem_triangle_cofinite [OF _
K]])
        using cont fd by auto
      then have f holomorphic_on ball z (min d e)
        by (metis open_ball at_within_open derivative_is_holomorphic)
      then show ?thesis
        unfolding holomorphic_on_def
          by (metis open_ball ‹0 < d› ‹0 < e› at_within_open centre_in_ball
min_less_iff_conj)
    qed
  }
  with holf S K show ?thesis
   by (simp add: holomorphic_on_open open_Diff finite_imp_closed field_differentiable_def
[symmetric])
qed

lemma no_isolated_singularity′:
  fixes z::complex
  assumes f: ⋀z. z ∈ K ⟹ (f ⟶ f z) (at z within S)
      and holf: f holomorphic_on (S − K) and S: open S and K: finite K
    shows f holomorphic_on S
proof (rule no_isolated_singularity[OF _ assms(2−)])
  show continuous_on S f unfolding continuous_on_def
```

```
  proof
    fix z assume z: z ∈ S
    have continuous_on (S - K) f
      using holf holomorphic_on_imp_continuous_on by auto
    then show (f ⟶ f z) (at z within S)
    by (metis Diff_iff K S at_within_interior continuous_on_def f finite_imp_closed
interior_eq open_Diff z)
  qed
qed

proposition Cauchy_integral_formula_convex:
  assumes S: convex S and K: finite K and contf: continuous_on S f
    and fcd: (⋀x. x ∈ interior S - K ⟹ f field_differentiable at x)
    and z: z ∈ interior S and vpg: valid_path γ
    and pasz: path_image γ ⊆ S - {z} and loop: pathfinish γ = pathstart γ
  shows ((λw. f w / (w - z)) has_contour_integral (2*pi * i * winding_number
γ z * f z)) γ
proof -
  have *: ⋀x. x ∈ interior S ⟹ f field_differentiable at x
    unfolding holomorphic_on_open [symmetric] field_differentiable_def
    using no_isolated_singularity [where S = interior S]
   by (meson K contf continuous_at_imp_continuous_on continuous_on_interior
fcd
        field_differentiable_at_within field_differentiable_def holomorphic_onI
        holomorphic_on_imp_differentiable_at open_interior)
  show ?thesis
     by (rule Cauchy_integral_formula_weak [OF S finite.emptyI contf]) (use *
assms in auto)
qed
```

Formula for higher derivatives.

```
lemma Cauchy_has_contour_integral_higher_derivative_circlepath:
  assumes contf: continuous_on (cball z r) f
      and holf: f holomorphic_on ball z r
      and w: w ∈ ball z r
    shows ((λu. f u / (u - w) ^ (Suc k)) has_contour_integral ((2 * pi * i) /
(fact k) * (deriv ^^ k) f w))
          (circlepath z r)
using w
proof (induction k arbitrary: w)
  case 0 then show ?case
   using assms by (auto simp: Cauchy_integral_circlepath dist_commute dist_norm)
next
  case (Suc k)
  have [simp]: r > 0 using w
    using ball_eq_empty by fastforce
  have f: continuous_on (path_image (circlepath z r)) f
    by (rule continuous_on_subset [OF contf]) (force simp: cball_def sphere_def
less_imp_le)
```

obtain X **where** X: $((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ (Suc\ k))\ has\_contour\_integral\ X)\ (circlepath\ z\ r)$
using *Cauchy_next_derivative_circlepath*(*1*) [*OF f Suc.IH _ Suc.prems*]
by (*auto simp*: *contour_integrable_on_def*)
then have *con*: $contour\_integral\ (circlepath\ z\ r)\ ((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ (Suc\ k))) = X$
by (*rule contour_integral_unique*)
have $\bigwedge n.\ ((deriv\ \hat{\hat{}}\ n)\ f\ has\_field\_derivative\ deriv\ ((deriv\ \hat{\hat{}}\ n)\ f)\ w)\ (at\ w)$
using *Suc.prems assms has_field_derivative_higher_deriv* **by** *auto*
then have *dnf_diff*: $\bigwedge n.\ (deriv\ \hat{\hat{}}\ n)\ f\ field\_differentiable\ (at\ w)$
by (*force simp*: *field_differentiable_def*)
have $deriv\ (\lambda w.\ complex\_of\_real\ (2 * pi) * \mathrm{i}\ /\ (fact\ k) * (deriv\ \hat{\hat{}}\ k)\ f\ w)\ w =$
$of\_nat\ (Suc\ k) * contour\_integral\ (circlepath\ z\ r)\ (\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ (Suc\ k))$
by (*force intro*!: *DERIV_imp_deriv Cauchy_next_derivative_circlepath* [*OF f Suc.IH _ Suc.prems*])
also have $\ldots = of\_nat\ (Suc\ k) * X$
by (*simp only*: *con*)
finally have $deriv\ (\lambda w.\ ((2 * pi) * \mathrm{i}\ /\ (fact\ k)) * (deriv\ \hat{\hat{}}\ k)\ f\ w)\ w = of\_nat\ (Suc\ k) * X$ **.**
then have $((2 * pi) * \mathrm{i}\ /\ (fact\ k)) * deriv\ (\lambda w.\ (deriv\ \hat{\hat{}}\ k)\ f\ w)\ w = of\_nat\ (Suc\ k) * X$
by (*metis deriv_cmult dnf_diff*)
then have $deriv\ (\lambda w.\ (deriv\ \hat{\hat{}}\ k)\ f\ w)\ w = of\_nat\ (Suc\ k) * X\ /\ ((2 * pi) * \mathrm{i}\ /\ (fact\ k))$
by (*simp add*: *field_simps*)
then show *?case*
using *of_nat_eq_0_iff X* **by** *fastforce*
qed

lemma *Cauchy_higher_derivative_integral_circlepath*:
assumes *contf*: *continuous_on* (*cball z r*) *f*
and *holf*: *f holomorphic_on ball z r*
and *w*: $w \in ball\ z\ r$
shows $(\lambda u.\ f\ u\ /\ (u - w)\hat{}(Suc\ k))\ contour\_integrable\_on\ (circlepath\ z\ r)$
(**is** *?thes1*)
and $(deriv\ \hat{\hat{}}\ k)\ f\ w = (fact\ k)\ /\ (2 * pi * \mathrm{i}) * contour\_integral(circlepath\ z\ r)\ (\lambda u.\ f\ u/(u - w)\hat{}(Suc\ k))$
(**is** *?thes2*)
proof −
have $*$: $((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ k)\ has\_contour\_integral\ (2 * pi) * \mathrm{i}\ /\ (fact\ k) * (deriv\ \hat{\hat{}}\ k)\ f\ w)$
$(circlepath\ z\ r)$
using *Cauchy_has_contour_integral_higher_derivative_circlepath* [*OF assms*]
by *simp*
show *?thes1* **using** $*$
using *contour_integrable_on_def* **by** *blast*
show *?thes2*
unfolding *contour_integral_unique* [*OF* $*$] **by** (*simp add*: *field_split_simps*)

qed

corollary *Cauchy_contour_integral_circlepath*:
assumes *continuous_on* (*cball z r*) *f f holomorphic_on ball z r w* ∈ *ball z r*
shows *contour_integral*(*circlepath z r*) (λ*u*. *f u*/(*u* − *w*)^(*Suc k*)) = (2 ∗ *pi* ∗ i) ∗ (*deriv* ^^ *k*) *f w* / (*fact k*)
by (*simp add*: *Cauchy_higher_derivative_integral_circlepath* [*OF assms*])

lemma *Cauchy_contour_integral_circlepath_2*:
assumes *continuous_on* (*cball z r*) *f f holomorphic_on ball z r w* ∈ *ball z r*
shows *contour_integral*(*circlepath z r*) (λ*u*. *f u*/(*u* − *w*)^2) = (2 ∗ *pi* ∗ i) ∗ *deriv f w*
using *Cauchy_contour_integral_circlepath* [*OF assms*, *of 1*]
by (*simp add*: *power2_eq_square*)

4.6 A holomorphic function is analytic, i.e. has local power series

theorem *holomorphic_power_series*:
assumes *holf*: *f holomorphic_on ball z r*
and *w*: *w* ∈ *ball z r*
shows ((λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*)^*n*) *sums f w*)
proof −
— Replacing *r* and the original (weak) premises with stronger ones
obtain *r* **where** *r* > *0* **and** *holfc*: *f holomorphic_on cball z r* **and** *w*: *w* ∈ *ball z r*
proof
have *cball z* ((*r* + *dist w z*) / *2*) ⊆ *ball z r*
using *w* **by** (*simp add*: *dist_commute field_sum_of_halves subset_eq*)
then show *f holomorphic_on cball z* ((*r* + *dist w z*) / *2*)
by (*rule holomorphic_on_subset* [*OF holf*])
have *r* > *0*
using *w* **by** *clarsimp* (*metis dist_norm le_less_trans norm_ge_zero*)
then show *0* < (*r* + *dist w z*) / *2*
by *simp* (*use zero_le_dist* [*of w z*] **in** *linarith*)
qed (*use w* **in** ‹*auto simp*: *dist_commute*›)
then have *holf*: *f holomorphic_on ball z r*
using *ball_subset_cball holomorphic_on_subset* **by** *blast*
have *contf*: *continuous_on* (*cball z r*) *f*
by (*simp add*: *holfc holomorphic_on_imp_continuous_on*)
have *cint*: ⋀*k*. (λ*u*. *f u* / (*u* − *z*) ^ *Suc k*) *contour_integrable_on circlepath z r*
by (*rule Cauchy_higher_derivative_integral_circlepath* [*OF contf holf*]) (*simp add*: ‹*0* < *r*›)
obtain *B* **where** *0* < *B* **and** *B*: ⋀*u*. *u* ∈ *cball z r* ⟹ *norm*(*f u*) ≤ *B*
by (*metis* (*no_types*) *bounded_pos compact_cball compact_continuous_image compact_imp_bounded contf image_eqI*)
obtain *k* **where** *k*: *0* < *k k* ≤ *r* **and** *wz_eq*: *norm*(*w* − *z*) = *r* − *k*
and *kle*: ⋀*u*. *norm*(*u* − *z*) = *r* ⟹ *k* ≤ *norm*(*u* − *w*)
proof

```
    show ⋀u. cmod (u − z) = r ⟹ r − dist z w ≤ cmod (u − w)
      by (metis add_diff_eq diff_add_cancel dist_norm norm_diff_ineq)
  qed (use w in ‹auto simp: dist_norm norm_minus_commute›)
  have ul: uniform_limit (sphere z r) (λn x. (∑k<n. (w − z) ^ k * (f x / (x −
z) ^ Suc k))) (λx. f x / (x − w)) sequentially
    unfolding uniform_limit_iff dist_norm
  proof clarify
    fix e::real
    assume 0 < e
    have rr: 0 ≤ (r − k) / r (r − k) / r < 1 using  k by auto
    obtain n where n: ((r − k) / r) ^ n < e / B * k
      using real_arch_pow_inv [of e/B*k (r − k)/r] ‹0 < e› ‹0 < B› k by force
    have norm ((∑k<N. (w − z) ^ k * f u / (u − z) ^ Suc k) − f u / (u − w))
< e
        if n ≤ N and r: r = dist z u  for N u
    proof −
      have N: ((r − k) / r) ^ N < e / B * k
        using le_less_trans [OF power_decreasing n]
        using ‹n ≤ N› k by auto
      have u [simp]: (u ≠ z) ∧ (u ≠ w)
        using ‹0 < r› r w by auto
      have wzu_not1: (w − z) / (u − z) ≠ 1
          by (metis (no_types) dist_norm divide_eq_1_iff less_irrefl mem_ball
norm_minus_commute r w)
      have norm ((∑k<N. (w − z) ^ k * f u / (u − z) ^ Suc k) * (u − w) − f u)
          = norm ((∑k<N. (((w − z) / (u − z)) ^ k)) * f u * (u − w) / (u −
z) − f u)
        unfolding sum_distrib_right sum_divide_distrib power_divide by (simp
add: algebra_simps)
      also have ... = norm ((((((w − z) / (u − z)) ^ N − 1) * (u − w) / (((w −
z) / (u − z) − 1) * (u − z)) − 1) * norm (f u)
        using ‹0 < B›
        apply (auto simp: geometric_sum [OF wzu_not1])
        apply (simp add: field_simps norm_mult [symmetric])
        done
      also have ... = norm ((u−z) ^ N * (w − u) − ((w − z) ^ N − (u−z) ^
N) * (u−w)) / (r ^ N * norm (u−w)) * norm (f u)
        using ‹0 < r› r by (simp add: divide_simps norm_mult norm_divide
norm_power dist_norm norm_minus_commute)
      also have ... = norm ((w − z) ^ N * (w − u)) / (r ^ N * norm (u − w))
* norm (f u)
        by (simp add: algebra_simps)
      also have ... = norm (w − z) ^ N * norm (f u) / r ^ N
        by (simp add: norm_mult norm_power norm_minus_commute)
      also have ... ≤ (((r − k)/r)^N) * B
        using ‹0 < r› w k
        by (simp add: B divide_simps mult_mono r wz_eq)
      also have ... < e * k
        using ‹0 < B› N by (simp add: divide_simps)
```

```
    also have ... ≤ e * norm (u - w)
      using r kle ‹0 < e› by (simp add: dist_commute dist_norm)
    finally show ?thesis
      by (simp add: field_split_simps norm_divide del: power_Suc)
  qed
  with ‹0 < r› show ∀F n in sequentially. ∀x∈sphere z r.
          norm ((∑k<n. (w - z) ^ k * (f x / (x - z) ^ Suc k)) - f x / (x -
w)) < e
    by (auto simp: mult_ac less_imp_le eventually_sequentially Ball_def)
 qed
 have §: ⋀x k. k∈ {..<x} ⟹
          (λu. (w - z) ^ k * (f u / (u - z) ^ Suc k)) contour_integrable_on
circlepath z r
   using contour_integrable_lmul [OF cint, of (w - z) ^ a for a] by (simp add:
field_simps)
 have eq: ∀F x in sequentially.
          contour_integral (circlepath z r) (λu. ∑k<x. (w - z) ^ k * (f u / (u
- z) ^ Suc k)) =
          (∑k<x. contour_integral (circlepath z r) (λu. f u / (u - z) ^ Suc k) *
(w - z) ^ k)
   apply (rule eventuallyI)
   apply (subst contour_integral_sum, simp)
   apply (simp_all only: § contour_integral_lmul cint algebra_simps)
   done
 have ⋀u k. k ∈ {..<u} ⟹ (λx. f x / (x - z) ^ Suc k) contour_integrable_on
circlepath z r
   using ‹0 < r› by (force intro!: Cauchy_higher_derivative_integral_circlepath
[OF contf holf])
  then have ⋀u. (λy. ∑k<u. (w - z) ^ k * (f y / (y - z) ^ Suc k)) con-
tour_integrable_on circlepath z r
   by (intro contour_integrable_sum contour_integrable_lmul, simp)
 then have (λk. contour_integral (circlepath z r) (λu. f u/(u - z)^(Suc k)) * (w
- z)^k)
       sums contour_integral (circlepath z r) (λu. f u/(u - w))
   unfolding sums_def using ‹0 < r›
  by (intro Lim_transform_eventually [OF _ eq] contour_integral_uniform_limit_circlepath
[OF eventuallyI ul]) auto
 then have (λk. contour_integral (circlepath z r) (λu. f u/(u - z)^(Suc k)) * (w
- z)^k)
          sums (2 * of_real pi * i * f w)
  using w by (auto simp: dist_commute dist_norm contour_integral_unique [OF
Cauchy_integral_circlepath_simple [OF holfc]])
 then have (λk. contour_integral (circlepath z r) (λu. f u / (u - z) ^ Suc k) *
(w - z)^k / (i * (of_real pi * 2)))
          sums ((2 * of_real pi * i * f w) / (i * (complex_of_real pi * 2)))
   by (rule sums_divide)
 then have (λn. (w - z) ^ n * contour_integral (circlepath z r) (λu. f u / (u -
z) ^ Suc n) / (i * (of_real pi * 2)))
          sums f w
```

by (*simp add*: *field_simps*)
then show *?thesis*
by (*simp add*: *field_simps* ‹*0 < r*› *Cauchy_higher_derivative_integral_circlepath* [*OF contf holf*])
qed

4.7 The Liouville theorem and the Fundamental Theorem of Algebra

These weak Liouville versions don't even need the derivative formula.

lemma *Liouville_weak_0*:
assumes *holf*: *f holomorphic_on UNIV* **and** *inf*: (*f* ⟶ *0*) *at_infinity*
shows *f z = 0*
proof (*rule ccontr*)
assume *fz*: *f z ≠ 0*
with *inf* [*unfolded Lim_at_infinity*, *rule_format*, *of norm*(*f z*)/*2*]
obtain *B* **where** *B*: ⋀*x*. *B* ≤ *cmod x* ⟹ *norm* (*f x*) * *2* < *cmod* (*f z*)
by (*auto simp*: *dist_norm*)
define *R* **where** *R = 1 + |B| + norm z*
have *R > 0*
unfolding *R_def* **by** (*smt* (*verit*) *norm_ge_zero*)
have ∗: ((λ*u*. *f u* / (*u* − *z*)) *has_contour_integral 2* ∗ *complex_of_real pi* ∗ i ∗ *f z*) (*circlepath z R*)
using *continuous_on_subset holf holomorphic_on_subset* ‹*0 < R*›
by (*force intro*: *holomorphic_on_imp_continuous_on Cauchy_integral_circlepath*)
have *cmod* (*x* − *z*) = *R* ⟹ *cmod* (*f x*) * *2* < *cmod* (*f z*) **for** *x*
unfolding *R_def* **by** (*rule B*) (*use norm_triangle_ineq4* [*of x z*] **in** *auto*)
with ‹*R > 0*› *fz* **show** *False*
using *has_contour_integral_bound_circlepath* [*OF* ∗, *of norm*(*f z*)/*2*/*R*]
by (*auto simp*: *less_imp_le norm_mult norm_divide field_split_simps*)
qed

proposition *Liouville_weak*:
assumes *f holomorphic_on UNIV* **and** (*f* ⟶ *l*) *at_infinity*
shows *f z = l*
using *Liouville_weak_0* [*of* λ*z*. *f z* − *l*]
by (*simp add*: *assms holomorphic_on_diff LIM_zero*)

proposition *Liouville_weak_inverse*:
assumes *f holomorphic_on UNIV* **and** *unbounded*: ⋀*B*. *eventually* (λ*x*. *norm* (*f x*) ≥ *B*) *at_infinity*
obtains *z* **where** *f z = 0*
proof −
{ assume *f*: ⋀*z*. *f z ≠ 0*
have *1*: (λ*x*. *1* / *f x*) *holomorphic_on UNIV*
by (*simp add*: *holomorphic_on_divide assms f*)
have *2*: ((λ*x*. *1* / *f x*) ⟶ *0*) *at_infinity*
proof (*rule tendstoI* [*OF eventually_mono*])
fix *e*::*real*

```
    assume e > 0
    show eventually (λx. 2/e ≤ cmod (f x)) at_infinity
      by (rule_tac B=2/e in unbounded)
  qed (simp add: dist_norm norm_divide field_split_simps)
  have False
    using Liouville_weak_0 [OF 1 2] f by simp
 }
 then show ?thesis
  using that by blast
qed
```

In particular we get the Fundamental Theorem of Algebra.

```
theorem fundamental_theorem_of_algebra:
  fixes a :: nat ⇒ complex
 assumes a 0 = 0 ∨ (∃ i ∈ {1..n}. a i ≠ 0)
 obtains z where (∑ i≤n. a i * z^i) = 0
using assms
proof (elim disjE bexE)
 assume a 0 = 0 then show ?thesis
  by (auto simp: that [of 0])
next
 fix i
 assume i: i ∈ {1..n} and nz: a i ≠ 0
 have 1: (λz. ∑ i≤n. a i * z^i) holomorphic_on UNIV
  by (rule holomorphic_intros)+
 show thesis
 proof (rule Liouville_weak_inverse [OF 1])
  show ∀F x in at_infinity. B ≤ cmod (∑ i≤n. a i * x ^ i) for B
    using i nz by (intro polyfun_extremal exI[of _ i]) auto
 qed (use that in auto)
qed
```

4.8 Weierstrass convergence theorem

```
lemma holomorphic_uniform_limit:
 assumes cont: eventually (λn. continuous_on (cball z r) (f n) ∧ (f n) holomorphic_on ball z r) F
     and ulim: uniform_limit (cball z r) f g F
     and F: ¬ trivial_limit F
 obtains continuous_on (cball z r) g ∧ g holomorphic_on ball z r
proof (cases r 0::real rule: linorder_cases)
 case less then show ?thesis by (force simp: ball_empty less_imp_le continuous_on_def holomorphic_on_def intro: that)
next
 case equal then show ?thesis
  by (force simp: holomorphic_on_def intro: that)
next
 case greater
 have contg: continuous_on (cball z r) g
```

```
    using cont uniform_limit_theorem [OF eventually_mono ulim F] by blast
  have path_image (circlepath z r) ⊆ cball z r
    using ‹0 < r› by auto
  then have 1: continuous_on (path_image (circlepath z r)) (λx. 1 / (2 * com-
plex_of_real pi * i) * g x)
    by (intro continuous_intros continuous_on_subset [OF contg])
  have 2: ((λu. 1 / (2 * of_real pi * i) * g u / (u - w) ^ 1) has_contour_integral
g w) (circlepath z r)
      if w: w ∈ ball z r for w
  proof -
    define d where d = (r - norm(w - z))
     have 0 < d  d ≤ r using w by (auto simp: norm_minus_commute d_def
dist_norm)
    have dle: ⋀u. cmod (z - u) = r ⟹ d ≤ cmod (u - w)
       unfolding d_def by (metis add_diff_eq diff_add_cancel norm_diff_ineq
norm_minus_commute)
    have ev_int: ∀F n in F. (λu. f n u / (u - w)) contour_integrable_on circlepath
z r
      using w
     by (auto intro: eventually_mono [OF cont] Cauchy_higher_derivative_integral_circlepath
[where k=0, simplified])
    have ⋀e. ⟦0 < r; 0 < d; 0 < e⟧
         ⟹ ∀F n in F.
               ∀x∈sphere z r.
                  x ≠ w ⟶
                  cmod (f n x - g x) < e * cmod (x - w)
       apply (rule_tac e1=e * d in eventually_mono [OF uniform_limitD [OF
ulim]])
       apply (force simp: dist_norm intro: dle mult_left_mono less_le_trans)+
      done
    then have ul_less: uniform_limit (sphere z r) (λn x. f n x / (x - w)) (λx. g
x / (x - w)) F
      using greater ‹0 < d›
     by (auto simp add: uniform_limit_iff dist_norm norm_divide diff_divide_distrib
[symmetric] divide_simps)
    have g_cint: (λu. g u/(u - w)) contour_integrable_on circlepath z r
     by (rule contour_integral_uniform_limit_circlepath [OF ev_int ul_less F ‹0
< r›])
     have cif_tends_cig: ((λn. contour_integral(circlepath z r) (λu. f n u / (u -
w))) ⟶ contour_integral(circlepath z r) (λu. g u/(u - w))) F
      by (rule contour_integral_uniform_limit_circlepath [OF ev_int ul_less F ‹0
< r›])
     have f_tends_cig: ((λn. 2 * of_real pi * i * f n w) ⟶ contour_integral
(circlepath z r) (λu. g u / (u - w))) F
    proof (rule Lim_transform_eventually)
      show ∀F x in F. contour_integral (circlepath z r) (λu. f x u / (u - w))
                    = 2 * of_real pi * i * f x w
        using w‹0 < d› d_def
        by (auto intro: eventually_mono [OF cont contour_integral_unique [OF
```

Cauchy_integral_circlepath]])
qed (*auto simp*: *cif_tends_cig*)
have $\bigwedge e.\ 0 < e \Longrightarrow \forall_F\ n\ in\ F.\ dist\ (f\ n\ w)\ (g\ w) < e$
by (*rule eventually_mono* [*OF uniform_limitD* [*OF ulim*]]) (*use w* **in** *auto*)
then have $((\lambda n.\ 2 * of\_real\ pi * \mathrm{i} * f\ n\ w) \longrightarrow 2 * of\_real\ pi * \mathrm{i} * g\ w)\ F$
by (*rule tendsto_mult_left* [*OF tendstoI*])
then have $((\lambda u.\ g\ u\ /\ (u - w))\ has\_contour\_integral\ 2 * of\_real\ pi * \mathrm{i} * g\ w)\ (circlepath\ z\ r)$
using *has_contour_integral_integral* [*OF g_cint*] *tendsto_unique* [*OF F f_tends_cig*] *w*
by *fastforce*
then have $((\lambda u.\ g\ u\ /\ (2 * of\_real\ pi * \mathrm{i} * (u - w)))\ has\_contour\_integral\ g\ w)\ (circlepath\ z\ r)$
using *has_contour_integral_div* [**where** $c = 2 * of\_real\ pi * \mathrm{i}$]
by (*force simp*: *field_simps*)
then show *?thesis*
by (*simp add*: *dist_norm*)
qed
show *?thesis*
using *Cauchy_next_derivative_circlepath*(*2*) [*OF 1 2*, *simplified*]
by (*fastforce simp add*: *holomorphic_on_open contg intro*: *that*)
qed

Version showing that the limit is the limit of the derivatives.

proposition *has_complex_derivative_uniform_limit*:
fixes *z*::*complex*
assumes *cont*: *eventually* $(\lambda n.\ continuous\_on\ (cball\ z\ r)\ (f\ n) \wedge (\forall w \in ball\ z\ r.\ ((f\ n)\ has\_field\_derivative\ (f'\ n\ w))\ (at\ w)))\ F$
and *ulim*: *uniform_limit* $(cball\ z\ r)\ f\ g\ F$
and *F*: $\neg\ trivial\_limit\ F$ **and** $0 < r$
obtains g' **where**
$continuous\_on\ (cball\ z\ r)\ g$
$\bigwedge w.\ w \in ball\ z\ r \Longrightarrow (g\ has\_field\_derivative\ (g'\ w))\ (at\ w) \wedge ((\lambda n.\ f'\ n\ w) \longrightarrow g'\ w)\ F$
proof −
let *?conint* = *contour_integral* (*circlepath z r*)
have *g*: *continuous_on* (*cball z r*) *g g holomorphic_on ball z r*
by (*rule holomorphic_uniform_limit* [*OF eventually_mono* [*OF cont*] *ulim F*]; *auto simp*: *holomorphic_on_open field_differentiable_def*)+
then obtain g' **where** g': $\bigwedge x.\ x \in ball\ z\ r \Longrightarrow (g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
using *DERIV_deriv_iff_has_field_derivative*
by (*fastforce simp add*: *holomorphic_on_open*)
then have *derg*: $\bigwedge x.\ x \in ball\ z\ r \Longrightarrow deriv\ g\ x = g'\ x$
by (*simp add*: *DERIV_imp_deriv*)
have *tends_f'n_g'*: $((\lambda n.\ f'\ n\ w) \longrightarrow g'\ w)\ F$ **if** *w*: $w \in ball\ z\ r$ **for** *w*
proof −
have *eq_f'*: $?conint\ (\lambda x.\ f\ n\ x\ /\ (x - w)^2) - ?conint\ (\lambda x.\ g\ x\ /\ (x - w)^2) =$

```
(f' n w - g' w) * (2 * of_real pi * i)
          if cont_fn: continuous_on (cball z r) (f n)
          and fnd: ⋀w. w ∈ ball z r ⟹ (f n has_field_derivative f' n w) (at w)
for n
  proof -
    have hol_fn: f n holomorphic_on ball z r
      using fnd by (force simp: holomorphic_on_open)
    have (f n has_field_derivative 1 / (2 * of_real pi * i) * ?conint (λu. f n u
/ (u - w)^2)) (at w)
      by (rule Cauchy_derivative_integral_circlepath [OF cont_fn hol_fn w])
    then have f': f' n w = 1 / (2 * of_real pi * i) * ?conint (λu. f n u / (u -
w)^2)
      using DERIV_unique [OF fnd] w by blast
    show ?thesis
     by (simp add: f' Cauchy_contour_integral_circlepath_2 [OF g w] derg [OF
w] field_split_simps)
  qed
  define d where d = (r - norm(w - z))^2
  have d > 0
    using w by (simp add: dist_commute dist_norm d_def)
  have dle: d ≤ cmod ((y - w)^2) if r = cmod (z - y) for y
  proof -
    have cmod (w - z) ≤ cmod (z - y)
      by (metis dist_commute dist_norm mem_ball order_less_imp_le that w)
    moreover have cmod (z - y) - cmod (w - z) ≤ cmod (y - w)
    by (metis diff_add_cancel diff_diff_eq2 norm_minus_commute norm_triangle_ineq2)
    ultimately show ?thesis
      using that by (simp add: d_def norm_power power_mono)
  qed
  have 1: ∀F n in F. (λx. f n x / (x - w)^2) contour_integrable_on circlepath z
r
   by (force simp: holomorphic_on_open intro: w Cauchy_derivative_integral_circlepath
eventually_mono [OF cont])
  have 2: uniform_limit (sphere z r) (λn x. f n x / (x - w)^2) (λx. g x / (x -
w)^2) F
    unfolding uniform_limit_iff
  proof clarify
    fix e::real
    assume e > 0
    with ‹r > 0›
    have ∀F n in F. ∀x. x ≠ w ⟶ cmod (z - x) = r ⟶ cmod (f n x - g x)
< e * cmod ((x - w)^2)
     by (force simp: ‹0 < d› dist_norm dle intro: less_le_trans eventually_mono
[OF uniform_limitD [OF ulim], of e*d])
    with ‹r > 0› ‹e > 0›
    show ∀F n in F. ∀x∈sphere z r. dist (f n x / (x - w)^2) (g x / (x - w)^2) < e
      by (simp add: norm_divide field_split_simps sphere_def dist_norm)
  qed
  have ((λn. contour_integral (circlepath z r) (λx. f n x / (x - w)^2))
```

```
        ⟶ contour_integral (circlepath z r) ((λx. g x / (x − w)²))) F
    by (rule contour_integral_uniform_limit_circlepath [OF 1 2 F ‹0 < r›])
   then have tendsto_0: ((λn. 1 / (2 ∗ of_real pi ∗ i) ∗ (?conint (λx. f n x / (x
− w)²) − ?conint (λx. g x / (x − w)²))) ⟶ 0) F
    using Lim_null by (force intro!: tendsto_mult_right_zero)
   have ((λn. f′ n w − g′ w) ⟶ 0) F
    apply (rule Lim_transform_eventually [OF tendsto_0])
    apply (force simp: divide_simps intro: eq_f′ eventually_mono [OF cont])
    done
   then show ?thesis using Lim_null by blast
  qed
  obtain g′ where ⋀w. w ∈ ball z r ⟹ (g has_field_derivative (g′ w)) (at w) ∧
((λn. f′ n w) ⟶ g′ w) F
    by (blast intro: tends_f′n_g′ g′)
  then show ?thesis using g
   using that by blast
qed
```

4.9 Some more simple/convenient versions for applications

```
lemma holomorphic_uniform_sequence:
  assumes S: open S
     and hol_fn: ⋀n. (f n) holomorphic_on S
      and ulim_g: ⋀x. x ∈ S ⟹ ∃ d. 0 < d ∧ cball x d ⊆ S ∧ uniform_limit
(cball x d) f g sequentially
  shows g holomorphic_on S
proof −
  have ∃f′. (g has_field_derivative f′) (at z) if z ∈ S for z
  proof −
   obtain r where 0 < r and r: cball z r ⊆ S
             and ul: uniform_limit (cball z r) f g sequentially
     using ulim_g [OF ‹z ∈ S›] by blast
    have ∗: ∀F n in sequentially. continuous_on (cball z r) (f n) ∧ f n holomor-
phic_on ball z r
    by (smt (verit, best) ball_subset_cball hol_fn holomorphic_on_imp_continuous_on

        holomorphic_on_subset not_eventuallyD r)
   show ?thesis
    using ‹0 < r› centre_in_ball ul
    by (auto simp: holomorphic_on_open intro: holomorphic_uniform_limit [OF
∗])
  qed
  with S show ?thesis
   by (simp add: holomorphic_on_open)
qed

lemma has_complex_derivative_uniform_sequence:
  fixes S :: complex set
  assumes S: open S
```

```
    and hfd: ⋀n x. x ∈ S ⟹ ((f n) has_field_derivative f' n x) (at x)
    and ulim_g: ⋀x. x ∈ S
        ⟹ ∃d. 0 < d ∧ cball x d ⊆ S ∧ uniform_limit (cball x d) f g sequentially
  shows ∃g'. ∀x ∈ S. (g has_field_derivative g' x) (at x) ∧ ((λn. f' n x) ⟶
g' x) sequentially
proof −
  have y: ∃y. (g has_field_derivative y) (at z) ∧ (λn. f' n z) ⟶ y if z ∈ S
for z
  proof −
    obtain r where 0 < r and r: cball z r ⊆ S
              and ul: uniform_limit (cball z r) f g sequentially
      using ulim_g [OF ‹z ∈ S›] by blast
    have ∗: ∀F n in sequentially. continuous_on (cball z r) (f n) ∧
                                  (∀w ∈ ball z r. ((f n) has_field_derivative (f' n w))
(at w))
    proof (intro eventuallyI conjI ballI)
      show continuous_on (cball z r) (f x) for x
      by (meson S continuous_on_subset hfd holomorphic_on_imp_continuous_on
holomorphic_on_open r)
      show w ∈ ball z r ⟹ (f x has_field_derivative f' x w) (at w) for w x
        using ball_subset_cball hfd r by blast
    qed
    show ?thesis
      by (rule has_complex_derivative_uniform_limit [OF ∗, of g]) (use ‹0 < r›
ul in ‹force+›)
  qed
  show ?thesis
    by (rule bchoice) (blast intro: y)
qed
```

4.10 On analytic functions defined by a series

```
lemma series_and_derivative_comparison:
  fixes S :: complex set
  assumes S: open S
      and h: summable h
      and hfd: ⋀n x. x ∈ S ⟹ (f n has_field_derivative f' n x) (at x)
      and to_g: ∀F n in sequentially. ∀x∈S. norm (f n x) ≤ h n
  obtains g g' where ∀x ∈ S. ((λn. f n x) sums g x) ∧ ((λn. f' n x) sums g' x)
∧ (g has_field_derivative g' x) (at x)
proof −
  obtain g where g: uniform_limit S (λn x. ∑i<n. f i x) g sequentially
    using Weierstrass_m_test_ev [OF to_g h]  by force
  have ∗: ∃d>0. cball x d ⊆ S ∧ uniform_limit (cball x d) (λn x. ∑i<n. f i x)
g sequentially
    if x ∈ S for x
    using open_contains_cball [of S] ‹x ∈ S› S g uniform_limit_on_subset by
blast
  have ⋀x. x ∈ S ⟹ (λn. ∑i<n. f i x) ⟶ g x
```

by (*metis tendsto_uniform_limitI* [*OF g*])
moreover have $\exists g'.\ \forall x{\in}S.\ (g\ has\_field\_derivative\ g'\ x)\ (at\ x) \wedge (\lambda n.\ \sum i{<}n.\ f'\ i\ x) \longrightarrow g'\ x$
by (*rule has_complex_derivative_uniform_sequence* [*OF S*]) (*auto intro*: ∗ *hfd DERIV_sum*)+
ultimately show *?thesis*
by (*metis sums_def that*)
qed

A version where we only have local uniform/comparative convergence.

lemma *series_and_derivative_comparison_local*:
fixes S :: *complex set*
assumes S: *open S*
and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x)$
and *to_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists d\ h.\ 0 < d \wedge summable\ h \wedge (\forall_F\ n\ in\ sequentially.\ \forall y{\in}ball\ x\ d \cap S.\ norm\ (f\ n\ y) \leq h\ n)$
shows $\exists g\ g'.\ \forall x \in S.\ ((\lambda n.\ f\ n\ x)\ sums\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ sums\ g'\ x) \wedge (g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
proof −
have $\exists y.\ (\lambda n.\ f\ n\ z)\ sums\ (\sum n.\ f\ n\ z) \wedge (\lambda n.\ f'\ n\ z)\ sums\ y \wedge ((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ y)\ (at\ z)$
if $z \in S$ **for** z
proof −
obtain $d\ h$ **where** $0 < d$ *summable h* **and** *le_h*: $\forall_F\ n\ in\ sequentially.\ \forall y{\in}ball\ z\ d \cap S.\ norm\ (f\ n\ y) \leq h\ n$
using *to_g* ‹$z \in S$› **by** *meson*
then obtain r **where** $r{>}0$ **and** r: $ball\ z\ r \subseteq ball\ z\ d \cap S$ **using** ‹$z \in S$› S
by (*metis Int_iff open_ball centre_in_ball open_Int open_contains_ball_eq*)
have *1*: *open* $(ball\ z\ d \cap S)$
by (*simp add*: *open_Int S*)
have *2*: $\bigwedge n\ x.\ x \in ball\ z\ d \cap S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x)$
by (*auto simp*: *hfd*)
obtain $g\ g'$ **where** gg': $\forall x \in ball\ z\ d \cap S.\ ((\lambda n.\ f\ n\ x)\ sums\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ sums\ g'\ x) \wedge (g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
by (*auto intro*: *le_h series_and_derivative_comparison* [*OF 1* ‹*summable h*› *hfd*])
then have $(\lambda n.\ f'\ n\ z)\ sums\ g'\ z$
by (*meson* ‹$0 < r$› *centre_in_ball contra_subsetD r*)
moreover have $(\lambda n.\ f\ n\ z)\ sums\ (\sum n.\ f\ n\ z)$
using *summable_sums centre_in_ball* ‹$0 < d$› ‹*summable h*› *le_h*
by (*metis* (*full_types*) *Int_iff gg' summable_def that*)
moreover have $((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ g'\ z)\ (at\ z)$
by (*metis* (*no_types*, *lifting*) *1 r* ‹$0 < r$› *gg' has_field_derivative_transform_within_open open_contains_ball_eq sums_unique*)
ultimately show *?thesis* **by** *auto*
qed
then show *?thesis*

by *meson*
qed

Sometimes convenient to compare with a complex series of positive reals. (?)

lemma *series_and_derivative_comparison_complex*:
fixes S :: *complex set*
assumes S: *open* S
and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ \textit{has\_field\_derivative}\ f'\ n\ x)\ (\textit{at}\ x)$
and *to_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists d\ h.\ 0 < d \wedge \textit{summable}\ h \wedge \textit{range}\ h \subseteq \mathbb{R}_{\geq 0} \wedge (\forall_F\ n\ \textit{in sequentially}.\ \forall y \in \textit{ball}\ x\ d \cap S.\ \textit{cmod}(f\ n\ y) \leq \textit{cmod}\ (h\ n))$
shows $\exists g\ g'.\ \forall x \in S.\ ((\lambda n.\ f\ n\ x)\ \textit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \textit{sums}\ g'\ x) \wedge (g\ \textit{has\_field\_derivative}\ g'\ x)\ (\textit{at}\ x)$
apply (*rule series_and_derivative_comparison_local* [*OF S hfd*], *assumption*)
apply (*rule ex_forward* [*OF to_g*], *assumption*)
apply (*erule exE*)
apply (*rule_tac x*=*Re* ∘ *h* **in** *exI*)
apply (*force simp*: *summable_Re o_def nonneg_Reals_cmod_eq_Re image_subset_iff*)
done

Sometimes convenient to compare with a complex series of positive reals. (?)

lemma *series_differentiable_comparison_complex*:
fixes S :: *complex set*
assumes S: *open* S
and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow f\ n\ \textit{field\_differentiable}\ (\textit{at}\ x)$
and *to_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists d\ h.\ 0 < d \wedge \textit{summable}\ h \wedge \textit{range}\ h \subseteq \mathbb{R}_{\geq 0} \wedge (\forall_F\ n\ \textit{in sequentially}.\ \forall y \in \textit{ball}\ x\ d \cap S.\ \textit{cmod}(f\ n\ y) \leq \textit{cmod}\ (h\ n))$
obtains g **where** $\forall x \in S.\ ((\lambda n.\ f\ n\ x)\ \textit{sums}\ g\ x) \wedge g\ \textit{field\_differentiable}\ (\textit{at}\ x)$
proof −
have *hfd'*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ \textit{has\_field\_derivative}\ \textit{deriv}\ (f\ n)\ x)\ (\textit{at}\ x)$
using *hfd field_differentiable_derivI* **by** *blast*
show *?thesis*
by (*metis field_differentiable_def that series_and_derivative_comparison_complex* [*OF S hfd' to_g*])
qed

In particular, a power series is analytic inside circle of convergence.

lemma *power_series_and_derivative_0*:
fixes a :: $nat \Rightarrow complex$ **and** r::*real*
assumes *summable* $(\lambda n.\ a\ n * r\hat{}n)$
shows $\exists g\ g'.\ \forall z.\ \textit{cmod}\ z < r \longrightarrow ((\lambda n.\ a\ n * z\hat{}n)\ \textit{sums}\ g\ z) \wedge ((\lambda n.\ \textit{of\_nat}\ n * a\ n * z\hat{}(n-1))\ \textit{sums}\ g'\ z) \wedge (g\ \textit{has\_field\_derivative}\ g'\ z)\ (\textit{at}\ z)$
proof (*cases* $0 < r$)
case *True*
have *der*: $\bigwedge n\ z.\ ((\lambda x.\ a\ n * x\ \hat{}\ n)\ \textit{has\_field\_derivative}\ \textit{of\_nat}\ n * a\ n * z\ \hat{}\ (n-1))\ (\textit{at}\ z)$

```
    by (rule derivative_eq_intros | simp)+
  have y_le: cmod y ≤ cmod (of_real r + of_real (cmod z)) / 2
    if cmod (z − y) * 2 < r − cmod z for z y
  by (smt (verit, best) field_sum_of_halves norm_minus_commute norm_of_real
norm_triangle_ineq2 of_real_add that)
  have summable (λn. a n * complex_of_real r ^ n)
    using assms ‹r > 0› by simp
  moreover have ⋀z. cmod z < r ⟹ cmod ((of_real r + of_real (cmod z)) /
2) < cmod (of_real r)
    using ‹r > 0›
    by (simp flip: of_real_add)
  ultimately have sum: ⋀z. cmod z < r ⟹ summable (λn. of_real (cmod (a
n)) * ((of_real r + complex_of_real (cmod z)) / 2) ^ n)
    by (rule power_series_conv_imp_absconv_weak)
  have ∃g g'. ∀z ∈ ball 0 r. (λn. (a n) * z ^ n) sums g z ∧
        (λn. of_nat n * (a n) * z ^(n − 1)) sums g' z ∧ (g has_field_derivative
g' z) (at z)
   apply (rule series_and_derivative_comparison_complex [OF open_ball der])
    apply (rule_tac x=(r − norm z)/2 in exI)
    apply (rule_tac x=λn. of_real(norm(a n)*((r + norm z)/2)^n) in exI)
    using ‹r > 0›
      apply (auto simp: sum eventually_sequentially norm_mult norm_power
dist_norm intro!: mult_left_mono power_mono y_le)
    done
 then show ?thesis
   by (simp add: ball_def)
next
 case False then show ?thesis
   unfolding not_less using less_le_trans norm_not_less_zero by blast
qed

proposition power_series_and_derivative:
 fixes a :: nat ⇒ complex and r::real
 assumes summable (λn. a n * r^n)
   obtains g g' where ∀z ∈ ball w r.
        ((λn. a n * (z − w) ^ n) sums g z) ∧ ((λn. of_nat n * a n * (z − w) ^
(n − 1)) sums g' z) ∧
          (g has_field_derivative g' z) (at z)
 using power_series_and_derivative_0 [OF assms]
 apply clarify
 apply (rule_tac g=(λz. g(z − w)) in that)
 using DERIV_shift [where z=−w]
 apply (auto simp: norm_minus_commute Ball_def dist_norm)
 done

proposition power_series_holomorphic:
 assumes ⋀w. w ∈ ball z r ⟹ ((λn. a n*(w − z)^n) sums f w)
   shows f holomorphic_on ball z r
proof −
```

```
  have ∃f'. (f has_field_derivative f') (at w) if w: dist z w < r for w
  proof −
    have wz: cmod (w − z) < r using w
      by (auto simp: field_split_simps dist_norm norm_minus_commute)
    then have 0 ≤ r
      by (meson less_eq_real_def norm_ge_zero order_trans)
    have inb: z + complex_of_real ((dist z w + r) / 2) ∈ ball z r
      using w by (simp add: dist_norm ‹0≤r› flip: of_real_add)
    have sum: summable (λn. a n * of_real (((cmod (z − w) + r) / 2) ^ n))
      using assms [OF inb] by (force simp: summable_def dist_norm)
    obtain g g' where gg': ⋀u. u ∈ ball z ((cmod (z − w) + r) / 2) ⟹
                           (λn. a n * (u − z) ^ n) sums g u ∧
                           (λn. of_nat n * a n * (u − z) ^ (n − 1)) sums g' u ∧ (g
has_field_derivative g' u) (at u)
      by (rule power_series_and_derivative [OF sum, of z]) fastforce
    have [simp]: g u = f u if cmod (u − w) < (r − cmod (z − w)) / 2 for u
    proof −
      have less: cmod (z − u) * 2 < cmod (z − w) + r
        using that dist_triangle2 [of z u w]
        by (simp add: dist_norm [symmetric] algebra_simps)
      have (λn. a n * (u − z) ^ n) sums g u (λn. a n * (u − z) ^ n) sums f u
        using gg' [of u] less w by (auto simp: assms dist_norm)
      then show ?thesis
        by (metis sums_unique2)
    qed
    have (f has_field_derivative g' w) (at w)
      by (rule has_field_derivative_transform_within [where d=(r − norm(z −
w))/2])
       (use w gg' [of w] in ‹(force simp: dist_norm)+›)
    then show ?thesis ..
  qed
  then show ?thesis by (simp add: holomorphic_on_open)
qed

corollary holomorphic_iff_power_series:
     f holomorphic_on ball z r ⟷
      (∀w ∈ ball z r. (λn. (deriv ^^ n) f z / (fact n) * (w − z)^n) sums f w)
  apply (intro iffI ballI holomorphic_power_series, assumption+)
  apply (force intro: power_series_holomorphic [where a = λn. (deriv ^^ n) f z
/ (fact n)])
  done

lemma power_series_analytic:
     (⋀w. w ∈ ball z r ⟹ (λn. a n*(w − z)^n) sums f w) ⟹ f analytic_on ball
z r
  by (force simp: analytic_on_open intro!: power_series_holomorphic)

lemma analytic_iff_power_series:
     f analytic_on ball z r ⟷
```

(∀ w ∈ *ball z r*. (λn. (*deriv* ^^ n) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*) *sums f w*)
by (*simp add*: *analytic_on_open holomorphic_iff_power_series*)

4.11 Equality between holomorphic functions, on open ball then connected set

lemma *holomorphic_fun_eq_on_ball*:
⟦*f holomorphic_on ball z r*; *g holomorphic_on ball z r*;
w ∈ *ball z r*;
⋀n. (*deriv* ^^ n) *f z* = (*deriv* ^^ n) *g z*⟧
⟹ *f w* = *g w*
by (*auto simp*: *holomorphic_iff_power_series sums_unique2* [*of* λn. (*deriv* ^^ n) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*])

lemma *holomorphic_fun_eq_0_on_ball*:
⟦*f holomorphic_on ball z r*; *w* ∈ *ball z r*;
⋀n. (*deriv* ^^ n) *f z* = 0⟧
⟹ *f w* = 0
using *holomorphic_fun_eq_on_ball* [**where** *g* = λz. 0] **by** *simp*

lemma *holomorphic_fun_eq_0_on_connected*:
assumes *holf*: *f holomorphic_on S* **and** *open S*
and *cons*: *connected S*
and *der*: ⋀n. (*deriv* ^^ n) *f z* = 0
and *z* ∈ *S w* ∈ *S*
shows *f w* = 0
proof −
have ∗: *ball x e* ⊆ (⋂ n. {*w* ∈ *S*. (*deriv* ^^ n) *f w* = 0})
if ∀ u. (*deriv* ^^ u) *f x* = 0 *ball x e* ⊆ *S* **for** *x e*
proof −
have (*deriv* ^^ m) ((*deriv* ^^ n) *f*) *x* = 0 **for** *m n*
by (*metis funpow_add o_apply that*(1))
then have ⋀x′ n. *dist x x*′ < *e* ⟹ (*deriv* ^^ n) *f x*′ = 0
using ‹*open S*›
by (*meson holf holomorphic_fun_eq_0_on_ball holomorphic_higher_deriv holomorphic_on_subset mem_ball that*(2))
with *that* **show** *?thesis* **by** *auto*
qed
obtain *e* **where** *e*>0 **and** *e*: *ball w e* ⊆ *S* **using** *openE* [*OF* ‹*open S*› ‹*w* ∈ *S*›]
.
then have *holfb*: *f holomorphic_on ball w e*
using *holf holomorphic_on_subset* **by** *blast*
have *open* (⋂ n. {*w* ∈ *S*. (*deriv* ^^ n) *f w* = 0})
using ‹*open S*›
apply (*simp add*: *open_contains_ball Ball_def image_iff*)
by (*metis* (*mono_tags*) ∗ *mem_Collect_eq*)
then have *openin* (*top_of_set S*) (⋂ n. {*w* ∈ *S*. (*deriv* ^^ n) *f w* = 0})
by (*force intro*: *open_subset*)
moreover have *closedin* (*top_of_set S*) (⋂ n. {*w* ∈ *S*. (*deriv* ^^ n) *f w* = 0})

using *assms*
by (*auto intro*: *continuous_closedin_preimage_constant holomorphic_on_imp_continuous_on holomorphic_higher_deriv*)
moreover have $(\bigcap n.\ \{w \in S.\ (deriv\ \widehat{\ \ }\ n)\ f\ w = 0\}) = S \Longrightarrow f\ w = 0$
using ‹*e*>0› *e* **by** (*force intro*: *holomorphic_fun_eq_0_on_ball* [*OF holfb*])
ultimately show *?thesis*
using *cons der* ‹$z \in S$›
by (*auto simp add*: *connected_clopen*)
qed

lemma *holomorphic_fun_eq_on_connected*:
assumes *f holomorphic_on S g holomorphic_on S* **and** *open S connected S*
and $\bigwedge n.\ (deriv\ \widehat{\ \ }\ n)\ f\ z = (deriv\ \widehat{\ \ }\ n)\ g\ z$
and $z \in S\ w \in S$
shows $f\ w = g\ w$
proof (*rule holomorphic_fun_eq_0_on_connected* [*of* $\lambda x.\ f\ x - g\ x\ S\ z$, *simplified*])
show $(\lambda x.\ f\ x - g\ x)$ *holomorphic_on S*
by (*intro assms holomorphic_intros*)
show $\bigwedge n.\ (deriv\ \widehat{\ \ }\ n)\ (\lambda x.\ f\ x - g\ x)\ z = 0$
using *assms higher_deriv_diff* **by** *auto*
qed (*use assms* **in** *auto*)

lemma *holomorphic_fun_eq_const_on_connected*:
assumes *holf*: *f holomorphic_on S* **and** *open S*
and *cons*: *connected S*
and *der*: $\bigwedge n.\ 0 < n \Longrightarrow (deriv\ \widehat{\ \ }\ n)\ f\ z = 0$
and $z \in S\ w \in S$
shows $f\ w = f\ z$
proof (*rule holomorphic_fun_eq_0_on_connected* [*of* $\lambda w.\ f\ w - f\ z\ S\ z$, *simplified*])
show $(\lambda w.\ f\ w - f\ z)$ *holomorphic_on S*
by (*intro assms holomorphic_intros*)
show $\bigwedge n.\ (deriv\ \widehat{\ \ }\ n)\ (\lambda w.\ f\ w - f\ z)\ z = 0$
by (*subst higher_deriv_diff*) (*use assms* **in** ‹*auto intro*: *holomorphic_intros*›)
qed (*use assms* **in** *auto*)

4.12 Some basic lemmas about poles/singularities

lemma *pole_lemma*:
assumes *holf*: *f holomorphic_on S* **and** *a*: $a \in interior\ S$
shows $(\lambda z.\ if\ z = a\ then\ deriv\ f\ a$
$else\ (f\ z - f\ a)\ /\ (z - a))\ holomorphic\_on\ S$ (**is** *?F holomorphic_on S*)
proof −
have ∗: *?F field_differentiable* (*at u within S*) **if** $u \in S\ u \neq a$ **for** *u*
proof −
have *fcd*: *f field_differentiable at u within S*
using *holf holomorphic_on_def* **by** (*simp add*: ‹$u \in S$›)

```
    have cd: (λz. (f z − f a) / (z − a)) field_differentiable at u within S
      by (rule fcd derivative_intros | simp add: that)+
    have 0 < dist a u using that dist_nz by blast
    then show ?thesis
      by (rule field_differentiable_transform_within [OF _ _ _ cd]) (auto simp:
‹u ∈ S›)
  qed
  moreover
  have ?F field_differentiable at a if 0 < e ball a e ⊆ S for e
  proof −
    have holfb: f holomorphic_on ball a e
      by (rule holomorphic_on_subset [OF holf ‹ball a e ⊆ S›])
    have 2: ?F holomorphic_on ball a e − {a}
      using mem_ball that
      by (auto simp add: holomorphic_on_def simp flip: field_differentiable_def
intro: ∗ field_differentiable_within_subset)
    have isCont (λz. if z = a then deriv f a else (f z − f a) / (z − a)) x
          if dist a x < e for x
    proof (cases x=a)
      case True
      then have f field_differentiable at a
        using holfb ‹0 < e› holomorphic_on_imp_differentiable_at by auto
      with True show ?thesis
        by (smt (verit) DERIV_deriv_iff_field_differentiable LIM_equal continu-
ous_at has_field_derivativeD)
    next
      case False with 2 that show ?thesis
      by (simp add: field_differentiable_imp_continuous_at holomorphic_on_imp_differentiable_at
open_Diff)
    qed
    then have 1: continuous_on (ball a e) ?F
      by (clarsimp simp: continuous_on_eq_continuous_at)
    have ?F holomorphic_on ball a e
      by (auto intro: no_isolated_singularity [OF 1 2])
    with that show ?thesis
      by (simp add: holomorphic_on_open field_differentiable_def [symmetric]
                    field_differentiable_at_within)
  qed
  ultimately show ?thesis
    by (metis (no_types, lifting) holomorphic_onI a field_differentiable_at_within
interior_subset openE open_interior subset_iff)
qed

lemma pole_theorem:
  assumes holg: g holomorphic_on S and a: a ∈ interior S
      and eq: ⋀z. z ∈ S − {a} ⟹ g z = (z − a) ∗ f z
    shows (λz. if z = a then deriv g a
               else f z − g a/(z − a)) holomorphic_on S
  using pole_lemma [OF holg a]
```

by (*rule holomorphic_transform*) (*simp add*: *eq field_split_simps*)

lemma *pole_lemma_open*:
assumes *f holomorphic_on S open S*
shows (*λz. if z = a then deriv f a else* (*f z − f a*)/(*z − a*)) *holomorphic_on S*
proof (*cases a ∈ S*)
case *True* **with** *assms interior_eq pole_lemma*
show *?thesis* **by** *fastforce*
next
case *False* **with** *assms* **show** *?thesis*
apply (*simp add*: *holomorphic_on_def field_differentiable_def* [*symmetric*], *clarify*)
apply (*rule field_differentiable_transform_within* [**where** *f* = *λz.* (*f z − f a*)/(*z − a*) **and** *d* = *1*])
apply (*rule derivative_intros* | *force*)+
done
qed

lemma *pole_theorem_open*:
assumes *holg*: *g holomorphic_on S* **and** *S*: *open S*
and *eq*: ⋀*z*. *z ∈ S − {a} ⟹ g z* = (*z − a*) ∗ *f z*
shows (*λz. if z = a then deriv g a*
else f z − g a/(*z − a*)) *holomorphic_on S*
using *pole_lemma_open* [*OF holg S*]
by (*rule holomorphic_transform*) (*auto simp*: *eq divide_simps*)

lemma *pole_theorem_0*:
assumes *holg*: *g holomorphic_on S* **and** *a*: *a ∈ interior S*
and *eq*: ⋀*z*. *z ∈ S − {a} ⟹ g z* = (*z − a*) ∗ *f z*
and [*simp*]: *f a* = *deriv g a g a* = *0*
shows *f holomorphic_on S*
using *pole_theorem* [*OF holg a eq*]
by (*rule holomorphic_transform*) (*auto simp*: *eq field_split_simps*)

lemma *pole_theorem_open_0*:
assumes *holg*: *g holomorphic_on S* **and** *S*: *open S*
and *eq*: ⋀*z*. *z ∈ S − {a} ⟹ g z* = (*z − a*) ∗ *f z*
and [*simp*]: *f a* = *deriv g a g a* = *0*
shows *f holomorphic_on S*
using *pole_theorem_open* [*OF holg S eq*]
by (*rule holomorphic_transform*) (*auto simp*: *eq field_split_simps*)

lemma *pole_theorem_analytic*:
assumes *g*: *g analytic_on S*
and *eq*: ⋀*z*. *z ∈ S*
⟹ ∃ *d*. *0 < d* ∧ (∀ *w ∈ ball z d − {a}*. *g w* = (*w − a*) ∗ *f w*)
shows (*λz. if z = a then deriv g a else f z − g a*/(*z − a*)) *analytic_on S* (**is** *?F analytic_on S*)
unfolding *analytic_on_def*

proof
fix x
assume $x \in S$
with g **obtain** e **where** $0 < e$ **and** e: g *holomorphic_on ball* x e
by (*auto simp add*: *analytic_on_def*)
obtain d **where** $0 < d$ **and** d: $\bigwedge w.\ w \in \mathit{ball}\ x\ d - \{a\} \Longrightarrow g\ w = (w - a) * f\ w$
using ‹$x \in S$› *eq* **by** *blast*
have *?F holomorphic_on ball* x (*min* d e)
using d e ‹$x \in S$› **by** (*fastforce simp*: *holomorphic_on_subset subset_ball intro*!: *pole_theorem_open*)
then show $\exists\, e{>}0.$ *?F holomorphic_on ball* x e
using ‹$0 < d$› ‹$0 < e$› *not_le* **by** *fastforce*
qed

lemma *pole_theorem_analytic_0*:
assumes g: g *analytic_on* S
and *eq*: $\bigwedge z.\ z \in S \Longrightarrow \exists\, d.\ 0 < d \wedge (\forall\, w \in \mathit{ball}\ z\ d - \{a\}.\ g\ w = (w - a) * f\ w)$
and [*simp*]: $f\ a = \mathit{deriv}\ g\ a\ g\ a = 0$
shows f *analytic_on* S
proof −
have [*simp*]: $(\lambda z.\ \mathit{if}\ z = a\ \mathit{then}\ \mathit{deriv}\ g\ a\ \mathit{else}\ f\ z - g\ a\ /\ (z - a)) = f$
by *auto*
show *?thesis*
using *pole_theorem_analytic* [*OF g eq*] **by** *simp*
qed

lemma *pole_theorem_analytic_open_superset*:
assumes g: g *analytic_on* S **and** $S \subseteq T$ *open* T
and *eq*: $\bigwedge z.\ z \in T - \{a\} \Longrightarrow g\ z = (z - a) * f\ z$
shows $(\lambda z.\ \mathit{if}\ z = a\ \mathit{then}\ \mathit{deriv}\ g\ a$
else $f\ z - g\ a/(z - a))$ *analytic_on* S
proof (*rule pole_theorem_analytic* [*OF g*])
fix z
assume $z \in S$
then obtain e **where** $0 < e$ **and** e: *ball* $z\ e \subseteq T$
using *assms openE* **by** *blast*
then show $\exists\, d{>}0.\ \forall\, w{\in}\mathit{ball}\ z\ d - \{a\}.\ g\ w = (w - a) * f\ w$
using *eq* **by** *auto*
qed

lemma *pole_theorem_analytic_open_superset_0*:
assumes g: g *analytic_on* S $S \subseteq T$ *open* T $\bigwedge z.\ z \in T - \{a\} \Longrightarrow g\ z = (z - a) * f\ z$
and [*simp*]: $f\ a = \mathit{deriv}\ g\ a\ g\ a = 0$
shows f *analytic_on* S
proof −
have [*simp*]: $(\lambda z.\ \mathit{if}\ z = a\ \mathit{then}\ \mathit{deriv}\ g\ a\ \mathit{else}\ f\ z - g\ a\ /\ (z - a)) = f$

```
    by auto
  have (λz. if z = a then deriv g a else f z − g a/(z − a)) analytic_on S
    by (rule pole_theorem_analytic_open_superset [OF g])
  then show ?thesis by simp
qed
```

4.13 General, homology form of Cauchy's theorem

Proof is based on Dixon's, as presented in Lang's "Complex Analysis" book (page 147).

```
lemma contour_integral_continuous_on_linepath_2D:
  assumes open U and cont_dw: ⋀w. w ∈ U ⟹ F w contour_integrable_on (linepath a b)
      and cond_uu: continuous_on (U × U) (λ(x,y). F x y)
      and abu: closed_segment a b ⊆ U
    shows continuous_on U (λw. contour_integral (linepath a b) (F w))
proof −
  have ∗: ∃ d>0. ∀ x′∈U. dist x′ w < d ⟶
                      dist (contour_integral (linepath a b) (F x′))
                           (contour_integral (linepath a b) (F w)) ≤ ε
        if w ∈ U 0 < ε a ≠ b for w ε
  proof −
    obtain δ where δ>0 and δ: cball w δ ⊆ U using open_contains_cball ‹open U› ‹w ∈ U› by force
    let ?TZ = cball w δ  × closed_segment a b
    have uniformly_continuous_on ?TZ (λ(x,y). F x y)
    proof (rule compact_uniformly_continuous)
      show continuous_on ?TZ (λ(x,y). F x y)
        by (rule continuous_on_subset[OF cond_uu]) (use SigmaE δ abu in blast)
      show compact ?TZ
        by (simp add: compact_Times)
    qed
    then obtain η where η>0
       and η: ⋀x x′. ⟦x∈?TZ; x′∈?TZ; dist x′ x < η⟧ ⟹
                      dist ((λ(x,y). F x y) x′) ((λ(x,y). F x y) x) < ε/norm(b − a)
      using ‹0 < ε› ‹a ≠ b›
      by (auto elim: uniformly_continuous_onE [where e = ε/norm(b − a)])
    have η: ⟦norm (w − x1) ≤ δ;  x2 ∈ closed_segment a b;
             norm (w − x1′) ≤ δ;  x2′ ∈ closed_segment a b; norm ((x1′, x2′) − (x1, x2)) < η⟧
            ⟹ norm (F x1′ x2′ − F x1 x2) ≤ ε / cmod (b − a)
          for x1 x2 x1′ x2′
      using η [of (x1,x2) (x1′,x2′)] by (force simp: dist_norm)
    have le_ee: cmod (contour_integral (linepath a b) (λx. F x′ x − F w x)) ≤ ε
              if x′ ∈ U cmod (x′ − w) < δ cmod (x′ − w) < η  for x′
    proof −
      have (λx. F x′ x − F w x) contour_integrable_on linepath a b
        by (simp add: ‹w ∈ U› cont_dw contour_integrable_diff that)
      then have cmod (contour_integral (linepath a b) (λx. F x′ x − F w x)) ≤
```

```
ε/norm(b − a) * norm(b − a)
      using has_contour_integral_bound_linepath [OF has_contour_integral_integral
_ η]
        using ‹0 < ε› ‹0 < δ› that by (force simp: norm_minus_commute)
      also have ... = ε using ‹a ≠ b› by simp
      finally show ?thesis .
    qed
    show ?thesis
      apply (rule_tac x=min δ η in exI)
      using ‹0 < δ› ‹0 < η›
        by (auto simp: dist_norm contour_integral_diff [OF cont_dw cont_dw,
symmetric] ‹w ∈ U› intro: le_ee)
  qed
  show ?thesis
  proof (cases a=b)
    case False
    show ?thesis
      by (rule continuous_onI) (use False in ‹auto intro: *›)
  qed auto
qed
```

This version has *polynomial_function* γ as an additional assumption.

```
lemma Cauchy_integral_formula_global_weak:
  assumes open U and holf: f holomorphic_on U
        and z: z ∈ U and γ: polynomial_function γ
        and pasz: path_image γ ⊆ U − {z} and loop: pathfinish γ = pathstart γ
        and zero: ⋀w. w ∉ U ⟹ winding_number γ w = 0
    shows ((λw. f w / (w − z)) has_contour_integral (2*pi * ɪ * winding_number
γ z * f z)) γ
proof −
  obtain γ' where pfγ': polynomial_function γ' and γ': ⋀x. (γ has_vector_derivative
(γ' x)) (at x)
    using has_vector_derivative_polynomial_function [OF γ] by blast
  then have bounded(path_image γ')
   by (simp add: path_image_def compact_imp_bounded compact_continuous_image
continuous_on_polymonial_function)
  then obtain B where B>0 and B: ⋀x. x ∈ path_image γ' ⟹ norm x ≤ B
    using bounded_pos by force
  define d where [abs_def]: d z w = (if w = z then deriv f z else (f w − f z)/(w
− z)) for z w
  define v where v = {w. w ∉ path_image γ ∧ winding_number γ w = 0}
  have path γ valid_path γ using γ
    by (auto simp: path_polynomial_function valid_path_polynomial_function)
  then have ov: open v
    by (simp add: v_def open_winding_number_levelsets loop)
  have uv_Un: U ∪ v = UNIV
    using pasz zero by (auto simp: v_def)
  have conf: continuous_on U f
    by (metis holf holomorphic_on_imp_continuous_on)
```

```
  have hol_d: (d y) holomorphic_on U if y ∈ U for y
  proof −
    have ∗: (λc. if c = y then deriv f y else (f c − f y) / (c − y)) holomorphic_on
U
      by (simp add: holf pole_lemma_open ‹open U›)
    then have isCont (λx. if x = y then deriv f y else (f x − f y) / (x − y)) y
        using at_within_open field_differentiable_imp_continuous_at holomor-
phic_on_def that ‹open U› by fastforce
    then have continuous_on U (d y)
      using ∗ d_def holomorphic_on_imp_continuous_on by auto
    moreover have d y holomorphic_on U − {y}
    proof −
     have (λw. if w = y then deriv f y else (f w − f y) / (w − y)) field_differentiable
at w
        if w ∈ U − {y} for w
      proof (rule field_differentiable_transform_within)
        show (λw. (f w − f y) / (w − y)) field_differentiable at w
          using that ‹open U› holf
         by (auto intro!: holomorphic_on_imp_differentiable_at derivative_intros)
        show dist w y > 0
          using that by auto
      qed (auto simp: dist_commute)
      then show ?thesis
      unfolding field_differentiable_def by (simp add: d_def holomorphic_on_open
‹open U› open_delete)
    qed
    ultimately show ?thesis
      by (rule no_isolated_singularity) (auto simp: ‹open U›)
  qed
   have cint_fxy: (λx. (f x − f y) / (x − y)) contour_integrable_on γ if y ∉
path_image γ for y
  proof (rule contour_integrable_holomorphic_simple [where S = U−{y}])
    show (λx. (f x − f y) / (x − y)) holomorphic_on U − {y}
      by (force intro: holomorphic_intros holomorphic_on_subset [OF holf])
    show path_image γ ⊆ U − {y}
      using pasz that by blast
  qed (auto simp: ‹open U› open_delete ‹valid_path γ›)
  define h where
    h z = (if z ∈ U then contour_integral γ (d z) else contour_integral γ (λw. f
w/(w − z))) for z
  have U: ((d z) has_contour_integral h z) γ if z ∈ U for z
  proof −
    have d z holomorphic_on U
      by (simp add: hol_d that)
    with that show ?thesis
    by (metis Diff_subset ‹valid_path γ› ‹open U› contour_integrable_holomorphic_simple
h_def has_contour_integral_integral pasz subset_trans)
  qed
  have V: ((λw. f w / (w − z)) has_contour_integral h z) γ if z: z ∈ v for z
```

proof −
have *0*: *0* = (*f z*) ∗ *2* ∗ *of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number* *γ z*
using *v_def z* **by** *auto*
then have ((λ*x*. *1* / (*x* − *z*)) *has_contour_integral 0*) *γ*
using *z v_def has_contour_integral_winding_number* [*OF* ‹*valid_path γ*›]
by *fastforce*
then have ((λ*x*. *f z* ∗ (*1* / (*x* − *z*))) *has_contour_integral 0*) *γ*
using *has_contour_integral_lmul* **by** *fastforce*
then have ((λ*x*. *f z* / (*x* − *z*)) *has_contour_integral 0*) *γ*
by (*simp add*: *field_split_simps*)
moreover have ((λ*x*. (*f x* − *f z*) / (*x* − *z*)) *has_contour_integral contour_integral γ* (*d z*)) *γ*
using *z*
apply (*simp add*: *v_def*)
apply (*metis* (*no_types*, *lifting*) *contour_integrable_eq d_def has_contour_integral_eq has_contour_integral_integral cint_fxy*)
done
ultimately have ∗: ((λ*x*. *f z* / (*x* − *z*) + (*f x* − *f z*) / (*x* − *z*)) *has_contour_integral* (*0* + *contour_integral γ* (*d z*))) *γ*
by (*rule has_contour_integral_add*)
have ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral contour_integral γ* (*d z*)) *γ*
if *z* ∈ *U*
using ∗ **by** (*auto simp*: *divide_simps has_contour_integral_eq*)
moreover have ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral contour_integral γ* (λ*w*. *f w* / (*w* − *z*))) *γ*
if *z* ∉ *U*
proof (*rule has_contour_integral_integral* [*OF contour_integrable_holomorphic_simple* [**where** *S*=*U*]])
show (λ*w*. *f w* / (*w* − *z*)) *holomorphic_on U*
by (*rule holomorphic_intros assms* | *use that* **in** *force*)+
qed (*use* ‹*open U*› *pasz* ‹*valid_path γ*› **in** *auto*)
ultimately show *?thesis*
using *z* **by** (*simp add*: *h_def*)
qed
have *znot*: *z* ∉ *path_image γ*
using *pasz* **by** *blast*
obtain *d0* **where** *d0*>*0* **and** *d0*: ⋀*x y*. *x* ∈ *path_image γ* ⟹ *y* ∈ − *U* ⟹ *d0* ≤ *dist x y*
using *separate_compact_closed* [*of path_image γ* −*U*] *pasz* ‹*open U*› ‹*path γ*› *compact_path_image*
by *blast*
obtain *dd* **where** *0* < *dd* **and** *dd*: {*y* + *k* | *y k*. *y* ∈ *path_image γ* ∧ *k* ∈ *ball 0 dd*} ⊆ *U*
proof
show *0* < *d0* / *2* **using** ‹*0* < *d0*› **by** *auto*
qed (*use* ‹*0* < *d0*› *d0* **in** ‹*force simp*: *dist_norm*›)
define *T* **where** *T* ≡ {*y* + *k* |*y k*. *y* ∈ *path_image γ* ∧ *k* ∈ *cball 0* (*dd* / *2*)}
have ⋀*x x'*. ⟦*x* ∈ *path_image γ*; *dist x x'* ∗ *2* < *dd*⟧ ⟹ ∃*y k*. *x'* = *y* + *k* ∧ *y* ∈ *path_image γ* ∧ *dist 0 k* ∗ *2* ≤ *dd*

```
    apply (rule_tac x=x in exI)
    apply (rule_tac x=x'−x in exI)
    apply (force simp: dist_norm)
    done
  then have subt: path_image γ ⊆ interior T
    using ‹0 < dd›
    apply (clarsimp simp add: mem_interior T_def)
    apply (rule_tac x=dd/2 in exI, auto)
    done
  have compact T
    unfolding T_def
   using ‹valid_path γ› compact_cball compact_sums compact_valid_path_image
by blast
  have T: T ⊆ U
    unfolding T_def using ‹0 < dd› dd by fastforce
  obtain L where L>0
          and L: ⋀f B. ⟦f holomorphic_on interior T; ⋀z. z∈interior T ⟹ cmod
(f z) ≤ B⟧ ⟹
                       cmod (contour_integral γ f) ≤ L * B
      using contour_integral_bound_exists [OF open_interior ‹valid_path γ› subt]
       by blast
  have bounded(f ‘ T)
     by (meson ‹compact T› compact_continuous_image compact_imp_bounded
conf continuous_on_subset T)
  then obtain D where D>0 and D: ⋀x. x ∈ T ⟹ norm (f x) ≤ D
    by (auto simp: bounded_pos)
  obtain C where C>0 and C: ⋀x. x ∈ T ⟹ norm x ≤ C
    using ‹compact T› bounded_pos compact_imp_bounded by force
  have dist (h y) 0 ≤ e if 0 < e and le: D * L / e + C ≤ cmod y for e y
  proof −
    have D * L / e > 0 using ‹D>0› ‹L>0› ‹e>0› by simp
    with le have ybig: norm y > C by force
    with C have y ∉ T  by force
    then have ynot: y ∉ path_image γ
      using subt interior_subset by blast
    have [simp]: winding_number γ y = 0
    proof (rule winding_number_zero_outside)
      show path_image γ ⊆ cball 0 C
        by (meson C interior_subset mem_cball_0 subset_eq subt)
    qed (use ybig loop ‹path γ› in auto)
    have [simp]: h y = contour_integral γ (λw. f w/(w − y))
      by (rule contour_integral_unique [symmetric]) (simp add: v_def ynot V)
    have holint: (λw. f w / (w − y)) holomorphic_on interior T
    proof (intro holomorphic_intros)
      show f holomorphic_on interior T
        using holf holomorphic_on_subset interior_subset T by blast
    qed (use ‹y ∉ T› interior_subset in auto)
    have leD: cmod (f z / (z − y)) ≤ D * (e / L / D) if z: z ∈ interior T for z
    proof −
```

```
    have D * L / e + cmod z ≤ cmod y
      using le C [of z] z using interior_subset by force
    then have DL2: D * L / e ≤ cmod (z - y)
     using norm_triangle_ineq2 [of y z] by (simp add: norm_minus_commute)
    have cmod (f z / (z - y)) = cmod (f z) * inverse (cmod (z - y))
    by (simp add: norm_mult norm_inverse Fields.field_class.field_divide_inverse)
    also have ... ≤ D * (e / L / D)
    proof (rule mult_mono)
      show cmod (f z) ≤ D
        using D interior_subset z by blast
      show inverse (cmod (z - y)) ≤ e / L / D D ≥ 0
     using ‹L>0› ‹e>0› ‹D>0› DL2 by (auto simp: norm_divide field_split_simps)
    qed auto
    finally show ?thesis .
  qed
  have dist (h y) 0 = cmod (contour_integral γ (λw. f w / (w - y)))
    by (simp add: dist_norm)
  also have ... ≤ L * (D * (e / L / D))
    by (rule L [OF holint leD])
  also have ... = e
    using ‹L>0› ‹0 < D› by auto
  finally show ?thesis .
 qed
 then have (h ⟶ 0) at_infinity
  by (meson Lim_at_infinityI)
 moreover have h holomorphic_on UNIV
 proof −
  have con_ff: continuous (at (x,z)) (λ(x,y). (f y − f x) / (y − x))
              if x ∈ U z ∈ U x ≠ z for x z
    using that conf
    apply (simp add: split_def continuous_on_eq_continuous_at ‹open U›)
    apply (simp | rule continuous_intros continuous_within_compose2 [where
g=f])+
    done
  have con_fstsnd: continuous_on UNIV (λx. (fst x − snd x) ::complex)
    by (rule continuous_intros)+
  have open_uu_Id: open (U × U − Id)
  proof (rule open_Diff)
    show open (U × U)
      by (simp add: open_Times ‹open U›)
    show closed (Id :: complex rel)
     using continuous_closed_preimage_constant [OF con_fstsnd closed_UNIV,
of 0]
      by (auto simp: Id_fstsnd_eq algebra_simps)
  qed
  have con_derf: continuous (at z) (deriv f) if z ∈ U for z
     by (meson analytic_at analytic_at_imp_isCont assms(1) holf holomor-
phic_deriv that)
  have tendsto_f': ((λ(x,y). if y = x then deriv f (x)
```

```
                    else (f (y) − f (x)) / (y − x)) ⟶ deriv f x)
                (at (x, x) within U × U) if x ∈ U for x
  proof (rule Lim_withinI)
    fix e::real assume 0 < e
    obtain k1 where k1>0 and k1: ⋀x'. norm (x' − x) ≤ k1 ⟹ norm (deriv
f x' − deriv f x) < e
      using ‹0 < e› continuous_within_E [OF con_derf [OF ‹x ∈ U›]]
      by (metis UNIV_I dist_norm)
    obtain k2 where k2>0 and k2: ball x k2 ⊆ U
      by (blast intro: openE [OF ‹open U›] ‹x ∈ U›)
    have neq: norm ((f z' − f x') / (z' − x') − deriv f x) ≤ e
                  if z' ≠ x' and less_k1: norm (x'−x, z'−x) < k1 and less_k2:
norm (x'−x, z'−x) < k2
              for x' z'
    proof −
      have cs_less: w ∈ closed_segment x' z' ⟹ cmod (w − x) ≤ norm (x'−x,
z'−x) for w
        using segment_furthest_le [of w x' z' x]
       by (metis (no_types) dist_commute dist_norm norm_fst_le norm_snd_le
order_trans)
      have derf_le: w ∈ closed_segment x' z' ⟹ z' ≠ x' ⟹ cmod (deriv f w −
deriv f x) ≤ e for w
      by (blast intro: cs_less less_k1 k1 [unfolded divide_const_simps dist_norm]
less_imp_le le_less_trans)
       have f_has_der: ⋀x. x ∈ U ⟹ (f has_field_derivative deriv f x) (at x
within U)
         by (metis DERIV_deriv_iff_field_differentiable at_within_open holf
holomorphic_on_def ‹open U›)
      have closed_segment x' z' ⊆ U
       by (rule order_trans [OF _ k2]) (simp add: cs_less le_less_trans [OF _
less_k2] dist_complex_def norm_minus_commute subset_iff)
      then have cint_derf: (deriv f has_contour_integral f z' − f x') (linepath x'
z')
        using contour_integral_primitive [OF f_has_der valid_path_linepath]
pasz by simp
      then have ∗: ((λx. deriv f x / (z' − x')) has_contour_integral (f z' − f x')
/ (z' − x')) (linepath x' z')
       by (rule has_contour_integral_div)
       have norm ((f z' − f x') / (z' − x') − deriv f x) ≤ e/norm(z' − x') ∗
norm(z' − x')
     apply (rule has_contour_integral_bound_linepath [OF has_contour_integral_diff
[OF ∗]])
     using has_contour_integral_div [where c = z' − x', OF has_contour_integral_const_linepath
[of deriv f x z' x']]
           ‹e > 0› ‹z' ≠ x'›
       apply (auto simp: norm_divide divide_simps derf_le)
       done
     also have ... ≤ e using ‹0 < e› by simp
     finally show ?thesis .
```

```
    qed
    show ∃ d>0. ∀ xa∈U × U.
              0 < dist xa (x, x) ∧ dist xa (x, x) < d ⟶
             dist (case xa of (x, y) ⇒ if y = x then deriv f x else (f y − f x) / (y
− x)) (deriv f x) ≤ e
      apply (rule_tac x=min k1 k2 in exI)
      using ‹k1>0› ‹k2>0› ‹e>0›
    by (force simp: dist_norm neq intro: dual_order.strict_trans2 k1 less_imp_le
norm_fst_le)
  qed
  have con_pa_f: continuous_on (path_image γ) f
   by (meson holf holomorphic_on_imp_continuous_on holomorphic_on_subset
interior_subset subt T)
  have le_B: ⋀T. T ∈ {0..1} ⟹ cmod (vector_derivative γ (at T)) ≤ B
  using γ' B by (simp add: path_image_def vector_derivative_at rev_image_eqI)
  have f_has_cint: ⋀w. w ∈ v − path_image γ ⟹ ((λu. f u / (u − w) ^ 1)
has_contour_integral h w) γ
    by (simp add: V)
  have cond_uu: continuous_on (U × U) (λ(x,y). d x y)
      apply (simp add: continuous_on_eq_continuous_within d_def continu-
ous_within tendsto_f')
     apply (simp add: tendsto_within_open_NO_MATCH open_Times ‹open
U›, clarify)
    apply (rule Lim_transform_within_open [OF _ open_uu_Id, where f =
(λ(x,y). (f y − f x) / (y − x))])
    using con_ff
    apply (auto simp: continuous_within)
    done
  have hol_dw: (λz. d z w) holomorphic_on U if w ∈ U for w
  proof −
    have continuous_on U ((λ(x,y). d x y) ∘ (λz. (w,z)))
     by (rule continuous_on_compose continuous_intros continuous_on_subset
[OF cond_uu] | force intro: that)+
    then have ∗: continuous_on U (λz. if w = z then deriv f z else (f w − f z)
/ (w − z))
     by (rule rev_iffD1 [OF _ continuous_on_cong [OF refl]]) (simp add: d_def
field_simps)
  have ∗∗: (λz. if w = z then deriv f z else (f w − f z) / (w − z)) field_differentiable
at x
     if x ∈ U x ≠ w for x
     proof (rule_tac f = λx. (f w − f x)/(w − x) and d = dist x w in
field_differentiable_transform_within)
     show (λx. (f w − f x) / (w − x)) field_differentiable at x
      using that ‹open U›
       by (intro derivative_intros holomorphic_on_imp_differentiable_at [OF
holf]; force)
    qed (use that ‹open U› in ‹auto simp: dist_commute›)
    show ?thesis
     unfolding d_def
```

```
    proof (rule no_isolated_singularity [OF * _ ‹open U›])
     show (λz. if w = z then deriv f z else (f w − f z) / (w − z)) holomorphic_on
U − {w}
      by (auto simp: field_differentiable_def [symmetric] holomorphic_on_open
open_Diff ‹open U› **)
    qed auto
   qed
   { fix a b
    assume abu: closed_segment a b ⊆ U
     have cont_cint_d: continuous_on U (λw. contour_integral (linepath a b)
(λz. d z w))
    proof (rule contour_integral_continuous_on_linepath_2D [OF ‹open U› _
_ abu])
     show ⋀w. w ∈ U ⟹ (λz. d z w) contour_integrable_on (linepath a b)
     by (metis abu hol_dw continuous_on_subset contour_integrable_continuous_linepath
holomorphic_on_imp_continuous_on)
      show continuous_on (U × U) (λ(x, y). d y x)
       by (auto intro: continuous_on_swap_args cond_uu)
    qed
    have cont_cint_dγ: continuous_on {0..1} ((λw. contour_integral (linepath
a b) (λz. d z w)) ∘ γ)
    proof (rule continuous_on_compose)
     show continuous_on {0..1} γ
      using ‹path γ› path_def by blast
     show continuous_on (γ ‘ {0..1}) (λw. contour_integral (linepath a b) (λz.
d z w))
      using pasz unfolding path_image_def
      by (auto intro!: continuous_on_subset [OF cont_cint_d])
    qed
    have continuous_on {0..1} (λx. vector_derivative γ (at x))
    using pfγ′ by (simp add: continuous_on_polymonial_function vector_derivative_at
[OF γ′])
    then     have cint_cint: (λw. contour_integral (linepath a b) (λz. d z w))
contour_integrable_on γ
     apply (simp add: contour_integrable_on)
     apply (rule integrable_continuous_real)
     by (rule continuous_on_mult [OF cont_cint_dγ [unfolded o_def]])
   have contour_integral (linepath a b) h = contour_integral (linepath a b) (λz.
contour_integral γ (d z))
     using abu  by (force simp: h_def intro: contour_integral_eq)
   also have ... =  contour_integral γ (λw. contour_integral (linepath a b) (λz.
d z w))
    proof (rule contour_integral_swap)
     show continuous_on (path_image (linepath a b) × path_image γ) (λ(y1,
y2). d y1 y2)
      using abu pasz by (auto intro: continuous_on_subset [OF cond_uu])
     show continuous_on {0..1} (λt. vector_derivative (linepath a b) (at t))
      by (auto intro!: continuous_intros)
     show continuous_on {0..1} (λt. vector_derivative γ (at t))
```

```
        by (metis γ' continuous_on_eq path_def path_polynomial_function pfγ' vector_derivative_at)
      qed (use ‹valid_path γ› in auto)
      finally have cint_h_eq:
        contour_integral (linepath a b) h =
                  contour_integral γ (λw. contour_integral (linepath a b) (λz. d z w)) .
      note cint_cint cint_h_eq
    } note cint_h = this
    have conthu: continuous_on U h
    proof (simp add: continuous_on_sequentially, clarify)
      fix a x
      assume x: x ∈ U and au: ∀n. a n ∈ U and ax: a ⟶ x
      then have A1: ∀F n in sequentially. d (a n) contour_integrable_on γ
        by (meson U contour_integrable_on_def eventuallyI)
      obtain dd where dd>0 and dd: cball x dd ⊆ U using open_contains_cball ‹open U› x by force
      have A2: uniform_limit (path_image γ) (λn. d (a n)) (d x) sequentially
        unfolding uniform_limit_iff dist_norm
      proof clarify
        fix ee::real
        assume 0 < ee
        show ∀F n in sequentially. ∀ξ∈path_image γ. cmod (d (a n) ξ - d x ξ) < ee
        proof -
          let ?ddpa = {(w,z) |w z. w ∈ cball x dd ∧ z ∈ path_image γ}
          have uniformly_continuous_on ?ddpa (λ(x,y). d x y)
          proof (rule compact_uniformly_continuous [OF continuous_on_subset[OF cond_uu]])
            show compact {(w, z) |w z. w ∈ cball x dd ∧ z ∈ path_image γ}
              using ‹valid_path γ›
              by (auto simp: compact_Times compact_valid_path_image simp del: mem_cball)
          qed (use dd pasz in auto)
          then obtain kk where kk>0
            and kk: ⋀x x'. ⟦x ∈ ?ddpa; x' ∈ ?ddpa; dist x' x < kk⟧ ⟹
                           dist ((λ(x,y). d x y) x') ((λ(x,y). d x y) x) < ee
            by (rule uniformly_continuous_onE [where e = ee]) (use ‹0 < ee› in auto)
          have kk: ⟦norm (w - x) ≤ dd; z ∈ path_image γ; norm ((w, z) - (x, z)) < kk⟧ ⟹ norm (d w z - d x z) < ee
            for w z
            using ‹dd>0› kk [of (x,z) (w,z)] by (force simp: norm_minus_commute dist_norm)
          obtain no where ∀n≥no. dist (a n) x < min dd kk
            using ax unfolding lim_sequentially
            by (meson ‹0 < dd› ‹0 < kk› min_less_iff_conj)
          then show ?thesis
            using ‹dd > 0› ‹kk > 0› by (fastforce simp: eventually_sequentially kk
```

```
dist_norm)
      qed
    qed
    have (λn. contour_integral γ (d (a n))) ⟶ contour_integral γ (d x)
      by (rule contour_integral_uniform_limit [OF A1 A2 le_B]) (auto simp:
‹valid_path γ›)
    then have tendsto_hx: (λn. contour_integral γ (d (a n))) ⟶ h x
      by (simp add: h_def x)
    then show (h ∘ a) ⟶ h x
      by (simp add: h_def x au o_def)
  qed
  show ?thesis
  proof (simp add: holomorphic_on_open field_differentiable_def [symmetric],
clarify)
    fix z0
    consider z0 ∈ v | z0 ∈ U using uv_Un by blast
    then show h field_differentiable at z0
    proof cases
      assume z0 ∈ v then show ?thesis
        using Cauchy_next_derivative [OF con_pa_f le_B f_has_cint _ ov] V
f_has_cint ‹valid_path γ›
        by (auto simp: field_differentiable_def v_def)
    next
      assume z0 ∈ U then
       obtain e where e>0 and e: ball z0 e ⊆ U by (blast intro: openE [OF
‹open U›])
      have ∗: contour_integral (linepath a b) h + contour_integral (linepath b c)
h + contour_integral (linepath c a) h = 0
            if abc_subset: convex hull {a, b, c} ⊆ ball z0 e  for a b c
      proof −
       have ∗: ⋀x1 x2 z. z ∈ U ⟹ closed_segment x1 x2 ⊆ U ⟹ (λw. d w z)
contour_integrable_on linepath x1 x2
          using  hol_dw holomorphic_on_imp_continuous_on ‹open U›
          by (auto intro!: contour_integrable_holomorphic_simple)
       have abc: closed_segment a b ⊆ U  closed_segment b c ⊆ U  closed_segment
c a ⊆ U
          using that e segments_subset_convex_hull by fastforce+
        have eq0: ⋀w. w ∈ U ⟹ contour_integral (linepath a b +++ linepath b
c +++ linepath c a) (λz. d z w) = 0
        proof (rule contour_integral_unique [OF Cauchy_theorem_triangle])
         show ⋀w. w ∈ U ⟹ (λz. d z w) holomorphic_on convex hull {a, b, c}
              using e abc_subset by (auto intro: holomorphic_on_subset [OF
hol_dw])
        qed
        have contour_integral γ
              (λx. contour_integral (linepath a b) (λz. d z x) +
                   (contour_integral (linepath b c) (λz. d z x) +
                    contour_integral (linepath c a) (λz. d z x))) = 0
          apply (rule contour_integral_eq_0)
```

```
          using abc pasz U
          apply (subst contour_integral_join [symmetric], auto intro: eq0 *)+
          done
        then show ?thesis
          by (simp add: cint_h abc contour_integrable_add contour_integral_add
[symmetric] add_ac)
      qed
      show ?thesis
        using e ‹e > 0›
       by (auto intro!: holomorphic_on_imp_differentiable_at [OF _ open_ball]
analytic_imp_holomorphic
                    Morera_triangle continuous_on_subset [OF conthu] *)
    qed
  qed
 qed
 ultimately have [simp]: h z = 0 for z
  by (meson Liouville_weak)
 have ((λw. 1 / (w − z)) has_contour_integral complex_of_real (2 * pi) * i *
winding_number γ z) γ
  by (rule has_contour_integral_winding_number [OF ‹valid_path γ› znot])
 then have ((λw. f z * (1 / (w − z))) has_contour_integral complex_of_real (2
* pi) * i * winding_number γ z * f z) γ
  by (metis mult.commute has_contour_integral_lmul)
 then have 1: ((λw. f z / (w − z)) has_contour_integral complex_of_real (2 *
pi) * i * winding_number γ z * f z) γ
  by (simp add: field_split_simps)
 moreover have 2: ((λw. (f w − f z) / (w − z)) has_contour_integral 0) γ
   using U [OF z] pasz d_def by (force elim: has_contour_integral_eq [where
g = λw. (f w − f z)/(w − z)])
 show ?thesis
  using has_contour_integral_add [OF 1 2]  by (simp add: diff_divide_distrib)
qed

theorem Cauchy_integral_formula_global:
   assumes S: open S and holf: f holomorphic_on S
      and z: z ∈ S and vpg: valid_path γ
      and pasz: path_image γ ⊆ S − {z} and loop: pathfinish γ = pathstart γ
      and zero: ⋀w. w ∉ S ⟹ winding_number γ w = 0
    shows ((λw. f w / (w − z)) has_contour_integral (2*pi * i * winding_number
γ z * f z)) γ
proof −
 have path γ using vpg by (blast intro: valid_path_imp_path)
 have hols: (λw. f w / (w − z)) holomorphic_on S − {z} (λw. 1 / (w − z))
holomorphic_on S − {z}
   by (rule holomorphic_intros holomorphic_on_subset [OF holf] | force)+
 then have cint_fw: (λw. f w / (w − z)) contour_integrable_on γ
  by (meson contour_integrable_holomorphic_simple holomorphic_on_imp_continuous_on
open_delete S vpg pasz)
 obtain d where d>0
```

```
    and d: ⋀g h. ⟦valid_path g; valid_path h; ∀t∈{0..1}. cmod (g t − γ t) < d
∧ cmod (h t − γ t) < d;
                pathstart h = pathstart g ∧ pathfinish h = pathfinish g⟧
                ⟹ path_image h ⊆ S − {z} ∧ (∀f. f holomorphic_on S − {z}
⟶ contour_integral h f = contour_integral g f)
   using contour_integral_nearby_ends [OF _ ‹path γ› pasz] S by (simp add:
open_Diff) metis
 obtain p where polyp: polynomial_function p
          and ps: pathstart p = pathstart γ and pf: pathfinish p = pathfinish γ
and led: ∀t∈{0..1}. cmod (p t − γ t) < d
   using path_approx_polynomial_function [OF ‹path γ› ‹d > 0›] by metis
 then have ploop: pathfinish p = pathstart p using loop by auto
 have vpp: valid_path p  using polyp valid_path_polynomial_function by blast
 have [simp]: z ∉ path_image γ using pasz by blast
 have paps: path_image p ⊆ S − {z} and cint_eq: (⋀f. f holomorphic_on S −
{z} ⟹ contour_integral p f = contour_integral γ f)
   using pf ps led d [OF vpg vpp] ‹d > 0› by auto
 have wn_eq: winding_number p z = winding_number γ z
   using vpp paps
   by (simp add: subset_Diff_insert vpg valid_path_polynomial_function wind-
ing_number_valid_path cint_eq hols)
 have winding_number p w = winding_number γ w if w ∉ S for w
 proof −
   have hol: (λv. 1 / (v − w)) holomorphic_on S − {z}
     using that by (force intro: holomorphic_intros holomorphic_on_subset [OF
holf])
   have w ∉ path_image p w ∉ path_image γ using paps pasz that by auto
   then show ?thesis
   using vpp vpg by (simp add: subset_Diff_insert valid_path_polynomial_function
winding_number_valid_path cint_eq [OF hol])
 qed
 then have wn0: ⋀w. w ∉ S ⟹ winding_number p w = 0
   by (simp add: zero)
 show ?thesis
    using Cauchy_integral_formula_global_weak [OF S holf z polyp paps ploop
wn0] hols
   by (metis wn_eq cint_eq has_contour_integral_eqpath cint_fw cint_eq)
qed

theorem Cauchy_theorem_global:
   assumes S: open S and holf: f holomorphic_on S
       and vpg: valid_path γ and loop: pathfinish γ = pathstart γ
       and pas: path_image γ ⊆ S
       and zero: ⋀w. w ∉ S ⟹ winding_number γ w = 0
     shows (f has_contour_integral 0) γ
proof −
 obtain z where z ∈ S and znot: z ∉ path_image γ
 proof −
   have path_image γ ≠ S
```

by (*metis compact_valid_path_image vpg compact_open path_image_nonempty S*)
with *pas* **show** *?thesis* **by** (*blast intro*: *that*)
qed
then have *pasz*: *path_image* $\gamma \subseteq S - \{z\}$ **using** *pas* **by** *blast*
have *hol*: $(\lambda w.\ (w - z) * f\ w)$ *holomorphic_on S*
by (*rule holomorphic_intros holf*)+
show *?thesis*
using *Cauchy_integral_formula_global* [*OF S hol* ‹$z \in S$› *vpg pasz loop zero*]
by (*auto simp*: *znot elim*!: *has_contour_integral_eq*)
qed

corollary *Cauchy_theorem_global_outside*:
assumes *open S f holomorphic_on S valid_path* γ *pathfinish* γ = *pathstart* γ *path_image* $\gamma \subseteq S$
$\bigwedge w.\ w \notin S \Longrightarrow w \in$ *outside*(*path_image* γ)
shows (*f has_contour_integral 0*) γ
by (*metis Cauchy_theorem_global assms winding_number_zero_in_outside valid_path_imp_path*)

lemma *simply_connected_imp_winding_number_zero*:
assumes *simply_connected S path g*
path_image g $\subseteq S$ *pathfinish g* = *pathstart g* $z \notin S$
shows *winding_number g z* = *0*
proof −
have *hom*: *homotopic_loops S g* (*linepath* (*pathstart g*) (*pathstart g*))
by (*meson assms homotopic_paths_imp_homotopic_loops pathfinish_linepath simply_connected_eq_contractible_path*)
then have *homotopic_paths* $(- \{z\})$ *g* (*linepath* (*pathstart g*) (*pathstart g*))
by (*meson* ‹$z \notin S$› *homotopic_loops_imp_homotopic_paths_null homotopic_paths_subset subset_Compl_singleton*)
then have *winding_number g z* = *winding_number*(*linepath* (*pathstart g*) (*pathstart g*)) *z*
by (*rule winding_number_homotopic_paths*)
also have ... = *0*
using *assms* **by** (*force intro*: *winding_number_trivial*)
finally show *?thesis* **.**
qed

lemma *Cauchy_theorem_simply_connected*:
assumes *open S simply_connected S f holomorphic_on S valid_path g*
path_image g $\subseteq S$ *pathfinish g* = *pathstart g*
shows (*f has_contour_integral 0*) *g*
by (*meson assms Cauchy_theorem_global simply_connected_imp_winding_number_zero valid_path_imp_path*)

proposition *holomorphic_logarithm_exists*:
assumes *A*: *convex A open A*
and *f*: *f holomorphic_on A* $\bigwedge x.\ x \in A \Longrightarrow f\ x \neq 0$
and *z0*: *z0* $\in A$

```
    obtains g where g holomorphic_on A and ⋀x. x ∈ A ⟹ exp (g x) = f x
proof −
  note f' = holomorphic_derivI [OF f(1) A(2)]
  obtain g where g: ⋀x. x ∈ A ⟹ (g has_field_derivative deriv f x / f x) (at x)
  proof (rule holomorphic_convex_primitive' [OF A])
    show (λx. deriv f x / f x) holomorphic_on A
      by (intro holomorphic_intros f A)
  qed (auto simp: A at_within_open[of _ A])
  define h where h = (λx. −g z0 + ln (f z0) + g x)
  from g and A have g_holo: g holomorphic_on A
   by (auto simp: holomorphic_on_def at_within_open[of _ A] field_differentiable_def)
  hence h_holo: h holomorphic_on A
    by (auto simp: h_def intro!: holomorphic_intros)
    note [simp] = at_within_open[OF _ ‹open A›]
    have ∃ c. ∀ x∈A. f x / exp (h x) − 1 = c
      using ‹convex A› z0 f
    by (force simp: h_def exp_diff field_simps intro!: has_field_derivative_zero_constant
derivative_eq_intros g f')
  then obtain c where c: ⋀x. x ∈ A ⟹ f x / exp (h x) − 1 = c
    by blast
  from c[OF z0] and z0 and f have c = 0
    by (simp add: h_def)
  with c have ⋀x. x ∈ A ⟹ exp (h x) = f x by simp
  from that[OF h_holo this] show ?thesis .
qed
```

4.14 Cauchy's inequality and more versions of Liouville

```
lemma Cauchy_higher_deriv_bound:
    assumes holf: f holomorphic_on (ball z r)
        and contf: continuous_on (cball z r) f
        and fin : ⋀w. w ∈ ball z r ⟹ f w ∈ ball y B0
        and 0 < r and 0 < n
      shows norm ((deriv ^^ n) f z) ≤ (fact n) * B0 / r^n
proof −
  have 0 < B0 using ‹0 < r› fin [of z]
    by (metis ball_eq_empty ex_in_conv fin not_less)
  have le_B0: cmod (f w − y) ≤ B0 if cmod (w − z) ≤ r for w
  proof (rule continuous_on_closure_norm_le [of ball z r λw. f w − y], use ‹0
< r› in simp_all)
    show continuous_on (cball z r) (λw. f w − y)
      by (intro continuous_intros contf)
    show dist z w ≤ r
      by (simp add: dist_commute dist_norm that)
  qed (use fin in ‹auto simp: dist_norm less_eq_real_def norm_minus_commute›)
  have (deriv ^^ n) f z = (deriv ^^ n) (λw. f w) z − (deriv ^^ n) (λw. y) z
    using ‹0 < n› by simp
  also have ... = (deriv ^^ n) (λw. f w − y) z
    by (rule higher_deriv_diff [OF holf, symmetric]) (auto simp: ‹0 < r›)
```

```
  finally have (deriv ^^ n) f z = (deriv ^^ n) (λw. f w - y) z .
  have contf': continuous_on (cball z r) (λu. f u - y)
    by (rule contf continuous_intros)+
  have holf': (λu. (f u - y)) holomorphic_on (ball z r)
    by (simp add: holf holomorphic_on_diff)
  define a where a = (2 * pi)/(fact n)
  have 0 < a  by (simp add: a_def)
  have B0/r^(Suc n)*2 * pi * r = a*((fact n)*B0/r^n)
    using ‹0 < r› by (simp add: a_def field_split_simps)
  have der_dif: (deriv ^^ n) (λw. f w - y) z = (deriv ^^ n) f z
    using ‹0 < r› ‹0 < n›
    by (auto simp: higher_deriv_diff [OF holf holomorphic_on_const])
  have norm ((2 * of_real pi * i)/(fact n) * (deriv ^^ n) (λw. f w - y) z)
        ≤ (B0/r^(Suc n)) * (2 * pi * r)
    apply (rule has_contour_integral_bound_circlepath [of (λu. (f u - y)/(u -
z)^(Suc n)) _ z])
    using Cauchy_has_contour_integral_higher_derivative_circlepath [OF contf'
holf']
    using ‹0 < B0› ‹0 < r›
   apply (auto simp: norm_divide norm_mult norm_power divide_simps le_B0)
    done
  then show ?thesis
    using ‹0 < r›
     by (auto simp: norm_divide norm_mult norm_power field_simps der_dif
le_B0)
qed

lemma Cauchy_inequality:
    assumes holf: f holomorphic_on (ball ξ r)
       and contf: continuous_on (cball ξ r) f
       and 0 < r
       and nof: ⋀x. norm(ξ-x) = r ⟹ norm(f x) ≤ B
     shows norm ((deriv ^^ n) f ξ) ≤ (fact n) * B / r^n
proof -
  obtain x where norm (ξ-x) = r
    by (metis ‹0 < r› dist_norm order_less_imp_le vector_choose_dist)
  then have 0 ≤ B
    by (metis nof norm_not_less_zero not_le order_trans)
  have ξ ∈ ball ξ r
    using ‹0 < r› by simp
  then have ((λu. f u / (u-ξ) ^ Suc n) has_contour_integral (2 * pi) * i / fact
n * (deriv ^^ n) f ξ)
        (circlepath ξ r)
   by (rule Cauchy_has_contour_integral_higher_derivative_circlepath [OF contf
holf])
  have norm ((2 * pi * i)/(fact n) * (deriv ^^ n) f ξ) ≤ (B / r^(Suc n)) * (2 *
pi * r)
  proof (rule has_contour_integral_bound_circlepath)
    have ξ ∈ ball ξ r
```

```
      using ‹0 < r› by simp
    then show ((λu. f u / (u−ξ) ^ Suc n) has_contour_integral (2 * pi) * i /
fact n * (deriv ^^ n) f ξ)
        (circlepath ξ r)
      by (rule Cauchy_has_contour_integral_higher_derivative_circlepath [OF
contf holf])
    show ⋀x. cmod (x−ξ) = r ⟹ cmod (f x / (x−ξ) ^ Suc n) ≤ B / r ^ Suc n
      using ‹0 ≤ B› ‹0 < r›
      by (simp add: norm_divide norm_power nof frac_le norm_minus_commute
del: power_Suc)
  qed (use ‹0 ≤ B› ‹0 < r› in auto)
  then show ?thesis using ‹0 < r›
    by (simp add: norm_divide norm_mult field_simps)
qed

lemma Liouville_polynomial:
    assumes holf: f holomorphic_on UNIV
      and nof: ⋀z. A ≤ norm z ⟹ norm(f z) ≤ B * norm z ^ n
      shows f ξ = (∑ k≤n. (deriv^^k) f 0 / fact k * ξ ^ k)
proof (cases rule: le_less_linear [THEN disjE])
  assume B ≤ 0
  then have ⋀z. A ≤ norm z ⟹ norm(f z) = 0
    by (metis nof less_le_trans zero_less_mult_iff neqE norm_not_less_zero
norm_power not_le)
  then have f0: (f ⟶ 0) at_infinity
    using Lim_at_infinity by force
  then have [simp]: f = (λw. 0)
    using Liouville_weak [OF holf, of 0]
    by (simp add: eventually_at_infinity f0) meson
  show ?thesis by simp
next
  assume 0 < B
  have ((λk. (deriv ^^ k) f 0 / (fact k) * (ξ − 0)^k) sums f ξ)
  proof (rule holomorphic_power_series [where r = norm ξ + 1])
    show f holomorphic_on ball 0 (cmod ξ + 1) ξ ∈ ball 0 (cmod ξ + 1)
      using holf holomorphic_on_subset by auto
  qed
  then have sumsf: ((λk. (deriv ^^ k) f 0 / (fact k) * ξ^k) sums f ξ) by simp
  have (deriv ^^ k) f 0 / fact k * ξ ^ k = 0 if k>n for k
  proof (cases (deriv ^^ k) f 0 = 0)
    case True then show ?thesis by simp
  next
    case False
    define w where w = complex_of_real (fact k * B / cmod ((deriv ^^ k) f 0)
+ (|A| + 1))
    have 1 ≤ abs (fact k * B / cmod ((deriv ^^ k) f 0) + (|A| + 1))
      using ‹0 < B› by simp
    then have wge1: 1 ≤ norm w
      by (metis norm_of_real w_def)
```

```
    then have w ≠ 0 by auto
    have kB: 0 < fact k * B
      using ‹0 < B› by simp
    then have 0 ≤ fact k * B / cmod ((deriv ^^ k) f 0)
      by simp
    then have wgeA: A ≤ cmod w
      by (simp only: w_def norm_of_real)
    have fact k * B / cmod ((deriv ^^ k) f 0) < abs (fact k * B / cmod ((deriv ^^
k) f 0) + (|A| + 1))
      using ‹0 < B› by simp
    then have wge: fact k * B / cmod ((deriv ^^ k) f 0) < norm w
      by (metis norm_of_real w_def)
    then have fact k * B / norm w < cmod ((deriv ^^ k) f 0)
    using False by (simp add: field_split_simps mult.commute split: if_split_asm)
    also have ... ≤ fact k * (B * norm w ^ n) / norm w ^ k
    proof (rule Cauchy_inequality)
      show f holomorphic_on ball 0 (cmod w)
        using holf holomorphic_on_subset by force
      show continuous_on (cball 0 (cmod w)) f
        using holf holomorphic_on_imp_continuous_on holomorphic_on_subset
by blast
      show ⋀x. cmod (0 - x) = cmod w ⟹ cmod (f x) ≤ B * cmod w ^ n
        by (metis nof wgeA dist_0_norm dist_norm)
    qed (use ‹w ≠ 0› in auto)
    also have ... = fact k * B / cmod w ^(k-n)
      using ‹k>n› by (simp add: divide_simps flip: power_add)
    finally have fact k * B / cmod w < fact k * B / cmod w ^(k - n) .
    then have 1 / cmod w < 1 / cmod w ^(k - n)
      by (metis kB divide_inverse inverse_eq_divide mult_less_cancel_left_pos)
    then have cmod w ^(k - n) < cmod w
      by (smt (verit, best) ‹w ≠ 0› frac_le zero_less_norm_iff)
    with self_le_power [OF wge1] show ?thesis
      by (meson diff_is_0_eq not_gr0 not_le that)
  qed
  then have (deriv ^^ (k + Suc n)) f 0 / fact (k + Suc n) * ξ ^(k + Suc n) =
0 for k
    using not_less_eq by blast
  then have (λi. (deriv ^^ (i + Suc n)) f 0 / fact (i + Suc n) * ξ ^(i + Suc n))
sums 0
    by (rule sums_0)
  with sums_split_initial_segment [OF sumsf, where n = Suc n]
  show ?thesis
    using atLeast0AtMost lessThan_Suc_atMost sums_unique2 by fastforce
qed
```

Every bounded entire function is a constant function.

```
theorem Liouville_theorem:
  assumes holf: f holomorphic_on UNIV
    and bf: bounded (range f)
```

shows *f constant_on UNIV*
using *Liouville_polynomial* [*OF holf*, *of 0 _ 0*, *simplified*]
by (*metis bf bounded_iff constant_on_def rangeI*)

A holomorphic function f has only isolated zeros unless f is 0.

lemma *powser_0_nonzero*:
fixes *a* :: *nat* ⇒ *'a*::{*real_normed_field*,*banach*}
assumes *r*: *0* < *r*
and *sm*: ⋀*x*. *norm* (*x*−ξ) < *r* ⟹ (λ*n*. *a n* ∗ (*x*−ξ) ^ *n*) *sums* (*f x*)
and [*simp*]: *f* ξ = *0*
and *m0*: *a m* ≠ *0* **and** *m*>*0*
obtains *s* **where** *0* < *s* **and** ⋀*z*. *z* ∈ *cball* ξ *s* − {ξ} ⟹ *f z* ≠ *0*
proof −
have *r* ≤ *conv_radius a*
using *sm sums_summable* **by** (*auto simp*: *le_conv_radius_iff* [**where** ξ=ξ])
obtain *m* **where** *am*: *a m* ≠ *0* **and** *az* [*simp*]: (⋀*n*. *n*<*m* ⟹ *a n* = *0*)
proof
show *a* (*LEAST n*. *a n* ≠ *0*) ≠ *0*
by (*metis* (*mono_tags*, *lifting*) *m0 LeastI*)
qed (*fastforce dest*!: *not_less_Least*)
define *b* **where** *b i* = *a* (*i*+*m*) / *a m* **for** *i*
define *g* **where** *g x* = *suminf* (λ*i*. *b i* ∗ (*x*−ξ) ^ *i*) **for** *x*
have [*simp*]: *b 0* = *1*
by (*simp add*: *am b_def*)
{ fix *x*::*'a*
assume *norm* (*x*−ξ) < *r*
then have (λ*n*. (*a m* ∗ (*x*−ξ)^*m*) ∗ (*b n* ∗ (*x*−ξ)^*n*)) *sums* (*f x*)
using *am az sm sums_zero_iff_shift* [*of m* (λ*n*. *a n* ∗ (*x*−ξ) ^ *n*) *f x*]
by (*simp add*: *b_def monoid_mult_class.power_add algebra_simps*)
then have *x* ≠ ξ ⟹ (λ*n*. *b n* ∗ (*x*−ξ)^*n*) *sums* (*f x* / (*a m* ∗ (*x*−ξ)^*m*))
using *am* **by** (*simp add*: *sums_mult_D*)
} note *bsums* = *this*
then have *norm* (*x*−ξ) < *r* ⟹ *summable* (λ*n*. *b n* ∗ (*x*−ξ)^*n*) **for** *x*
using *sums_summable* **by** (*cases x*=ξ) *auto*
then have *r* ≤ *conv_radius b*
by (*simp add*: *le_conv_radius_iff* [**where** ξ=ξ])
then have *r*/*2* < *conv_radius b*
using *not_le order_trans r* **by** *fastforce*
then have *continuous_on* (*cball* ξ (*r*/*2*)) *g*
using *powser_continuous_suminf* [*of r*/*2 b* ξ] **by** (*simp add*: *g_def*)
then obtain *s* **where** *s*>*0* ⋀*x*. ⟦*norm* (*x*−ξ) ≤ *s*; *norm* (*x*−ξ) ≤ *r*/*2*⟧ ⟹ *dist* (*g x*) (*g* ξ) < *1*/*2*
proof (*rule continuous_onE*)
show ξ ∈ *cball* ξ (*r* / *2*) *1*/*2* > (*0*::*real*)
using *r* **by** *auto*
qed (*auto simp*: *dist_commute dist_norm*)
moreover have *g* ξ = *1*
by (*simp add*: *g_def*)
ultimately have *gnz*: ⋀*x*. ⟦*norm* (*x*−ξ) ≤ *s*; *norm* (*x*−ξ) ≤ *r*/*2*⟧ ⟹ (*g x*) ≠ *0*

```
    by fastforce
  have f x ≠ 0 if x ≠ ξ norm (x−ξ) ≤ s norm (x−ξ) ≤ r/2 for x
    using bsums [of x] that gnz [of x] r sums_iff unfolding g_def by fastforce
  then show ?thesis
    apply (rule_tac s=min s (r/2) in that)
    using ‹0 < r› ‹0 < s› by (auto simp: dist_commute dist_norm)
qed
```

4.15 Complex functions and power series

The following defines the power series expansion of a complex function at a given point (assuming that it is analytic at that point).

```
definition fps_expansion :: (complex ⇒ complex) ⇒ complex ⇒ complex fps
where
  fps_expansion f z0 = Abs_fps (λn. (deriv ^^ n) f z0 / fact n)

lemma fps_expansion_cong:
  assumes ∀F w in nhds x. f w =g w
  shows fps_expansion f x = fps_expansion g x
  unfolding fps_expansion_def using assms higher_deriv_cong_ev by fastforce

lemma
  fixes r :: ereal
  assumes f holomorphic_on eball z0 r
  shows   conv_radius_fps_expansion: fps_conv_radius (fps_expansion f z0) ≥ r
    and   eval_fps_expansion: ⋀z. z ∈ eball z0 r ⟹ eval_fps (fps_expansion f
z0) (z − z0) = f z
    and   eval_fps_expansion': ⋀z. norm z < r ⟹ eval_fps (fps_expansion f z0)
z = f (z0 + z)
proof −
  have (λn. fps_nth (fps_expansion f z0) n * (z − z0) ^ n) sums f z
    if z ∈ ball z0 r' ereal r' < r for z r'
  proof −
    have f holomorphic_on ball z0 r'
      using holomorphic_on_subset[OF _ ball_eball_mono] assms that by force
    then show ?thesis
      using fps_expansion_def holomorphic_power_series that by auto
  qed
  hence ∗: (λn. fps_nth (fps_expansion f z0) n * (z − z0) ^ n) sums f z
    if z ∈ eball z0 r for z
    using that by (subst (asm) eball_conv_UNION_balls) blast
 show fps_conv_radius (fps_expansion f z0) ≥ r unfolding fps_conv_radius_def
  proof (rule conv_radius_geI_ex)
    fix r' :: real assume r': r' > 0 ereal r' < r
    thus ∃z. norm z = r' ∧ summable (λn. fps_nth (fps_expansion f z0) n * z ^
n)
      using ∗[of z0 + of_real r']
      by (intro exI[of _ of_real r']) (auto simp: summable_def dist_norm)
  qed
```

```
  show eval_fps (fps_expansion f z0) (z − z0) = f z if z ∈ eball z0 r for z
    using ∗[OF that] by (simp add: eval_fps_def sums_iff)
  show eval_fps (fps_expansion f z0) z = f (z0 + z) if ereal (norm z) < r for z
    using ∗[of z0 + z] and that by (simp add: eval_fps_def sums_iff dist_norm)
qed
```

We can now show several more facts about power series expansions (at least in the complex case) with relative ease that would have been trickier without complex analysis.

```
lemma
  fixes f :: complex fps and r :: ereal
  assumes ⋀z. ereal (norm z) < r ⟹ eval_fps f z ≠ 0
 shows  fps_conv_radius_inverse: fps_conv_radius (inverse f) ≥ min r (fps_conv_radius
f)
    and    eval_fps_inverse: ⋀z. ereal (norm z) < fps_conv_radius f ⟹ ereal
(norm z) < r ⟹
                          eval_fps (inverse f) z = inverse (eval_fps f z)
proof −
  define R where R = min (fps_conv_radius f) r
  have ∗: fps_conv_radius (inverse f) ≥ min r (fps_conv_radius f) ∧
        (∀ z∈eball 0 (min (fps_conv_radius f) r). eval_fps (inverse f) z = inverse
(eval_fps f z))
  proof (cases min r (fps_conv_radius f) > 0)
    case True
    define f' where f' = fps_expansion (λz. inverse (eval_fps f z)) 0
   have holo: (λz. inverse (eval_fps f z)) holomorphic_on eball 0 (min r (fps_conv_radius
f))
      using assms by (intro holomorphic_intros) auto
    from holo have radius: fps_conv_radius f' ≥ min r (fps_conv_radius f)
      unfolding f'_def by (rule conv_radius_fps_expansion)
    have eval_f': eval_fps f' z = inverse (eval_fps f z)
      if norm z < fps_conv_radius f norm z < r for z
      using that unfolding f'_def by (subst eval_fps_expansion'[OF holo]) auto

    have f ∗ f' = 1
    proof (rule eval_fps_eqD)
     from radius and True have 0 < min (fps_conv_radius f) (fps_conv_radius
f')
        by (auto simp: min_def split: if_splits)
      also have ... ≤ fps_conv_radius (f ∗ f') by (rule fps_conv_radius_mult)
      finally show ... > 0 .
    next
      from True have R > 0 by (auto simp: R_def)
      hence eventually (λz. z ∈ eball 0 R) (nhds 0)
        by (intro eventually_nhds_in_open) (auto simp: zero_ereal_def)
      thus eventually (λz. eval_fps (f ∗ f') z = eval_fps 1 z) (nhds 0)
      proof eventually_elim
        case (elim z)
        hence eval_fps (f ∗ f') z = eval_fps f z ∗ eval_fps f' z
```

```
      using radius by (intro eval_fps_mult)
                      (auto simp: R_def min_def split: if_splits intro: less_trans)
    also have eval_fps f' z = inverse (eval_fps f z)
      using elim by (intro eval_f') (auto simp: R_def)
    also from elim have eval_fps f z ≠ 0
      by (intro assms) (auto simp: R_def)
    hence eval_fps f z * inverse (eval_fps f z) = eval_fps 1 z
      by simp
    finally show eval_fps (f * f') z = eval_fps 1 z .
  qed
 qed simp_all
 hence f' = inverse f
  by (intro fps_inverse_unique [symmetric]) (simp_all add: mult_ac)
 with eval_f' and radius show ?thesis by simp
next
 case False
 hence *: eball 0 R = {}
  by (intro eball_empty) (auto simp: R_def min_def split: if_splits)
 show ?thesis
 proof safe
  from False have min r (fps_conv_radius f) ≤ 0
   by (simp add: min_def)
  also have 0 ≤ fps_conv_radius (inverse f)
   by (simp add: fps_conv_radius_def conv_radius_nonneg)
  finally show min r (fps_conv_radius f) ≤ ... .
 qed (unfold * [unfolded R_def], auto)
qed

from * show fps_conv_radius (inverse f) ≥ min r (fps_conv_radius f) by blast
from * show eval_fps (inverse f) z = inverse (eval_fps f z)
 if ereal (norm z) < fps_conv_radius f ereal (norm z) < r for z
 using that by auto
qed

lemma
 fixes f g :: complex fps and r :: ereal
 defines R ≡ Min {r, fps_conv_radius f, fps_conv_radius g}
 assumes fps_conv_radius f > 0 fps_conv_radius g > 0 r > 0
 assumes nz: ⋀z. z ∈ eball 0 r ⟹ eval_fps g z ≠ 0
 shows   fps_conv_radius_divide': fps_conv_radius (f / g) ≥ R
  and    eval_fps_divide':
         ereal (norm z) < R ⟹ eval_fps (f / g) z = eval_fps f z / eval_fps g z
proof −
 from nz[of 0] and ‹r > 0› have nz': fps_nth g 0 ≠ 0
  by (auto simp: eval_fps_at_0 zero_ereal_def)
 have R ≤ min r (fps_conv_radius g)
  by (auto simp: R_def intro: min.coboundedI2)
 also have min r (fps_conv_radius g) ≤ fps_conv_radius (inverse g)
  by (intro fps_conv_radius_inverse assms) (auto simp: zero_ereal_def)
```

```
  finally have radius: fps_conv_radius (inverse g) ≥ R .
  have R ≤ min (fps_conv_radius f) (fps_conv_radius (inverse g))
    by (intro radius min.boundedI) (auto simp: R_def intro: min.coboundedI1
min.coboundedI2)
  also have ... ≤ fps_conv_radius (f * inverse g)
    by (rule fps_conv_radius_mult)
  also have f * inverse g = f / g
    by (intro fps_divide_unit [symmetric] nz')
  finally show fps_conv_radius (f / g) ≥ R .

  assume z: ereal (norm z) < R
  have eval_fps (f * inverse g) z = eval_fps f z * eval_fps (inverse g) z
    using radius by (intro eval_fps_mult less_le_trans[OF z])
                    (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  also have eval_fps (inverse g) z = inverse (eval_fps g z) using ‹r > 0›
    by (intro eval_fps_inverse[where r = r] less_le_trans[OF z] nz)
      (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  also have f * inverse g = f / g by fact
  finally show eval_fps (f / g) z = eval_fps f z / eval_fps g z
    by (simp add: field_split_simps)
qed

lemma
  fixes f g :: complex fps and r :: ereal
  defines R ≡ Min {r, fps_conv_radius f, fps_conv_radius g}
  assumes subdegree g ≤ subdegree f
  assumes fps_conv_radius f > 0 fps_conv_radius g > 0 r > 0
  assumes ⋀z. z ∈ eball 0 r ⟹ z ≠ 0 ⟹ eval_fps g z ≠ 0
  shows   fps_conv_radius_divide: fps_conv_radius (f / g) ≥ R
    and   eval_fps_divide:
        ereal (norm z) < R ⟹ c = fps_nth f (subdegree g) / fps_nth g (subdegree
g) ⟹
            eval_fps (f / g) z = (if z = 0 then c else eval_fps f z / eval_fps g z)
proof −
  define f' g' where f' = fps_shift (subdegree g) f and g' = fps_shift (subdegree
g) g
  have f_eq: f = f' * fps_X ^ subdegree g and g_eq: g = g' * fps_X ^ subdegree
g
    unfolding f'_def g'_def by (rule subdegree_decompose' le_refl | fact)+
  have subdegree: subdegree f' = subdegree f − subdegree g subdegree g' = 0
    using assms(2) by (simp_all add: f'_def g'_def)
   have [simp]: fps_conv_radius f' = fps_conv_radius f fps_conv_radius g' =
fps_conv_radius g
    by (simp_all add: f'_def g'_def)
  have [simp]: fps_nth f' 0 = fps_nth f (subdegree g)
               fps_nth g' 0 = fps_nth g (subdegree g) by (simp_all add: f'_def
g'_def)
  have g_nz: g ≠ 0
  proof −
```

```
    define z :: complex where z = (if r = ∞ then 1 else of_real (real_of_ereal r
/ 2))
    from ‹r > 0› have z ∈ eball 0 r
      by (cases r) (auto simp: z_def eball_def)
    moreover have z ≠ 0 using ‹r > 0›
      by (cases r) (auto simp: z_def)
    ultimately have eval_fps g z ≠ 0 by (rule assms(6))
    thus g ≠ 0 by auto
  qed
  have fg: f / g = f' * inverse g'
    by (subst f_eq, subst (2) g_eq) (insert g_nz, simp add: fps_divide_unit)

  have g'_nz: eval_fps g' z ≠ 0 if z: norm z < min r (fps_conv_radius g) for z
  proof (cases z = 0)
    case False
    with assms and z have eval_fps g z ≠ 0 by auto
    also from z have eval_fps g z = eval_fps g' z * z ^ subdegree g
      by (subst g_eq) (auto simp: eval_fps_mult)
    finally show ?thesis by auto
  qed (use ‹g ≠ 0› in ‹auto simp: g'_def eval_fps_at_0›)

  have R ≤ min (min r (fps_conv_radius g)) (fps_conv_radius g')
    by (auto simp: R_def min.coboundedI1 min.coboundedI2)
  also have ... ≤ fps_conv_radius (inverse g')
    using g'_nz by (rule fps_conv_radius_inverse)
  finally have conv_radius_inv: R ≤ fps_conv_radius (inverse g') .
  hence R ≤ fps_conv_radius (f' * inverse g')
    by (intro order.trans[OF _ fps_conv_radius_mult])
       (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  thus fps_conv_radius (f / g) ≥ R by (simp add: fg)

  fix z c :: complex assume z: ereal (norm z) < R
  assume c: c = fps_nth f (subdegree g) / fps_nth g (subdegree g)
  show eval_fps (f / g) z = (if z = 0 then c else eval_fps f z / eval_fps g z)
  proof (cases z = 0)
    case False
   from z and conv_radius_inv have ereal (norm z) < fps_conv_radius (inverse
g')
      by simp
    with z have eval_fps (f / g) z = eval_fps f' z * eval_fps (inverse g') z
      unfolding fg by (subst eval_fps_mult) (auto simp: R_def)
    also have eval_fps (inverse g') z = inverse (eval_fps g' z)
      using z by (intro eval_fps_inverse[of min r (fps_conv_radius g')] g'_nz)
(auto simp: R_def)
    also have eval_fps f' z * ... = eval_fps f z / eval_fps g z
      using z False assms(2) by (simp add: f'_def g'_def eval_fps_shift R_def)
    finally show ?thesis using False by simp
  qed (simp_all add: eval_fps_at_0 fg field_simps c)
qed
```

lemma *has_fps_expansion_fps_expansion* [*intro*]:
assumes *open A 0 ∈ A f holomorphic_on A*
shows *f has_fps_expansion fps_expansion f 0*
proof −
from *assms* **obtain** *r* **where** *r > 0* **and** *r*: *ball 0 r ⊆ A*
by (*auto simp*: *open_contains_ball*)
with *assms* **have** *holo*: *f holomorphic_on eball 0* (*ereal r*)
by *auto*
have *r ≤ fps_conv_radius* (*fps_expansion f 0*)
using *holo* **by** (*intro conv_radius_fps_expansion*) *auto*
then have *... > 0*
by (*simp add*: *ereal_le_less ‹r > 0› zero_ereal_def*)
moreover have *eventually* (*λz. z ∈ ball 0 r*) (*nhds 0*)
using *‹r > 0›* **by** (*intro eventually_nhds_in_open*) *auto*
hence *eventually* (*λz. eval_fps* (*fps_expansion f 0*) *z* = *f z*) (*nhds 0*)
by *eventually_elim* (*subst eval_fps_expansion'*[*OF holo*], *auto*)
ultimately show *?thesis* **using** *‹r > 0›* **by** (*auto simp*: *has_fps_expansion_def*)
qed

lemma *fps_conv_radius_tan*:
fixes *c* :: *complex*
assumes *c ≠ 0*
shows *fps_conv_radius* (*fps_tan c*) ≥ *pi* / (2 ∗ *norm c*)
proof −
have *fps_conv_radius* (*fps_tan c*) ≥
Min {*pi* / (2 ∗ *norm c*), *fps_conv_radius* (*fps_sin c*), *fps_conv_radius* (*fps_cos c*)}
unfolding *fps_tan_def*
proof (*rule fps_conv_radius_divide*)
fix *z* :: *complex* **assume** *z ∈ eball 0* (*pi* / (2 ∗ *norm c*))
with *cos_eq_zero_imp_norm_ge*[*of c∗z*] *assms*
show *eval_fps* (*fps_cos c*) *z ≠ 0* **by** (*auto simp*: *norm_mult field_simps*)
qed (*insert assms*, *auto*)
thus *?thesis* **by** (*simp add*: *min_def*)
qed

lemma *eval_fps_tan*:
fixes *c* :: *complex*
assumes *norm z < pi* / (2 ∗ *norm c*)
shows *eval_fps* (*fps_tan c*) *z* = *tan* (*c* ∗ *z*)
proof (*cases c = 0*)
case *False*
show *?thesis* **unfolding** *fps_tan_def*
proof (*subst eval_fps_divide'*[**where** *r* = *pi* / (2 ∗ *norm c*)])
fix *z* :: *complex* **assume** *z ∈ eball 0* (*pi* / (2 ∗ *norm c*))
with *cos_eq_zero_imp_norm_ge*[*of c∗z*] *assms*
show *eval_fps* (*fps_cos c*) *z ≠ 0* **using** *False* **by** (*auto simp*: *norm_mult field_simps*)

qed (*use False assms* **in** ‹*auto simp*: *field_simps tan_def*›)
qed *simp_all*

end

5 Conformal Mappings and Consequences of Cauchy's Integral Theorem

By John Harrison et al. Ported from HOL Light by L C Paulson (2016)

Also Cauchy's residue theorem by Wenda Li (2016)

theory *Conformal_Mappings*
imports *Cauchy_Integral_Formula*

begin

5.1 Analytic continuation

proposition *isolated_zeros*:
assumes *holf*: *f holomorphic_on S*
and *open S connected S ξ ∈ S f ξ = 0 β ∈ S f β ≠ 0*
obtains *r* **where** *0 < r* **and** *ball ξ r ⊆ S* **and**
⋀*z*. *z ∈ ball ξ r − {ξ} ⟹ f z ≠ 0*
proof −
obtain *r* **where** *0 < r* **and** *r*: *ball ξ r ⊆ S*
using ‹*open S*› ‹*ξ ∈ S*› *open_contains_ball_eq* **by** *blast*
have *powf*: ((λ*n*. (*deriv* ^^ *n*) *f ξ* / (*fact n*) ∗ (*z* − *ξ*) ^ *n*) *sums f z*) **if** *z ∈ ball ξ r* **for** *z*
by (*intro holomorphic_power_series* [*OF* _ *that*] *holomorphic_on_subset* [*OF holf r*])
obtain *m* **where** *m*: (*deriv* ^^ *m*) *f ξ* / (*fact m*) ≠ *0*
using *holomorphic_fun_eq_0_on_connected* [*OF holf* ‹*open S*› ‹*connected S*› _ ‹*ξ ∈ S*› ‹*β ∈ S*›] ‹*f β ≠ 0*›
by *auto*
then have *m ≠ 0* **using** *assms*(*5*) *funpow_0* **by** *fastforce*
obtain *s* **where** *0 < s* **and** *s*: ⋀*z*. *z ∈ cball ξ s − {ξ} ⟹ f z ≠ 0*
using *powser_0_nonzero* [*OF* ‹*0 < r*› *powf* ‹*f ξ = 0*› *m*]
by (*metis* ‹*m ≠ 0*› *dist_norm mem_ball norm_minus_commute not_gr_zero*)
have *0 < min r s* **by** (*simp add*: ‹*0 < r*› ‹*0 < s*›)
then show *thesis*
apply (*rule that*)
using *r s* **by** *auto*
qed

proposition *analytic_continuation*:
assumes *holf*: *f holomorphic_on S*
and *open S* **and** *connected S*
and *U ⊆ S* **and** *ξ ∈ S*

```
    and ξ islimpt U
    and fU0 [simp]: ⋀z. z ∈ U ⟹ f z = 0
    and w ∈ S
  shows f w = 0
proof −
  obtain e where 0 < e and e: cball ξ e ⊆ S
    using ‹open S› ‹ξ ∈ S› open_contains_cball_eq by blast
  define T where T = cball ξ e ∩ U
  have contf: continuous_on (closure T) f
   by (metis T_def closed_cball closure_minimal e holf holomorphic_on_imp_continuous_on
          holomorphic_on_subset inf.cobounded1)
  have fT0 [simp]: ⋀x. x ∈ T ⟹ f x = 0
    by (simp add: T_def)
  have ⋀r. ⟦∀e>0. ∃x'∈U. x' ≠ ξ ∧ dist x' ξ < e; 0 < r⟧ ⟹ ∃x'∈cball ξ e ∩
U. x' ≠ ξ ∧ dist x' ξ < r
   by (metis ‹0 < e› IntI dist_commute less_eq_real_def mem_cball min_less_iff_conj)
  then have ξ islimpt T using ‹ξ islimpt U›
    by (auto simp: T_def islimpt_approachable)
  then have ξ ∈ closure T
    by (simp add: closure_def)
  then have f ξ = 0
    by (auto simp: continuous_constant_on_closure [OF contf])
  moreover have ⋀r. ⟦0 < r; ⋀z. z ∈ ball ξ r − {ξ} ⟹ f z ≠ 0⟧ ⟹ False
     by (metis open_ball ‹ξ islimpt T› centre_in_ball fT0 insertE insert_Diff
islimptE)
  ultimately show ?thesis
    by (metis ‹open S› ‹connected S› ‹ξ ∈ S› ‹w ∈ S› holf isolated_zeros)
qed

corollary analytic_continuation_open:
  assumes open s and open s' and s ≠ {} and connected s'
      and s ⊆ s'
  assumes f holomorphic_on s' and g holomorphic_on s'
      and ⋀z. z ∈ s ⟹ f z = g z
  assumes z ∈ s'
  shows   f z = g z
proof −
  from ‹s ≠ {}› obtain ξ where ξ ∈ s by auto
  with ‹open s› have ξ: ξ islimpt s
    by (intro interior_limit_point) (auto simp: interior_open)
  have f z − g z = 0
    by (rule analytic_continuation[of λz. f z − g z s' s ξ])
       (insert assms ‹ξ ∈ s› ξ, auto intro: holomorphic_intros)
  thus ?thesis by simp
qed

corollary analytic_continuation':
  assumes f holomorphic_on S open S connected S
      and U ⊆ S ξ ∈ S ξ islimpt U
```

```
    and f constant_on U
  shows f constant_on S
proof −
  obtain c where c: ⋀x. x ∈ U ⟹ f x − c = 0
    by (metis ‹f constant_on U› constant_on_def diff_self)
  have (λz. f z − c) holomorphic_on S
    using assms by (intro holomorphic_intros)
  with c analytic_continuation assms have ⋀x. x ∈ S ⟹ f x − c = 0
    by blast
  then show ?thesis
    unfolding constant_on_def by force
qed

lemma holomorphic_compact_finite_zeros:
  assumes S: f holomorphic_on S open S connected S
      and compact K K ⊆ S
      and ¬ f constant_on S
    shows finite {z∈K. f z = 0}
proof (rule ccontr)
  assume infinite {z∈K. f z = 0}
  then obtain z where z ∈ K and z: z islimpt {z∈K. f z = 0}
    using ‹compact K› by (auto simp: compact_eq_Bolzano_Weierstrass)
  moreover have {z∈K. f z = 0} ⊆ S
    using ‹K ⊆ S› by blast
    ultimately show False
    using assms analytic_continuation [OF S] unfolding constant_on_def
    by blast
qed

lemma holomorphic_countable_zeros:
  assumes S: f holomorphic_on S open S connected S and fsigma S
      and ¬ f constant_on S
    shows countable {z∈S. f z = 0}
proof −
  obtain F::nat ⇒ complex set
      where F: range F ⊆ Collect compact and Seq: S = (⋃ i. F i)
    using ‹fsigma S› by (meson fsigma_Union_compact)
  have fin: finite {z ∈ F i. f z = 0} for i
    using holomorphic_compact_finite_zeros assms F Seq Union_iff by blast
  have {z ∈ S. f z = 0} = (⋃ i. {z ∈ F i. f z = 0})
    using Seq by auto
  with fin show ?thesis
    by (simp add: countable_finite)
qed

lemma holomorphic_countable_equal:
 assumes f holomorphic_on S g holomorphic_on S open S connected S and fsigma
S
    and eq: uncountable {z∈S. f z = g z}
```

```
  shows S ⊆ {z∈S. f z = g z}
proof −
  obtain z where z: z∈S f z = g z
    using eq not_finite_existsD uncountable_infinite by blast
  have (λx. f x − g x) holomorphic_on S
    by (simp add: assms holomorphic_on_diff)
  then have (λx. f x − g x) constant_on S
    using holomorphic_countable_zeros assms by force
  with z have ⋀x. x∈S ⟹ f x − g x = 0
    unfolding constant_on_def by force
  then show ?thesis
    by auto
qed

lemma holomorphic_countable_equal_UNIV:
  assumes fg: f holomorphic_on UNIV g holomorphic_on UNIV
    and eq: uncountable {z. f z = g z}
  shows f=g
  using holomorphic_countable_equal [OF fg] eq by fastforce
```

5.2 Open mapping theorem

```
lemma holomorphic_contract_to_zero:
  assumes contf: continuous_on (cball ξ r) f
      and holf: f holomorphic_on ball ξ r
      and 0 < r
      and norm_less: ⋀z. norm(ξ − z) = r ⟹ norm(f ξ) < norm(f z)
  obtains z where z ∈ ball ξ r f z = 0
proof −
  { assume fnz: ⋀w. w ∈ ball ξ r ⟹ f w ≠ 0
    then have 0 < norm (f ξ)
      by (simp add: ‹0 < r›)
    have fnz': ⋀w. w ∈ cball ξ r ⟹ f w ≠ 0
      by (metis norm_less dist_norm fnz less_eq_real_def mem_ball mem_cball
norm_not_less_zero norm_zero)
    have frontier(cball ξ r) ≠ {}
      using ‹0 < r› by simp
    define g where [abs_def]: g z = inverse (f z) for z
    have contg: continuous_on (cball ξ r) g
      unfolding g_def using contf continuous_on_inverse fnz' by blast
    have holg: g holomorphic_on ball ξ r
      unfolding g_def using fnz holf holomorphic_on_inverse by blast
    have frontier (cball ξ r) ⊆ cball ξ r
      by (simp add: subset_iff)
    then have contf': continuous_on (frontier (cball ξ r)) f
        and contg': continuous_on (frontier (cball ξ r)) g
      by (blast intro: contf contg continuous_on_subset)+
    have froc: frontier(cball ξ r) ≠ {}
      using ‹0 < r› by simp
```

```
    moreover have continuous_on (frontier (cball ξ r)) (norm o f)
      using contf' continuous_on_compose continuous_on_norm_id by blast
    ultimately obtain w where w: w ∈ frontier(cball ξ r)
                          and now: ⋀x. x ∈ frontier(cball ξ r) ⟹ norm (f w) ≤ norm
(f x)
      using continuous_attains_inf [OF compact_frontier [OF compact_cball]]
      by (metis comp_apply)
    then have fw: 0 < norm (f w)
      by (simp add: fnz')
    have continuous_on (frontier (cball ξ r)) (norm o g)
      using contg' continuous_on_compose continuous_on_norm_id by blast
    then obtain v where v: v ∈ frontier(cball ξ r)
                    and nov: ⋀x. x ∈ frontier(cball ξ r) ⟹ norm (g v) ≥ norm (g x)
      using continuous_attains_sup [OF compact_frontier [OF compact_cball]
froc] by force
    then have fv: 0 < norm (f v)
      by (simp add: fnz')
    have norm ((deriv ^^ 0) g ξ) ≤ fact 0 * norm (g v) / r ^ 0
      by (rule Cauchy_inequality [OF holg contg ‹0 < r›]) (simp add: dist_norm
nov)
    then have cmod (g ξ) ≤ cmod (g v)
      by simp
    moreover have cmod (ξ - w) = r
      by (metis (no_types) dist_norm frontier_cball mem_sphere w)
    ultimately obtain wr: norm (ξ - w) = r and nfw: norm (f w) ≤ norm (f ξ)
      unfolding g_def
      by (metis (no_types) ‹0 < cmod (f ξ)› less_imp_inverse_less norm_inverse
not_le now order_trans v)
    with fw have False
      using norm_less by force
  }
  with that show ?thesis by blast
qed

theorem open_mapping_thm:
  assumes holf: f holomorphic_on S
      and S: open S and connected S
      and open U and U ⊆ S
      and fne: ¬ f constant_on S
    shows open (f ‘ U)
proof -
  have ∗: open (f ‘ U)
          if U ≠ {} and U: open U connected U and f holomorphic_on U and
fneU: ⋀x. ∃ y ∈ U. f y ≠ x
          for U
  proof (clarsimp simp: open_contains_ball)
    fix ξ assume ξ: ξ ∈ U
    show ∃ e>0. ball (f ξ) e ⊆ f ‘ U
    proof -
```

```
    have hol: (λz. f z − f ξ) holomorphic_on U
      by (rule holomorphic_intros that)+
    obtain s where 0 < s and sbU: ball ξ s ⊆ U
              and sne: ⋀z. z ∈ ball ξ s − {ξ} ⟹ (λz. f z − f ξ) z ≠ 0
      using isolated_zeros [OF hol U ξ]  by (metis fneU right_minus_eq)
    obtain r where 0 < r and r: cball ξ r ⊆ ball ξ s
      using ‹0 < s› by (rule_tac r=s/2 in that) auto
    have cball ξ r ⊆ U
      using sbU r by blast
    then have frsbU: frontier (cball ξ r) ⊆ U
      using Diff_subset frontier_def order_trans by fastforce
    then have cof: compact (frontier(cball ξ r))
      by blast
    have frne: frontier (cball ξ r) ≠ {}
      using ‹0 < r› by auto
    have contfr: continuous_on (frontier (cball ξ r)) (λz. norm (f z − f ξ))
      by (metis continuous_on_norm continuous_on_subset frsbU hol holomor-
phic_on_imp_continuous_on)
    obtain w where norm (ξ − w) = r
              and w: (⋀z. norm (ξ − z) = r ⟹ norm (f w − f ξ) ≤ norm(f z −
f ξ))
    using continuous_attains_inf [OF cof frne contfr] by (auto simp: dist_norm)
    moreover define ε where ε ≡ norm (f w − f ξ) / 3
    ultimately have 0 < ε
      using ‹0 < r› dist_complex_def r sne by auto
    have ball (f ξ) ε ⊆ f ‘ U
    proof
      fix γ
      assume γ: γ ∈ ball (f ξ) ε
      have ∗: cmod (γ − f ξ) < cmod (γ − f z) if cmod (ξ − z) = r for z
      proof −
        have lt: cmod (f w − f ξ) / 3 < cmod (γ − f z)
          using w [OF that] γ
          using dist_triangle2 [of f ξ γ  f z] dist_triangle2 [of f ξ f z γ]
          by (simp add: ε_def dist_norm norm_minus_commute)
        show ?thesis
          by (metis ε_def dist_commute dist_norm less_trans lt mem_ball γ)
      qed
     have continuous_on (cball ξ r) (λz. γ − f z)
        using ‹cball ξ r ⊆ U› ‹f holomorphic_on U›
     by (force intro: continuous_intros continuous_on_subset holomorphic_on_imp_continuous_on)
      moreover have (λz. γ − f z) holomorphic_on ball ξ r
        using ‹cball ξ r ⊆ U› ball_subset_cball holomorphic_on_subset that(4)
        by (intro holomorphic_intros) blast
      ultimately obtain z where z ∈ ball ξ r γ − f z = 0
        using ∗ ‹0 < r› holomorphic_contract_to_zero by blast
      then show γ ∈ f ‘ U
        using ‹cball ξ r ⊆ U› by fastforce
    qed
```

```
    then show ?thesis using ‹0 < ε› by blast
   qed
  qed
  have open (f ‘ X) if X ∈ components U for X
  proof −
   have holfU: f holomorphic_on U
    using ‹U ⊆ S› holf holomorphic_on_subset by blast
   have X ≠ {}
    using that by (simp add: in_components_nonempty)
   moreover have open X
    using that ‹open U› open_components by auto
   moreover have connected X
    using that in_components_maximal by blast
   moreover have f holomorphic_on X
    by (meson that holfU holomorphic_on_subset in_components_maximal)
   moreover have ∃ y∈X. f y ≠ x for x
   proof (rule ccontr)
    assume not: ¬ (∃ y∈X. f y ≠ x)
    have X ⊆ S
     using ‹U ⊆ S› in_components_subset that by blast
    obtain w where w: w ∈ X using ‹X ≠ {}› by blast
    have wis: w islimpt X
     using w ‹open X› interior_eq by auto
    have hol: (λz. f z − x) holomorphic_on S
     by (simp add: holf holomorphic_on_diff)
    with fne [unfolded constant_on_def]
        analytic_continuation[OF hol S ‹connected S› ‹X ⊆ S› _ wis] not ‹X ⊆
S› w
    show False by auto
   qed
   ultimately show ?thesis
    by (rule ∗)
  qed
  then have open (f ‘ ⋃(components U))
   by (metis (no_types, lifting) imageE image_Union open_Union)
  then show ?thesis
   by force
qed
```

No need for S to be connected. But the nonconstant condition is stronger.

```
corollary open_mapping_thm2:
  assumes holf: f holomorphic_on S
     and S: open S
     and open U U ⊆ S
     and fnc: ⋀X. ⟦open X; X ⊆ S; X ≠ {}⟧ ⟹ ¬ f constant_on X
   shows open (f ‘ U)
proof −
  have S = ⋃(components S) by simp
  with ‹U ⊆ S› have U = (⋃ C ∈ components S. C ∩ U) by auto
```

```
  then have f ‘ U = (⋃C∈components S. f ‘ (C ∩ U))
    using image_UN by fastforce
  moreover
  { fix C assume C ∈ components S
    with S ‹C ∈ components S› open_components in_components_connected
    have C: open C connected C by auto
    have C ⊆ S
      by (metis ‹C ∈ components S› in_components_maximal)
    have nf: ¬ f constant_on C
      using ‹open C› ‹C ∈ components S› ‹C ⊆ S› fnc in_components_nonempty
by blast
    have f holomorphic_on C
      by (metis holf holomorphic_on_subset ‹C ⊆ S›)
    then have open (f ‘ (C ∩ U))
      by (meson C ‹open U› inf_le1 nf open_Int open_mapping_thm)
  } ultimately show ?thesis
    by force
qed

corollary open_mapping_thm3:
  assumes holf: f holomorphic_on S
      and open S and injf: inj_on f S
    shows  open (f ‘ S)
proof (rule open_mapping_thm2 [OF holf])
  show ⋀X. ⟦open X; X ⊆ S; X ≠ {}⟧ ⟹ ¬ f constant_on X
    using inj_on_subset injective_not_constant injf by blast
qed (use assms in auto)
```

5.3 Maximum modulus principle

If f is holomorphic, then its norm (modulus) cannot exhibit a true local maximum that is properly within the domain of f.

```
proposition maximum_modulus_principle:
  assumes holf: f holomorphic_on S
      and S: open S and connected S
      and open U and U ⊆ S and ξ ∈ U
      and no: ⋀z. z ∈ U ⟹ norm(f z) ≤ norm(f ξ)
    shows f constant_on S
proof (rule ccontr)
  assume ¬ f constant_on S
  then have open (f ‘ U)
    using open_mapping_thm assms by blast
  moreover have ¬ open (f ‘ U)
  proof −
    have ∃t. cmod (f ξ − t) < e ∧ t ∉ f ‘ U if 0 < e for e
      using that
      apply (rule_tac x=if 0 < Re(f ξ) then f ξ + (e/2) else f ξ − (e/2) in exI)
      apply (simp add: dist_norm)
      apply (fastforce simp: cmod_Re_le_iff dest!: no dest: sym)
```

```
    done
  then show ?thesis
    unfolding open_contains_ball by (metis ‹ξ ∈ U› contra_subsetD dist_norm
imageI mem_ball)
 qed
 ultimately show False
  by blast
qed

proposition maximum_modulus_frontier:
 assumes holf: f holomorphic_on (interior S)
     and contf: continuous_on (closure S) f
     and bos: bounded S
     and leB: ⋀z. z ∈ frontier S ⟹ norm(f z) ≤ B
     and ξ ∈ S
   shows norm(f ξ) ≤ B
proof −
 have compact (closure S) using bos
  by (simp add: bounded_closure compact_eq_bounded_closed)
 moreover have continuous_on (closure S) (cmod ∘ f)
  using contf continuous_on_compose continuous_on_norm_id by blast
 ultimately obtain z where z ∈ closure S and z: ⋀y. y ∈ closure S ⟹ (cmod
∘ f) y ≤ (cmod ∘ f) z
  using continuous_attains_sup [of closure S norm o f] ‹ξ ∈ S› by auto
 then consider z ∈ frontier S | z ∈ interior S using frontier_def by auto
 then have norm(f z) ≤ B
 proof cases
  case 1 then show ?thesis using leB by blast
 next
  case 2
  have f constant_on (connected_component_set (interior S) z)
  proof (rule maximum_modulus_principle)
   show f holomorphic_on connected_component_set (interior S) z
    by (metis connected_component_subset holf holomorphic_on_subset)
   show zin: z ∈ connected_component_set (interior S) z
    by (simp add: 2)
   show ⋀W. W ∈ connected_component_set (interior S) z ⟹ cmod (f W)
≤ cmod (f z)
    using closure_def connected_component_subset z by fastforce
  qed (auto simp: open_connected_component)
  then obtain c where c: ⋀w. w ∈ connected_component_set (interior S) z
⟹ f w = c
   by (auto simp: constant_on_def)
  have f ‘ closure(connected_component_set (interior S) z) ⊆ {c}
  proof (rule image_closure_subset)
   show continuous_on (closure (connected_component_set (interior S) z)) f
       by (meson closure_mono connected_component_subset contf continu-
ous_on_subset interior_subset)
  qed (use c in auto)
```

```
    then have cc: ⋀w. w ∈ closure(connected_component_set (interior S) z) ⟹
f w = c by blast
    have connected_component (interior S) z z
      by (simp add: 2)
    moreover have connected_component_set (interior S) z ≠ UNIV
    by (metis bos bounded_interior connected_component_eq_UNIV not_bounded_UNIV)
    ultimately have frontier(connected_component_set (interior S) z) ≠ {}
      by (meson 2 connected_component_eq_empty frontier_not_empty)
   then obtain w where w: w ∈ frontier(connected_component_set (interior S)
z)
      by auto
    then have norm (f z) = norm (f w)  by (simp add: 2 c cc frontier_def)
    also have ... ≤ B
      using w frontier_interior_subset frontier_of_connected_component_subset
      by (blast intro: leB)
    finally show ?thesis .
  qed
  then show ?thesis
    using z ‹ξ ∈ S› closure_subset by fastforce
qed

corollary maximum_real_frontier:
  assumes holf: f holomorphic_on (interior S)
      and contf: continuous_on (closure S) f
      and bos: bounded S
      and leB: ⋀z. z ∈ frontier S ⟹ Re(f z) ≤ B
      and ξ ∈ S
    shows Re(f ξ) ≤ B
using maximum_modulus_frontier [of exp o f S exp B]
    Transcendental.continuous_on_exp holomorphic_on_compose holomorphic_on_exp
assms
by auto
```

5.4 Factoring out a zero according to its order

```
lemma holomorphic_factor_order_of_zero:
  assumes holf: f holomorphic_on S
      and os: open S
      and ξ ∈ S 0 < n
      and dnz: (deriv ^^ n) f ξ ≠ 0
      and dfz: ⋀i. ⟦0 < i; i < n⟧ ⟹ (deriv ^^ i) f ξ = 0
   obtains g r where 0 < r
                     g holomorphic_on ball ξ r
                     ⋀w. w ∈ ball ξ r ⟹ f w − f ξ = (w − ξ)^n ∗ g w
                     ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof −
  obtain r where r>0 and r: ball ξ r ⊆ S using assms by (blast elim!: openE)
  then have holfb: f holomorphic_on ball ξ r
    using holf holomorphic_on_subset by blast
```

```
  define g where g w = suminf (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w
− ξ)^i) for w
  have sumsg: (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w − ξ)^i) sums g w
   and feq: f w − f ξ = (w − ξ)^n * g w
       if w: w ∈ ball ξ r for w
  proof −
    define powf where powf = (λi. (deriv ^^ i) f ξ/(fact i) * (w − ξ)^i)
    have [simp]: powf 0 = f ξ
      by (simp add: powf_def)
    have sing: {..<n} − {i. powf i = 0} = (if f ξ = 0 then {} else {0})
      unfolding powf_def using ‹0 < n› dfz by (auto simp: dfz; metis funpow_0
not_gr0)
    have powf sums f w
      unfolding powf_def by (rule holomorphic_power_series [OF holfb w])
    moreover have (∑ i<n. powf i) = f ξ
      by (subst sum.setdiff_irrelevant [symmetric]; simp add: dfz sing)
    ultimately have fsums: (λi. powf (i+n)) sums (f w − f ξ)
      using w sums_iff_shift' by metis
    then have ∗: summable (λi. (w − ξ) ^ n * ((deriv ^^ (i + n)) f ξ * (w − ξ)
^ i / fact (i + n)))
      unfolding powf_def using sums_summable
      by (auto simp: power_add mult_ac)
    have summable (λi. (deriv ^^ (i + n)) f ξ * (w − ξ) ^ i / fact (i + n))
    proof (cases w=ξ)
      case False then show ?thesis
        using summable_mult [OF ∗, of 1 / (w − ξ) ^ n] by simp
    next
      case True then show ?thesis
      by (auto simp: Power.semiring_1_class.power_0_left intro!: summable_finite
[of {0}]
                split: if_split_asm)
    qed
    then show sumsg: (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w − ξ)^i)
sums g w
      by (simp add: summable_sums_iff g_def)
    show f w − f ξ = (w − ξ)^n * g w
      using sums_mult [OF sumsg, of (w − ξ) ^ n]
    by (intro sums_unique2 [OF fsums]) (auto simp: power_add mult_ac powf_def)
  qed
  then have holg: g holomorphic_on ball ξ r
    by (meson sumsg power_series_holomorphic)
  then have contg: continuous_on (ball ξ r) g
    by (blast intro: holomorphic_on_imp_continuous_on)
  have g ξ ≠ 0
    using dnz unfolding g_def
    by (subst suminf_finite [of {0}]) auto
  obtain d where 0 < d and d: ⋀w. w ∈ ball ξ d ⟹ g w ≠ 0
    using ‹0 < r› continuous_on_avoid [OF contg _ ‹g ξ ≠ 0›]
    by (metis centre_in_ball le_cases mem_ball mem_ball_leI)
```

```
  show ?thesis
  proof
    show g holomorphic_on ball ξ (min r d)
      using holg by (auto simp: feq holomorphic_on_subset subset_ball d)
  qed (use ‹0 < r› ‹0 < d› in ‹auto simp: feq d›)
qed


lemma holomorphic_factor_order_of_zero_strong:
  assumes holf: f holomorphic_on S open S  ξ ∈ S 0 < n
      and (deriv ^^ n) f ξ ≠ 0
      and ⋀i. ⟦0 < i; i < n⟧ ⟹ (deriv ^^ i) f ξ = 0
   obtains g r where 0 < r
                     g holomorphic_on ball ξ r
                     ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = ((w - ξ) * g w) ^ n
                     ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof -
  obtain g r where 0 < r
                   and holg: g holomorphic_on ball ξ r
                   and feq: ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = (w - ξ)^n * g w
                   and gne: ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
    by (auto intro: holomorphic_factor_order_of_zero [OF assms])
  have con: continuous_on (ball ξ r) (λz. deriv g z / g z)
    by (rule continuous_intros) (auto simp: gne holg holomorphic_deriv holomorphic_on_imp_continuous_on)
  have cd: (λz. deriv g z / g z) field_differentiable at x if dist ξ x < r for x
  proof (intro derivative_intros)
    show deriv g field_differentiable at x
        using that holg mem_ball by (blast intro: holomorphic_deriv holomorphic_on_imp_differentiable_at)
    show g field_differentiable at x
     by (metis that open_ball at_within_open holg holomorphic_on_def mem_ball)
    qed (simp add: gne that)
    obtain h where h: ⋀x. x ∈ ball ξ r ⟹ (h has_field_derivative deriv g x / g x) (at x)
      using holomorphic_convex_primitive [of ball ξ r {} λz. deriv g z / g z]
       by (metis (no_types, lifting) Diff_empty at_within_interior cd con convex_ball infinite_imp_nonempty interior_ball mem_ball)
  then have continuous_on (ball ξ r) h
   by (metis open_ball holomorphic_on_imp_continuous_on holomorphic_on_open)
  then have con: continuous_on (ball ξ r) (λx. exp (h x) / g x)
   by (auto intro!: continuous_intros simp add: holg holomorphic_on_imp_continuous_on gne)
  have 0: dist ξ x < r ⟹ ((λx. exp (h x) / g x) has_field_derivative 0) (at x)
for x
   apply (rule h derivative_eq_intros DERIV_deriv_iff_field_differentiable [THEN iffD2] | simp)+
    using holg by (auto simp: holomorphic_on_imp_differentiable_at gne h)
  obtain c where c: ⋀x. x ∈ ball ξ r ⟹ exp (h x) / g x = c
```

```
    by (rule DERIV_zero_connected_constant [of ball ξ r {} λx. exp(h x) / g x])
(auto simp: con 0)
  have hol: (λz. exp ((Ln (inverse c) + h z) / of_nat n)) holomorphic_on ball ξ r
   proof (intro holomorphic_intros holomorphic_on_compose [unfolded o_def,
where g = exp])
    show h holomorphic_on ball ξ r
      using h holomorphic_on_open by blast
  qed (use ‹0 < n› in auto)
  show ?thesis
  proof
    show ⋀w. w ∈ ball ξ r ⟹ f w − f ξ = ((w − ξ) * exp ((Ln (inverse c) + h
w) / of_nat n)) ^ n
      using ‹0 < n›
       by (auto simp: feq power_mult_distrib exp_divide_power_eq exp_add gne
simp flip: c)
  qed (use hol ‹0 < r› in auto)
qed


lemma
  fixes k :: 'a::wellorder
  assumes a_def: a == LEAST x. P x and P: P k
  shows def_LeastI: P a and def_Least_le: a ≤ k
unfolding a_def
by (rule LeastI Least_le; rule P)+

lemma holomorphic_factor_zero_nonconstant:
  assumes holf: f holomorphic_on S and S: open S connected S
     and ξ ∈ S f ξ = 0
     and nonconst: ¬ f constant_on S
   obtains g r n
     where 0 < n  0 < r  ball ξ r ⊆ S
           g holomorphic_on ball ξ r
           ⋀w. w ∈ ball ξ r ⟹ f w = (w − ξ)^n * g w
           ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof (cases ∀ n>0. (deriv ^^ n) f ξ = 0)
  case True then show ?thesis
    using holomorphic_fun_eq_const_on_connected [OF holf S _ ‹ξ ∈ S›] non-
const by (simp add: constant_on_def)
next
  case False
  then obtain n0 where n0 > 0 and n0: (deriv ^^ n0) f ξ ≠ 0 by blast
  obtain r0 where r0 > 0 ball ξ r0 ⊆ S using S openE ‹ξ ∈ S› by auto
  define n where n ≡ LEAST n. (deriv ^^ n) f ξ ≠ 0
  have n_ne: (deriv ^^ n) f ξ ≠ 0
    by (rule def_LeastI [OF n_def]) (rule n0)
  then have 0 < n using ‹f ξ = 0›
    using funpow_0 by fastforce
  have n_min: ⋀k. k < n ⟹ (deriv ^^ k) f ξ = 0
```

```
    using def_Least_le [OF n_def] not_le by blast
  then obtain g r1
    where g: 0 < r1 g holomorphic_on ball ξ r1
          and geq: ⋀w. w ∈ ball ξ r1 ⟹ f w = (w - ξ) ^ n * g w
          and g0: ⋀w. w ∈ ball ξ r1 ⟹ g w ≠ 0
    by (auto intro: holomorphic_factor_order_of_zero [OF holf ‹open S› ‹ξ ∈ S›
‹n > 0› n_ne] simp: ‹f ξ = 0›)
  show ?thesis
  proof
    show g holomorphic_on ball ξ (min r0 r1)
      using g by auto
    show ⋀w. w ∈ ball ξ (min r0 r1) ⟹ f w = (w - ξ) ^ n * g w
      by (simp add: geq)
  qed (use ‹0 < n› ‹0 < r0› ‹0 < r1› ‹ball ξ r0 ⊆ S› g0 in auto)
qed


lemma holomorphic_lower_bound_difference:
  assumes holf: f holomorphic_on S and S: open S connected S
      and ξ ∈ S and φ ∈ S
      and fne: f φ ≠ f ξ
   obtains k n r
      where 0 < k  0 < r
            ball ξ r ⊆ S
            ⋀w. w ∈ ball ξ r ⟹ k * norm(w - ξ)^n ≤ norm(f w - f ξ)
proof -
  define n where n = (LEAST n. 0 < n ∧ (deriv ^^ n) f ξ ≠ 0)
  obtain n0 where 0 < n0 and n0: (deriv ^^ n0) f ξ ≠ 0
    using fne holomorphic_fun_eq_const_on_connected [OF holf S] ‹ξ ∈ S› ‹φ ∈
S› by blast
  then have 0 < n and n_ne: (deriv ^^ n) f ξ ≠ 0
    unfolding n_def by (metis (mono_tags, lifting) LeastI)+
  have n_min: ⋀k. ⟦0 < k; k < n⟧ ⟹ (deriv ^^ k) f ξ = 0
    unfolding n_def by (blast dest: not_less_Least)
  then obtain g r
    where 0 < r and holg: g holomorphic_on ball ξ r
      and fne: ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = (w - ξ) ^ n * g w
      and gnz: ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
      by (auto intro: holomorphic_factor_order_of_zero [OF holf ‹open S› ‹ξ ∈
S› ‹n > 0› n_ne])
  obtain e where e>0 and e: ball ξ e ⊆ S using assms by (blast elim!: openE)
  then have holfb: f holomorphic_on ball ξ e
    using holf holomorphic_on_subset by blast
  define d where d = (min e r) / 2
  have 0 < d using ‹0 < r› ‹0 < e› by (simp add: d_def)
  have d < r
    using ‹0 < r› by (auto simp: d_def)
  then have cbb: cball ξ d ⊆ ball ξ r
    by (auto simp: cball_subset_ball_iff)
```

```
  then have g holomorphic_on cball ξ d
    by (rule holomorphic_on_subset [OF holg])
  then have closed (g ‘ cball ξ d)
   by (simp add: compact_imp_closed compact_continuous_image holomorphic_on_imp_continuous_on)
  moreover have g ‘ cball ξ d ≠ {}
    using ‹0 < d› by auto
  ultimately obtain x where x: x ∈ g ‘ cball ξ d and ⋀y. y ∈ g ‘ cball ξ d ⟹
dist 0 x ≤ dist 0 y
    by (rule distance_attains_inf) blast
  then have leg: ⋀w. w ∈ cball ξ d ⟹ norm x ≤ norm (g w)
    by auto
  have ball ξ d ⊆ cball ξ d by auto
  also have ... ⊆ ball ξ e using ‹0 < d› d_def by auto
  also have ... ⊆ S by (rule e)
  finally have dS: ball ξ d ⊆ S .
  have x ≠ 0 using gnz x ‹d < r› by auto
  show thesis
  proof
    show ⋀w. w ∈ ball ξ d ⟹ cmod x * cmod (w - ξ) ^ n ≤ cmod (f w - f ξ)
     using ‹d < r› leg by (auto simp: fne norm_mult norm_power algebra_simps
mult_right_mono)
  qed (use dS ‹x ≠ 0› ‹d > 0› in auto)
qed

lemma
  assumes holf: f holomorphic_on (S - {ξ}) and ξ: ξ ∈ interior S
    shows holomorphic_on_extend_lim:
          (∃g. g holomorphic_on S ∧ (∀z ∈ S - {ξ}. g z = f z)) ⟷
           ((λz. (z - ξ) * f z) ⟶ 0) (at ξ)
          (is ?P = ?Q)
     and holomorphic_on_extend_bounded:
          (∃g. g holomorphic_on S ∧ (∀z ∈ S - {ξ}. g z = f z)) ⟷
           (∃B. eventually (λz. norm(f z) ≤ B) (at ξ))
          (is ?P = ?R)
proof -
  obtain δ where 0 < δ and δ: ball ξ δ ⊆ S
    using ξ mem_interior by blast
  have ?R if holg: g holomorphic_on S and gf: ⋀z. z ∈ S - {ξ} ⟹ g z = f z
for g
  proof -
    have §: cmod (f x) ≤ cmod (g ξ) + 1 if x ≠ ξ dist x ξ < δ dist (g x) (g ξ) <
1 for x
    proof -
      have x ∈ S
        by (metis δ dist_commute mem_ball subsetD that(2))
      with that gf [of x] show ?thesis
        using norm_triangle_ineq2 [of f x g ξ] dist_complex_def by auto
    qed
    then have ∗: ∀F z in at ξ. dist (g z) (g ξ) < 1 ⟶ cmod (f z) ≤ cmod (g ξ)
```

```
+ 1
      using ‹0 < δ› eventually_at by blast
    have continuous_on (interior S) g
     by (meson continuous_on_subset holg holomorphic_on_imp_continuous_on
interior_subset)
    then have ⋀x. x ∈ interior S ⟹ (g ⟶ g x) (at x)
    using continuous_on_interior continuous_within holg holomorphic_on_imp_continuous_on
by blast
    then have (g ⟶ g ξ) (at ξ)
      by (simp add: ξ)
    then show ?thesis
      apply (rule_tac x=norm(g ξ) + 1 in exI)
      apply (rule eventually_mp [OF ∗ tendstoD [where e=1]], auto)
      done
  qed
  moreover have ?Q if ∀ F z in at ξ. cmod (f z) ≤ B for B
    by (rule lim_null_mult_right_bounded [OF _ that]) (simp add: LIM_zero)
  moreover have ?P if (λz. (z − ξ) ∗ f z) −ξ→ 0
  proof −
    define h where [abs_def]: h z = (z − ξ)^2 ∗ f z for z
    have h0: (h has_field_derivative 0) (at ξ)
      apply (simp add: h_def has_field_derivative_iff)
    apply (auto simp: field_split_simps power2_eq_square Lim_transform_within
[OF that, of 1])
      done
    have holh: h holomorphic_on S
    proof (simp add: holomorphic_on_def, clarify)
      fix z assume z ∈ S
      show h field_differentiable at z within S
      proof (cases z = ξ)
        case True then show ?thesis
          using field_differentiable_at_within field_differentiable_def h0 by blast
      next
        case False
        then have f field_differentiable at z within S
          using holomorphic_onD [OF holf, of z] ‹z ∈ S›
          unfolding field_differentiable_def has_field_derivative_iff
          by (force intro: exI [where x=dist ξ z] elim: Lim_transform_within_set
[unfolded eventually_at])
        then show ?thesis
          by (simp add: h_def power2_eq_square derivative_intros)
      qed
    qed
    define g where [abs_def]: g z = (if z = ξ then deriv h ξ else (h z − h ξ) / (z
− ξ)) for z
    have holg: g holomorphic_on S
      unfolding g_def by (rule pole_lemma [OF holh ξ])
    have §: ∀ z∈S − {ξ}. (g z − g ξ) / (z − ξ) = f z
    using h0 by (auto simp: g_def power2_eq_square divide_simps DERIV_imp_deriv
```

```
h_def)
    show ?thesis
      apply (intro exI conjI)
       apply (rule pole_lemma [OF holg ξ])
      apply (simp add: §)
      done
  qed
  ultimately show ?P = ?Q and ?P = ?R
    by meson+
qed

lemma pole_at_infinity:
 assumes holf: f holomorphic_on UNIV and lim: ((inverse o f) ⟶ l) at_infinity
  obtains a n where ⋀z. f z = (∑ i≤n. a i * z^i)
proof (cases l = 0)
  case False
  show thesis
  proof
    show f z = (∑ i≤0. inverse l * z ^ i) for z
       using False tendsto_inverse [OF lim] by (simp add: Liouville_weak [OF
holf])
  qed
next
  case True
  then have [simp]: l = 0 .
  show ?thesis
  proof (cases ∃ r. 0 < r ∧ (∀ z ∈ ball 0 r - {0}. f(inverse z) ≠ 0))
    case True
      then obtain r where 0 < r and r: ⋀z. z ∈ ball 0 r - {0} ⟹ f(inverse
z) ≠ 0
            by auto
      have 1: inverse ∘ f ∘ inverse holomorphic_on ball 0 r - {0}
     by (rule holomorphic_on_compose holomorphic_intros holomorphic_on_subset
[OF holf] | force simp: r)+
      have 2: 0 ∈ interior (ball 0 r)
        using ‹0 < r› by simp
      have ∃ B. 0<B ∧ eventually (λz. cmod ((inverse ∘ f ∘ inverse) z) ≤ B) (at
0)
        apply (rule exI [where x=1])
        using tendstoD [OF lim [unfolded lim_at_infinity_0] zero_less_one]
        by (simp add: eventually_mono)
      with holomorphic_on_extend_bounded [OF 1 2]
      obtain g where holg: g holomorphic_on ball 0 r
               and geq: ⋀z. z ∈ ball 0 r - {0} ⟹ g z = (inverse ∘ f ∘ inverse) z
        by meson
      have ifi0: (inverse ∘ f ∘ inverse) -0→ 0
        using ‹l = 0› lim lim_at_infinity_0 by blast
      have g2g0: g -0→ g 0
     using ‹0 < r› centre_in_ball continuous_at continuous_on_eq_continuous_at
```

```
holg
      by (blast intro: holomorphic_on_imp_continuous_on)
    have g2g1: g -0→ 0
      apply (rule Lim_transform_within_open [OF ifi0 open_ball [of 0 r]])
      using ‹0 < r› by (auto simp: geq)
    have [simp]: g 0 = 0
      by (rule tendsto_unique [OF _ g2g0 g2g1]) simp
    have ball 0 r - {0::complex} ≠ {}
    using ‹0 < r› by (metis 2 Diff_iff insert_Diff interior_ball interior_singleton)
    then obtain w::complex where w ≠ 0 and w: norm w < r by force
    then have g w ≠ 0 by (simp add: geq r)
    obtain B n e where 0 < B 0 < e e ≤ r
                  and leg: ⋀w. norm w < e ⟹ B * cmod w ^ n ≤ cmod (g w)
     proof (rule holomorphic_lower_bound_difference [OF holg open_ball con-
nected_ball])
      show g w ≠ g 0
        by (simp add: ‹g w ≠ 0›)
      show w ∈ ball 0 r
        using mem_ball_0 w by blast
    qed (use ‹0 < r› in ‹auto simp: ball_subset_ball_iff›)
    have §: cmod (f z) ≤ cmod z ^ n / B if 2/e ≤ cmod z for z
    proof -
      have ize: inverse z ∈ ball 0 e - {0} using that ‹0 < e›
        by (auto simp: norm_divide field_split_simps algebra_simps)
      then have [simp]: z ≠ 0 and izr: inverse z ∈ ball 0 r - {0} using ‹e ≤
r›
        by auto
      then have [simp]: f z ≠ 0
        using r [of inverse z] by simp
      have [simp]: f z = inverse (g (inverse z))
        using izr geq [of inverse z] by simp
      show ?thesis using ize leg [of inverse z] ‹0 < B› ‹0 < e›
        by (simp add: field_split_simps norm_divide algebra_simps)
    qed
    show thesis
    proof
      show f z = (∑ i≤n. (deriv ^^ i) f 0 / fact i * z ^ i) for z
        using § by (rule_tac A = 2/e and B = 1/B in Liouville_polynomial
[OF holf], simp)
    qed
  next
    case False
    then have fi0: ⋀r. r > 0 ⟹ ∃ z∈ball 0 r - {0}. f (inverse z) = 0
      by simp
    have fz0: f z = 0 if 0 < r and lt1: ⋀x. x ≠ 0 ⟹ cmod x < r ⟹ inverse
(cmod (f (inverse x))) < 1
          for z r
    proof -
      have f0: (f ⟶ 0) at_infinity
```

```
    proof −
     have DIM_complex[intro]: 2 ≤ DIM(complex)  — should not be necessary!
       by simp
     have f (inverse x) ≠ 0 ⟹ x ≠ 0 ⟹ cmod x < r ⟹ 1 < cmod (f (inverse
x)) for x
       using lt1[of x] by (auto simp: field_simps)
      then have ∗∗: cmod (f (inverse x)) ≤ 1 ⟹ x ≠ 0 ⟹ cmod x < r ⟹ f
(inverse x) = 0 for x
       by force
      then have ∗: (f ∘ inverse) ‘ (ball 0 r − {0}) ⊆ {0} ∪ − ball 0 1
       by force
      have continuous_on (inverse ‘ (ball 0 r − {0})) f
        using continuous_on_subset holf holomorphic_on_imp_continuous_on
by blast
      then have connected ((f ∘ inverse) ‘ (ball 0 r − {0}))
       using connected_punctured_ball
       by (intro connected_continuous_image continuous_intros; force)
     then have {0} ∩ (f ∘ inverse) ‘ (ball 0 r − {0}) = {} ∨ − ball 0 1 ∩ (f ∘
inverse) ‘ (ball 0 r − {0}) = {}
       by (rule connected_closedD) (use ∗ in auto)
      then have f (inverse w) = 0 if w ≠ 0 cmod w < r for w
       using ∗∗[of w] fi0 ‹0 < r› that by force
      then show ?thesis
       unfolding lim_at_infinity_0
      using eventually_at ‹r > 0› by (force simp add: intro: tendsto_eventually)
    qed
    obtain w where w ∈ ball 0 r − {0} and f (inverse w) = 0
      using False ‹0 < r› by blast
    then show ?thesis
      by (auto simp: f0 Liouville_weak [OF holf, of 0])
  qed
  show thesis
  proof
    show ⋀z. f z = (∑ i≤0. 0 ∗ z ^ i)
      using lim
      apply (simp add: lim_at_infinity_0 Lim_at dist_norm norm_inverse)
      using fz0 zero_less_one by blast
  qed
 qed
qed
```

5.5 Entire proper functions are precisely the non-trivial polynomials

```
lemma proper_map_polyfun:
  fixes c :: nat ⇒ 'a::{real_normed_div_algebra,heine_borel}
 assumes closed S and compact K and c: c i ≠ 0 1 ≤ i i ≤ n
  shows compact (S ∩ {z. (∑ i≤n. c i ∗ z^i) ∈ K})
proof −
```

```
  obtain B where B > 0 and B: ⋀x. x ∈ K ⟹ norm x ≤ B
    by (metis compact_imp_bounded ‹compact K› bounded_pos)
  have *: norm x ≤ b
          if ⋀x. b ≤ norm x ⟹ B + 1 ≤ norm (∑ i≤n. c i * x ^ i)
            (∑ i≤n. c i * x ^ i) ∈ K  for b x
  proof −
    have norm (∑ i≤n. c i * x ^ i) ≤ B
      using B that by blast
    moreover have ¬ B + 1 ≤ B
      by simp
    ultimately show norm x ≤ b
      using that by (metis (no_types) less_eq_real_def not_less order_trans)
  qed
  have bounded {z. (∑ i≤n. c i * z ^ i) ∈ K}
    using Limits.polyfun_extremal [where c=c and B=B+1, OF c]
    by (auto simp: bounded_pos eventually_at_infinity_pos *)
  moreover have closed ((λz. (∑ i≤n. c i * z ^ i)) −‘ K)
    using ‹compact K› compact_eq_bounded_closed
    by (intro allI continuous_closed_vimage continuous_intros; force)
  ultimately show ?thesis
    using closed_Int_compact [OF ‹closed S›] compact_eq_bounded_closed
    by (auto simp add: vimage_def)
qed

lemma proper_map_polyfun_univ:
    fixes c :: nat ⇒ 'a::{real_normed_div_algebra,heine_borel}
  assumes compact K c i ≠ 0 1 ≤ i i ≤ n
    shows compact ({z. (∑ i≤n. c i * z^i) ∈ K})
  using proper_map_polyfun [of UNIV K c i n] assms by simp

lemma proper_map_polyfun_eq:
  assumes f holomorphic_on UNIV
    shows (∀ k. compact k ⟶ compact {z. f z ∈ k}) ⟷
          (∃ c n. 0 < n ∧ (c n ≠ 0) ∧ f = (λz. ∑ i≤n. c i * z^i))
          (is ?lhs = ?rhs)
proof
  assume compf [rule_format]: ?lhs
  have 2: ∃ k. 0 < k ∧ a k ≠ 0 ∧ f = (λz. ∑ i ≤ k. a i * z ^ i)
        if ⋀z. f z = (∑ i≤n. a i * z ^ i) for a n
  proof (cases ∀ i≤n. 0<i ⟶ a i = 0)
    case True
    then have [simp]: ⋀z. f z = a 0
      by (simp add: that sum.atMost_shift)
    have False using compf [of {a 0}] by simp
    then show ?thesis ..
  next
    case False
    then obtain k where k: 0 < k k≤n a k ≠ 0 by force
    define m where m = (GREATEST k. k≤n ∧ a k ≠ 0)
```

```
    have m: m≤n ∧ a m ≠ 0
      unfolding m_def
      using GreatestI_nat [where b = n] k by (metis (mono_tags, lifting))
    have [simp]: a i = 0 if m < i i ≤ n for i
      using Greatest_le_nat [where b = n and P = λk. k≤n ∧ a k ≠ 0]
      using m_def not_le that by auto
    have k ≤ m
      unfolding m_def
      using Greatest_le_nat [where b = n] k by (metis (mono_tags, lifting))
    with k m show ?thesis
    by (rule_tac x=m in exI) (auto simp: that comm_monoid_add_class.sum.mono_neutral_right)
  qed
  have §: ((inverse ∘ f) ⟶ 0) at_infinity
  proof (rule Lim_at_infinityI)
    fix e::real assume 0 < e
    with compf [of cball 0 (inverse e)]
    show ∃B. ∀x. B ≤ cmod x ⟶ dist ((inverse ∘ f) x) 0 ≤ e
      apply simp
    apply (clarsimp simp add: compact_eq_bounded_closed bounded_pos norm_inverse)
    by (metis (no_types, opaque_lifting) inverse_inverse_eq le_less_trans less_eq_real_def
less_imp_inverse_less linordered_field_no_ub not_less)
  qed
  then obtain a n where ⋀z. f z = (∑ i≤n. a i * z^i)
    using assms pole_at_infinity by blast
  with § 2 show ?rhs by blast
next
  assume ?rhs
  then obtain c n where 0 < n c n ≠ 0 f = (λz. ∑ i≤n. c i * z ^ i) by blast
  then have compact {z. f z ∈ k} if compact k for k
    by (auto intro: proper_map_polyfun_univ [OF that])
  then show ?lhs by blast
qed
```

5.6 Relating invertibility and nonvanishing of derivative

```
lemma has_complex_derivative_locally_injective:
  assumes holf: f holomorphic_on S
      and S: ξ ∈ S open S
      and dnz: deriv f ξ ≠ 0
  obtains r where r > 0 ball ξ r ⊆ S inj_on f (ball ξ r)
proof −
  have ∗: ∃d>0. ∀x. dist ξ x < d ⟶ onorm (λv. deriv f x * v − deriv f ξ * v)
< e if e > 0 for e
  proof −
    have contdf: continuous_on S (deriv f)
      by (simp add: holf holomorphic_deriv holomorphic_on_imp_continuous_on
‹open S›)
    obtain δ where δ>0 and δ: ⋀x. ⟦x ∈ S; dist x ξ ≤ δ⟧ ⟹ cmod (deriv f x −
deriv f ξ) ≤ e/2
```

```
    using continuous_onE [OF contdf ‹ξ ∈ S›, of e/2] ‹0 < e›
    by (metis dist_complex_def half_gt_zero less_imp_le)
  have §: ⋀ζ. ⟦ζ ∈ S; dist ξ ζ < δ⟧ ⟹ cmod (deriv f ζ − deriv f ξ) * cmod
z ≤ e/2 * cmod z
    by (intro mult_right_mono [OF δ]) (auto simp: dist_commute)
  obtain ε where ε>0 ball ξ ε ⊆ S
    by (metis openE [OF ‹open S› ‹ξ ∈ S›])
  with ‹δ>0› have ∃δ>0. ∀x. dist ξ x < δ ⟶ onorm (λv. deriv f x * v − deriv
f ξ * v) ≤ e/2
    using §
    apply (rule_tac x=min δ ε in exI)
    apply (intro conjI allI impI Operator_Norm.onorm_le)
   apply (force simp: mult_right_mono norm_mult [symmetric] left_diff_distrib
δ)+
    done
  with ‹e>0› show ?thesis by force
 qed
 have inj ((*) (deriv f ξ))
  using dnz by simp
 then obtain g' where g': linear g' g' ∘ (*) (deriv f ξ) = id
  using linear_injective_left_inverse [of (*) (deriv f ξ)] by auto
 show ?thesis
  apply (rule has_derivative_locally_injective [OF S, where f=f and f' = λz
h. deriv f z * h and g' = g'])
  using g' *
  apply (simp_all add: linear_conv_bounded_linear that)
 using ‹open S› has_field_derivative_imp_has_derivative holf holomorphic_derivI
by blast
qed

lemma has_complex_derivative_locally_invertible:
 assumes holf: f holomorphic_on S
    and S: ξ ∈ S open S
    and dnz: deriv f ξ ≠ 0
 obtains r where r > 0 ball ξ r ⊆ S open (f ‘ (ball ξ r)) inj_on f (ball ξ r)
proof −
 obtain r where r > 0 ball ξ r ⊆ S inj_on f (ball ξ r)
  by (blast intro: that has_complex_derivative_locally_injective [OF assms])
 then have ξ: ξ ∈ ball ξ r by simp
 then have nc: ¬ f constant_on ball ξ r
  using ‹inj_on f (ball ξ r)› injective_not_constant by fastforce
 have holf': f holomorphic_on ball ξ r
  using ‹ball ξ r ⊆ S› holf holomorphic_on_subset by blast
 have open (f ‘ ball ξ r)
  by (simp add: ‹inj_on f (ball ξ r)› holf' open_mapping_thm3)
 then show ?thesis
  using ‹0 < r› ‹ball ξ r ⊆ S› ‹inj_on f (ball ξ r)› that by blast
qed
```

```
lemma holomorphic_injective_imp_regular:
  assumes holf: f holomorphic_on S
      and open S and injf: inj_on f S
      and ξ ∈ S
    shows deriv f ξ ≠ 0
proof −
  obtain r where r>0 and r: ball ξ r ⊆ S using assms by (blast elim!: openE)
  have holf': f holomorphic_on ball ξ r
    using ‹ball ξ r ⊆ S› holf holomorphic_on_subset by blast
  show ?thesis
  proof (cases ∀ n>0. (deriv ^^ n) f ξ = 0)
    case True
    have fcon: f w = f ξ if w ∈ ball ξ r for w
      by (meson open_ball True ‹0 < r› centre_in_ball connected_ball holf'
            holomorphic_fun_eq_const_on_connected that)
    have False
      using fcon [of ξ + r/2] ‹0 < r› r injf unfolding inj_on_def
     by (metis ‹ξ ∈ S› contra_subsetD dist_commute fcon mem_ball perfect_choose_dist)
    then show ?thesis ..
  next
    case False
    then obtain n0 where n0: n0 > 0 ∧ (deriv ^^ n0) f ξ ≠ 0 by blast
    define n where [abs_def]: n = (LEAST n. n > 0 ∧ (deriv ^^ n) f ξ ≠ 0)
    have n_ne: n > 0 (deriv ^^ n) f ξ ≠ 0
      using def_LeastI [OF n_def n0] by auto
    have n_min: ⋀k. 0 < k ⟹ k < n ⟹ (deriv ^^ k) f ξ = 0
      using def_Least_le [OF n_def] not_le by auto
    obtain g δ where 0 < δ
          and holg: g holomorphic_on ball ξ δ
          and fd: ⋀w. w ∈ ball ξ δ ⟹ f w − f ξ = ((w − ξ) * g w) ^ n
          and gnz: ⋀w. w ∈ ball ξ δ ⟹ g w ≠ 0
     by (blast intro: n_min holomorphic_factor_order_of_zero_strong [OF holf
‹open S› ‹ξ ∈ S› n_ne])
    show ?thesis
    proof (cases n=1)
      case True
      with n_ne show ?thesis by auto
    next
      case False
      have g holomorphic_on ball ξ (min r δ)
        using holg by (simp add: holomorphic_on_subset subset_ball)
      then have holgw: (λw. (w − ξ) * g w) holomorphic_on ball ξ (min r δ)
        by (intro holomorphic_intros)
      have gd: ⋀w. dist ξ w < δ ⟹ (g has_field_derivative deriv g w) (at w)
        using holg
        by (simp add: DERIV_deriv_iff_field_differentiable holomorphic_on_def
at_within_open_NO_MATCH)
      have *: ⋀w. w ∈ ball ξ (min r δ)
          ⟹ ((λw. (w − ξ) * g w) has_field_derivative ((w − ξ) * deriv g w + g
```

w))
(*at w*)
by (*rule gd derivative_eq_intros* | *simp*)+
have [*simp*]: *deriv* (λ*w*. (*w* − ξ) ∗ *g w*) ξ ≠ *0*
using ∗ [*of* ξ] ‹*0* < δ› ‹*0* < *r*› **by** (*simp add*: *DERIV_imp_deriv gnz*)
obtain *T* **where** ξ ∈ *T open T* **and** *Tsb*: *T* ⊆ *ball* ξ (*min r* δ) **and** *oimT*: *open* ((λ*w*. (*w* − ξ) ∗ *g w*) ‘ *T*)
using ‹*0* < *r*› ‹*0* < δ› *has_complex_derivative_locally_invertible* [*OF holgw*, *of* ξ]
by *force*
define *U* **where** *U* = (λ*w*. (*w* − ξ) ∗ *g w*) ‘ *T*
have *open U* **by** (*metis oimT U_def*)
moreover have *0* ∈ *U*
using ‹ξ ∈ *T*› **by** (*auto simp*: *U_def intro*: *image_eqI* [**where** *x* = ξ])
ultimately obtain ε **where** ε>*0* **and** ε: *cball 0* ε ⊆ *U*
using ‹*open U*› *open_contains_cball* **by** *blast*
then have ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*0*/*n*)) ∈ *cball 0* ε
ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*1*/*n*)) ∈ *cball 0* ε
by (*auto simp*: *norm_mult*)
with ε **have** ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*0*/*n*)) ∈ *U*
ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*1*/*n*)) ∈ *U* **by** *blast*+
then obtain *y0 y1* **where** *y0* ∈ *T* **and** *y0*: (*y0* − ξ) ∗ *g y0* = ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*0*/*n*))
and *y1* ∈ *T* **and** *y1*: (*y1* − ξ) ∗ *g y1* = ε ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ (*1*/*n*))
by (*auto simp*: *U_def*)
then have *y0* ∈ *ball* ξ δ *y1* ∈ *ball* ξ δ **using** *Tsb* **by** *auto*
then have *f y0* − *f* ξ = ((*y0* − ξ) ∗ *g y0*) ^ *n f y1* − *f* ξ = ((*y1* − ξ) ∗ *g y1*) ^ *n*
using *fd* **by** *blast*+
moreover have *y0* ≠ *y1*
using *y0 y1* ‹ε > *0*› *complex_root_unity_eq_1* [*of n 1*] ‹*n* > *0*› *False* **by** *auto*
moreover have *T* ⊆ *S*
by (*meson Tsb min.cobounded1 order_trans r subset_ball*)
ultimately have *False*
using *inj_onD* [*OF injf*, *of y0 y1*] ‹*y0* ∈ *T*› ‹*y1* ∈ *T*›
using *complex_root_unity* [*of n 1*]
apply (*simp add*: *y0 y1 power_mult_distrib*)
apply (*force simp*: *algebra_simps*)
done
then show *?thesis* **..**
qed
qed
qed

5.6.1 Hence a nice clean inverse function theorem

lemma *has_field_derivative_inverse_strong*:

fixes $f :: 'a::\{euclidean\_space,real\_normed\_field\} \Rightarrow 'a$
shows $[\![DERIV\ f\ x :> f'; f' \neq 0; open\ S; x \in S; continuous\_on\ S\ f;$
$\bigwedge z.\ z \in S \Longrightarrow g\ (f\ z) = z]\!]$
$\Longrightarrow DERIV\ g\ (f\ x) :> inverse\ (f')$
unfolding *has_field_derivative_def*
by (*rule has_derivative_inverse_strong* [*of S x f g*]) *auto*

lemma *has_field_derivative_inverse_strong_x*:
fixes $f :: 'a::\{euclidean\_space,real\_normed\_field\} \Rightarrow 'a$
shows $[\![DERIV\ f\ (g\ y) :> f'; f' \neq 0; open\ S; continuous\_on\ S\ f; g\ y \in S; f(g\ y) = y;$
$\bigwedge z.\ z \in S \Longrightarrow g\ (f\ z) = z]\!]$
$\Longrightarrow DERIV\ g\ y :> inverse\ (f')$
unfolding *has_field_derivative_def*
by (*rule has_derivative_inverse_strong_x* [*of S g y f*]) *auto*

proposition *holomorphic_has_inverse*:
assumes *holf*: *f holomorphic_on S*
and *open S* **and** *injf*: *inj_on f S*
obtains *g* **where** *g holomorphic_on* (*f ' S*)
$\bigwedge z.\ z \in S \Longrightarrow deriv\ f\ z * deriv\ g\ (f\ z) = 1$
$\bigwedge z.\ z \in S \Longrightarrow g(f\ z) = z$
proof −
have *ofs*: *open* (*f ' S*)
by (*rule open_mapping_thm3* [*OF assms*])
have *contf*: *continuous_on S f*
by (*simp add*: *holf holomorphic_on_imp_continuous_on*)
have ∗: (*the_inv_into S f has_field_derivative inverse* (*deriv f z*)) (*at* (*f z*)) **if** $z \in S$ **for** *z*
proof −
have *1*: (*f has_field_derivative deriv f z*) (*at z*)
using *DERIV_deriv_iff_field_differentiable* ‹*z* ∈ *S*› ‹*open S*› *holf holomorphic_on_imp_differentiable_at*
by *blast*
have *2*: *deriv f z* ≠ *0*
using ‹*z* ∈ *S*› ‹*open S*› *holf holomorphic_injective_imp_regular injf* **by** *blast*
show *?thesis*
proof (*rule has_field_derivative_inverse_strong* [*OF 1 2* ‹*open S*› ‹*z* ∈ *S*›])
show *continuous_on S f*
by (*simp add*: *holf holomorphic_on_imp_continuous_on*)
show $\bigwedge z.\ z \in S \Longrightarrow$ *the_inv_into S f* (*f z*) = *z*
by (*simp add*: *injf the_inv_into_f_f*)
qed
qed
show *?thesis*
proof
show *the_inv_into S f holomorphic_on f ' S*
by (*simp add*: *holomorphic_on_open ofs*) (*blast intro*: ∗)
next

```
    fix z assume z ∈ S
    have deriv f z ≠ 0
      using ‹z ∈ S› ‹open S› holf holomorphic_injective_imp_regular injf by
blast
    then show deriv f z * deriv (the_inv_into S f) (f z) = 1
      using * [OF ‹z ∈ S›] by (simp add: DERIV_imp_deriv)
  next
    fix z assume z ∈ S
    show the_inv_into S f (f z) = z
      by (simp add: ‹z ∈ S› injf the_inv_into_f_f)
 qed
qed
```

5.6.2 Holomorphism of covering maps and lifts.

```
lemma covering_space_lift_is_holomorphic:
 assumes cov: covering_space C p S
    and C: open C p holomorphic_on C
    and holf: f holomorphic_on U and fim: f ∈ U → S and gim: g ∈ U → C
    and contg: continuous_on U g and pg_f: ⋀x. x ∈ U ⟹ p(g x) = f x
  shows g holomorphic_on U
 unfolding holomorphic_on_def
proof (intro strip)
 fix z
 assume z ∈ U
 with fim have f z ∈ S by blast
 then obtain T 𝒱 where f z ∈ T and opeT: openin (top_of_set S) T
      and UV: ⋃𝒱 = C ∩ p -‘ T
      and ⋀W. W ∈ 𝒱 ⟹ openin (top_of_set C) W
      and disV: pairwise disjnt 𝒱 and homeV: ⋀W. W ∈ 𝒱 ⟹ ∃q. homeomor-
phism W T p q
   using cov fim unfolding covering_space_def by meson
 then have ⋀W. W ∈ 𝒱 ⟹ open W ∧ W ⊆ C
   by (metis ‹open C› inf_le1 open_Int openin_open)
 then obtain V where V ∈ 𝒱 g z ∈ V open V V ⊆ C
   by (metis IntI UnionE image_subset_iff vimageI UV ‹f z ∈ T› ‹z ∈ U› gim
pg_f image_subset_iff_funcset)
 have holp: p holomorphic_on V
   using ‹V ⊆ C› ‹p holomorphic_on C› holomorphic_on_subset by blast
 moreover have injp: inj_on p V
   by (metis ‹V ∈ 𝒱› homeV homeomorphism_def inj_on_inverseI)
 ultimately
 obtain p' where holp': p' holomorphic_on (p ‘ V) and pp': ⋀z. z ∈ V ⟹ p'(p
z) = z
   using ‹open V› holomorphic_has_inverse by metis
 have z ∈ U ∩ g -‘ V
   using ‹g z ∈ V› ‹z ∈ U› by blast
 moreover have openin (top_of_set U) (U ∩ g -‘ V)
   using continuous_openin_preimage [OF contg gim]
```

```
    by (meson ‹open V› contg continuous_openin_preimage_eq)
  ultimately obtain ε where ε>0 and e: ball z ε ∩ U ⊆ g −‘ V
    by (force simp add: openin_contains_ball)
  show g field_differentiable at z within U
  proof (rule field_differentiable_transform_within)
    show (0::real) < ε
      by (simp add: ‹0 < ε›)
    show z ∈ U
      by (simp add: ‹z ∈ U›)
    show (p' o f) x' = g x' if x' ∈ U dist x' z < ε for x'
      using that
       by (metis Int_iff comp_apply dist_commute e mem_ball pg_f pp' subsetD
vimage_eq)
    have open (p ‘ V)
      using ‹open V› holp injp open_mapping_thm3 by force
    moreover have f z ∈ p ‘ V
      by (metis ‹z ∈ U› image_iff pg_f ‹g z ∈ V›)
    ultimately have p' field_differentiable at (f z)
      using holomorphic_on_imp_differentiable_at holp' by blast
    moreover have f field_differentiable at z within U
      by (metis (no_types) ‹z ∈ U› holf holomorphic_on_def)
    ultimately show (p' o f) field_differentiable at z within U
     by (metis (no_types) field_differentiable_at_within field_differentiable_compose_within)
  qed
qed

lemma covering_space_lift_holomorphic:
  assumes cov: covering_space C p S
      and C: open C p holomorphic_on C
      and f: f holomorphic_on U f ∈ U → S
      and U: simply_connected U locally path_connected U
  obtains g where g holomorphic_on U g ∈ U → C ⋀y. y ∈ U ⟹ p(g y) = f y
proof −
  obtain g where continuous_on U g g ∈ U → C ⋀y. y ∈ U ⟹ p(g y) = f y
    using covering_space_lift [OF cov U] f holomorphic_on_imp_continuous_on
by blast
  then show ?thesis
   by (metis C cov covering_space_lift_is_holomorphic f image_subset_iff_funcset
that)
qed
```

5.7 The Schwarz Lemma

```
lemma Schwarz1:
  assumes holf: f holomorphic_on S
      and contf: continuous_on (closure S) f
      and S: open S connected S
      and boS: bounded S
      and S ≠ {}
```

```
  obtains w where w ∈ frontier S
      ⋀z. z ∈ closure S ⟹ norm (f z) ≤ norm (f w)
proof −
  have connf: continuous_on (closure S) (norm o f)
    using contf continuous_on_compose continuous_on_norm_id by blast
  have coc: compact (closure S)
    by (simp add: ‹bounded S› bounded_closure compact_eq_bounded_closed)
  then obtain x where x: x ∈ closure S and xmax: ⋀z. z ∈ closure S ⟹ norm(f
z) ≤ norm(f x)
    using continuous_attains_sup [OF _ _ connf] ‹S ≠ {}› by auto
  then show ?thesis
  proof (cases x ∈ frontier S)
    case True
    then show ?thesis using that xmax by blast
  next
    case False
    then have x ∈ S
      using ‹open S› frontier_def interior_eq x by auto
    then have f constant_on S
    proof (rule maximum_modulus_principle [OF holf S ‹open S› order_refl])
      show ⋀z. z ∈ S ⟹ cmod (f z) ≤ cmod (f x)
        using closure_subset by (blast intro: xmax)
    qed
    then have f constant_on (closure S)
      by (rule constant_on_closureI [OF _ contf])
    then obtain c where c: ⋀x. x ∈ closure S ⟹ f x = c
      by (meson constant_on_def)
    obtain w where w ∈ frontier S
      by (metis coc all_not_in_conv assms(6) closure_UNIV frontier_eq_empty
not_compact_UNIV)
    then show ?thesis
      by (simp add: c frontier_def that)
  qed
qed

lemma Schwarz2:
 ⟦f holomorphic_on ball 0 r;
    0 < s; ball w s ⊆ ball 0 r;
    ⋀z. norm (w−z) < s ⟹ norm(f z) ≤ norm(f w)⟧
    ⟹ f constant_on ball 0 r
by (rule maximum_modulus_principle [where U = ball w s and ξ = w]) (simp_all
add: dist_norm)

lemma Schwarz3:
  assumes holf: f holomorphic_on (ball 0 r) and [simp]: f 0 = 0
  obtains h where h holomorphic_on (ball 0 r) and ⋀z. norm z < r ⟹ f z =
z * (h z) and deriv f 0 = h 0
proof −
  define h where h z = (if z = 0 then deriv f 0 else f z / z) for z
```

```
  have d0: deriv f 0 = h 0
    by (simp add: h_def)
  moreover have h holomorphic_on (ball 0 r)
    by (rule pole_theorem_open_0 [OF holf, of 0]) (auto simp: h_def)
  moreover have norm z < r ⟹ f z = z * h z for z
    by (simp add: h_def)
  ultimately show ?thesis
    using that by blast
qed

proposition Schwarz_Lemma:
  assumes holf: f holomorphic_on (ball 0 1) and [simp]: f 0 = 0
      and no: ⋀z. norm z < 1 ⟹ norm (f z) < 1
      and ξ: norm ξ < 1
    shows norm (f ξ) ≤ norm ξ and norm(deriv f 0) ≤ 1
      and ((∃ z. norm z < 1 ∧ z ≠ 0 ∧ norm(f z) = norm z)
           ∨ norm(deriv f 0) = 1)
          ⟹ ∃ α. (∀ z. norm z < 1 ⟶ f z = α * z) ∧ norm α = 1
      (is ?P ⟹ ?Q)
proof −
  obtain h where holh: h holomorphic_on (ball 0 1)
            and fz_eq: ⋀z. norm z < 1 ⟹ f z = z * (h z) and df0: deriv f 0 = h
0
    by (rule Schwarz3 [OF holf]) auto
  have noh_le: norm (h z) ≤ 1 if z: norm z < 1 for z
  proof −
    have norm (h z) < a if a: 1 < a for a
    proof −
      have max (inverse a) (norm z) < 1
        using z a by (simp_all add: inverse_less_1_iff)
      then obtain r where r: max (inverse a) (norm z) < r and r < 1
        using Rats_dense_in_real by blast
      then have nzr: norm z < r and ira: inverse r < a
        using z a less_imp_inverse_less by force+
      then have 0 < r
        by (meson norm_not_less_zero not_le order.strict_trans2)
      have holh': h holomorphic_on ball 0 r
        by (meson holh ‹r < 1› holomorphic_on_subset less_eq_real_def sub-
set_ball)
      have conth': continuous_on (cball 0 r) h
      by (meson ‹r < 1› dual_order.trans holh holomorphic_on_imp_continuous_on
holomorphic_on_subset mem_ball_0 mem_cball_0 not_less subsetI)
      obtain w where w: norm w = r and lenw: ⋀z. norm z < r ⟹ norm(h z)
≤ norm(h w)
        apply (rule Schwarz1 [OF holh']) using conth' ‹0 < r› by auto
      have h w = f w / w using fz_eq ‹r < 1› nzr w by auto
      then have cmod (h z) < inverse r
        by (metis ‹0 < r› ‹r < 1› divide_strict_right_mono inverse_eq_divide
              le_less_trans lenw no norm_divide nzr w)
```

```
    then show ?thesis using ira by linarith
   qed
   then show norm (h z) ≤ 1
    using not_le by blast
  qed
  show cmod (f ξ) ≤ cmod ξ
  proof (cases ξ = 0)
   case True then show ?thesis by auto
  next
   case False
   then show ?thesis
    by (simp add: noh_le fz_eq ξ mult_left_le norm_mult)
  qed
  show no_df0: norm(deriv f 0) ≤ 1
   by (simp add: ‹⋀z. cmod z < 1 ⟹ cmod (h z) ≤ 1› df0)
  show ?Q if ?P
   using that
  proof
   assume ∃ z. cmod z < 1 ∧ z ≠ 0 ∧ cmod (f z) = cmod z
   then obtain γ where γ: cmod γ < 1 γ ≠ 0 cmod (f γ) = cmod γ by blast
   then have [simp]: norm (h γ) = 1
    by (simp add: fz_eq norm_mult)
   have §: ball γ (1 − cmod γ) ⊆ ball 0 1
    by (simp add: ball_subset_ball_iff)
   moreover have ⋀z. cmod (γ − z) < 1 − cmod γ ⟹ cmod (h z) ≤ cmod (h
γ)
     by (metis ‹cmod (h γ) = 1› § dist_0_norm dist_complex_def in_mono
mem_ball noh_le)
   ultimately obtain c where c: ⋀z. norm z < 1 ⟹ h z = c
    using Schwarz2 [OF holh, of 1 − norm γ γ, unfolded constant_on_def] γ
by auto
   then have norm c = 1
    using γ by force
   with c show ?thesis
    using fz_eq by auto
  next
   assume [simp]: cmod (deriv f 0) = 1
   then obtain c where c: ⋀z. norm z < 1 ⟹ h z = c
    using Schwarz2 [OF holh zero_less_one, of 0, unfolded constant_on_def]
df0 noh_le
    by auto
   moreover have norm c = 1 using df0 c by auto
   ultimately show ?thesis
    using fz_eq by auto
  qed
qed

corollary Schwarz_Lemma':
  assumes holf: f holomorphic_on (ball 0 1) and [simp]: f 0 = 0
```

```
    and no: ⋀z. norm z < 1 ⟹ norm (f z) < 1
  shows ((∀ξ. norm ξ < 1 ⟶ norm (f ξ) ≤ norm ξ)
         ∧ norm(deriv f 0) ≤ 1)
         ∧ (((∃z. norm z < 1 ∧ z ≠ 0 ∧ norm(f z) = norm z)
           ∨ norm(deriv f 0) = 1)
           ⟶ (∃α. (∀z. norm z < 1 ⟶ f z = α * z) ∧ norm α = 1))
  using Schwarz_Lemma [OF assms]
  by (metis (no_types) norm_eq_zero zero_less_one)
```

5.8 The Schwarz reflection principle

```
lemma hol_pal_lem0:
  assumes d · a ≤ k k ≤ d · b
  obtains c where
    c ∈ closed_segment a b d · c = k
    ⋀z. z ∈ closed_segment a c ⟹ d · z ≤ k
    ⋀z. z ∈ closed_segment c b ⟹ k ≤ d · z
proof −
  obtain c where cin: c ∈ closed_segment a b and keq: k = d · c
    using connected_ivt_hyperplane [of closed_segment a b a b d k]
    by (auto simp: assms)
  have closed_segment a c ⊆ {z. d · z ≤ k}  closed_segment c b ⊆ {z. k ≤ d · z}
    unfolding segment_convex_hull using assms keq
    by (auto simp: convex_halfspace_le convex_halfspace_ge hull_minimal)
  then show ?thesis using cin that by fastforce
qed

lemma hol_pal_lem1:
  assumes convex S open S
    and abc: a ∈ S b ∈ S c ∈ S
        d ≠ 0 and lek: d · a ≤ k d · b ≤ k d · c ≤ k
    and holf1: f holomorphic_on {z. z ∈ S ∧ d · z < k}
    and contf: continuous_on S f
  shows contour_integral (linepath a b) f +
        contour_integral (linepath b c) f +
        contour_integral (linepath c a) f = 0
proof −
  have interior (convex hull {a, b, c}) ⊆ interior(S ∩ {x. d · x ≤ k})
  proof (intro interior_mono hull_minimal)
    show {a, b, c} ⊆ S ∩ {x. d · x ≤ k}
      by (simp add: abc lek)
    show convex (S ∩ {x. d · x ≤ k})
      by (rule convex_Int [OF ‹convex S› convex_halfspace_le])
  qed
  also have ... ⊆ {z ∈ S. d · z < k}
    by (force simp: interior_open [OF ‹open S›] ‹d ≠ 0›)
  finally have ∗: interior (convex hull {a, b, c}) ⊆ {z ∈ S. d · z < k} .
  have continuous_on (convex hull {a,b,c}) f
    using ‹convex S› contf abc continuous_on_subset subset_hull
```

```
    by fastforce
  moreover have f holomorphic_on interior (convex hull {a,b,c})
    by (rule holomorphic_on_subset [OF holf1 *])
  ultimately show ?thesis
   using Cauchy_theorem_triangle_interior has_chain_integral_chain_integral3
     by blast
qed

lemma hol_pal_lem2:
  assumes S: convex S open S
     and abc: a ∈ S b ∈ S c ∈ S
     and d ≠ 0 and lek: d · a ≤ k d · b ≤ k
     and holf1: f holomorphic_on {z. z ∈ S ∧ d · z < k}
     and holf2: f holomorphic_on {z. z ∈ S ∧ k < d · z}
     and contf: continuous_on S f
   shows contour_integral (linepath a b) f +
         contour_integral (linepath b c) f +
         contour_integral (linepath c a) f = 0
proof (cases d · c ≤ k)
  case True show ?thesis
    by (rule hol_pal_lem1 [OF S abc ‹d ≠ 0› lek True holf1 contf])
next
  case False
  then have d · c > k by force
  obtain a' where a': a' ∈ closed_segment b c and d · a' = k
     and ba': ⋀z. z ∈ closed_segment b a' ⟹ d · z ≤ k
     and a'c: ⋀z. z ∈ closed_segment a' c ⟹ k ≤ d · z
    using False hol_pal_lem0 [of d b k c, OF ‹d · b ≤ k›] by auto
  obtain b' where b': b' ∈ closed_segment a c and d · b' = k
     and ab': ⋀z. z ∈ closed_segment a b' ⟹ d · z ≤ k
     and b'c: ⋀z. z ∈ closed_segment b' c ⟹ k ≤ d · z
    using False hol_pal_lem0 [of d a k c, OF ‹d · a ≤ k›] by auto
  have a'b': a' ∈ S ∧ b' ∈ S
    using a' abc b' convex_contains_segment ‹convex S› by auto
  have continuous_on (closed_segment c a) f
    by (meson abc contf continuous_on_subset convex_contains_segment ‹convex
S›)
  then have 1: contour_integral (linepath c a) f =
              contour_integral (linepath c b') f + contour_integral (linepath b' a) f
    using b' closed_segment_commute contour_integral_split_linepath by blast
  have continuous_on (closed_segment b c) f
    by (meson abc contf continuous_on_subset convex_contains_segment ‹convex
S›)
  then have 2: contour_integral (linepath b c) f =
              contour_integral (linepath b a') f + contour_integral (linepath a' c) f
    by (rule contour_integral_split_linepath [OF _ a'])
  have 3: contour_integral (reversepath (linepath b' a')) f =
               − contour_integral (linepath b' a') f
    by (rule contour_integral_reversepath [OF valid_path_linepath])
```

```
  have fcd_le: f field_differentiable at x
            if x ∈ interior S ∧ x ∈ interior {x. d • x ≤ k} for x
  proof −
    have f holomorphic_on S ∩ {c. d • c < k}
      by (metis (no_types) Collect_conj_eq Collect_mem_eq holf1)
    then have ∃ C D. x ∈ interior C ∩ interior D ∧ f holomorphic_on interior C
∩ interior D
      using that
      by (metis Collect_mem_eq Int_Collect ‹d ≠ 0› interior_halfspace_le inte-
rior_open ‹open S›)
    then show f field_differentiable at x
     by (metis at_within_interior holomorphic_on_def interior_Int interior_interior)
  qed
  have ab_le: ⋀x. x ∈ closed_segment a b ⟹ d • x ≤ k
  proof −
    fix x :: complex
    assume x ∈ closed_segment a b
    then have ⋀C. x ∈ C ∨ b ∉ C ∨ a ∉ C ∨ ¬ convex C
      by (meson contra_subsetD convex_contains_segment)
    then show d • x ≤ k
      by (metis lek convex_halfspace_le mem_Collect_eq)
  qed
  have cs: closed_segment a' b' ⊆ {x. d • x ≤ k} ∧ closed_segment b' a ⊆ {x. d
• x ≤ k}
   by (simp add: ‹d • a' = k› ‹d • b' = k› closed_segment_subset convex_halfspace_le
lek(1))
  have continuous_on (S ∩ {x. d • x ≤ k}) f using contf
    by (simp add: continuous_on_subset)
  then have (f has_contour_integral 0)
        (linepath a b +++ linepath b a' +++ linepath a' b' +++ linepath b' a)
    apply (rule Cauchy_theorem_convex [where K = {}])
     by (simp_all add: path_image_join convex_Int convex_halfspace_le ‹convex
S› fcd_le ab_le
              closed_segment_subset abc a'b' ba' cs)
  then have 4: contour_integral (linepath a b) f +
              contour_integral (linepath b a') f +
              contour_integral (linepath a' b') f +
              contour_integral (linepath b' a) f = 0
    by (rule has_chain_integral_chain_integral4)
  have fcd_ge: f field_differentiable at x
            if x ∈ interior S ∧ x ∈ interior {x. k ≤ d • x} for x
  proof −
    have f2: f holomorphic_on S ∩ {c. k < d • c}
      by (metis (full_types) Collect_conj_eq Collect_mem_eq holf2)
    have f3: interior S = S
      by (simp add: interior_open ‹open S›)
    then have x ∈ S ∩ interior {c. k ≤ d • c}
      using that by simp
    then show f field_differentiable at x
```

```
    using f3 f2 unfolding holomorphic_on_def
   by (metis (no_types) ‹d ≠ 0› at_within_interior interior_Int interior_halfspace_ge
interior_interior)
  qed
  have cs: closed_segment c b' ⊆ {x. k ≤ d • x} ∧ closed_segment b' a' ⊆ {x. k
≤ d • x}
   by (simp add: ‹d • a' = k› b'c closed_segment_subset convex_halfspace_ge)
  have continuous_on (S ∩ {x. k ≤ d • x}) f using contf
   by (simp add: continuous_on_subset)
  then have (f has_contour_integral 0) (linepath a' c +++ linepath c b' +++
linepath b' a')
   apply (rule Cauchy_theorem_convex [where K = {}])
   by (simp_all add: path_image_join convex_Int convex_halfspace_ge ‹convex
S›
                     fcd_ge closed_segment_subset abc a'b' a'c cs)
  then have 5: contour_integral (linepath a' c) f + contour_integral (linepath c
b') f + contour_integral (linepath b' a') f = 0
   by (rule has_chain_integral_chain_integral3)
  show ?thesis
   using 1 2 3 4 5 by (metis add.assoc eq_neg_iff_add_eq_0 reversepath_linepath)
qed

lemma hol_pal_lem3:
  assumes S: convex S open S
     and abc: a ∈ S b ∈ S c ∈ S
     and d ≠ 0 and lek: d • a ≤ k
     and holf1: f holomorphic_on {z. z ∈ S ∧ d • z < k}
     and holf2: f holomorphic_on {z. z ∈ S ∧ k < d • z}
     and contf: continuous_on S f
   shows contour_integral (linepath a b) f +
         contour_integral (linepath b c) f +
         contour_integral (linepath c a) f = 0
proof (cases d • b ≤ k)
  case True show ?thesis
   by (rule hol_pal_lem2 [OF S abc ‹d ≠ 0› lek True holf1 holf2 contf])
next
  case False
  show ?thesis
  proof (cases d • c ≤ k)
   case True
   have contour_integral (linepath c a) f +
        contour_integral (linepath a b) f +
        contour_integral (linepath b c) f = 0
     by (rule hol_pal_lem2 [OF S ‹c ∈ S› ‹a ∈ S› ‹b ∈ S› ‹d ≠ 0› ‹d • c ≤ k›
lek holf1 holf2 contf])
   then show ?thesis
     by (simp add: algebra_simps)
  next
   case False
```

```
    have contour_integral (linepath b c) f +
         contour_integral (linepath c a) f +
         contour_integral (linepath a b) f = 0
      using hol_pal_lem2 [OF S ‹b ∈ S› ‹c ∈ S› ‹a ∈ S›, of −d −k]
      using ‹d ≠ 0› ‹¬ d · b ≤ k› False by (simp_all add: holf1 holf2 contf)
    then show ?thesis
      by (simp add: algebra_simps)
  qed
qed

lemma hol_pal_lem4:
  assumes S: convex S open S
      and abc: a ∈ S b ∈ S c ∈ S and d ≠ 0
      and holf1: f holomorphic_on {z. z ∈ S ∧ d · z < k}
      and holf2: f holomorphic_on {z. z ∈ S ∧ k < d · z}
      and contf: continuous_on S f
    shows contour_integral (linepath a b) f +
          contour_integral (linepath b c) f +
          contour_integral (linepath c a) f = 0
proof (cases d · a ≤ k)
  case True show ?thesis
    by (rule hol_pal_lem3 [OF S abc ‹d ≠ 0› True holf1 holf2 contf])
next
  case False
  show ?thesis
    using ‹d ≠ 0› hol_pal_lem3 [OF S abc, of −d −k] False
    by (simp_all add: holf1 holf2 contf)
qed

lemma holomorphic_on_paste_across_line:
  assumes S: open S and d ≠ 0
      and holf1: f holomorphic_on (S ∩ {z. d · z < k})
      and holf2: f holomorphic_on (S ∩ {z. k < d · z})
      and contf: continuous_on S f
    shows f holomorphic_on S
proof −
  have ∗: ∃ t. open t ∧ p ∈ t ∧ continuous_on t f ∧
            (∀ a b c. convex hull {a, b, c} ⊆ t ⟶
                      contour_integral (linepath a b) f +
                      contour_integral (linepath b c) f +
                      contour_integral (linepath c a) f = 0)
        if p ∈ S for p
  proof −
    obtain e where e>0 and e: ball p e ⊆ S
      using ‹p ∈ S› openE S by blast
    then have continuous_on (ball p e) f
      using contf continuous_on_subset by blast
    moreover have f holomorphic_on {z. dist p z < e ∧ d · z < k}
      apply (rule holomorphic_on_subset [OF holf1])
```

```
    using e by auto
  moreover have f holomorphic_on {z. dist p z < e ∧ k < d * z}
    apply (rule holomorphic_on_subset [OF holf2])
    using e by auto
  ultimately show ?thesis
    apply (rule_tac x=ball p e in exI)
    using ‹e > 0› e ‹d ≠ 0› hol_pal_lem4 [of ball p e _ _ _ d _ k]
    by (force simp add: subset_hull)
 qed
 show ?thesis
  by (blast intro: * Morera_local_triangle analytic_imp_holomorphic)
qed

proposition Schwarz_reflection:
 assumes open S and cnjs: cnj ‘ S ⊆ S
    and  holf: f holomorphic_on (S ∩ {z. 0 < Im z})
    and contf: continuous_on (S ∩ {z. 0 ≤ Im z}) f
    and f: ⋀z. ⟦z ∈ S; z ∈ ℝ⟧ ⟹ (f z) ∈ ℝ
  shows (λz. if 0 ≤ Im z then f z else cnj(f(cnj z))) holomorphic_on S
proof −
 have 1: (λz. if 0 ≤ Im z then f z else cnj (f (cnj z))) holomorphic_on (S ∩ {z.
0 < Im z})
  by (force intro: iffD1 [OF holomorphic_cong [OF refl] holf])
 have cont_cfc: continuous_on (S ∩ {z. Im z ≤ 0}) (cnj o f o cnj)
  using cnjs
   by (intro continuous_intros continuous_on_compose continuous_on_subset
[OF contf]) auto
 have cnj ∘ f ∘ cnj field_differentiable at x within S ∩ {z. Im z < 0}
      if x ∈ S Im x < 0 f field_differentiable at (cnj x) within S ∩ {z. 0 < Im z}
for x
  using that
   apply (clarsimp simp add: field_differentiable_def has_field_derivative_iff
Lim_within dist_norm)
  apply (rule_tac x=cnj f' in exI)
  apply (elim all_forward ex_forward conj_forward imp_forward asm_rl, clar-
ify)
  apply (drule_tac x=cnj xa in bspec)
  using cnjs apply force
   apply (metis complex_cnj_cnj complex_cnj_diff complex_cnj_divide com-
plex_mod_cnj)
  done
 then have hol_cfc: (cnj o f o cnj) holomorphic_on (S ∩ {z. Im z < 0})
  using holf cnjs
  by (force simp: holomorphic_on_def)
 have 2: (λz. if 0 ≤ Im z then f z else cnj (f (cnj z))) holomorphic_on (S ∩ {z.
Im z < 0})
  apply (rule iffD1 [OF holomorphic_cong [OF refl]])
  using hol_cfc by auto
 have [simp]: (S ∩ {z. 0 ≤ Im z}) ∪ (S ∩ {z. Im z ≤ 0}) = S
```

```
    by force
  have eq: ⋀z. ⟦z ∈ S; Im z ≤ 0; 0 ≤ Im z⟧ ⟹ f z = cnj (f (cnj z))
    using f Reals_cnj_iff complex_is_Real_iff by auto
  have continuous_on ((S ∩ {z. 0 ≤ Im z}) ∪ (S ∩ {z. Im z ≤ 0}))
                     (λz. if 0 ≤ Im z then f z else cnj (f (cnj z)))
    apply (rule continuous_on_cases_local)
    using cont_cfc contf
   by (simp_all add: closedin_closed_Int closed_halfspace_Im_le closed_halfspace_Im_ge
eq)
  then have 3: continuous_on S (λz. if 0 ≤ Im z then f z else cnj (f (cnj z)))
    by force
  show ?thesis
    apply (rule holomorphic_on_paste_across_line [OF ‹open S›, of − i _ 0])
    using 1 2 3 by auto
qed
```

5.9 Bloch's theorem

```
lemma Bloch_lemma_0:
  assumes holf: f holomorphic_on cball 0 r and 0 < r
      and [simp]: f 0 = 0
      and le: ⋀z. norm z < r ⟹ norm(deriv f z) ≤ 2 * norm(deriv f 0)
    shows ball 0 ((3 − 2 * sqrt 2) * r * norm(deriv f 0)) ⊆ f ‘ ball 0 r
proof −
  have sqrt 2 < 3/2
    by (rule real_less_lsqrt) (auto simp: power2_eq_square)
  then have sq3: 0 < 3 − 2 * sqrt 2 by simp
  show ?thesis
  proof (cases deriv f 0 = 0)
    case True then show ?thesis by simp
  next
    case False
    define C where C = 2 * norm(deriv f 0)
    have 0 < C using False by (simp add: C_def)
    have holf': f holomorphic_on ball 0 r using holf
      using ball_subset_cball holomorphic_on_subset by blast
    then have holdf': deriv f holomorphic_on ball 0 r
      by (rule holomorphic_deriv [OF _ open_ball])
    have Le1: norm(deriv f z − deriv f 0) ≤ norm z / (r − norm z) * C
              if norm z < r for z
    proof −
      have T1: norm(deriv f z − deriv f 0) ≤ norm z / (R − norm z) * C
            if R: norm z < R R < r for R
      proof −
        have 0 < R using R
          by (metis less_trans norm_zero zero_less_norm_iff)
        have df_le: ⋀x. norm x < r ⟹ norm (deriv f x) ≤ C
          using le by (simp add: C_def)
        have hol_df: deriv f holomorphic_on cball 0 R
```

```
      using R holdf' holomorphic_on_subset by auto
    have *: ((λw. deriv f w / (w − z)) has_contour_integral 2 * pi * i * deriv
f z) (circlepath 0 R)
        if norm z < R for z
     using ‹0 < R› that Cauchy_integral_formula_convex_simple [OF con-
vex_cball hol_df, of _ circlepath 0 R]
      by (force simp: winding_number_circlepath)
    have **: ((λx. deriv f x / (x − z) − deriv f x / x) has_contour_integral
          of_real (2 * pi) * i * (deriv f z − deriv f 0))
         (circlepath 0 R)
     using has_contour_integral_diff [OF * [of z] * [of 0]] ‹0 < R› that
     by (simp add: algebra_simps)
    have [simp]: ⋀x. norm x = R ⟹ x ≠ z using that(1) by blast
    have norm (deriv f x / (x − z) − deriv f x / x)
            ≤ C * norm z / (R * (R − norm z))
         if norm x = R for x
    proof −
     have [simp]: norm (deriv f x * x − deriv f x * (x − z)) =
              norm (deriv f x) * norm z
      by (simp add: norm_mult right_diff_distrib')
     show ?thesis
      using ‹0 < R› ‹0 < C› R that
        by (auto simp add: norm_mult norm_divide divide_simps df_le
mult_mono norm_triangle_ineq2)
    qed
    then show ?thesis
     using has_contour_integral_bound_circlepath
         [OF **, of C * norm z/(R*(R − norm z))]
        ‹0 < R› ‹0 < C› R
     apply (simp add: norm_mult norm_divide)
     apply (simp add: divide_simps mult.commute)
     done
   qed
   obtain r' where r': norm z < r' r' < r
    using Rats_dense_in_real [of norm z r] ‹norm z < r› by blast
   then have [simp]: closure {r'<..<r} = {r'..r} by simp
   show ?thesis
    apply (rule continuous_ge_on_closure
         [where f = λr. norm z / (r − norm z) * C and s = {r'<..<r},
          OF _ _ T1])
    using that r'
    by (auto simp: not_le intro!: continuous_intros)
  qed
  have *: (norm z − norm z^2/(r − norm z)) * norm(deriv f 0) ≤ norm(f z)
       if r: norm z < r for z
  proof −
   have 1: ⋀x. x ∈ ball 0 r ⟹
        ((λz. f z − deriv f 0 * z) has_field_derivative deriv f x − deriv f 0)
         (at x within ball 0 r)
```

```
    by (rule derivative_eq_intros holomorphic_derivI holf' | simp)+
  have 2: closed_segment 0 z ⊆ ball 0 r
   by (metis ‹0 < r› convex_ball convex_contains_segment dist_self mem_ball
mem_ball_0 that)
  have 4: norm (deriv f (x *R z) − deriv f 0) * norm z ≤ norm z * norm z *
x * C / (r − norm z)
        if x: 0 ≤ x x ≤ 1 for x
  proof −
    have [simp]: x * norm z < r
   using r x by (meson le_less_trans mult_le_cancel_right2 norm_not_less_zero)
    have norm (deriv f (x *R z) − deriv f 0) ≤ norm (x *R z) / (r − norm (x
*R z)) * C
      apply (rule Le1) using r x ‹0 < r› by simp
    also have ... ≤ norm (x *R z) / (r − norm z) * C
      using r x ‹0 < r›
      apply (simp add: field_split_simps)
   by (simp add: ‹0 < C› mult.assoc mult_left_le_one_le ordered_comm_semiring_class.comm_mult_le
    finally have norm (deriv f (x *R z) − deriv f 0) * norm z ≤ norm (x *R
z)  / (r − norm z) * C * norm z
      by (rule mult_right_mono) simp
    with x show ?thesis by (simp add: algebra_simps)
  qed
  have le_norm: abc ≤ norm d − e ⟹ norm(f − d) ≤ e ⟹ abc ≤ norm f
for abc d e and f::complex
  by (metis add_diff_cancel_left' add_diff_eq diff_left_mono norm_diff_ineq
order_trans)
  have norm (integral {0..1} (λx. (deriv f (x *R z) − deriv f 0) * z))
       ≤ integral {0..1} (λt. (norm z)^2 * t / (r − norm z) * C)
  proof (rule integral_norm_bound_integral)
    show (λx. (deriv f (x *R z) − deriv f 0) * z) integrable_on {0..1}
      using contour_integral_primitive [OF 1, of linepath 0 z] 2
   by (simp add: has_contour_integral_linepath has_integral_integrable_integral)
    have (*) ((cmod z)^2) integrable_on {0..1}
   by (metis ident_integrable_on integrable_0 integrable_eq integrable_on_cmult_iff
lambda_zero)
    then show (λt. (norm z)^2 * t / (r − norm z) * C) integrable_on {0..1}
   using integrable_on_cmult_right[where 'b=real, simplified] integrable_on_divide
[where 'b=real, simplified]
      by blast
  qed (simp add: norm_mult power2_eq_square 4)
  then have int_le: norm (f z − deriv f 0 * z) ≤ (norm z)^2 * norm(deriv f 0)
/ ((r − norm z))
    using contour_integral_primitive [OF 1, of linepath 0 z] 2
  by (simp add: has_contour_integral_linepath has_integral_integrable_integral
C_def)
   have norm z * (norm (deriv f 0) * (r − norm z − norm z)) ≤ norm z *
(norm (deriv f 0) * (r − norm z) − norm (deriv f 0) * norm z)
    by (simp add: algebra_simps)
  then have §: (norm z * (r − norm z) − norm z * norm z) * norm (deriv f
```

```
0) ≤ norm (deriv f 0) * norm z * (r - norm z) - norm z * norm z * norm (deriv
f 0)
        by (simp add: algebra_simps)
      show ?thesis
        apply (rule le_norm [OF _ int_le])
        using ‹norm z < r›
        by (simp add: power2_eq_square divide_simps C_def norm_mult §)
    qed
    have sq201 [simp]: 0 < (1 - sqrt 2 / 2) (1 - sqrt 2 / 2)  < 1
      by (auto simp:  sqrt2_less_2)
    have 1: continuous_on (closure (ball 0 ((1 - sqrt 2 / 2) * r))) f
    proof (rule continuous_on_subset [OF holomorphic_on_imp_continuous_on
[OF holf]])
      show closure (ball 0 ((1 - sqrt 2 / 2) * r)) ⊆ cball 0 r
      proof -
        have (1 - sqrt 2 / 2) * r ≤ r
          by (simp add: ‹0 < r›)
        then show ?thesis
        by (meson ball_subset_cball closed_cball closure_minimal dual_order.trans
subset_ball)
      qed
    qed
    have 2: open (f ‘ interior (ball 0 ((1 - sqrt 2 / 2) * r)))
    proof (rule open_mapping_thm [OF holf' open_ball connected_ball])
      show interior (ball 0 ((1 - sqrt 2 / 2) * r)) ⊆ ball (0::complex) r
        using ‹0 < r› mult_pos_pos sq201 by (simp add: ball_subset_ball_iff)
      show ¬ f constant_on ball 0 r
       using False ‹0 < r› centre_in_ball holf' holomorphic_nonconstant by blast
    qed auto
    have ball 0 ((3 - 2 * sqrt 2) * r * norm (deriv f 0)) =
          ball (f 0) ((3 - 2 * sqrt 2) * r * norm (deriv f 0))
      by simp
    also have ...  ⊆ f ‘ ball 0 ((1 - sqrt 2 / 2) * r)
    proof -
      have 3: (3 - 2 * sqrt 2) * r * norm (deriv f 0) ≤ norm (f z)
        if norm z = (1 - sqrt 2 / 2) * r for z
        apply (rule order_trans [OF _ *])
        using  ‹0 < r›
         apply (simp_all add: field_simps power2_eq_square that)
        apply (simp add: mult.assoc [symmetric])
        done
      show ?thesis
        apply (rule ball_subset_open_map_image [OF 1 2 _ bounded_ball])
        using ‹0 < r› sq201 3 C_def ‹0 < C› sq3 by auto
    qed
    also have ...  ⊆ f ‘ ball 0 r
    proof -
      have ⋀x. (1 - sqrt 2 / 2) * r ≤ r
        using ‹0 < r› by (auto simp: field_simps)
```

```
    then show ?thesis
      by auto
  qed
  finally show ?thesis .
 qed
qed

lemma Bloch_lemma:
 assumes holf: f holomorphic_on cball a r and 0 < r
   and le: ⋀z. z ∈ ball a r ⟹ norm(deriv f z) ≤ 2 * norm(deriv f a)
 shows ball (f a) ((3 - 2 * sqrt 2) * r * norm(deriv f a)) ⊆ f ‘ ball a r (is ?lhs
⊆ ?rhs)
proof -
 have fz: (λz. f (a + z)) = f o (λz. (a + z))
   by (simp add: o_def)
 have hol0: (λz. f (a + z)) holomorphic_on cball 0 r
   unfolding fz by (intro holomorphic_intros holf holomorphic_on_compose |
simp)+
 then have [simp]: ⋀x. norm x < r ⟹ (λz. f (a + z)) field_differentiable at x
   by (metis open_ball at_within_open ball_subset_cball diff_0 dist_norm holo-
morphic_on_def holomorphic_on_subset mem_ball norm_minus_cancel)
 have [simp]: ⋀z. norm z < r ⟹ f field_differentiable at (a + z)
   by (metis holf open_ball add_diff_cancel_left' dist_complex_def holomor-
phic_on_imp_differentiable_at holomorphic_on_subset interior_cball interior_subset
mem_ball norm_minus_commute)
 then have [simp]: f field_differentiable at a
   by (metis add.comm_neutral ‹0 < r› norm_eq_zero)
 have hol1: (λz. f (a + z) - f a) holomorphic_on cball 0 r
   by (intro holomorphic_intros hol0)
 then have §: ball 0 ((3 - 2 * sqrt 2) * r * norm (deriv (λz. f (a + z) - f a)
0))
             ⊆ (λz. f (a + z) - f a) ‘ ball 0 r
   apply (rule Bloch_lemma_0)
   using ‹0 < r›
    apply (simp_all add: ‹0 < r›)
   apply (simp add: fz deriv_chain dist_norm le)
   done
 show ?thesis
 proof clarify
   fix x
   assume x ∈ ?lhs
   with subsetD [OF §, of x - f a] show x ∈ ?rhs
   by (force simp: fz ‹0 < r› dist_norm deriv_chain field_differentiable_compose)
 qed
qed

proposition Bloch_unit:
 assumes holf: f holomorphic_on ball a 1 and [simp]: deriv f a = 1
 obtains b r where 1/12 < r and ball b r ⊆ f ‘ (ball a 1)
```

```
proof −
  define r :: real where r = 249/256
  have 0 < r r < 1 by (auto simp: r_def)
  define g where g z = deriv f z * of_real(r − norm(z − a)) for z
  have deriv f holomorphic_on ball a 1
    by (rule holomorphic_deriv [OF holf open_ball])
  then have continuous_on (ball a 1) (deriv f)
    using holomorphic_on_imp_continuous_on by blast
  then have continuous_on (cball a r) (deriv f)
    by (rule continuous_on_subset) (simp add: cball_subset_ball_iff ‹r < 1›)
  then have continuous_on (cball a r) g
    by (simp add: g_def continuous_intros)
  then have 1: compact (g ‘ cball a r)
    by (rule compact_continuous_image [OF _ compact_cball])
  have 2: g ‘ cball a r ≠ {}
    using ‹r > 0› by auto
  obtain p where pr: p ∈ cball a r
            and pge: ⋀y. y ∈ cball a r ⟹ norm (g y) ≤ norm (g p)
    using distance_attains_sup [OF 1 2, of 0] by force
  define t where t = (r − norm(p − a)) / 2
  have norm (p − a) ≠ r
    using pge [of a] ‹r > 0› by (auto simp: g_def norm_mult)
  then have norm (p − a) < r using pr
    by (simp add: norm_minus_commute dist_norm)
  then have 0 < t
    by (simp add: t_def)
  have cpt: cball p t ⊆ ball a r
   using ‹0 < t› by (simp add: cball_subset_ball_iff dist_norm t_def field_simps)
  have gen_le_dfp: norm (deriv f y) * (r − norm (y − a)) / (r − norm (p − a))
≤ norm (deriv f p)
            if y ∈ cball a r for y
  proof −
    have [simp]: norm (y − a) ≤ r
      using that by (simp add: dist_norm norm_minus_commute)
    have norm (g y) ≤ norm (g p)
      using pge [OF that] by simp
    then have norm (deriv f y) * abs (r − norm (y − a)) ≤ norm (deriv f p) *
abs (r − norm (p − a))
      by (simp only: dist_norm g_def norm_mult norm_of_real)
    with that ‹norm (p − a) < r› show ?thesis
      by (simp add: dist_norm field_split_simps)
  qed
  have le_norm_dfp: r / (r − norm (p − a)) ≤ norm (deriv f p)
    using gen_le_dfp [of a] ‹r > 0› by auto
  have 1: f holomorphic_on cball p t
    using cpt ‹r < 1› order_subst1 subset_ball
    by (force simp add: intro!: holomorphic_on_subset [OF holf])
  have 2: norm (deriv f z) ≤ 2 * norm (deriv f p) if z ∈ ball p t for z
  proof −
```

```
    have z: z ∈ cball a r
      by (meson ball_subset_cball subsetD cpt that)
    then have norm(z − a) < r
    by (metis ball_subset_cball contra_subsetD cpt dist_norm mem_ball norm_minus_commute
that)
    have norm (deriv f z) ∗ (r − norm (z − a)) / (r − norm (p − a)) ≤ norm
(deriv f p)
      using gen_le_dfp [OF z] by simp
    with ‹norm (z − a) < r› ‹norm (p − a) < r›
    have norm (deriv f z) ≤ (r − norm (p − a)) / (r − norm (z − a)) ∗ norm
(deriv f p)
      by (simp add: field_simps)
    also have ... ≤ 2 ∗ norm (deriv f p)
    proof (rule mult_right_mono)
      show (r − cmod (p − a)) / (r − cmod (z − a)) ≤ 2
       using that ‹norm (p − a) < r› ‹norm(z − a) < r› dist_triangle3 [of z a p]
        by (simp add: field_simps t_def dist_norm [symmetric])
    qed auto
    finally show ?thesis .
  qed
  have sqrt2: sqrt 2 < 2113/1494
    by (rule real_less_lsqrt) (auto simp: power2_eq_square)
  then have sq3: 0 < 3 − 2 ∗ sqrt 2 by simp
  have 1 / 12 / ((3 − 2 ∗ sqrt 2) / 2) < r
    using sq3 sqrt2 by (auto simp: field_simps r_def)
  also have ... ≤ cmod (deriv f p) ∗ (r − cmod (p − a))
    using ‹norm (p − a) < r› le_norm_dfp  by (simp add: pos_divide_le_eq)
  finally have 1 / 12 < cmod (deriv f p) ∗ (r − cmod (p − a)) ∗ ((3 − 2 ∗ sqrt
2) / 2)
    using pos_divide_less_eq half_gt_zero_iff sq3 by blast
  then have ∗∗: 1 / 12 < (3 − 2 ∗ sqrt 2) ∗ t ∗ norm (deriv f p)
    using sq3 by (simp add: mult.commute t_def)
  have ball (f p) ((3 − 2 ∗ sqrt 2) ∗ t ∗ norm (deriv f p)) ⊆ f ‘ ball p t
    by (rule Bloch_lemma [OF 1 ‹0 < t› 2])
  also have ... ⊆ f ‘ ball a 1
  proof −
    have ball a r ⊆ ball a 1
      using ‹0 < t› ‹r < 1› by (simp add: ball_subset_ball_iff dist_norm)
    then show ?thesis
      using ball_subset_cball cpt by blast
  qed
  finally have ball (f p) ((3 − 2 ∗ sqrt 2) ∗ t ∗ norm (deriv f p)) ⊆ f ‘ ball a 1 .
  with ∗∗ show ?thesis
    by (rule that)
qed

theorem Bloch:
  assumes holf: f holomorphic_on ball a r and 0 < r
      and r': r' ≤ r ∗ norm (deriv f a) / 12
```

```
  obtains b where ball b r' ⊆ f ` (ball a r)
proof (cases deriv f a = 0)
  case True with r' show ?thesis
    using ball_eq_empty that by fastforce
next
  case False
  define C where C = deriv f a
  have 0 < norm C using False by (simp add: C_def)
  have dfa: f field_differentiable at a
    using ‹0 < r› holomorphic_on_imp_differentiable_at [OF holf] by auto
  have fo: (λz. f (a + of_real r * z)) = f o (λz. (a + of_real r * z))
    by (simp add: o_def)
  have holf': f holomorphic_on (λz. a + complex_of_real r * z) ` ball 0 1
    using ‹0 < r› holomorphic_on_subset [OF holf] by (force simp: dist_norm
norm_mult)
  have 1: (λz. f (a + r * z) / (C * r)) holomorphic_on ball 0 1
    using ‹0 < r› ‹0 < norm C›
    by (intro holomorphic_intros holomorphic_on_compose holf'; simp add: fo)+
  have ((λz. f (a + of_real r * z) / (C * of_real r)) has_field_derivative
        (deriv f (a + of_real r * z) / C)) (at z)
       if norm z < 1 for z
  proof −
    have fd: f field_differentiable at (a + complex_of_real r * z)
    using ‹0 < r› by (simp_all add: dist_norm norm_mult holomorphic_on_imp_differentiable_at
[OF holf] that)
    have ∗: ((λx. f (a + of_real r * x)) has_field_derivative
          (deriv f (a + of_real r * z) * of_real r)) (at z)
    by (rule fd DERIV_chain [OF field_differentiable_derivI] derivative_eq_intros
| simp add: fo)+
    show ?thesis
      apply (rule derivative_eq_intros ∗ | simp)+
      using ‹0 < r› by (auto simp: C_def False)
  qed
  have deriv (λz. f (a + of_real r * z) / (C * of_real r)) 0 = deriv (λz. f (a +
complex_of_real r * z)) 0 /
    (C * complex_of_real r)
    apply (rule deriv_cdivide_right)
   by (metis (no_types) DERIV_chain2 add.right_neutral dfa field_differentiable_add_const
field_differentiable_def field_differentiable_linear fo mult_zero_right)
  also have ... = 1
   using ‹0 < r› by (simp add: C_def False fo derivative_intros dfa deriv_chain)
  finally have 2: deriv (λz. f (a + of_real r * z) / (C * of_real r)) 0 = 1 .
  have sb1: (∗) (C * r) ` (λz. f (a + of_real r * z) / (C * r)) ` ball 0 1
            ⊆ f ` ball a r
    using ‹0 < r› by (auto simp: dist_norm norm_mult C_def False)
  have sb2: ball (C * r * b) r' ⊆ (∗) (C * r) ` ball b t
            if 1 / 12 < t for b t
  proof −
    have ∗: r * cmod (deriv f a) / 12 ≤ r * (t * cmod (deriv f a))
```

```
    using that ‹0 < r› less_eq_real_def mult.commute mult.right_neutral mult_left_mono
norm_ge_zero times_divide_eq_right
      by auto
    show ?thesis
      apply clarify
      apply (rule_tac x=x / (C * r) in image_eqI)
       using ‹0 < r› apply (simp_all add: dist_norm norm_mult norm_divide
C_def False field_simps)
      using * r' by linarith
  qed
  show ?thesis
    apply (rule Bloch_unit [OF 1 2])
    using image_mono sb1 sb2 that by fastforce
qed

corollary Bloch_general:
  assumes holf: f holomorphic_on S and a ∈ S
      and tle: ⋀z. z ∈ frontier S ⟹ t ≤ dist a z
      and rle: r ≤ t * norm(deriv f a) / 12
  obtains b where ball b r ⊆ f ‘ S
proof −
  consider r ≤ 0 | 0 < t * norm(deriv f a) / 12 using rle by force
  then show ?thesis
  proof cases
    case 1 then show ?thesis
      by (simp add: ball_empty that)
  next
    case 2
    show ?thesis
    proof (cases deriv f a = 0)
      case True then show ?thesis
        using rle by (simp add: ball_empty that)
    next
      case False
      then have t > 0
        using 2 by (force simp: zero_less_mult_iff)
      have ¬ ball a t ⊆ S ⟹ ball a t ∩ frontier S ≠ {}
       by (metis Diff_eq_empty_iff ‹0 < t› ‹a ∈ S› closure_Int_ball_not_empty
closure_subset connected_Int_frontier connected_ball inf.commute)
      with tle have *: ball a t ⊆ S by fastforce
      then have 1: f holomorphic_on ball a t
        using holf using holomorphic_on_subset by blast
      show ?thesis
        using Bloch [OF 1 ‹t > 0› rle] * by (metis image_mono order_trans that)
    qed
  qed
qed

end
```

theory *Complex_Singularities*
imports *Conformal_Mappings*
begin

5.10 Non-essential singular points

definition *is_pole* ::
(*'a*::*topological_space* ⇒ *'b*::*real_normed_vector*) ⇒ *'a* ⇒ *bool* **where**
is_pole f a = (*LIM x* (*at a*). *f x* :> *at_infinity*)

lemma *is_pole_cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) (*at a*) *a*=*b*
shows *is_pole f a* ⟷ *is_pole g b*
unfolding *is_pole_def* **using** *assms* **by** (*intro filterlim_cong*,*auto*)

lemma *is_pole_transform*:
assumes *is_pole f a eventually* (λ*x*. *f x* = *g x*) (*at a*) *a*=*b*
shows *is_pole g b*
using *is_pole_cong assms* **by** *auto*

lemma *is_pole_shift_iff*:
fixes *f* :: (*'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*)
shows *is_pole* (*f* ∘ (+) *d*) *a* = *is_pole f* (*a* + *d*)
by (*metis add_diff_cancel_right' filterlim_shift_iff is_pole_def*)

lemma *is_pole_tendsto*:
fixes *f*::(*'a*::*topological_space* ⇒ *'b*::*real_normed_div_algebra*)
shows *is_pole f x* ⟹ ((*inverse o f*) ⟶ *0*) (*at x*)
unfolding *is_pole_def*
by (*auto simp add*:*filterlim_inverse_at_iff*[*symmetric*] *comp_def filterlim_at*)

lemma *is_pole_shift_0*:
fixes *f* :: (*'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*)
shows *is_pole f z* ⟷ *is_pole* (λ*x*. *f* (*z* + *x*)) *0*
unfolding *is_pole_def* **by** (*subst at_to_0*) (*auto simp*: *filterlim_filtermap add_ac*)

lemma *is_pole_shift_0'*:
fixes *f* :: (*'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*)
shows *NO_MATCH 0 z* ⟹ *is_pole f z* ⟷ *is_pole* (λ*x*. *f* (*z* + *x*)) *0*
by (*metis is_pole_shift_0*)

lemma *is_pole_compose_iff*:
assumes *filtermap g* (*at x*) = (*at y*)
shows *is_pole* (*f* ∘ *g*) *x* ⟷ *is_pole f y*
unfolding *is_pole_def filterlim_def filtermap_compose assms* **..**

lemma *is_pole_inverse_holomorphic*:
assumes *open s*
and *f_holo*:*f holomorphic_on* (*s*−{*z*})

```
    and pole:is_pole f z
    and non_z:∀ x∈s−{z}. f x≠0
  shows (λx. if x=z then 0 else inverse (f x)) holomorphic_on s
proof −
  define g where g ≡ λx. if x=z then 0 else inverse (f x)
  have isCont g z unfolding isCont_def  using is_pole_tendsto[OF pole]
    by (simp add: g_def cong: LIM_cong)
 moreover have continuous_on (s−{z}) f using f_holo holomorphic_on_imp_continuous_on
by auto
  hence continuous_on (s−{z}) (inverse o f) unfolding comp_def
    by (auto elim!:continuous_on_inverse simp add:non_z)
  hence continuous_on (s−{z}) g unfolding g_def
    using continuous_on_eq by fastforce
  ultimately have continuous_on s g using open_delete[OF ‹open s›] ‹open s›
    by (auto simp add:continuous_on_eq_continuous_at)
  moreover have (inverse o f) holomorphic_on (s−{z})
    unfolding comp_def using f_holo
    by (auto elim!:holomorphic_on_inverse simp add:non_z)
  hence g holomorphic_on (s−{z})
    using g_def holomorphic_transform by force
  ultimately show ?thesis unfolding g_def using ‹open s›
    by (auto elim!: no_isolated_singularity)
qed

lemma not_is_pole_holomorphic:
  assumes open A x ∈ A f holomorphic_on A
  shows  ¬is_pole f x
proof −
  have continuous_on A f by (intro holomorphic_on_imp_continuous_on) fact
  with assms have isCont f x by (simp add: continuous_on_eq_continuous_at)
  hence f −x→ f x by (simp add: isCont_def)
  thus ¬is_pole f x unfolding is_pole_def
    using not_tendsto_and_filterlim_at_infinity[of at x f f x] by auto
qed

lemma is_pole_inverse_power: n > 0 ⟹ is_pole (λz::complex. 1 / (z − a) ^
n) a
  unfolding is_pole_def inverse_eq_divide [symmetric]
  by (intro filterlim_compose[OF filterlim_inverse_at_infinity] tendsto_intros)
     (auto simp: filterlim_at eventually_at intro!: exI[of _ 1] tendsto_eq_intros)

lemma is_pole_cmult_iff [simp]:
  c ≠ 0 ⟹ is_pole (λz. c * f z :: 'a :: real_normed_field) z ⟷ is_pole f z
proof
  assume *: c ≠ 0 is_pole (λz. c * f z) z
  have is_pole (λz. inverse c * (c * f z)) z unfolding is_pole_def
    by (rule tendsto_mult_filterlim_at_infinity tendsto_const)+ (use * in ‹auto
simp: is_pole_def›)
  thus is_pole f z
```

```
    using ∗(1) by (simp add: field_simps)
next
  assume ∗: c ≠ 0 is_pole f z
  show is_pole (λz. c ∗ f z) z unfolding is_pole_def
    by (rule tendsto_mult_filterlim_at_infinity tendsto_const)+ (use ∗ in ‹auto
simp: is_pole_def›)
qed

lemma is_pole_uminus_iff [simp]: is_pole (λz. −f z :: 'a :: real_normed_field)
z ⟷ is_pole f z
  using is_pole_cmult_iff[of −1 f] by (simp del: is_pole_cmult_iff)

lemma is_pole_inverse: is_pole (λz::complex. 1 / (z − a)) a
  using is_pole_inverse_power[of 1 a] by simp

lemma is_pole_divide:
  fixes f :: 'a :: t2_space ⇒ 'b :: real_normed_field
  assumes isCont f z filterlim g (at 0) (at z) f z ≠ 0
  shows   is_pole (λz. f z / g z) z
proof −
  have filterlim (λz. f z ∗ inverse (g z)) at_infinity (at z)
    using assms filterlim_compose filterlim_inverse_at_infinity isCont_def
      tendsto_mult_filterlim_at_infinity by blast
  thus ?thesis by (simp add: field_split_simps is_pole_def)
qed

lemma is_pole_basic:
  assumes f holomorphic_on A open A z ∈ A f z ≠ 0 n > 0
  shows   is_pole (λw. f w / (w − z) ^ n) z
proof (rule is_pole_divide)
  have continuous_on A f by (rule holomorphic_on_imp_continuous_on) fact
  with assms show isCont f z by (auto simp: continuous_on_eq_continuous_at)
  have filterlim (λw. (w − z) ^ n) (nhds 0) (at z)
    using assms by (auto intro!: tendsto_eq_intros)
  thus filterlim (λw. (w − z) ^ n) (at 0) (at z)
    by (intro filterlim_atI tendsto_eq_intros)
       (insert assms, auto simp: eventually_at_filter)
qed fact+

lemma is_pole_basic':
  assumes f holomorphic_on A open A 0 ∈ A f 0 ≠ 0 n > 0
  shows   is_pole (λw. f w / w ^ n) 0
  using is_pole_basic[of f A 0] assms by simp

lemma is_pole_compose:
  assumes is_pole f w g −z→ w eventually (λz. g z ≠ w) (at z)
  shows   is_pole (λx. f (g x)) z
  using assms(1) unfolding is_pole_def
  by (rule filterlim_compose) (use assms in ‹auto simp: filterlim_at›)
```

lemma *is_pole_plus_const_iff*:
is_pole f z $\longleftrightarrow$ *is_pole* $(\lambda x.\ f\ x + c)\ z$
proof
assume *is_pole f z*
then have *filterlim f at_infinity* (*at z*) **unfolding** *is_pole_def* **.**
moreover have $((\lambda\_.\ c) \longrightarrow c)$ (*at z*) **by** *auto*
ultimately have *LIM x* (*at z*). *f x* + *c* :> *at_infinity*
using *tendsto_add_filterlim_at_infinity'*[*of f at z*] **by** *auto*
then show *is_pole* $(\lambda x.\ f\ x + c)\ z$ **unfolding** *is_pole_def* **.**
next
assume *is_pole* $(\lambda x.\ f\ x + c)\ z$
then have *filterlim* $(\lambda x.\ f\ x + c)$ *at_infinity* (*at z*)
unfolding *is_pole_def* **.**
moreover have $((\lambda\_.\ -c) \longrightarrow -c)$ (*at z*) **by** *auto*
ultimately have *LIM x* (*at z*). *f x* :> *at_infinity*
using *tendsto_add_filterlim_at_infinity'*[*of* $(\lambda x.\ f\ x + c)$
at z $(\lambda\_.\ -\ c)\ -c$]
by *auto*
then show *is_pole f z* **unfolding** *is_pole_def* **.**
qed

lemma *is_pole_minus_const_iff*:
is_pole $(\lambda x.\ f\ x - c)\ z \longleftrightarrow$ *is_pole f z*
using *is_pole_plus_const_iff* [*of f z* $-c$] **by** *simp*

lemma *is_pole_alt*:
is_pole f x $= (\forall B{>}0.\ \exists U.\ open\ U \wedge x{\in}U \wedge (\forall y{\in}U.\ y{\neq}x \longrightarrow norm\ (f\ y){\geq}B))$
unfolding *is_pole_def*
unfolding *filterlim_at_infinity*[*of 0*,*simplified*] *eventually_at_topological*
by *auto*

lemma *is_pole_mult_analytic_nonzero1*:
assumes *is_pole g x f analytic_on* $\{x\}$ $f\ x \neq 0$
shows *is_pole* $(\lambda x.\ f\ x * g\ x)\ x$
unfolding *is_pole_def*
proof (*rule tendsto_mult_filterlim_at_infinity*)
show $f \longrightarrow x \rightarrow f\ x$
using *assms* **by** (*simp add*: *analytic_at_imp_isCont isContD*)
qed (*use assms* **in** ‹*auto simp*: *is_pole_def*›)

lemma *is_pole_mult_analytic_nonzero2*:
assumes *is_pole f x g analytic_on* $\{x\}$ $g\ x \neq 0$
shows *is_pole* $(\lambda x.\ f\ x * g\ x)\ x$
by (*subst mult.commute*, *rule is_pole_mult_analytic_nonzero1*) (*use assms* **in** *auto*)

lemma *is_pole_mult_analytic_nonzero1_iff*:
assumes *f analytic_on* $\{x\}$ $f\ x \neq 0$

```
  shows   is_pole (λx. f x * g x) x ⟷ is_pole g x
proof
  assume is_pole g x
  thus is_pole (λx. f x * g x) x
    by (intro is_pole_mult_analytic_nonzero1 assms)
next
  assume is_pole (λx. f x * g x) x
  hence is_pole (λx. inverse (f x) * (f x * g x)) x
    by (rule is_pole_mult_analytic_nonzero1)
       (use assms in ‹auto intro!: analytic_intros›)
  also have ?this ⟷ is_pole g x
  proof (rule is_pole_cong)
    have eventually (λx. f x ≠ 0) (at x)
      using assms by (simp add: analytic_at_neq_imp_eventually_neq)
    thus eventually (λx. inverse (f x) * (f x * g x) = g x) (at x)
      by eventually_elim auto
  qed auto
  finally show is_pole g x .
qed

lemma is_pole_mult_analytic_nonzero2_iff:
  assumes g analytic_on {x} g x ≠ 0
  shows   is_pole (λx. f x * g x) x ⟷ is_pole f x
 by (subst mult.commute, rule is_pole_mult_analytic_nonzero1_iff) (fact assms)+

lemma frequently_const_imp_not_is_pole:
  fixes z :: 'a::first_countable_topology
  assumes frequently (λw. f w = c) (at z)
  shows   ¬ is_pole f z
proof
  assume is_pole f z
  from assms have z islimpt {w. f w = c}
    by (simp add: islimpt_conv_frequently_at)
  then obtain g where g: ⋀n. g n ∈ {w. f w = c} - {z} g ⟶ z
    unfolding islimpt_sequential by blast
  then have (f ∘ g) ⟶ c
    by (simp add: tendsto_eventually)
  moreover have *: filterlim g (at z) sequentially
    using g by (auto simp: filterlim_at)
  have filterlim (f ∘ g) at_infinity sequentially
    unfolding o_def by (rule filterlim_compose [OF _ *])
                       (use ‹is_pole f z› in ‹simp add: is_pole_def›)
  ultimately show False
     using not_tendsto_and_filterlim_at_infinity trivial_limit_sequentially by
blast
qed
```

The proposition $\exists x.\ f \mathrel{-z\to} x \vee$ *is_pole f z* can be interpreted as the complex function f has a non-essential singularity at z (i.e. the singularity is

either removable or a pole).

definition *not_essential*::[*complex* ⇒ *complex*, *complex*] ⇒ *bool* **where**
not_essential f z = (∃ *x*. *f*−*z*→*x* ∨ *is_pole f z*)

definition *isolated_singularity_at*::[*complex* ⇒ *complex*, *complex*] ⇒ *bool* **where**
isolated_singularity_at f z = (∃ *r*>*0*. *f analytic_on ball z r*−{*z*})

lemma *not_essential_cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) (*at z*) *z* = *z*′
shows *not_essential f z* ⟷ *not_essential g z*′
unfolding *not_essential_def* **using** *assms filterlim_cong is_pole_cong* **by** *fastforce*

lemma *not_essential_compose_iff*:
assumes *filtermap g* (*at z*) = *at z*′
shows *not_essential* (*f* ∘ *g*) *z* = *not_essential f z*′
unfolding *not_essential_def filterlim_def filtermap_compose assms is_pole_compose_iff*[*OF assms*]
by *blast*

lemma *isolated_singularity_at_cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) (*at z*) *z* = *z*′
shows *isolated_singularity_at f z* ⟷ *isolated_singularity_at g z*′
proof −
have *isolated_singularity_at g z*
if *isolated_singularity_at f z eventually* (λ*x*. *f x* = *g x*) (*at z*) **for** *f g*
proof −
from *that*(*1*) **obtain** *r* **where** *r*: *r* > *0 f analytic_on ball z r* − {*z*}
by (*auto simp*: *isolated_singularity_at_def*)
from *that*(*2*) **obtain** *r*′ **where** *r*′: *r*′ > *0* ∀ *x*∈*ball z r*′−{*z*}. *f x* = *g x*
unfolding *eventually_at_filter eventually_nhds_metric* **by** (*auto simp*: *dist_commute*)

have *f holomorphic_on ball z r* − {*z*}
using *r*(*2*) **by** (*subst* (*asm*) *analytic_on_open*) *auto*
hence *f holomorphic_on ball z* (*min r r*′) − {*z*}
by (*rule holomorphic_on_subset*) *auto*
also have *?this* ⟷ *g holomorphic_on ball z* (*min r r*′) − {*z*}
using *r*′ **by** (*intro holomorphic_cong*) *auto*
also have ... ⟷ *g analytic_on ball z* (*min r r*′) − {*z*}
by (*subst analytic_on_open*) *auto*
finally show *?thesis*
unfolding *isolated_singularity_at_def*
by (*intro exI*[*of* _ *min r r*′]) (*use* ‹*r* > *0*› ‹*r*′ > *0*› **in** *auto*)
qed
from *this*[*of f g*] *this*[*of g f*] *assms* **show** *?thesis*
by (*auto simp*: *eq_commute*)
qed

```
lemma removable_singularity:
  assumes f holomorphic_on A − {x} open A
  assumes f −x→ c
  shows   (λy. if y = x then c else f y) holomorphic_on A (is ?g holomorphic_on
_)
proof −
  have continuous_on A ?g
    unfolding continuous_on_def
  proof
    fix y assume y: y ∈ A
    show (?g ⟶ ?g y) (at y within A)
    proof (cases y = x)
      case False
      have continuous_on (A − {x}) f
        using assms(1) by (meson holomorphic_on_imp_continuous_on)
      hence (f ⟶ ?g y) (at y within A − {x})
        using False y by (auto simp: continuous_on_def)
      also have ?this ⟷ (?g ⟶ ?g y) (at y within A − {x})
        by (intro filterlim_cong refl) (auto simp: eventually_at_filter)
      also have at y within A − {x} = at y within A
        using y assms False
        by (intro at_within_nhd[of _ A − {x}]) auto
      finally show ?thesis .
    next
      case [simp]: True
      have f −x→ c
        by fact
      also have ?this ⟷ (?g ⟶ ?g y) (at y)
        by (intro filterlim_cong) (auto simp: eventually_at_filter)
      finally show ?thesis
        by (meson Lim_at_imp_Lim_at_within)
    qed
  qed
  moreover {
    have ?g holomorphic_on A − {x}
      using assms(1) holomorphic_transform by fastforce
  }
  ultimately show ?thesis
    by (rule no_isolated_singularity) (use assms in auto)
qed

lemma removable_singularity':
  assumes isolated_singularity_at f z
  assumes f −z→ c
  shows   (λy. if y = z then c else f y) analytic_on {z}
proof −
  from assms obtain r where r: r > 0 f analytic_on ball z r − {z}
    by (auto simp: isolated_singularity_at_def)
  have (λy. if y = z then c else f y) holomorphic_on ball z r
```

```
  proof (rule removable_singularity)
    show f holomorphic_on ball z r − {z}
      using r(2) by (subst (asm) analytic_on_open) auto
  qed (use assms in auto)
  thus ?thesis
    using r(1) unfolding analytic_at by (intro exI[of _ ball z r]) auto
qed

named_theorems singularity_intros introduction rules for singularities

lemma holomorphic_factor_unique:
  fixes f::complex ⇒ complex and z::complex and r::real and m n::int
  assumes r>0 g z≠0 h z≠0
    and asm:∀ w∈ball z r−{z}. f w = g w ∗ (w−z) powi n ∧ g w≠0 ∧ f w =  h w
∗ (w − z) powi m ∧ h w≠0
    and g_holo:g holomorphic_on ball z r and h_holo:h holomorphic_on ball z r
  shows n=m
proof −
  have [simp]:at z within ball z r ≠ bot using ‹r>0›
      by (auto simp add:at_within_ball_bot_iff)
  have False when n>m
  proof −
    have (h ⟶ 0) (at z within ball z r)
    proof (rule Lim_transform_within[OF _ ‹r>0›, where f=λw. (w − z) powi
(n − m) ∗ g w])
      have ∀ w∈ball z r−{z}. h w = (w−z)powi(n−m) ∗ g w
        using ‹n>m› asm ‹r>0› by (simp add: field_simps power_int_diff) force
      then show ⟦x′ ∈ ball z r; 0 < dist x′ z;dist x′ z < r⟧
            ⟹ (x′ − z) powi (n − m) ∗ g x′ = h x′ for x′ by auto
    next
      define F where F ≡ at z within ball z r
      define f′ where f′ ≡ λx. (x − z) powi (n−m)
      have f′ z=0 using ‹n>m› unfolding f′_def by auto
      moreover have continuous F f′ unfolding f′_def F_def continuous_def
        using ‹n>m›
          by (auto simp add: Lim_ident_at  intro!:tendsto_powr_complex_0 tend-
sto_eq_intros)
      ultimately have (f′ ⟶ 0) F unfolding F_def
        by (simp add: continuous_within)
      moreover have (g ⟶ g z) F
        unfolding F_def
     using ‹r>0› centre_in_ball continuous_on_def g_holo holomorphic_on_imp_continuous_on
by blast
       ultimately show  ((λw. f′ w ∗ g w) ⟶ 0) F using tendsto_mult by
fastforce
    qed
    moreover have (h ⟶ h z) (at z within ball z r)
      using holomorphic_on_imp_continuous_on[OF h_holo]
      by (auto simp add:continuous_on_def ‹r>0›)
```

```
    ultimately have h z=0 by (auto intro!: tendsto_unique)
    thus False using ‹h z≠0› by auto
  qed
  moreover have False when m>n
  proof −
    have (g ⟶ 0) (at z within ball z r)
    proof (rule Lim_transform_within[OF _ ‹r>0›, where f=λw. (w − z) powi
(m − n) ∗ h w])
      have ∀ w∈ball z r −{z}. g w = (w−z) powi (m−n) ∗ h w using ‹m>n› asm
        by (simp add:field_simps power_int_diff) force
      then show ⟦x′ ∈ ball z r; 0 < dist x′ z;dist x′ z < r⟧
          ⟹ (x′ − z) powi (m − n) ∗ h x′ = g x′ for x′ by auto
    next
      define F where F ≡ at z within ball z r
      define f′ where f′ ≡λx. (x − z) powi (m−n)
      have f′ z=0 using ‹m>n› unfolding f′_def by auto
      moreover have continuous F f′ unfolding f′_def F_def continuous_def
        using ‹m>n›
      by (auto simp: Lim_ident_at intro!:tendsto_powr_complex_0 tendsto_eq_intros)
      ultimately have (f′ ⟶ 0) F unfolding F_def
        by (simp add: continuous_within)
      moreover have (h ⟶ h z) F
        using holomorphic_on_imp_continuous_on[OF h_holo,unfolded continu-
ous_on_def] ‹r>0›
        unfolding F_def by auto
      ultimately show ((λw. f′ w ∗ h w) ⟶ 0) F using tendsto_mult by
fastforce
    qed
    moreover have (g ⟶ g z) (at z within ball z r)
      using holomorphic_on_imp_continuous_on[OF g_holo]
      by (auto simp add:continuous_on_def ‹r>0›)
    ultimately have g z=0 by (auto intro!: tendsto_unique)
    thus False using ‹g z≠0› by auto
  qed
  ultimately show n=m by fastforce
qed

lemma holomorphic_factor_puncture:
  assumes f_iso:isolated_singularity_at f z
      and not_essential f z — f has either a removable singularity or a pole at z
      and non_zero:∃ F w in (at z). f w≠0 — f will not be constantly zero in a
neighbour of z
  shows ∃!n::int. ∃ g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀ w∈cball z r−{z}. f w = g w ∗ (w−z) powi n ∧ g w≠0)
proof −
  define P where P = (λf n g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀ w∈cball z r − {z}. f w = g w ∗ (w−z) powi n  ∧ g w≠0))
  have imp_unique:∃!n::int. ∃ g r. P f n g r when ∃ n g r. P f n g r
  proof (rule ex_ex1I[OF that])
```

fix *n1 n2* :: *int*
assume *g1_asm*:∃ *g1 r1*. *P f n1 g1 r1* **and** *g2_asm*:∃ *g2 r2*. *P f n2 g2 r2*
define *fac* **where** *fac* ≡ λ*n g r*. ∀ *w*∈*cball z r*−{*z*}. *f w* = *g w* ∗ (*w* − *z*) *powi n* ∧ *g w* ≠ *0*
obtain *g1 r1* **where** *0* < *r1* **and** *g1_holo*: *g1 holomorphic_on cball z r1* **and** *g1 z*≠*0*
and *fac n1 g1 r1* **using** *g1_asm* **unfolding** *P_def fac_def* **by** *auto*
obtain *g2 r2* **where** *0* < *r2* **and** *g2_holo*: *g2 holomorphic_on cball z r2* **and** *g2 z*≠*0*
and *fac n2 g2 r2* **using** *g2_asm* **unfolding** *P_def fac_def* **by** *auto*
define *r* **where** *r* ≡ *min r1 r2*
have *r*>*0* **using** ‹*r1*>*0*› ‹*r2*>*0*› **unfolding** *r_def* **by** *auto*
moreover have ∀ *w*∈*ball z r*−{*z*}. *f w* = *g1 w* ∗ (*w*−*z*) *powi n1* ∧ *g1 w*≠*0*
∧ *f w* = *g2 w* ∗ (*w* − *z*) *powi n2* ∧ *g2 w*≠*0*
using ‹*fac n1 g1 r1*› ‹*fac n2 g2 r2*› **unfolding** *fac_def r_def*
by *fastforce*
ultimately show *n1*=*n2* **using** *g1_holo g2_holo* ‹*g1 z*≠*0*› ‹*g2 z*≠*0*›
apply (*elim holomorphic_factor_unique*)
by (*auto simp add*:*r_def*)
qed

have *P_exist*:∃ *n g r*. *P h n g r* **when**
∃ *z'*. (*h* ⟶ *z'*) (*at z*) *isolated_singularity_at h z* ∃ *F w in* (*at z*). *h w*≠*0*
for *h*
proof −
from *that*(*2*) **obtain** *r* **where** *r*>*0* **and** *r*: *h analytic_on ball z r* − {*z*}
unfolding *isolated_singularity_at_def* **by** *auto*
obtain *z'* **where** (*h* ⟶ *z'*) (*at z*) **using** ‹∃ *z'*. (*h* ⟶ *z'*) (*at z*)› **by** *auto*
define *h'* **where** *h'*=(λ*x*. *if x*=*z then z' else h x*)
have *h' holomorphic_on ball z r*
proof (*rule no_isolated_singularity'*[*of* {*z*}])
show ⋀*w*. *w* ∈ {*z*} ⟹ (*h'* ⟶ *h' w*) (*at w within ball z r*)
by (*simp add*: *LIM_cong Lim_at_imp_Lim_at_within* ‹*h* −*z*→ *z'*› *h'_def*)
show *h' holomorphic_on ball z r* − {*z*}
using *r analytic_imp_holomorphic h'_def holomorphic_transform* **by** *fastforce*
qed *auto*
have *?thesis* **when** *z'*=*0*
proof −
have *h' z*=*0* **using** *that* **unfolding** *h'_def* **by** *auto*
moreover have ¬ *h' constant_on ball z r*
using ‹∃ *F w in* (*at z*). *h w*≠*0*› **unfolding** *constant_on_def frequently_def eventually_at h'_def*
by (*metis* ‹*0* < *r*› *centre_in_ball dist_commute mem_ball that*)
moreover note ‹*h' holomorphic_on ball z r*›
ultimately obtain *g r1 n* **where** *0* < *n 0* < *r1 ball z r1* ⊆ *ball z r* **and**
g:*g holomorphic_on ball z r1*
⋀*w*. *w* ∈ *ball z r1* ⟹ *h' w* = (*w* − *z*) ^ *n* ∗ *g w*
⋀*w*. *w* ∈ *ball z r1* ⟹ *g w* ≠ *0*

```
      using holomorphic_factor_zero_nonconstant[of _ ball z r z thesis,simplified,
            OF ‹h' holomorphic_on ball z r› ‹r>0› ‹h' z=0› ‹¬ h' constant_on
ball z r›]
       by (auto simp add:dist_commute)
     define rr where rr=r1/2
     have P h' n g rr
       unfolding P_def rr_def
       using ‹n>0› ‹r1>0› g by (auto simp add:powr_nat)
     then have P h n g rr
       unfolding h'_def P_def by auto
     then show ?thesis unfolding P_def by blast
   qed
   moreover have ?thesis when z'≠0
   proof −
     have h' z≠0 using that unfolding h'_def by auto
     obtain r1 where r1>0 cball z r1 ⊆ ball z r ∀x∈cball z r1. h' x≠0
     proof −
       have isCont h' z h' z≠0
         by (auto simp add: Lim_cong_within ‹h −z→ z'› ‹z'≠0› continuous_at
h'_def)
       then obtain r2 where r2:r2>0 ∀x∈ball z r2. h' x≠0
         using continuous_at_avoid[of z h' 0 ] unfolding ball_def by auto
       define r1 where r1=min r2 r / 2
       have 0 < r1 cball z r1 ⊆ ball z r
         using ‹r2>0› ‹r>0› unfolding r1_def by auto
       moreover have ∀x∈cball z r1. h' x ≠ 0
         using r2 unfolding r1_def by simp
       ultimately show ?thesis using that by auto
     qed
    then have P h' 0 h' r1 using ‹h' holomorphic_on ball z r› unfolding P_def
by auto
     then have P h 0 h' r1 unfolding P_def h'_def by auto
     then show ?thesis unfolding P_def by blast
   qed
   ultimately show ?thesis by auto
 qed

 have ?thesis when ∃x. (f ⟶ x) (at z)
   apply (rule_tac imp_unique[unfolded P_def])
   using P_exist[OF that(1) f_iso non_zero] unfolding P_def .
 moreover have ?thesis when is_pole f z
 proof (rule imp_unique[unfolded P_def])
   obtain e where [simp]:e>0 and e_holo:f holomorphic_on ball z e − {z} and
e_nz: ∀x∈ball z e−{z}. f x≠0
   proof −
     have ∀F z in at z. f z ≠ 0
       using ‹is_pole f z› filterlim_at_infinity_imp_eventually_ne unfolding
is_pole_def
       by auto
```

```
    then obtain e1 where e1:e1>0 ∀x∈ball z e1−{z}. f x≠0
      using that eventually_at[of λx. f x≠0 z UNIV,simplified] by (auto simp
add:dist_commute)
    obtain e2 where e2:e2>0 f holomorphic_on ball z e2 − {z}
    using f_iso analytic_imp_holomorphic unfolding isolated_singularity_at_def
by auto
    show ?thesis
      using e1 e2 by (force intro: that[of min e1 e2])
  qed

  define h where h ≡ λx. inverse (f x)
  have ∃ n g r. P h n g r
  proof −
    have (λx. inverse (f x)) analytic_on ball z e − {z}
    by (metis e_holo e_nz open_ball analytic_on_open holomorphic_on_inverse
open_delete)
    moreover have h −z→ 0
      using Lim_transform_within_open assms(2) h_def is_pole_tendsto that
by fastforce
    moreover have ∃F w in (at z). h w≠0
      using non_zero by (simp add: h_def)
    ultimately show ?thesis
      using P_exist[of h] ‹e > 0›
      unfolding isolated_singularity_at_def h_def
      by blast
  qed
  then obtain n g r
    where 0 < r and
        g_holo:g holomorphic_on cball z r and g z≠0 and
        g_fac:(∀w∈cball z r−{z}. h w = g w ∗ (w − z) powi n ∧ g w ≠ 0)
    unfolding P_def by auto
  have P f (−n) (inverse o g) r
  proof −
    have f w = inverse (g w) ∗ (w − z) powi (− n) when w∈cball z r − {z} for
w
    by (metis g_fac h_def inverse_inverse_eq inverse_mult_distrib power_int_minus
that)
    then show ?thesis
      unfolding P_def comp_def
      using ‹r>0› g_holo g_fac ‹g z≠0› by (auto intro:holomorphic_intros)
  qed
  then show ∃x g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z ≠ 0
          ∧ (∀w∈cball z r − {z}. f w = g w ∗ (w − z) powi x ∧ g w ≠ 0)
    unfolding P_def by blast
 qed
 ultimately show ?thesis using ‹not_essential f z› unfolding not_essential_def
by presburger
qed
```

lemma *not_essential_transform*:
assumes *not_essential g z*
assumes $\forall_F$ *w in* (*at z*). *g w* = *f w*
shows *not_essential f z*
using *assms* **unfolding** *not_essential_def*
by (*simp add*: *filterlim_cong is_pole_cong*)

lemma *isolated_singularity_at_transform*:
assumes *isolated_singularity_at g z*
assumes $\forall_F$ *w in* (*at z*). *g w* = *f w*
shows *isolated_singularity_at f z*
proof −
obtain *r1* **where** *r1*>*0* **and** *r1*:*g analytic_on ball z r1* − {*z*}
using *assms*(*1*) **unfolding** *isolated_singularity_at_def* **by** *auto*
obtain *r2* **where** *r2*>*0* **and** *r2*: ∀*x*. *x* ≠ *z* ∧ *dist x z* < *r2* ⟶ *g x* = *f x*
using *assms*(*2*) **unfolding** *eventually_at* **by** *auto*
define *r3* **where** *r3*=*min r1 r2*
have *r3*>*0* **unfolding** *r3_def* **using** ‹*r1*>*0*› ‹*r2*>*0*› **by** *auto*
moreover have *f analytic_on ball z r3* − {*z*}
proof −
have *g holomorphic_on ball z r3* − {*z*}
using *r1* **unfolding** *r3_def* **by** (*subst* (*asm*) *analytic_on_open*,*auto*)
then have *f holomorphic_on ball z r3* − {*z*}
using *r2* **unfolding** *r3_def*
by (*auto simp add*:*dist_commute elim*!:*holomorphic_transform*)
then show *?thesis* **by** (*subst analytic_on_open*,*auto*)
qed
ultimately show *?thesis* **unfolding** *isolated_singularity_at_def* **by** *auto*
qed

lemma *not_essential_powr*[*singularity_intros*]:
assumes *LIM w* (*at z*). *f w* :> (*at x*)
shows *not_essential* (λ*w*. (*f w*) *powi n*) *z*
proof −
define *fp* **where** *fp*=(λ*w*. (*f w*) *powi n*)
have *?thesis* **when** *n*>*0*
proof −
have (λ*w*. (*f w*) ^ (*nat n*)) −*z*→ *x* ^ *nat n*
using *that assms* **unfolding** *filterlim_at* **by** (*auto intro*!:*tendsto_eq_intros*)
then have *fp* −*z*→ *x* ^ *nat n* **unfolding** *fp_def*
by (*smt* (*verit*) *LIM_cong power_int_def that*)
then show *?thesis* **unfolding** *not_essential_def fp_def* **by** *auto*
qed
moreover have *?thesis* **when** *n*=*0*
proof −
have $\forall_F$ *x in at z*. *fp x* = *1*
using *that filterlim_at_within_not_equal*[*OF assms*] **by** (*auto simp*: *fp_def*)
then have *fp* −*z*→ *1*
by (*simp add*: *tendsto_eventually*)

```
    then show ?thesis unfolding fp_def not_essential_def by auto
  qed
  moreover have ?thesis when n<0
  proof (cases x=0)
    case True
    have (λx. f x ^ nat (- n)) -z→ 0
    using assms True that unfolding filterlim_at by (auto intro!:tendsto_eq_intros)
    moreover have ∀\_F x in at z. f x ^ nat (- n) ≠ 0
    by (smt (verit) True assms eventually_at_topological filterlim_at power_eq_0_iff)
    ultimately have LIM w (at z). inverse ((f w) ^ (nat (-n))) :> at_infinity
      by (metis filterlim_atI filterlim_compose filterlim_inverse_at_infinity)
    then have LIM w (at z). fp w :> at_infinity
    proof (elim filterlim_mono_eventually)
      show ∀\_F x in at z. inverse (f x ^ nat (- n)) = fp x
        using filterlim_at_within_not_equal[OF assms,of 0] unfolding fp_def
        by (smt (verit) eventuallyI power_int_def power_inverse that)
    qed auto
    then show ?thesis unfolding fp_def not_essential_def is_pole_def by auto
  next
    case False
    let ?xx= inverse (x ^ (nat (-n)))
    have (λw. inverse ((f w) ^ (nat (-n)))) -z→?xx
      using assms False unfolding filterlim_at by (auto intro!:tendsto_eq_intros)
    then have fp -z→ ?xx
      by (smt (verit, best) LIM_cong fp_def power_int_def power_inverse that)
    then show ?thesis unfolding fp_def not_essential_def by auto
  qed
  ultimately show ?thesis by linarith
qed

lemma isolated_singularity_at_powr[singularity_intros]:
  assumes isolated_singularity_at f z ∀\_F w in (at z). f w≠0
  shows isolated_singularity_at (λw. (f w) powi n) z
proof -
  obtain r1 where r1>0 f analytic_on ball z r1 - {z}
    using assms(1) unfolding isolated_singularity_at_def by auto
  then have r1:f holomorphic_on ball z r1 - {z}
    using analytic_on_open[of ball z r1-{z} f] by blast
  obtain r2 where r2>0 and r2:∀ w. w ≠ z ∧ dist w z < r2 ⟶ f w ≠ 0
    using assms(2) unfolding eventually_at by auto
  define r3 where r3=min r1 r2
  have (λw. (f w) powi n) holomorphic_on ball z r3 - {z}
   by (intro holomorphic_on_power_int) (use r1 r2 in ‹auto simp: dist_commute
r3_def›)
  moreover have r3>0 unfolding r3_def using ‹0 < r1› ‹0 < r2› by linarith
  ultimately show ?thesis
   by (meson open_ball analytic_on_open isolated_singularity_at_def open_delete)
qed
```

```
lemma non_zero_neighbour:
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in (at z). f w≠0
    shows ∀ F w in (at z). f w≠0
proof −
  obtain fn fp fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
                ∀ w∈cball z fr − {z}. f w = fp w ∗ (w − z) powi fn ∧ fp w ≠ 0
   using holomorphic_factor_puncture[OF f_iso f_ness f_nconst,THEN ex1_implies_ex]
by auto
  have f w ≠ 0 when  w ≠ z dist w z < fr for w
  proof −
    have f w = fp w ∗ (w − z) powi fn fp w ≠ 0
     using fr(2)[rule_format, of w] using that by (auto simp add:dist_commute)
    moreover have (w − z) powi fn ≠0
      unfolding powr_eq_0_iff using ‹w≠z› by auto
    ultimately show ?thesis by auto
  qed
  then show ?thesis using ‹fr>0› unfolding eventually_at by auto
qed

lemma non_zero_neighbour_pole:
  assumes is_pole f z
  shows ∀ F w in (at z). f w≠0
  using assms filterlim_at_infinity_imp_eventually_ne[of f at z 0]
  unfolding is_pole_def by auto

lemma non_zero_neighbour_alt:
  assumes holo: f holomorphic_on S
      and open S connected S z ∈ S  β ∈ S f β ≠ 0
    shows ∀ F w in (at z). f w≠0 ∧ w∈S
proof (cases f z = 0)
  case True
  from isolated_zeros[OF holo ‹open S› ‹connected S› ‹z ∈ S› True ‹β ∈ S› ‹f β
≠ 0›]
  obtain r where 0 < r ball z r ⊆ S ∀ w ∈ ball z r − {z}.f w ≠ 0 by metis
  then show ?thesis
     by (smt (verit) open_ball centre_in_ball eventually_at_topological insertE
insert_Diff subsetD)
next
  case False
  obtain r1 where r1:r1>0 ∀ y. dist z y < r1 ⟶ f y ≠ 0
   using continuous_at_avoid[of z f, OF _ False] assms(2,4) continuous_on_eq_continuous_at
      holo holomorphic_on_imp_continuous_on by blast
  obtain r2 where r2:r2>0 ball z r2 ⊆ S
    using assms openE by blast
  show ?thesis unfolding eventually_at
   by (metis (no_types) dist_commute dual_order.strict_trans linorder_less_linear
```

mem_ball r1 r2 subsetD)
qed

lemma *not_essential_times*[*singularity_intros*]:
assumes *f_ness*:*not_essential f z* **and** *g_ness*:*not_essential g z*
assumes *f_iso*:*isolated_singularity_at f z* **and** *g_iso*:*isolated_singularity_at g z*
shows *not_essential* (λ*w*. *f w* ∗ *g w*) *z*
proof −
define *fg* **where** *fg* = (λ*w*. *f w* ∗ *g w*)
have *?thesis* **when** ¬ ((∃ $_F$*w in* (*at z*). *f w*≠*0*) ∧ (∃ $_F$*w in* (*at z*). *g w*≠*0*))
proof −
have ∀ $_F$*w in* (*at z*). *fg w*=*0*
using *that*[*unfolded frequently_def*, *simplified*] **unfolding** *fg_def*
by (*auto elim*: *eventually_rev_mp*)
from *tendsto_cong*[*OF this*] **have** *fg* −*z*→*0* **by** *auto*
then show *?thesis* **unfolding** *not_essential_def fg_def* **by** *auto*
qed
moreover have *?thesis* **when** *f_nconst*:∃ $_F$*w in* (*at z*). *f w*≠*0* **and** *g_nconst*:∃ $_F$*w in* (*at z*). *g w*≠*0*
proof −
obtain *fn fp fr* **where** [*simp*]:*fp z* ≠ *0* **and** *fr* > *0*
and *fr*: *fp holomorphic_on cball z fr*
∀ *w*∈*cball z fr* − {*z*}. *f w* = *fp w* ∗ (*w* − *z*) *powi fn* ∧ *fp w* ≠ *0*
using *holomorphic_factor_puncture*[*OF f_iso f_ness f_nconst*,*THEN ex1_implies_ex*]
by *auto*
obtain *gn gp gr* **where** [*simp*]:*gp z* ≠ *0* **and** *gr* > *0*
and *gr*: *gp holomorphic_on cball z gr*
∀ *w*∈*cball z gr* − {*z*}. *g w* = *gp w* ∗ (*w* − *z*) *powi gn* ∧ *gp w* ≠ *0*
using *holomorphic_factor_puncture*[*OF g_iso g_ness g_nconst*,*THEN ex1_implies_ex*]
by *auto*

define *r1* **where** *r1*=(*min fr gr*)
have *r1*>*0* **unfolding** *r1_def* **using** ‹*fr*>*0*› ‹*gr*>*0*› **by** *auto*
have *fg_times*:*fg w* = (*fp w* ∗ *gp w*) ∗ (*w* − *z*) *powi* (*fn*+*gn*) **and** *fgp_nz*:*fp w*∗*gp w*≠*0*
when *w*∈*ball z r1* − {*z*} **for** *w*
proof −
have *f w* = *fp w* ∗ (*w* − *z*) *powi fn fp w*≠*0*
using *fr*(*2*)[*rule_format*,*of w*] *that* **unfolding** *r1_def* **by** *auto*
moreover have *g w* = *gp w* ∗ (*w* − *z*) *powi gn gp w* ≠ *0*
using *gr*(*2*)[*rule_format*, *of w*] *that* **unfolding** *r1_def* **by** *auto*
ultimately show *fg w* = (*fp w* ∗ *gp w*) ∗ (*w* − *z*) *powi* (*fn*+*gn*) *fp w*∗*gp w*≠*0*
using *that* **unfolding** *fg_def* **by** (*auto simp add*:*power_int_add*)
qed

have [*intro*]: *fp* −*z*→*fp z gp* −*z*→*gp z*
using *fr*(*1*) ‹*fr*>*0*› *gr*(*1*) ‹*gr*>*0*›
by (*meson open_ball ball_subset_cball centre_in_ball*

```
      continuous_on_eq_continuous_at continuous_within holomorphic_on_imp_continuous_on
        holomorphic_on_subset)+
  have ?thesis when fn+gn>0
  proof −
    have (λw. (fp w ∗ gp w) ∗ (w − z) ^ (nat (fn+gn))) −z→0
      using that by (auto intro!:tendsto_eq_intros)
    then have fg −z→ 0
      apply (elim Lim_transform_within[OF _ ‹r1>0›])
    by (smt (verit, best) Diff_iff dist_commute fg_times mem_ball power_int_def
singletonD that zero_less_dist_iff)
    then show ?thesis unfolding not_essential_def fg_def by auto
  qed
  moreover have ?thesis when fn+gn=0
  proof −
    have (λw. fp w ∗ gp w) −z→fp z∗gp z
      using that by (auto intro!:tendsto_eq_intros)
    then have fg −z→ fp z∗gp z
      apply (elim Lim_transform_within[OF _ ‹r1>0›])
      apply (subst fg_times)
      by (auto simp add:dist_commute that)
    then show ?thesis unfolding not_essential_def fg_def by auto
  qed
  moreover have ?thesis when fn+gn<0
  proof −
    have LIM x at z. (x − z) ^ nat (− (fn + gn)) :> at 0
      using eventually_at_topological that
      by (force intro!: tendsto_eq_intros filterlim_atI)
    moreover have ∃ c. (λc. fp c ∗ gp c) −z→ c ∧ 0 ≠ c
      using ‹fp −z→ fp z› ‹gp −z→ gp z› tendsto_mult by fastforce
    ultimately have LIM w (at z). fp w ∗ gp w / (w−z)^nat (−(fn+gn)) :>
at_infinity
      using filterlim_divide_at_infinity by blast
    then have is_pole fg z unfolding is_pole_def
      apply (elim filterlim_transform_within[OF _ _ ‹r1>0›])
      using that
        by (simp_all add: dist_commute fg_times of_int_of_nat divide_simps
power_int_def del: minus_add_distrib)
    then show ?thesis unfolding not_essential_def fg_def by auto
  qed
  ultimately show ?thesis unfolding not_essential_def fg_def by fastforce
 qed
 ultimately show ?thesis by auto
qed

lemma not_essential_inverse[singularity_intros]:
 assumes f_ness:not_essential f z
 assumes f_iso:isolated_singularity_at f z
 shows not_essential (λw. inverse (f w)) z
proof −
```

```
  define vf where vf = (λw. inverse (f w))
  have ?thesis when ¬(∃ F w in (at z). f w≠0)
  proof −
    have ∀ F w in (at z). f w=0
     using that[unfolded frequently_def, simplified] by (auto elim: eventually_rev_mp)
    then have ∀ F w in (at z). vf w=0
      unfolding vf_def by auto
    from tendsto_cong[OF this] have vf −z→0 unfolding vf_def by auto
    then show ?thesis unfolding not_essential_def vf_def by auto
  qed
  moreover have ?thesis when is_pole f z
  proof −
    have vf −z→0
     using that filterlim_at filterlim_inverse_at_iff unfolding is_pole_def vf_def
by blast
    then show ?thesis unfolding not_essential_def vf_def by auto
  qed
  moreover have ?thesis when ∃ x. f−z→x and f_nconst:∃ F w in (at z). f w≠0
  proof −
    from that obtain fz where fz:f−z→fz by auto
    have ?thesis when fz=0
    proof −
      have (λw. inverse (vf w)) −z→0
        using fz that unfolding vf_def by auto
      moreover have ∀ F w in at z. inverse (vf w) ≠ 0
        using non_zero_neighbour[OF f_iso f_ness f_nconst]
        unfolding vf_def by auto
      ultimately have is_pole vf z
       using filterlim_inverse_at_iff[of vf at z] unfolding filterlim_at is_pole_def
by auto
      then show ?thesis unfolding not_essential_def vf_def by auto
    qed
    moreover have ?thesis when fz≠0
    proof −
      have vf −z→inverse fz
        using fz that unfolding vf_def by (auto intro:tendsto_eq_intros)
      then show ?thesis unfolding not_essential_def vf_def by auto
    qed
    ultimately show ?thesis by auto
  qed
  ultimately show ?thesis using f_ness unfolding not_essential_def by auto
qed

lemma isolated_singularity_at_inverse[singularity_intros]:
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
  shows isolated_singularity_at (λw. inverse (f w)) z
proof −
  define vf where vf = (λw. inverse (f w))
```

```
  have ?thesis when ¬(∃ F w in (at z). f w≠0)
  proof −
    have ∀ F w in (at z). f w=0
    using that[unfolded frequently_def, simplified] by (auto elim: eventually_rev_mp)
    then have ∀ F w in (at z). vf w=0
      unfolding vf_def by auto
    then obtain d1 where d1>0 and d1:∀ x. x ≠ z ∧ dist x z < d1 ⟶ vf x = 0
      unfolding eventually_at by auto
    then have vf holomorphic_on ball z d1−{z}
      apply (rule_tac holomorphic_transform[of λ_. 0])
      by (auto simp add:dist_commute)
    then have vf analytic_on ball z d1 − {z}
      by (simp add: analytic_on_open open_delete)
      then show ?thesis using ‹d1>0› unfolding isolated_singularity_at_def
vf_def by auto
  qed
  moreover have ?thesis when f_nconst:∃ F w in (at z). f w≠0
  proof −
     have ∀ F w in at z. f w ≠ 0 using non_zero_neighbour[OF f_iso f_ness
f_nconst] .
    then obtain d1 where d1:d1>0 ∀ x. x ≠ z ∧ dist x z < d1 ⟶ f x ≠ 0
      unfolding eventually_at by auto
    obtain d2 where d2>0 and d2:f analytic_on ball z d2 − {z}
      using f_iso unfolding isolated_singularity_at_def by auto
    define d3 where d3=min d1 d2
    have d3>0 unfolding d3_def using ‹d1>0› ‹d2>0› by auto
    moreover
    have f analytic_on ball z d3 − {z}
      by (smt (verit, best) Diff_iff analytic_on_analytic_at d2 d3_def mem_ball)
      then have vf analytic_on ball z d3 − {z}
      unfolding vf_def
      by (intro analytic_on_inverse; simp add: d1(2) d3_def dist_commute)
     ultimately show ?thesis unfolding isolated_singularity_at_def vf_def by
auto
  qed
  ultimately show ?thesis by auto
qed

lemma not_essential_divide[singularity_intros]:
  assumes f_ness:not_essential f z and g_ness:not_essential g z
  assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
  shows not_essential (λw. f w / g w) z
proof −
  have not_essential (λw. f w * inverse (g w)) z
     by (simp add: f_iso f_ness g_iso g_ness isolated_singularity_at_inverse
not_essential_inverse not_essential_times)
  then show ?thesis by (simp add:field_simps)
qed
```

lemma
assumes *f_iso*:*isolated_singularity_at f z*
and *g_iso*:*isolated_singularity_at g z*
shows *isolated_singularity_at_times*[*singularity_intros*]:
isolated_singularity_at (λ*w*. *f w* ∗ *g w*) *z* **and**
isolated_singularity_at_add[*singularity_intros*]:
isolated_singularity_at (λ*w*. *f w* + *g w*) *z*
proof −
obtain *d1 d2* **where** *d1*>*0 d2*>*0*
and *d1*:*f analytic_on ball z d1* − {*z*} **and** *d2*:*g analytic_on ball z d2* − {*z*}
using *f_iso g_iso* **unfolding** *isolated_singularity_at_def* **by** *auto*
define *d3* **where** *d3*=*min d1 d2*
have *d3*>*0* **unfolding** *d3_def* **using** ‹*d1*>*0*› ‹*d2*>*0*› **by** *auto*

have *fan*: *f analytic_on ball z d3* − {*z*}
by (*smt* (*verit*, *best*) *Diff_iff analytic_on_analytic_at d1 d3_def mem_ball*)
have *gan*: *g analytic_on ball z d3* − {*z*}
by (*smt* (*verit*, *best*) *Diff_iff analytic_on_analytic_at d2 d3_def mem_ball*)
have (λ*w*. *f w* ∗ *g w*) *analytic_on ball z d3* − {*z*}
using *analytic_on_mult fan gan* **by** *blast*
then show *isolated_singularity_at* (λ*w*. *f w* ∗ *g w*) *z*
using ‹*d3*>*0*› **unfolding** *isolated_singularity_at_def* **by** *auto*
have (λ*w*. *f w* + *g w*) *analytic_on ball z d3* − {*z*}
using *analytic_on_add fan gan* **by** *blast*
then show *isolated_singularity_at* (λ*w*. *f w* + *g w*) *z*
using ‹*d3*>*0*› **unfolding** *isolated_singularity_at_def* **by** *auto*
qed

lemma *isolated_singularity_at_uminus*[*singularity_intros*]:
assumes *f_iso*:*isolated_singularity_at f z*
shows *isolated_singularity_at* (λ*w*. − *f w*) *z*
using *assms* **unfolding** *isolated_singularity_at_def* **using** *analytic_on_neg* **by** *blast*

lemma *isolated_singularity_at_id*[*singularity_intros*]:
isolated_singularity_at (λ*w*. *w*) *z*
unfolding *isolated_singularity_at_def* **by** (*simp add*: *gt_ex*)

lemma *isolated_singularity_at_minus*[*singularity_intros*]:
assumes *f_iso*:*isolated_singularity_at f z*
and *g_iso*:*isolated_singularity_at g z*
shows *isolated_singularity_at* (λ*w*. *f w* − *g w*) *z*
using *isolated_singularity_at_uminus*[*THEN isolated_singularity_at_add*[*OF f_iso*,*of* λ*w*. − *g w*]
,*OF g_iso*] **by** *simp*

lemma *isolated_singularity_at_divide*[*singularity_intros*]:
assumes *f_iso*:*isolated_singularity_at f z*

```
    and g_iso:isolated_singularity_at g z
    and g_ness:not_essential g z
  shows isolated_singularity_at (λw. f w / g w) z
 using isolated_singularity_at_inverse[THEN isolated_singularity_at_times[OF
f_iso,
       of λw. inverse (g w)],OF g_iso g_ness] by (simp add:field_simps)

lemma isolated_singularity_at_const[singularity_intros]:
  isolated_singularity_at (λw. c) z
 unfolding isolated_singularity_at_def by (simp add: gt_ex)

lemma isolated_singularity_at_holomorphic:
 assumes f holomorphic_on s-{z} open s z∈s
 shows isolated_singularity_at f z
 using assms unfolding isolated_singularity_at_def
 by (metis analytic_on_holomorphic centre_in_ball insert_Diff openE open_delete
subset_insert_iff)

lemma isolated_singularity_at_altdef:
 isolated_singularity_at f z ⟷ eventually (λz. f analytic_on {z}) (at z)
proof
 assume isolated_singularity_at f z
 then obtain r where r: r > 0 f analytic_on ball z r - {z}
   unfolding isolated_singularity_at_def by blast
 have eventually (λw. w ∈ ball z r - {z}) (at z)
   using r(1) by (intro eventually_at_in_open) auto
 thus eventually (λz. f analytic_on {z}) (at z)
   by eventually_elim (use r analytic_on_subset in auto)
next
 assume eventually (λz. f analytic_on {z}) (at z)
 then obtain A where A: open A z ∈ A ⋀w. w ∈ A - {z} ⟹ f analytic_on
{w}
   unfolding eventually_at_topological by blast
 then show isolated_singularity_at f z
  by (meson analytic_imp_holomorphic analytic_on_analytic_at isolated_singularity_at_holomorphic)
qed

lemma isolated_singularity_at_shift:
 assumes isolated_singularity_at (λx. f (x + w)) z
 shows   isolated_singularity_at f (z + w)
proof -
 from assms obtain r where r: r > 0 and ana: (λx. f (x + w)) analytic_on
ball z r - {z}
   unfolding isolated_singularity_at_def by blast
 have ((λx. f (x + w)) ∘ (λx. x - w)) analytic_on (ball (z + w) r - {z + w})
   by (rule analytic_on_compose_gen[OF _ ana])
      (auto simp: dist_norm algebra_simps intro!: analytic_intros)
 hence f analytic_on (ball (z + w) r - {z + w})
   by (simp add: o_def)
```

thus *?thesis* **using** *r*
unfolding *isolated_singularity_at_def* **by** *blast*
qed

lemma *isolated_singularity_at_shift_iff*:
isolated_singularity_at f (*z* + *w*) ⟷ *isolated_singularity_at* (λ*x*. *f* (*x* + *w*)) *z*
using *isolated_singularity_at_shift*[*of f w z*]
isolated_singularity_at_shift[*of* λ*x*. *f* (*x* + *w*) −*w w* + *z*]
by (*auto simp*: *algebra_simps*)

lemma *isolated_singularity_at_shift_0*:
NO_MATCH 0 z ⟹ *isolated_singularity_at f z* ⟷ *isolated_singularity_at* (λ*x*. *f* (*z* + *x*)) *0*
using *isolated_singularity_at_shift_iff*[*of f 0 z*] **by** (*simp add*: *add_ac*)

lemma *not_essential_shift*:
assumes *not_essential* (λ*x*. *f* (*x* + *w*)) *z*
shows *not_essential f* (*z* + *w*)
proof −
from *assms* **consider** *c* **where** (λ*x*. *f* (*x* + *w*)) −*z*→ *c* | *is_pole* (λ*x*. *f* (*x* + *w*)) *z*
unfolding *not_essential_def* **by** *blast*
thus *?thesis*
proof *cases*
case (*1 c*)
hence *f* −*z* + *w*→ *c*
by (*smt* (*verit*, *ccfv_SIG*) *LIM_cong add.assoc filterlim_at_to_0*)
thus *?thesis*
by (*auto simp*: *not_essential_def*)
next
case *2*
hence *is_pole f* (*z* + *w*)
by (*subst is_pole_shift_iff* [*symmetric*]) (*auto simp*: *o_def add_ac*)
thus *?thesis*
by (*auto simp*: *not_essential_def*)
qed
qed

lemma *not_essential_shift_iff*: *not_essential f* (*z* + *w*) ⟷ *not_essential* (λ*x*. *f* (*x* + *w*)) *z*
using *not_essential_shift*[*of f w z*]
not_essential_shift[*of* λ*x*. *f* (*x* + *w*) −*w w* + *z*]
by (*auto simp*: *algebra_simps*)

lemma *not_essential_shift_0*:
NO_MATCH 0 z ⟹ *not_essential f z* ⟷ *not_essential* (λ*x*. *f* (*z* + *x*)) *0*
using *not_essential_shift_iff*[*of f 0 z*] **by** (*simp add*: *add_ac*)

lemma *not_essential_holomorphic*:

assumes *f holomorphic_on A x* ∈ *A open A*
shows *not_essential f x*
proof −
have *continuous_on A f*
using *assms holomorphic_on_imp_continuous_on* **by** *blast*
hence *f* −*x*→ *f x*
using *assms continuous_on_eq_continuous_at isContD* **by** *blast*
thus *?thesis*
by (*auto simp*: *not_essential_def*)
qed

lemma *not_essential_analytic*:
assumes *f analytic_on* {*z*}
shows *not_essential f z*
using *analytic_at assms not_essential_holomorphic* **by** *blast*

lemma *not_essential_id* [*singularity_intros*]: *not_essential* (λ*w*. *w*) *z*
by (*simp add*: *not_essential_analytic*)

lemma *is_pole_imp_not_essential* [*intro*]: *is_pole f z* ⟹ *not_essential f z*
by (*auto simp*: *not_essential_def*)

lemma *tendsto_imp_not_essential* [*intro*]: *f* −*z*→ *c* ⟹ *not_essential f z*
by (*auto simp*: *not_essential_def*)

lemma *eventually_not_pole*:
assumes *isolated_singularity_at f z*
shows *eventually* (λ*w*. ¬*is_pole f w*) (*at z*)
proof −
from *assms* **obtain** *r* **where** *r* > *0* **and** *r*: *f analytic_on ball z r* − {*z*}
by (*auto simp*: *isolated_singularity_at_def*)
then have *eventually* (λ*w*. *w* ∈ *ball z r* − {*z*}) (*at z*)
by (*intro eventually_at_in_open*) *auto*
thus *eventually* (λ*w*. ¬*is_pole f w*) (*at z*)
proof *eventually_elim*
case (*elim w*)
with *r* **show** *?case*
using *analytic_imp_holomorphic not_is_pole_holomorphic open_delete* **by**
blast
qed
qed

lemma *not_islimpt_poles*:
assumes *isolated_singularity_at f z*
shows ¬*z islimpt* {*w*. *is_pole f w*}
using *eventually_not_pole* [*OF assms*]
by (*auto simp*: *islimpt_conv_frequently_at frequently_def*)

lemma *analytic_at_imp_no_pole*: *f analytic_on* {*z*} ⟹ ¬*is_pole f z*

using *analytic_at not_is_pole_holomorphic* **by** *blast*

lemma *not_essential_const* [*singularity_intros*]: *not_essential* (λ_. *c*) *z*
by *blast*

lemma *not_essential_uminus* [*singularity_intros*]:
assumes *f_ness*: *not_essential f z*
assumes *f_iso*:*isolated_singularity_at f z*
shows *not_essential* (λ*w*. −*f w*) *z*
proof −
have *not_essential* (λ*w*. −*1* ∗ *f w*) *z*
by (*intro assms singularity_intros*)
thus *?thesis* **by** *simp*
qed

lemma *isolated_singularity_at_analytic*:
assumes *f analytic_on* {*z*}
shows *isolated_singularity_at f z*
proof −
from *assms* **obtain** *r* **where** *r*: *r* > *0 f holomorphic_on ball z r*
by (*auto simp*: *analytic_on_def*)
show *?thesis*
by (*rule isolated_singularity_at_holomorphic*[*of f ball z r*])
(*use* ‹*r* > *0*› **in** ‹*auto intro*!: *holomorphic_on_subset*[*OF r*(*2*)]›)
qed

5.11 The order of non-essential singularities (i.e. removable singularities or poles)

definition *zorder* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ *int* **where**
zorder f z = (*THE n*. (∃ *h r*. *r*>*0* ∧ *h holomorphic_on cball z r* ∧ *h z*≠*0*
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *h w* ∗ (*w*−*z*) *powi n*
∧ *h w* ≠*0*)))

definition *zor_poly*
::[*complex* ⇒ *complex*, *complex*] ⇒ *complex* ⇒ *complex* **where**
zor_poly f z = (*SOME h*. ∃ *r*. *r* > *0* ∧ *h holomorphic_on cball z r* ∧ *h z* ≠ *0*
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *h w* ∗ (*w* − *z*) *powi* (*zorder f z*)
∧ *h w* ≠*0*))

lemma *zorder_exist*:
fixes *f*::*complex* ⇒ *complex* **and** *z*::*complex*
defines *n*≡*zorder f z* **and** *g*≡*zor_poly f z*
assumes *f_iso*:*isolated_singularity_at f z*
and *f_ness*:*not_essential f z*
and *f_nconst*:∃ *w* *in* (*at z*). *f w*≠*0*
shows *g z*≠*0* ∧ (∃ *r*. *r*>*0* ∧ *g holomorphic_on cball z r*
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *g w* ∗ (*w*−*z*) *powi n* ∧ *g w* ≠*0*))
proof −

```
  define P where P = (λn g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀w∈cball z r − {z}. f w = g w * (w−z) powi n ∧ g w≠0))
  have ∃!n. ∃ g r. P n g r
   using holomorphic_factor_puncture[OF assms(3−)] unfolding P_def by auto
  then have ∃ g r. P n g r
    unfolding n_def P_def zorder_def
    by (drule_tac theI',argo)
  then have ∃ r. P n g r
    unfolding P_def zor_poly_def g_def n_def
    by (drule_tac someI_ex,argo)
  then obtain r1 where P n g r1 by auto
  then show ?thesis unfolding P_def by auto
qed

lemma zorder_shift:
  shows  zorder f z = zorder (λu. f (u + z)) 0
  unfolding zorder_def
  apply (rule arg_cong [of concl: The])
  apply (auto simp: fun_eq_iff Ball_def dist_norm)
  subgoal for x h r
    apply (rule_tac x=h o (+)z in exI)
    apply (rule_tac x=r in exI)
    apply (intro conjI holomorphic_on_compose holomorphic_intros)
      apply (simp_all flip: cball_translation)
    apply (simp add: add.commute)
    done
  subgoal for x h r
    apply (rule_tac x=h o (λu. u−z) in exI)
    apply (rule_tac x=r in exI)
    apply (intro conjI holomorphic_on_compose holomorphic_intros)
      apply (simp_all add: flip: cball_translation_subtract)
    by (metis diff_add_cancel eq_iff_diff_eq_0 norm_minus_commute)
  done

lemma zorder_shift': NO_MATCH 0 z ⟹ zorder f z = zorder (λu. f (u + z)) 0
  by (rule zorder_shift)

lemma
  fixes f::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in (at z). f w≠0
    shows zorder_inverse: zorder (λw. inverse (f w)) z = − zorder f z
      and zor_poly_inverse: ∀ F w in (at z). zor_poly (λw. inverse (f w)) z w
                                        = inverse (zor_poly f z w)
proof −
  define vf where vf = (λw. inverse (f w))
  define fn vfn where
    fn = zorder f z  and vfn = zorder vf z
```

define *fp vfp* **where**
fp = *zor_poly f z* **and** *vfp* = *zor_poly vf z*

obtain *fr* **where** [*simp*]:*fp z* ≠ *0* **and** *fr* > *0*
and *fr*: *fp holomorphic_on cball z fr*
∀ *w*∈*cball z fr* − {*z*}. *f w* = *fp w* ∗ (*w* − *z*) *powi fn* ∧ *fp w* ≠ *0*
using *zorder_exist*[*OF f_iso f_ness f_nconst*,*folded fn_def fp_def*]
by *auto*
have *fr_inverse*: *vf w* = (*inverse* (*fp w*)) ∗ (*w* − *z*) *powi* (−*fn*)
and *fr_nz*: *inverse* (*fp w*) ≠ *0*
when *w*∈*ball z fr* − {*z*} **for** *w*
proof −
have *f w* = *fp w* ∗ (*w* − *z*) *powi fn fp w*≠*0*
using *fr*(*2*)[*rule_format*,*of w*] *that* **by** *auto*
then show *vf w* = (*inverse* (*fp w*)) ∗ (*w* − *z*) *powi* (−*fn*) *inverse* (*fp w*)≠*0*
by (*simp_all add*: *power_int_minus vf_def*)
qed
obtain *vfr* **where** [*simp*]:*vfp z* ≠ *0* **and** *vfr*>*0* **and** *vfr*:*vfp holomorphic_on cball z vfr*
(∀ *w*∈*cball z vfr* − {*z*}. *vf w* = *vfp w* ∗ (*w* − *z*) *powi vfn* ∧ *vfp w* ≠ *0*)
proof −
have *isolated_singularity_at vf z*
using *isolated_singularity_at_inverse*[*OF f_iso f_ness*] **unfolding** *vf_def* .
moreover have *not_essential vf z*
using *not_essential_inverse*[*OF f_ness f_iso*] **unfolding** *vf_def* .
moreover have ∃ *F w in at z*. *vf w* ≠ *0*
using *f_nconst* **unfolding** *vf_def* **by** (*auto elim*:*frequently_elim1*)
ultimately show *?thesis* **using** *zorder_exist*[*of vf z*, *folded vfn_def vfp_def*] *that* **by** *auto*
qed

define *r1* **where** *r1* = *min fr vfr*
have *r1*>*0* **using** ‹*fr*>*0*› ‹*vfr*>*0*› **unfolding** *r1_def* **by** *simp*
show *vfn* = − *fn*
proof (*rule holomorphic_factor_unique*)
have §: ⋀*w*. ⟦*fp w* = *0*; *dist z w* < *fr*⟧ ⟹ *False*
using *fr_nz* **by** *force*
then show ∀ *w*∈*ball z r1* − {*z*}.
vf w = *vfp w* ∗ (*w* − *z*) *powi vfn* ∧
vfp w ≠ *0* ∧ *vf w* = *inverse* (*fp w*) ∗ (*w* − *z*) *powi* (− *fn*) ∧
inverse (*fp w*) ≠ *0*
using *fr_inverse r1_def vfr*(*2*)
by (*smt* (*verit*) *Diff_iff inverse_nonzero_iff_nonzero mem_ball mem_cball*)
show *vfp holomorphic_on ball z r1*
using *r1_def vfr*(*1*) **by** *auto*
show (*λw*. *inverse* (*fp w*)) *holomorphic_on ball z r1*
by (*metis* § *ball_subset_cball fr*(*1*) *holomorphic_on_inverse holomorphic_on_subset mem_ball min.cobounded2 min.commute r1_def subset_ball*)

```
  qed (use ‹r1>0› in auto)
  have vfp w = inverse (fp w) when w∈ball z r1−{z} for w
  proof −
    have w ∈ ball z fr − {z} w ∈ cball z vfr − {z}  w≠z using that unfolding
r1_def by auto
   from fr_inverse[OF this(1)] fr_nz[OF this(1)] vfr(2)[rule_format,OF this(2)]
‹vfn = − fn› ‹w≠z›
    show ?thesis by auto
  qed
  then show ∀ F w in (at z). vfp w = inverse (fp w)
    unfolding eventually_at using ‹r1>0›
    by (metis DiffI dist_commute mem_ball singletonD)
qed

lemma zor_poly_shift:
  assumes iso1: isolated_singularity_at f z
    and ness1: not_essential f z
    and nzero1: ∃ F w in at z. f w ≠ 0
  shows ∀ F w in nhds z. zor_poly f z w = zor_poly (λu. f (z + u)) 0 (w−z)
proof −
  obtain r1 where r1>0 zor_poly f z z ≠ 0 and
      holo1:zor_poly f z holomorphic_on cball z r1 and
      rball1:∀ w∈cball z r1 − {z}.
          f w = zor_poly f z w ∗ (w − z) powi (zorder f z) ∧
          zor_poly f z w ≠ 0
    using zorder_exist[OF iso1 ness1 nzero1] by blast

  define ff where ff=(λu. f (z + u))
  have isolated_singularity_at ff 0
    unfolding ff_def
    using iso1 isolated_singularity_at_shift_iff[of f 0 z]
    by (simp add:algebra_simps)
  moreover have not_essential ff 0
    unfolding ff_def
    using ness1 not_essential_shift_iff[of f 0 z]
    by (simp add:algebra_simps)
  moreover have ∃ F w in at 0. ff w ≠ 0
    unfolding ff_def using nzero1
    by (smt (verit, ccfv_SIG) add.commute eventually_at_to_0
       eventually_mono not_frequently)
  ultimately obtain r2 where r2>0 zor_poly ff 0 0 ≠ 0 and
      holo2:zor_poly ff 0 holomorphic_on cball 0 r2 and
      rball2:∀ w∈cball 0 r2 − {0}.
          ff w = zor_poly ff 0 w ∗ w powi (zorder ff 0) ∧
          zor_poly ff 0 w ≠ 0
    using zorder_exist[of ff 0] by auto

  define r where r=min r1 r2
  have r>0 using ‹r1>0› ‹r2>0› unfolding r_def by auto
```

```
have zor_poly f z w = zor_poly ff 0 (w − z)
  if w∈ball z r − {z} for w
proof −
  define n where n ≡ zorder f z

  have f w = zor_poly f z w ∗ (w − z) powi n
  proof −
    have w∈cball z r1 − {z}
      using r_def that by auto
    from rball1[rule_format, OF this]
    show ?thesis unfolding n_def by auto
  qed

  moreover have f w = zor_poly ff 0 (w − z) ∗ (w − z) powi n
  proof −
    have w−z∈cball 0 r2 − {0}
      using r_def that by (auto simp:dist_complex_def)
    from rball2[rule_format, OF this]
    have ff (w − z) = zor_poly ff 0 (w − z) ∗ (w − z)
                      powi (zorder ff 0)
      by simp
    moreover have of_int (zorder ff 0) = n
      unfolding n_def ff_def by (simp add:zorder_shift' add.commute)
    ultimately show ?thesis unfolding ff_def by auto
  qed
  ultimately have zor_poly f z w ∗ (w − z) powi n
            = zor_poly ff 0 (w − z) ∗ (w − z) powi n
    by auto
  moreover have (w − z) powi n ≠0
    using that by auto
  ultimately show ?thesis
    using mult_cancel_right by blast
qed
then have ∀F w in at z. zor_poly f z w
              = zor_poly ff 0 (w − z)
  unfolding eventually_at
  apply (rule_tac x=r in exI)
  using ‹r>0› by (auto simp:dist_commute)
moreover have isCont (zor_poly f z) z
  using holo1[THEN holomorphic_on_imp_continuous_on]
  apply (elim continuous_on_interior)
  using ‹r1>0› by auto
moreover have isCont (λw. zor_poly ff 0 (w − z)) z
proof −
  have isCont (zor_poly ff 0) 0
    using holo2[THEN holomorphic_on_imp_continuous_on]
    apply (elim continuous_on_interior)
    using ‹r2>0› by auto
```

```
    then show ?thesis
      unfolding isCont_iff by simp
  qed
  ultimately show ∀F w in nhds z. zor_poly f z w
                = zor_poly ff 0 (w − z)
    by (elim at_within_isCont_imp_nhds;auto)
qed

lemma
  fixes f g::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
      and f_ness:not_essential f z and g_ness:not_essential g z
      and fg_nconst: ∃F w in (at z). f w * g w≠ 0
  shows zorder_times:zorder (λw. f w * g w) z = zorder f z + zorder g z and
        zor_poly_times:∀F w in (at z). zor_poly (λw. f w * g w) z w
                                     = zor_poly f z w *zor_poly g z w
proof −
  define fg where fg = (λw. f w * g w)
  define fn gn fgn where
    fn = zorder f z and gn = zorder g z and fgn = zorder fg z
  define fp gp fgp where
    fp = zor_poly f z and gp = zor_poly g z and fgp = zor_poly fg z
  have f_nconst:∃F w in (at z). f w ≠ 0 and g_nconst:∃F w in (at z).g w≠ 0
    using fg_nconst by (auto elim!:frequently_elim1)
  obtain fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
              ∀ w∈cball z fr − {z}. f w = fp w * (w − z) powi fn ∧ fp w ≠ 0
    using zorder_exist[OF f_iso f_ness f_nconst,folded fp_def fn_def] by auto
  obtain gr where [simp]:gp z ≠ 0 and gr > 0
        and gr: gp holomorphic_on cball z gr
              ∀ w∈cball z gr − {z}. g w = gp w * (w − z) powi gn ∧ gp w ≠ 0
    using zorder_exist[OF g_iso g_ness g_nconst,folded gn_def gp_def] by auto
  define r1 where r1=min fr gr
  have r1>0 unfolding r1_def using ‹fr>0› ‹gr>0› by auto
  have fg_times:fg w = (fp w * gp w) * (w − z) powi (fn+gn) and fgp_nz:fp w*gp
w≠0
    when w∈ball z r1 − {z} for w
  proof −
    have f w = fp w * (w − z) powi fn fp w≠0
      using fr(2)[rule_format,of w] that unfolding r1_def by auto
    moreover have g w = gp w * (w − z) powi gn gp w ≠ 0
      using gr(2)[rule_format, of w] that unfolding r1_def by auto
    ultimately show fg w = (fp w * gp w) * (w − z) powi (fn+gn) fp w*gp w≠0
      using that
      unfolding fg_def by (auto simp add:power_int_add)
  qed

  obtain fgr where [simp]:fgp z ≠ 0 and fgr > 0
```

```
        and fgr: fgp holomorphic_on cball z fgr
              ∀ w∈cball z fgr − {z}. fg w = fgp w ∗ (w − z) powi fgn ∧ fgp w ≠ 0
  proof −
    have fgp z ≠ 0 ∧ (∃ r>0. fgp holomorphic_on cball z r
          ∧ (∀ w∈cball z r − {z}. fg w = fgp w ∗ (w − z) powi fgn ∧ fgp w ≠ 0))
      apply (rule zorder_exist[of fg z, folded fgn_def fgp_def])
       subgoal unfolding fg_def using isolated_singularity_at_times[OF f_iso
g_iso] .
       subgoal unfolding fg_def using not_essential_times[OF f_ness g_ness
f_iso g_iso] .
      subgoal unfolding fg_def using fg_nconst .
      done
    then show ?thesis using that by blast
  qed
  define r2 where r2 = min fgr r1
  have r2>0 using ‹r1>0› ‹fgr>0› unfolding r2_def by simp
  show fgn = fn + gn
    apply (rule holomorphic_factor_unique[of r2 fgp z λw. fp w ∗ gp w fg])
    subgoal using ‹r2>0› by simp
    subgoal by simp
    subgoal by simp
    subgoal
    proof (rule ballI)
      fix w assume w ∈ ball z r2 − {z}
     then have w ∈ ball z r1 − {z} w ∈ cball z fgr − {z}  unfolding r2_def by
auto
     from fg_times[OF this(1)] fgp_nz[OF this(1)] fgr(2)[rule_format,OF this(2)]
      show fg w = fgp w ∗ (w − z) powi fgn ∧ fgp w ≠ 0
            ∧ fg w = fp w ∗ gp w ∗ (w − z) powi (fn + gn) ∧ fp w ∗ gp w ≠ 0 by
auto
    qed
   subgoal using fgr(1) unfolding r2_def r1_def by (auto intro!:holomorphic_intros)
   subgoal using fr(1) gr(1) unfolding r2_def r1_def by (auto intro!:holomorphic_intros)
    done

  have fgp w = fp w ∗gp w when w∈ball z r2−{z} for w
  proof −
    have w ∈ ball z r1 − {z} w ∈ cball z fgr − {z} w≠z using that  unfolding
r2_def by auto
   from fg_times[OF this(1)] fgp_nz[OF this(1)] fgr(2)[rule_format,OF this(2)]
‹fgn = fn + gn› ‹w≠z›
    show ?thesis by auto
  qed
  then show ∀ F w in (at z). fgp w = fp w ∗ gp w
    using ‹r2>0› unfolding eventually_at by (auto simp add:dist_commute)
qed

lemma
  fixes f g::complex ⇒ complex and z::complex
```

assumes *f_iso*:*isolated_singularity_at f z* **and** *g_iso*:*isolated_singularity_at g z*
and *f_ness*:*not_essential f z* **and** *g_ness*:*not_essential g z*
and *fg_nconst*: $\exists_F w$ *in* (*at z*). *f w* $*$ *g w*$\neq$ *0*
shows *zorder_divide*:*zorder* (λw. *f w* / *g w*) *z* = *zorder f z* − *zorder g z* **and**
zor_poly_divide:$\forall_F w$ *in* (*at z*). *zor_poly* (λw. *f w* / *g w*) *z w*
= *zor_poly f z w* / *zor_poly g z w*
proof −
have *f_nconst*:$\exists_F w$ *in* (*at z*). *f w* ≠ *0* **and** *g_nconst*:$\exists_F w$ *in* (*at z*).*g w*$\neq$ *0*
using *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
define *vg* **where** *vg*=(λw. *inverse* (*g w*))
have *zorder* (λw. *f w* $*$ *vg w*) *z* = *zorder f z* + *zorder vg z*
apply (*rule zorder_times*[*OF f_iso _ f_ness*,*of vg*])
subgoal unfolding *vg_def* **using** *isolated_singularity_at_inverse*[*OF g_iso g_ness*] .
subgoal unfolding *vg_def* **using** *not_essential_inverse*[*OF g_ness g_iso*] .
subgoal unfolding *vg_def* **using** *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
done
then show *zorder* (λw. *f w* / *g w*) *z* = *zorder f z* − *zorder g z*
using *zorder_inverse*[*OF g_iso g_ness g_nconst*,*folded vg_def*] **unfolding** *vg_def*
by (*auto simp add*:*field_simps*)

have $\forall_F$ *w in at z*. *zor_poly* (λw. *f w* $*$ *vg w*) *z w* = *zor_poly f z w* $*$ *zor_poly vg z w*
apply (*rule zor_poly_times*[*OF f_iso _ f_ness*,*of vg*])
subgoal unfolding *vg_def* **using** *isolated_singularity_at_inverse*[*OF g_iso g_ness*] .
subgoal unfolding *vg_def* **using** *not_essential_inverse*[*OF g_ness g_iso*] .
subgoal unfolding *vg_def* **using** *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
done
then show $\forall_F w$ *in* (*at z*). *zor_poly* (λw. *f w* / *g w*) *z w* = *zor_poly f z w* / *zor_poly g z w*
using *zor_poly_inverse*[*OF g_iso g_ness g_nconst*,*folded vg_def*] **unfolding** *vg_def*
by *eventually_elim* (*auto simp add*:*field_simps*)
qed

lemma *zorder_exist_zero*:
fixes *f*::*complex* ⇒ *complex* **and** *z*::*complex*
defines *n*≡*zorder f z* **and** *g*≡*zor_poly f z*
assumes *holo*: *f holomorphic_on s* **and**
open s connected s z∈*s*
and *non_const*: ∃ *w*∈*s*. *f w* ≠ *0*
shows (*if f z*=*0 then n* > *0 else n*=*0*) ∧ (∃ *r*. *r*>*0* ∧ *cball z r* ⊆ *s* ∧ *g holomorphic_on cball z r*
∧ (∀ *w*∈*cball z r*. *f w* = *g w* $*$ (*w*−*z*) ^ *nat n* ∧ *g w* ≠*0*))
proof −
obtain *r* **where** *g z* ≠ *0* **and** *r*: *r*>*0 cball z r* ⊆ *s g holomorphic_on cball z r*

```
        (∀ w∈cball z r − {z}. f w = g w * (w − z) powi n ∧ g w ≠ 0)
  proof −
    have g z ≠ 0 ∧ (∃ r>0. g holomorphic_on cball z r
          ∧ (∀ w∈cball z r − {z}. f w = g w * (w − z) powi n ∧ g w ≠ 0))
    proof (rule zorder_exist[of f z,folded g_def n_def])
      show isolated_singularity_at f z unfolding isolated_singularity_at_def
        using holo assms(4,6)
          by (meson Diff_subset open_ball analytic_on_holomorphic holomor-
phic_on_subset openE)
      show not_essential f z unfolding not_essential_def
      using assms(4,6) at_within_open continuous_on holo holomorphic_on_imp_continuous_on
        by fastforce
      have ∀ F w in at z. f w ≠ 0 ∧ w∈s
      proof −
        obtain w where w∈s f w≠0 using non_const by auto
        then show ?thesis
          by (rule non_zero_neighbour_alt[OF holo ‹open s› ‹connected s› ‹z∈s›])
      qed
      then show ∃ F w in at z. f w ≠ 0
        apply (elim eventually_frequentlyE)
        by auto
    qed
    then obtain r1 where g z ≠ 0 r1>0 and r1:g holomorphic_on cball z r1
          (∀ w∈cball z r1 − {z}. f w = g w * (w − z) powi n ∧ g w ≠ 0)
      by auto
    obtain r2 where r2: r2>0 cball z r2 ⊆ s
      using assms(4,6) open_contains_cball_eq by blast
    define r3 where r3=min r1 r2
    have r3>0 cball z r3 ⊆ s using ‹r1>0› r2 unfolding r3_def by auto
    moreover have g holomorphic_on cball z r3
      using r1(1) unfolding r3_def by auto
    moreover have (∀ w∈cball z r3 − {z}. f w = g w * (w − z) powi n ∧ g w ≠
0)
      using r1(2) unfolding r3_def by auto
    ultimately show ?thesis using that[of r3] ‹g z≠0› by auto
  qed

  have fz_lim: f− z → f z
   by (metis assms(4,6) at_within_open continuous_on holo holomorphic_on_imp_continuous_on)
  have gz_lim: g −z→g z
    using r
   by (meson Elementary_Metric_Spaces.open_ball analytic_at analytic_at_imp_isCont

       ball_subset_cball centre_in_ball holomorphic_on_subset isContD)
  have if_0:if f z=0 then n > 0 else n=0
  proof −
    have (λw. g w * (w − z) powi n) −z→ f z
      using fz_lim Lim_transform_within_open[where s=ball z r] r by fastforce
    then have (λw. (g w * (w − z) powi n) / g w) −z→ f z/g z
```

```
    using gz_lim ‹g z ≠ 0› tendsto_divide by blast
  then have powi_tendsto:(λw. (w − z) powi n) −z→ f z/g z
    using Lim_transform_within_open[where s=ball z r] r by fastforce

  have ?thesis when n≥0 f z=0
  proof −
    have (λw. (w − z) ^ nat n) −z→ f z/g z
      using Lim_transform_within[OF powi_tendsto, where d=r]
      by (meson power_int_def r(1) that(1))
    then have ∗:(λw. (w − z) ^ nat n) −z→ 0 using ‹f z=0› by simp
    moreover have False when n=0
    proof −
      have (λw. (w − z) ^ nat n) −z→ 1
        using ‹n=0› by auto
      then show False using ∗ using LIM_unique zero_neq_one by blast
    qed
    ultimately show ?thesis using that by fastforce
  qed
  moreover have ?thesis when n≥0 f z≠0
  proof −
    have False when n>0
    proof −
      have (λw. (w − z) ^ nat n) −z→ f z/g z
        using Lim_transform_within[OF powi_tendsto, where d=r]
        by (meson ‹0 ≤ n› power_int_def r(1))
      moreover have (λw. (w − z) ^ nat n) −z→ 0
        using ‹n>0› by (auto intro!:tendsto_eq_intros)
        ultimately show False using ‹f z≠0› ‹g z≠0› using LIM_unique di-
vide_eq_0_iff by blast
    qed
    then show ?thesis using that by force
  qed
  moreover have False when n<0
  proof −
    have (λw. inverse ((w − z) ^ nat (− n))) −z→ f z/g z
     by (smt (verit) LIM_cong power_int_def power_inverse powi_tendsto that)
    moreover
    have (λw.((w − z) ^ nat (− n))) −z→ 0
      using that by (auto intro!:tendsto_eq_intros)
    ultimately
    have (λx. inverse ((x − z) ^ nat (− n)) ∗ (x − z) ^ nat (− n)) −z→ 0
      using tendsto_mult by fastforce
    then have (λx. 1::complex) −z→ 0
      by (elim Lim_transform_within_open[where s=UNIV],auto)
    then show False using LIM_const_eq by fastforce
  qed
  ultimately show ?thesis by fastforce
 qed
 moreover have f w = g w ∗ (w−z) ^ nat n ∧ g w ≠0 when w∈cball z r for w
```

```
  proof (cases w=z)
    case True
    then have f −z→f w
      using fz_lim by blast
    then have (λw. g w * (w−z) ^ nat n) −z→f w
    proof (elim Lim_transform_within[OF _ ‹r>0›])
      fix x assume 0 < dist x z dist x z < r
      then have x ∈ cball z r − {z} x≠z
        unfolding cball_def by (auto simp add: dist_commute)
      then have f x = g x * (x − z) powi n
        using r(4)[rule_format,of x] by simp
      also have ... = g x * (x − z) ^ nat n
        by (smt (verit, best) if_0 int_nat_eq power_int_of_nat)
      finally show f x = g x * (x − z) ^ nat n .
    qed
    moreover have (λw. g w * (w−z) ^ nat n) −z→ g w * (w−z) ^ nat n
      using True by (auto intro!:tendsto_eq_intros gz_lim)
    ultimately have f w = g w * (w−z) ^ nat n using LIM_unique by blast
    then show ?thesis using ‹g z≠0› True by auto
  next
    case False
    then have f w = g w * (w − z) powi n ∧ g w ≠ 0
      using r(4) that by auto
    then show ?thesis
      by (smt (verit, best) False if_0 int_nat_eq power_int_of_nat)
  qed
  ultimately show ?thesis using r by auto
qed

lemma zorder_exist_pole:
  fixes f::complex ⇒ complex and z::complex
  defines n≡zorder f z and g≡zor_poly f z
  assumes  holo: f holomorphic_on s−{z} and
        open s z∈s
      and is_pole f z
  shows n < 0 ∧ g z≠0 ∧ (∃r. r>0 ∧ cball z r ⊆ s ∧ g holomorphic_on cball z r
    ∧ (∀w∈cball z r − {z}. f w  = g w / (w−z) ^ nat (− n) ∧ g w ≠0))
proof −
  obtain r where g z ≠ 0 and r: r>0 cball z r ⊆ s g holomorphic_on cball z r
          (∀w∈cball z r − {z}. f w = g w * (w − z) powi n ∧ g w ≠ 0)
  proof −
    have g z ≠ 0 ∧ (∃r>0. g holomorphic_on cball z r
          ∧ (∀w∈cball z r − {z}. f w = g w * (w − z) powi n ∧ g w ≠ 0))
    proof (rule zorder_exist[of f z,folded g_def n_def])
      show isolated_singularity_at f z unfolding isolated_singularity_at_def
        using holo assms(4,5)
          by (metis analytic_on_holomorphic centre_in_ball insert_Diff openE
open_delete subset_insert_iff)
      show not_essential f z unfolding not_essential_def
```

```
    using assms(4,6) at_within_open continuous_on holo holomorphic_on_imp_continuous_on
      by fastforce
    from non_zero_neighbour_pole[OF ‹is_pole f z›] show ∃F w in at z. f w ≠
0
      apply (elim eventually_frequentlyE)
      by auto
  qed
  then obtain r1 where g z ≠ 0 r1>0 and r1:g holomorphic_on cball z r1
        (∀w∈cball z r1 − {z}. f w = g w ∗ (w − z) powi n ∧ g w ≠ 0)
    by auto
  obtain r2 where r2: r2>0 cball z r2 ⊆ s
    using assms(4,5) open_contains_cball_eq by metis
  define r3 where r3=min r1 r2
  have r3>0 cball z r3 ⊆ s using ‹r1>0› r2 unfolding r3_def by auto
  moreover have g holomorphic_on cball z r3
    using r1(1) unfolding r3_def by auto
  moreover have (∀w∈cball z r3 − {z}. f w = g w ∗ (w − z) powi n ∧ g w ≠
0)
    using r1(2) unfolding r3_def by auto
  ultimately show ?thesis using that[of r3] ‹g z≠0› by auto
 qed

 have n<0
 proof (rule ccontr)
  assume ¬ n < 0
  define c where c=(if n=0 then g z else 0)
  have [simp]:g −z→ g z
    by (metis open_ball at_within_open ball_subset_cball centre_in_ball
      continuous_on holomorphic_on_imp_continuous_on holomorphic_on_subset
r(1) r(3) )
  have ∀F x in at z. f x = g x ∗ (x − z) ^ nat n
    unfolding eventually_at_topological
    apply (rule_tac exI[where x=ball z r])
    by (simp add: ‹¬ n < 0› linorder_not_le power_int_def r(1) r(4))
  moreover have (λx. g x ∗ (x − z) ^ nat n) −z→ c
  proof (cases n=0)
    case True
    then show ?thesis unfolding c_def by simp
  next
    case False
    then have (λx. (x − z) ^ nat n) −z→ 0 using ‹¬ n < 0›
      by (auto intro!:tendsto_eq_intros)
    from tendsto_mult[OF _ this,of g g z,simplified]
    show ?thesis unfolding c_def using False by simp
  qed
  ultimately have f −z→c using tendsto_cong by fast
 then show False using ‹is_pole f z› at_neq_bot not_tendsto_and_filterlim_at_infinity
    unfolding is_pole_def by blast
 qed
```

moreover have $\forall\, w{\in}cball\ z\ r - \{z\}.\ f\ w\ = g\ w\ /\ (w{-}z)\ \hat{}\ nat\ (-\ n) \wedge g\ w \neq 0$
using *r(4)* ‹*n<0*›
by (*smt* (*verit*) *inverse_eq_divide mult.right_neutral power_int_def power_inverse times_divide_eq_right*)
ultimately show *?thesis* **using** *r(1−3)* ‹*g z≠0*› **by** *auto*
qed

lemma *zorder_eqI*:
assumes *open s z* ∈ *s g holomorphic_on s g z* ≠ *0*
assumes *fg_eq*:⋀*w*. ⟦*w* ∈ *s*;*w*≠*z*⟧ ⟹ *f w* = *g w* ∗ (*w* − *z*) *powi n*
shows *zorder f z* = *n*
proof −
have *continuous_on s g* **by** (*rule holomorphic_on_imp_continuous_on*) *fact*
moreover have *open* (−{*0*::*complex*}) **by** *auto*
ultimately have *open* ((*g* −‘ (−{*0*})) ∩ *s*)
unfolding *continuous_on_open_vimage*[*OF* ‹*open s*›] **by** *blast*
moreover from *assms* **have** *z* ∈ (*g* −‘ (−{*0*})) ∩ *s* **by** *auto*
ultimately obtain *r* **where** *r*: *r* > *0 cball z r* ⊆ *s* ∩ (*g* −‘ (−{*0*}))
unfolding *open_contains_cball* **by** *blast*

let *?gg*= (λ*w*. *g w* ∗ (*w* − *z*) *powi n*)
define *P* **where** *P* = (λ*n g r*. *0* < *r* ∧ *g holomorphic_on cball z r* ∧ *g z*≠*0*
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *g w* ∗ (*w*−*z*) *powi n* ∧ *g w*≠*0*))
have *P n g r*
unfolding *P_def* **using** *r assms*(*3*,*4*,*5*) **by** *auto*
then have ∃ *g r*. *P n g r* **by** *auto*
moreover have *unique*: ∃!*n*. ∃ *g r*. *P n g r* **unfolding** *P_def*
proof (*rule holomorphic_factor_puncture*)
have *ball z r*−{*z*} ⊆ *s* **using** *r* **using** *ball_subset_cball* **by** *blast*
then have *?gg holomorphic_on ball z r*−{*z*}
using ‹*g holomorphic_on s*› *r* **by** (*auto intro*!: *holomorphic_intros*)
then have *f holomorphic_on ball z r* − {*z*}
by (*smt* (*verit*, *best*) *DiffD2* ‹*ball z r*−{*z*} ⊆ *s*› *fg_eq holomorphic_cong singleton_iff subset_iff*)
then show *isolated_singularity_at f z* **unfolding** *isolated_singularity_at_def*
using *analytic_on_open open_delete r(1)* **by** *blast*
next
have *not_essential ?gg z*
proof (*intro singularity_intros*)
show *not_essential g z*
by (*meson* ‹*continuous_on s g*› *assms continuous_on_eq_continuous_at isCont_def not_essential_def*)
show ∀*F w in at z*. *w* − *z* ≠ *0* **by** (*simp add*: *eventually_at_filter*)
then show *LIM w at z*. *w* − *z* :> *at 0*
unfolding *filterlim_at* **by** (*auto intro*:*tendsto_eq_intros*)
show *isolated_singularity_at g z*
by (*meson Diff_subset open_ball analytic_on_holomorphic assms holomorphic_on_subset isolated_singularity_at_def openE*)
qed

```
    moreover
    have ∀F w in at z. g w * (w − z) powi n = f w
      unfolding eventually_at_topological using assms fg_eq by force
    ultimately show not_essential f z
      using not_essential_transform by blast
    show ∃F w in at z. f w ≠ 0 unfolding frequently_at
    proof (intro strip)
      fix d::real assume 0 < d
      define z' where z' ≡ z+min d r / 2
      have z' ≠ z  dist z' z < d
        unfolding z'_def using ‹d>0› ‹r>0› by (auto simp add:dist_norm)
      moreover have f z' ≠ 0
      proof (subst fg_eq[OF _ ‹z'≠z›])
        have z' ∈ cball z r
          unfolding z'_def using ‹r>0› ‹d>0› by (auto simp add:dist_norm)
        then show  z' ∈ s using r(2) by blast
        show g z' * (z' − z) powi n ≠ 0
          using P_def ‹P n g r› ‹z' ∈ cball z r› ‹z' ≠ z› by auto
      qed
      ultimately show ∃x∈UNIV. x ≠ z ∧ dist x z < d ∧ f x ≠ 0 by auto
    qed
  qed
  ultimately have (THE n. ∃ g r. P n g r) = n
    by (rule_tac the1_equality)
  then show ?thesis unfolding zorder_def P_def by blast
qed

lemma simple_zeroI:
  assumes open s z ∈ s g holomorphic_on s g z ≠ 0
  assumes ⋀w. w ∈ s ⟹ f w = g w * (w − z)
  shows   zorder f z = 1
  using assms zorder_eqI by force

lemma higher_deriv_power:
  shows   (deriv ^^ j) (λw. (w − z) ^ n) w =
            pochhammer (of_nat (Suc n − j)) j * (w − z) ^ (n − j)
proof (induction j arbitrary: w)
  case 0
  thus ?case by auto
next
  case (Suc j w)
  have (deriv ^^ Suc j) (λw. (w − z) ^ n) w = deriv ((deriv ^^ j) (λw. (w − z)
^ n)) w
    by simp
  also have (deriv ^^ j) (λw. (w − z) ^ n) =
              (λw. pochhammer (of_nat (Suc n − j)) j * (w − z) ^ (n − j))
    using Suc by (intro Suc.IH ext)
  also {
    have (... has_field_derivative of_nat (n − j) *
```

```
            pochhammer (of_nat (Suc n − j)) j ∗ (w − z) ^(n − Suc j)) (at w)
    using Suc.prems by (auto intro!: derivative_eq_intros)
  also have of_nat (n − j) ∗ pochhammer (of_nat (Suc n − j)) j =
            pochhammer (of_nat (Suc n − Suc j)) (Suc j)
    by (cases Suc j ≤ n, subst pochhammer_rec)
       (insert Suc.prems, simp_all add: algebra_simps Suc_diff_le pochhammer_0_left)
  finally have deriv (λw. pochhammer (of_nat (Suc n − j)) j ∗ (w − z) ^(n − j)) w =
                 ... ∗ (w − z) ^(n − Suc j)
    by (rule DERIV_imp_deriv)
 }
 finally show ?case .
qed

lemma zorder_zero_eqI:
 assumes f_holo:f holomorphic_on s and open s z ∈ s
 assumes zero: ⋀i. i < nat n ⟹ (deriv ^^ i) f z = 0
 assumes nz: (deriv ^^ nat n) f z ≠ 0 and n≥0
 shows   zorder f z = n
proof −
 obtain r where [simp]:r>0 and ball z r ⊆ s
   using ‹open s› ‹z∈s› openE by blast
 have nz':∃ w∈ball z r. f w ≠ 0
 proof (rule ccontr)
   assume ¬ (∃ w∈ball z r. f w ≠ 0)
   then have eventually (λu. f u = 0) (nhds z)
     using open_ball ‹0 < r› centre_in_ball eventually_nhds by blast
   then have (deriv ^^ nat n) f z = (deriv ^^ nat n) (λ_. 0) z
     by (intro higher_deriv_cong_ev) auto
   also have (deriv ^^ nat n) (λ_. 0) z = 0
     by (induction n) simp_all
   finally show False using nz by contradiction
 qed

 define zn g where zn = zorder f z and g = zor_poly f z
 obtain e where e_if:if f z = 0 then 0 < zn else zn = 0 and
         [simp]:e>0 and cball z e ⊆ ball z r and
         g_holo:g holomorphic_on cball z e and
         e_fac:(∀ w∈cball z e. f w = g w ∗ (w − z) ^ nat zn ∧ g w ≠ 0)
 proof −
   have f holomorphic_on ball z r
     using f_holo ‹ball z r ⊆ s› by auto
   from that zorder_exist_zero[of f ball z r z,simplified,OF this nz',folded zn_def
g_def]
   show ?thesis by blast
 qed
 then obtain zn ≥ 0 g z≠0
   by (metis centre_in_cball less_le_not_le order_refl)
```

```
  define A where A ≡ (λi. of_nat (i choose (nat zn)) * fact (nat zn) * (deriv ^^
(i - nat zn)) g z)
  have deriv_A:(deriv ^^ i) f z = (if zn ≤ int i then A i else 0) for i
  proof -
    have eventually (λw. w ∈ ball z e) (nhds z)
      using ‹cball z e ⊆ ball z r› ‹e>0› by (intro eventually_nhds_in_open) auto
    hence eventually (λw. f w = (w - z) ^ (nat zn) * g w) (nhds z)
      using e_fac eventually_mono by fastforce
    hence (deriv ^^ i) f z = (deriv ^^ i) (λw. (w - z) ^ nat zn * g w) z
      by (intro higher_deriv_cong_ev) auto
    also have ... = (∑j=0..i. of_nat (i choose j) *
                    (deriv ^^ j) (λw. (w - z) ^ nat zn) z * (deriv ^^ (i - j)) g z)
      using g_holo ‹e>0›
      by (intro higher_deriv_mult[of _ ball z e]) (auto intro!: holomorphic_intros)
    also have ... = (∑j=0..i. if j = nat zn then
                  of_nat (i choose nat zn) * fact (nat zn) * (deriv ^^ (i - nat zn))
g z else 0)
    proof (intro sum.cong refl, goal_cases)
      case (1 j)
      have (deriv ^^ j) (λw. (w - z) ^ nat zn) z =
            pochhammer (of_nat (Suc (nat zn) - j)) j * 0 ^ (nat zn - j)
        by (subst higher_deriv_power) auto
      also have ... = (if j = nat zn then fact j else 0)
        by (auto simp: not_less pochhammer_0_left pochhammer_fact)
      also have of_nat (i choose j) * ... * (deriv ^^ (i - j)) g z =
                  (if j = nat zn then of_nat (i choose (nat zn)) * fact (nat zn)
                      * (deriv ^^ (i - nat zn)) g z else 0)
        by simp
      finally show ?case .
    qed
    also have ... = (if i ≥ zn then A i else 0)
      by (auto simp: A_def)
    finally show (deriv ^^ i) f z = ... .
  qed

  have False when n<zn
  proof -
    have (deriv ^^ nat n) f z = 0
      using deriv_A[of nat n] that ‹n≥0› by auto
    with nz show False by auto
  qed
  moreover have n≤zn
  proof -
    have g z ≠ 0 using e_fac[rule_format,of z] ‹e>0› by simp
    then have (deriv ^^ nat zn) f z ≠ 0
      using deriv_A[of nat zn] by(auto simp add:A_def)
    then have nat zn ≥ nat n using zero[of nat zn] by linarith
    moreover have zn≥0 using e_if by (auto split:if_splits)
```

```
    ultimately show ?thesis using nat_le_eq_zle by blast
  qed
  ultimately show ?thesis unfolding zn_def by fastforce
qed

lemma
  assumes eventually (λz. f z = g z) (at z) z = z'
  shows zorder_cong:zorder f z = zorder g z' and zor_poly_cong:zor_poly f z =
zor_poly g z'
proof −
  define P where P = (λff n h r. 0 < r ∧ h holomorphic_on cball z r ∧ h z≠0
                ∧ (∀ w∈cball z r − {z}. ff w = h w ∗ (w−z) powi n ∧ h w≠0))
  have (∃ r. P f n h r) = (∃ r. P g n h r) for n h
  proof −
    have ∗: ∃ r. P g n h r if ∃ r. P f n h r and eventually (λx. f x = g x) (at z)
for f g
    proof −
      from that(1) obtain r1 where r1_P:P f n h r1 by auto
      from that(2) obtain r2 where r2>0 and r2_dist:∀ x. x ≠ z ∧ dist x z ≤
r2 ⟶ f x = g x
        unfolding eventually_at_le by auto
      define r where r=min r1 r2
      have r>0 h z≠0 using r1_P ‹r2>0› unfolding r_def P_def by auto
      moreover have h holomorphic_on cball z r
        using r1_P unfolding P_def r_def by auto
      moreover have g w = h w ∗ (w − z) powi n ∧ h w ≠ 0 when w∈cball z r
− {z} for w
      proof −
        have f w = h w ∗ (w − z) powi n ∧ h w ≠ 0
          using r1_P that unfolding P_def r_def by auto
        moreover have f w=g w using r2_dist[rule_format,of w] that unfolding
r_def
          by (simp add: dist_commute)
        ultimately show ?thesis by simp
      qed
      ultimately show ?thesis unfolding P_def by auto
    qed
    from assms have eq': eventually (λz. g z = f z) (at z)
      by (simp add: eq_commute)
    show ?thesis
      by (rule iffI[OF ∗[OF _ assms(1)] ∗[OF _ eq']])
  qed
  then show zorder f z = zorder g z' zor_poly f z = zor_poly g z'
      using ‹z=z'› unfolding P_def zorder_def zor_poly_def by auto
qed

lemma zorder_times_analytic':
  assumes isolated_singularity_at f z not_essential f z
  assumes g analytic_on {z} frequently (λz. f z ∗ g z ≠ 0) (at z)
```

shows *zorder* $(\lambda x.\ f\ x * g\ x)\ z = zorder\ f\ z + zorder\ g\ z$
proof (*rule zorder_times*)
show *isolated_singularity_at g z not_essential g z*
by (*intro isolated_singularity_at_analytic not_essential_analytic assms*)+
qed (*use assms* **in** *auto*)

lemma *zorder_cmult*:
assumes $c \neq 0$
shows *zorder* $(\lambda z.\ c * f\ z)\ z = zorder\ f\ z$
proof −
define P **where**
$P = (\lambda f\ n\ h\ r.\ 0 < r \wedge h\ holomorphic\_on\ cball\ z\ r \wedge$
$h\ z \neq 0 \wedge (\forall w \in cball\ z\ r - \{z\}.\ f\ w = h\ w * (w - z)\ powi\ n \wedge h\ w \neq 0))$
have $*$: $P\ (\lambda x.\ c * f\ x)\ n\ (\lambda x.\ c * h\ x)\ r$ **if** $P\ f\ n\ h\ r\ c \neq 0$ **for** $f\ n\ h\ r\ c$
using *that* **unfolding** P_def **by** (*auto intro*!: *holomorphic_intros*)
have $(\exists h\ r.\ P\ (\lambda x.\ c * f\ x)\ n\ h\ r) \longleftrightarrow (\exists h\ r.\ P\ f\ n\ h\ r)$ **for** n
using $*[of\ f\ n\ \_\ \_\ c]\ *[of\ \lambda x.\ c * f\ x\ n\ \_\ \_\ inverse\ c]\ \langle c \neq 0 \rangle$
by (*fastforce simp*: *field_simps*)
hence $(THE\ n.\ \exists h\ r.\ P\ (\lambda x.\ c * f\ x)\ n\ h\ r) = (THE\ n.\ \exists h\ r.\ P\ f\ n\ h\ r)$
by *simp*
thus *?thesis*
by (*simp add*: *zorder_def P_def*)
qed

lemma *zorder_nonzero_div_power*:
assumes *sz*: *open s z* $\in s$ *f holomorphic_on s f z* $\neq 0$ **and** $n > 0$
shows *zorder* $(\lambda w.\ f\ w\ /\ (w - z)\ \hat{}\ n)\ z = -\ n$
by (*intro zorder_eqI* [*OF sz*]) (*simp add*: *inverse_eq_divide power_int_minus*)

lemma *zor_poly_eq*:
assumes *isolated_singularity_at f z not_essential f z* $\exists_F\ w\ in\ at\ z.\ f\ w \neq 0$
shows *eventually* $(\lambda w.\ zor\_poly\ f\ z\ w = f\ w * (w - z)\ powi - zorder\ f\ z)\ (at\ z)$
proof −
obtain r **where** $r{:}r{>}0$
$(\forall w \in cball\ z\ r - \{z\}.\ f\ w = zor\_poly\ f\ z\ w * (w - z)\ powi\ (zorder\ f\ z))$
using *zorder_exist*[*OF assms*] **by** *blast*
then have $*$: $\forall w \in ball\ z\ r - \{z\}.\ zor\_poly\ f\ z\ w = f\ w * (w - z)\ powi - zorder\ f\ z$
by (*auto simp*: *field_simps power_int_minus*)
have *eventually* $(\lambda w.\ w \in ball\ z\ r - \{z\})\ (at\ z)$
using *r eventually_at_ball'*[*of r z UNIV*] **by** *auto*
thus *?thesis* **by** *eventually_elim* (*insert* $*$, *auto*)
qed

lemma *zor_poly_zero_eq*:
assumes *f holomorphic_on s open s connected s z* $\in s\ \exists w \in s.\ f\ w \neq 0$
shows *eventually* $(\lambda w.\ zor\_poly\ f\ z\ w = f\ w\ /\ (w - z)\ \hat{}\ nat\ (zorder\ f\ z))\ (at\ z)$
proof −

obtain r **where** $r{:}r{>}0$
$(\forall w{\in}cball\ z\ r - \{z\}.\ f\ w = zor\_poly\ f\ z\ w * (w - z)\ \hat{}\ nat\ (zorder\ f\ z))$
using *zorder_exist_zero*[*OF assms*] **by** *auto*
then have $*{:}\ \forall w{\in}ball\ z\ r - \{z\}.\ zor\_poly\ f\ z\ w = f\ w\ /\ (w - z)\ \hat{}\ nat\ (zorder\ f\ z)$
by (*auto simp*: *field_simps powr_minus*)
have *eventually* $(\lambda w.\ w \in ball\ z\ r - \{z\})\ (at\ z)$
using r *eventually_at_ball'*[*of r z UNIV*] **by** *auto*
thus *?thesis* **by** *eventually_elim* (*insert* $*$, *auto*)
qed

lemma *zor_poly_pole_eq*:
assumes *f_iso*:*isolated_singularity_at f z is_pole f z*
shows *eventually* $(\lambda w.\ zor\_poly\ f\ z\ w = f\ w * (w - z)\ \hat{}\ nat\ (-\ zorder\ f\ z))\ (at\ z)$
proof −
obtain e **where** [*simp*]:$e{>}0$ **and** *f_holo*:*f holomorphic_on ball z e − {z}*
using *f_iso analytic_imp_holomorphic* **unfolding** *isolated_singularity_at_def*
by *blast*
obtain r **where** $r{:}r{>}0$
$(\forall w{\in}cball\ z\ r - \{z\}.\ f\ w = zor\_poly\ f\ z\ w\ /\ (w - z)\ \hat{}\ nat\ (-\ zorder\ f\ z))$
using *zorder_exist_pole*[*OF f_holo*,*simplified*,*OF* ‹*is_pole f z*›] **by** *auto*
then have $*{:}\ \forall w{\in}ball\ z\ r - \{z\}.\ zor\_poly\ f\ z\ w = f\ w * (w - z)\ \hat{}\ nat\ (-\ zorder\ f\ z)$
by (*auto simp*: *field_simps*)
have *eventually* $(\lambda w.\ w \in ball\ z\ r - \{z\})\ (at\ z)$
using r *eventually_at_ball'*[*of r z UNIV*] **by** *auto*
thus *?thesis* **by** *eventually_elim* (*insert* $*$, *auto*)
qed

lemma *zor_poly_eqI*:
fixes $f :: complex \Rightarrow complex$ **and** $z0 :: complex$
defines $n \equiv zorder\ f\ z0$
assumes *isolated_singularity_at f z0 not_essential f z0* $\exists_F\ w\ in\ at\ z0.\ f\ w \neq 0$
assumes *lim*: $((\lambda x.\ f\ (g\ x) * (g\ x - z0)\ powi\ -\ n) \longrightarrow c)\ F$
assumes g: *filterlim g* $(at\ z0)\ F$ **and** $F \neq bot$
shows $zor\_poly\ f\ z0\ z0 = c$
proof −
from *zorder_exist*[*OF assms*(2−4)] **obtain** r **where**
$r{:}\ r > 0\ zor\_poly\ f\ z0\ holomorphic\_on\ cball\ z0\ r$
$\bigwedge w.\ w \in cball\ z0\ r - \{z0\} \Longrightarrow f\ w = zor\_poly\ f\ z0\ w * (w - z0)\ powi\ n$
unfolding *n_def* **by** *blast*
from $r(1)$ **have** *eventually* $(\lambda w.\ w \in ball\ z0\ r \wedge w \neq z0)\ (at\ z0)$
using *eventually_at_ball'*[*of r z0 UNIV*] **by** *auto*
hence *eventually* $(\lambda w.\ zor\_poly\ f\ z0\ w = f\ w * (w - z0)\ powi\ -\ n)\ (at\ z0)$
by *eventually_elim* (*insert r*, *auto simp*: *field_simps power_int_minus*)
moreover have *continuous_on* $(ball\ z0\ r)\ (zor\_poly\ f\ z0)$
using r **by** (*intro holomorphic_on_imp_continuous_on*) *auto*
with $r(1,2)$ **have** *isCont* $(zor\_poly\ f\ z0)\ z0$

```
    by (auto simp: continuous_on_eq_continuous_at)
  hence (zor_poly f z0 ⟶ zor_poly f z0 z0) (at z0)
    unfolding isCont_def .
  ultimately have ((λw. f w * (w - z0) powi - n) ⟶ zor_poly f z0 z0) (at
z0)
    by (blast intro: Lim_transform_eventually)
  hence ((λx. f (g x) * (g x - z0) powi - n) ⟶ zor_poly f z0 z0) F
    by (rule filterlim_compose[OF _ g])
  from tendsto_unique[OF ‹F ≠ bot› this lim] show ?thesis .
qed

lemma zor_poly_zero_eqI:
  fixes f :: complex ⇒ complex and z0 :: complex
  defines n ≡ zorder f z0
  assumes f holomorphic_on A open A connected A z0 ∈ A ∃ z∈A. f z ≠ 0
  assumes lim: ((λx. f (g x) / (g x - z0) ^ nat n) ⟶ c) F
  assumes g: filterlim g (at z0) F and F ≠ bot
  shows   zor_poly f z0 z0 = c
proof -
  from zorder_exist_zero[OF assms(2-6)] obtain r where
    r: r > 0 cball z0 r ⊆ A zor_poly f z0 holomorphic_on cball z0 r
       ⋀w. w ∈ cball z0 r ⟹ f w = zor_poly f z0 w * (w - z0) ^ nat n
    unfolding n_def by blast
  from r(1) have eventually (λw. w ∈ ball z0 r ∧ w ≠ z0) (at z0)
    using eventually_at_ball'[of r z0 UNIV] by auto
  hence eventually (λw. zor_poly f z0 w = f w / (w - z0) ^ nat n) (at z0)
    by eventually_elim (insert r, auto simp: field_simps)
  moreover have continuous_on (ball z0 r) (zor_poly f z0)
    using r by (intro holomorphic_on_imp_continuous_on) auto
  with r(1,2) have isCont (zor_poly f z0) z0
    by (auto simp: continuous_on_eq_continuous_at)
  hence (zor_poly f z0 ⟶ zor_poly f z0 z0) (at z0)
    unfolding isCont_def .
  ultimately have ((λw. f w / (w - z0) ^ nat n) ⟶ zor_poly f z0 z0) (at z0)
    by (blast intro: Lim_transform_eventually)
  hence ((λx. f (g x) / (g x - z0) ^ nat n) ⟶ zor_poly f z0 z0) F
    by (rule filterlim_compose[OF _ g])
  from tendsto_unique[OF ‹F ≠ bot› this lim] show ?thesis .
qed

lemma zor_poly_pole_eqI:
  fixes f :: complex ⇒ complex and z0 :: complex
  defines n ≡ zorder f z0
  assumes f_iso:isolated_singularity_at f z0 and is_pole f z0
  assumes lim: ((λx. f (g x) * (g x - z0) ^ nat (-n)) ⟶ c) F
  assumes g: filterlim g (at z0) F and F ≠ bot
  shows   zor_poly f z0 z0 = c
proof -
  obtain r where r: r > 0  zor_poly f z0 holomorphic_on cball z0 r
```

```
  proof −
    have ∃F w in at z0. f w ≠ 0
     using non_zero_neighbour_pole[OF ‹is_pole f z0›] by (auto elim:eventually_frequentlyE)
   moreover have not_essential f z0 unfolding not_essential_def using ‹is_pole
f z0› by simp
    ultimately show ?thesis using that zorder_exist[OF f_iso,folded n_def] by
auto
  qed
  from r(1) have eventually (λw. w ∈ ball z0 r ∧ w ≠ z0) (at z0)
    using eventually_at_ball'[of r z0 UNIV] by auto
  have eventually (λw. zor_poly f z0 w = f w * (w − z0) ^ nat (−n)) (at z0)
    using zor_poly_pole_eq[OF f_iso ‹is_pole f z0›] unfolding n_def .
  moreover have continuous_on (ball z0 r) (zor_poly f z0)
    using r by (intro holomorphic_on_imp_continuous_on) auto
  with r(1,2) have isCont (zor_poly f z0) z0
    by (auto simp: continuous_on_eq_continuous_at)
  hence (zor_poly f z0 ⟶ zor_poly f z0 z0) (at z0)
    unfolding isCont_def .
  ultimately have ((λw. f w * (w − z0) ^ nat (−n)) ⟶ zor_poly f z0 z0) (at
z0)
    by (blast intro: Lim_transform_eventually)
  hence ((λx. f (g x) * (g x − z0) ^ nat (−n)) ⟶ zor_poly f z0 z0) F
    by (rule filterlim_compose[OF _ g])
  from tendsto_unique[OF ‹F ≠ bot› this lim] show ?thesis .
qed

lemma
  assumes is_pole f (x :: complex) open A x ∈ A
  assumes ⋀y. y ∈ A − {x} ⟹ (f has_field_derivative f' y) (at y)
  shows   is_pole_deriv': is_pole f' x
    and   zorder_deriv':  zorder f' x = zorder f x − 1
proof −
  have holo: f holomorphic_on A − {x}
    using assms by (subst holomorphic_on_open) auto
  obtain r where r: r > 0 ball x r ⊆ A
    using assms(2,3) openE by blast
  moreover have open (ball x r − {x})
    by auto
  ultimately have isolated_singularity_at f x
    by (auto simp: isolated_singularity_at_def analytic_on_open
          intro!: exI[of _ r] holomorphic_on_subset[OF holo])
  hence ev: ∀F w in at x. zor_poly f x w = f w * (w − x) ^ nat (− zorder f x)
    using ‹is_pole f x› zor_poly_pole_eq by blast

  define P where P = zor_poly f x
  define n where n = nat (−zorder f x)

  obtain r where r: r > 0 cball x r ⊆ A P holomorphic_on cball x r zorder f x
< 0 P x ≠ 0
```

```
    ∀w∈cball x r − {x}. f w = P w / (w − x) ^ n ∧ P w ≠ 0
     unfolding P_def n_def using zorder_exist_pole[OF holo assms(2,3,1)] by
blast
  have n: n > 0
    using r(4) by (auto simp: n_def)

  have [derivative_intros]: (P has_field_derivative deriv P w) (at w)
    if w ∈ ball x r for w
   using that by (intro holomorphic_derivI[OF holomorphic_on_subset[OF r(3),
of ball x r]]) auto

  define D where D = (λw. (deriv P w * (w − x) − of_nat n * P w) / (w − x)
^ (n + 1))
  define n' where n' = n − 1
  have n': n = Suc n'
    using n by (simp add: n'_def)

  have eventually (λw. w ∈ ball x r) (nhds x)
    using ‹r > 0› by (intro eventually_nhds_in_open) auto
  hence ev'': eventually (λw. w ∈ ball x r − {x}) (at x)
    by (auto simp: eventually_at_filter elim: eventually_mono)

  {
    fix w assume w: w ∈ ball x r − {x}
    have ev': eventually (λw. w ∈ ball x r − {x}) (nhds w)
      using w by (intro eventually_nhds_in_open) auto

    have ((λw. P w / (w − x) ^ n) has_field_derivative D w) (at w)
      apply (rule derivative_eq_intros refl | use w in force)+
      using w
      apply (simp add: divide_simps D_def)
      apply (simp add: n' algebra_simps)
      done
    also have ?this ⟷ (f has_field_derivative D w) (at w)
      using r by (intro has_field_derivative_cong_ev refl eventually_mono[OF
ev']) auto
    finally have (f has_field_derivative D w) (at w) .
    moreover have (f has_field_derivative f' w) (at w)
      using w r by (intro assms) auto
    ultimately have D w = f' w
      using DERIV_unique by blast
  } note D_eq = this

  have is_pole D x
    unfolding D_def using n ‹r > 0› ‹P x ≠ 0›
     by (intro is_pole_basic[where A = ball x r] holomorphic_intros holomor-
phic_on_subset[OF r(3)]) auto
  also have ?this ⟷ is_pole f' x
    by (intro is_pole_cong eventually_mono[OF ev''] D_eq) auto
```

```
  finally show is_pole f' x .

  have zorder f' x = -int (Suc n)
  proof (rule zorder_eqI)
    show open (ball x r) x ∈ ball x r
      using ‹r > 0› by auto
    show f' w = (deriv P w * (w - x) - of_nat n * P w) * (w - x) powi (- int
(Suc n))
      if w ∈ ball x r w ≠ x for w
    using that D_eq[of w] n by (auto simp: D_def power_int_diff power_int_minus
powr_nat' divide_simps)
  qed (use r n in ‹auto intro!: holomorphic_intros›)
  thus zorder f' x = zorder f x - 1
    using n by (simp add: n_def)
qed

lemma
  assumes is_pole f (x :: complex) isolated_singularity_at f x
  shows   is_pole_deriv: is_pole (deriv f) x
    and   zorder_deriv:  zorder (deriv f) x = zorder f x - 1
proof -
  from assms(2) obtain r where r: r > 0 f analytic_on ball x r - {x}
    by (auto simp: isolated_singularity_at_def)
  hence holo: f holomorphic_on ball x r - {x}
    by (subst (asm) analytic_on_open) auto
  have *: x ∈ ball x r open (ball x r) open (ball x r - {x})
    using ‹r > 0› by auto
  show is_pole (deriv f) x zorder (deriv f) x = zorder f x - 1
    by (rule is_pole_deriv' zorder_deriv', (rule assms * holomorphic_derivI holo
| assumption)+)+
qed

lemma removable_singularity_deriv':
  assumes f -x→ c x ∈ A open (A :: complex set)
  assumes ⋀y. y ∈ A - {x} ⟹ (f has_field_derivative f' y) (at y)
  shows   ∃c. f' -x→ c
proof -
  have holo: f holomorphic_on A - {x}
    using assms by (subst holomorphic_on_open) auto

  define g where g = (λy. if y = x then c else f y)
  have deriv_g_eq: deriv g y = f' y if y ∈ A - {x} for y
  proof -
    have ev: eventually (λy. y ∈ A - {x}) (nhds y)
      using that assms by (intro eventually_nhds_in_open) auto
    have (f has_field_derivative f' y) (at y)
      using assms that by auto
    also have ?this ⟷ (g has_field_derivative f' y) (at y)
      by (intro has_field_derivative_cong_ev refl eventually_mono[OF ev]) (auto
```

```
simp: g_def)
    finally show ?thesis
      by (intro DERIV_imp_deriv assms)
  qed

  have g holomorphic_on A
    unfolding g_def using assms assms(1) holo by (intro removable_singularity)
auto
  hence deriv g holomorphic_on A
    by (intro holomorphic_deriv assms)
  hence continuous_on A (deriv g)
    by (meson holomorphic_on_imp_continuous_on)
  hence (deriv g ⟶ deriv g x) (at x within A)
    using assms by (auto simp: continuous_on_def)
  also have ?this ⟷ (f' ⟶ deriv g x) (at x within A)
    by (intro filterlim_cong refl) (auto simp: eventually_at_filter deriv_g_eq)
  finally have f' −x→ deriv g x
    using ‹open A› ‹x ∈ A› by (meson tendsto_within_open)
  thus ?thesis
    by blast
qed

lemma removable_singularity_deriv:
  assumes f −x→ c isolated_singularity_at f x
  shows   ∃c. deriv f −x→ c
proof −
  from assms(2) obtain r where r: r > 0 f analytic_on ball x r − {x}
    by (auto simp: isolated_singularity_at_def)
  hence holo: f holomorphic_on ball x r − {x}
    using analytic_imp_holomorphic by blast
  show ?thesis
    using assms(1)
  proof (rule removable_singularity_deriv')
    show x ∈ ball x r open (ball x r)
      using ‹r > 0› by auto
  qed (auto intro!: holomorphic_derivI[OF holo])
qed

lemma not_essential_deriv':
  assumes not_essential f x x ∈ A open A
  assumes ⋀y. y ∈ A − {x} ⟹ (f has_field_derivative f' y) (at y)
  shows   not_essential f' x
proof −
  have holo: f holomorphic_on A − {x}
    using assms by (subst holomorphic_on_open) auto
  from assms consider is_pole f x | c where f −x→ c
    by (auto simp: not_essential_def)
  thus ?thesis
  proof cases
```

```
    case 1
    hence is_pole f' x
      using is_pole_deriv' assms by blast
    thus ?thesis by (auto simp: not_essential_def)
  next
    case (2 c)
    from 2 have ∃ c. f' −x→ c
      by (rule removable_singularity_deriv'[OF _ assms(2−4)])
    thus ?thesis
      by (auto simp: not_essential_def)
  qed
qed

lemma not_essential_deriv[singularity_intros]:
  assumes not_essential f x isolated_singularity_at f x
  shows   not_essential (deriv f) x
proof −
  from assms(2) obtain r where r: r > 0 f analytic_on ball x r − {x}
    by (auto simp: isolated_singularity_at_def)
  hence holo: f holomorphic_on ball x r − {x}
    by (subst (asm) analytic_on_open) auto
  show ?thesis
    using assms(1)
  proof (rule not_essential_deriv')
    show x ∈ ball x r open (ball x r)
      using ‹r > 0› by auto
  qed (auto intro!: holomorphic_derivI[OF holo])
qed

lemma not_essential_frequently_0_imp_tendsto_0:
  fixes f :: complex ⇒ complex
  assumes sing: isolated_singularity_at f z not_essential f z
  assumes freq: frequently (λz. f z = 0) (at z)
  shows   f −z→ 0
proof −
  from freq obtain g :: nat ⇒ complex where g: filterlim g (at z) at_top ⋀n. f
(g n) = 0
    using frequently_atE by blast
  have eventually (λx. f (g x) = 0) sequentially
    using g by auto
  hence fg: (λx. f (g x)) ⟶ 0
    by (simp add: tendsto_eventually)

  from assms(2) consider c where f −z→ c | is_pole f z
    unfolding not_essential_def by blast
  thus ?thesis
  proof cases
    case (1 c)
    have (λx. f (g x)) ⟶ c
```

```
    by (rule filterlim_compose[OF 1 g(1)])
  with fg have c = 0
    using LIMSEQ_unique by blast
  with 1 show ?thesis by simp
 next
  case 2
  have filterlim (λx. f (g x)) at_infinity sequentially
    by (rule filterlim_compose[OF _ g(1)]) (use 2 in ‹auto simp: is_pole_def›)
  with fg have False
    by (meson not_tendsto_and_filterlim_at_infinity sequentially_bot)
  thus ?thesis ..
 qed
qed

lemma not_essential_frequently_0_imp_eventually_0:
 fixes f :: complex ⇒ complex
 assumes sing: isolated_singularity_at f z not_essential f z
 assumes freq: frequently (λz. f z = 0) (at z)
 shows   eventually (λz. f z = 0) (at z)
proof −
 from sing obtain r where r: r > 0 and f analytic_on ball z r − {z}
   by (auto simp: isolated_singularity_at_def)
 hence holo: f holomorphic_on ball z r − {z}
   by (subst (asm) analytic_on_open) auto
 have eventually (λw. w ∈ ball z r − {z}) (at z)
   using r by (intro eventually_at_in_open) auto
 from freq and this have frequently (λw. f w = 0 ∧ w ∈ ball z r − {z}) (at z)
   using frequently_eventually_frequently by blast
 hence frequently (λw. w ∈ {w∈ball z r − {z}. f w = 0}) (at z)
   by (simp add: conj_commute)
 hence limpt: z islimpt {w∈ball z r − {z}. f w = 0}
   using islimpt_conv_frequently_at by blast

 define g where g = (λw. if w = z then 0 else f w)
 have f −z→ 0
   by (intro not_essential_frequently_0_imp_tendsto_0 assms)
 hence g_holo: g holomorphic_on ball z r
   unfolding g_def by (intro removable_singularity holo) auto

 have g_eq_0: g w = 0 if w ∈ ball z r for w
 proof (rule analytic_continuation[where f = g])
   show open (ball z r) connected (ball z r)
     using r by auto
   show z islimpt {w∈ball z r − {z}. f w = 0}
     by fact
   show g w = 0 if w ∈ {w ∈ ball z r − {z}. f w = 0} for w
     using that by (auto simp: g_def)
 qed (use r that g_holo in auto)
```

```
  have eventually (λw. w ∈ ball z r - {z}) (at z)
    using r by (intro eventually_at_in_open) auto
  thus eventually (λw. f w = 0) (at z)
  proof eventually_elim
    case (elim w)
    thus ?case using g_eq_0[of w]
      by (auto simp: g_def)
  qed
qed

lemma pole_imp_not_constant:
  fixes f :: 'a :: {perfect_space} ⇒ _
  assumes is_pole f x open A x ∈ A A ⊆ insert x B
  shows   ¬f constant_on B
proof
  assume ∗: f constant_on B
  then obtain c where c: ∀x∈B. f x = c
    by (auto simp: constant_on_def)
  have eventually (λy. y ∈ A - {x}) (at x)
    using assms by (intro eventually_at_in_open) auto
  hence eventually (λy. f y = c) (at x)
    by eventually_elim (use c assms in auto)
  hence ∗∗: f -x→ c
    by (simp add: tendsto_eventually)
  show False
   using not_tendsto_and_filterlim_at_infinity[OF _ ∗∗ assms(1)[unfolded is_pole_def]]
by simp
qed


lemma neg_zorder_imp_is_pole:
  assumes iso:isolated_singularity_at f z and f_ness:not_essential f z
      and zorder f z < 0 and fre_nz:∃F w in at z. f w ≠ 0
    shows is_pole f z
proof -
  define P where P = zor_poly f z
  define n where n = zorder f z
  have n<0 unfolding n_def by (simp add: assms(3))
  define nn where nn = nat (-n)

  obtain r where P z ≠ 0 r>0 and r_holo:P holomorphic_on cball z r and
      w_Pn:(∀w∈cball z r - {z}. f w = P w ∗ (w - z) powi n ∧ P w ≠ 0)
    using zorder_exist[OF iso f_ness fre_nz,folded P_def n_def] by auto

  have is_pole (λw. P w ∗ (w - z) powi n) z
    unfolding is_pole_def
  proof (rule tendsto_mult_filterlim_at_infinity)
    show P -z→ P z
      by (meson open_ball ‹0 < r› ball_subset_cball centre_in_ball
```

```
        continuous_on_eq_continuous_at continuous_on_subset
        holomorphic_on_imp_continuous_on isContD r_holo)
    show P z≠0 by (simp add: ‹P z ≠ 0›)

    have LIM x at z. inverse ((x − z) ^ nat (−n)) :> at_infinity
      apply (subst filterlim_inverse_at_iff[symmetric])
      using ‹n<0›
      by (auto intro!:tendsto_eq_intros filterlim_atI
            simp add:eventually_at_filter)
    then show LIM x at z. (x − z) powi n :> at_infinity
    proof (elim filterlim_mono_eventually)
      have inverse ((x − z) ^ nat (−n)) = (x − z) powi n
        if x≠z for x
        by (metis ‹n < 0› linorder_not_le power_int_def power_inverse)
      then show ∀F x in at z. inverse ((x − z) ^ nat (−n))
                = (x − z) powi n
        by (simp add: eventually_at_filter)
    qed auto
  qed
  moreover have ∀F w in at z. f w =  P w * (w − z) powi n
    unfolding eventually_at_le
    apply (rule exI[where x=r])
    using w_Pn ‹r>0› by (simp add: dist_commute)
  ultimately show ?thesis using is_pole_cong by fast
qed

lemma is_pole_divide_zorder:
  fixes f g::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
      and f_ness:not_essential f z and g_ness:not_essential g z
      and fg_nconst: ∃F w in (at z). f w * g w≠ 0
      and z_less:zorder f z < zorder g z
    shows is_pole (λz. f z / g z) z
proof −
  define fn gn fg where fn=zorder f z and gn=zorder g z
                    and fg=(λw. f w / g w)

  have isolated_singularity_at fg z
    unfolding fg_def using f_iso g_iso g_ness
    by (auto intro:singularity_intros)
  moreover have not_essential fg z
    unfolding fg_def using f_iso g_iso g_ness f_ness
    by (auto intro:singularity_intros)
  moreover have zorder fg z < 0
  proof −
    have zorder fg z = fn − gn
      using zorder_divide[OF f_iso g_iso f_ness g_ness
          fg_nconst,folded fn_def gn_def fg_def] .
```

```
    then show ?thesis
      using z_less by (simp add: fn_def gn_def)
  qed
  moreover have ∃\_F w in at z. fg w ≠ 0
    using fg_nconst unfolding fg_def by force
  ultimately show is_pole fg z
    using neg_zorder_imp_is_pole by auto
qed

lemma isolated_pole_imp_nzero_times:
  assumes f_iso:isolated_singularity_at f z
    and is_pole f z
  shows ∃\_F w in (at z). deriv f w * f w ≠ 0
proof (rule ccontr)
  assume ¬ (∃\_F w in at z.  deriv f w  * f w ≠ 0)
  then have ∀\_F x in at z. deriv f x * f x = 0
    unfolding not_frequently by simp
  moreover have ∀\_F w in at z. f w ≠ 0
    using non_zero_neighbour_pole[OF ‹is_pole f z›] .
  moreover have ∀\_F w in at z. deriv f w ≠ 0
    using is_pole_deriv[OF ‹is_pole f z› f_iso,THEN non_zero_neighbour_pole]
    .
  ultimately have ∀\_F w in at z. False
    apply eventually_elim
    by auto
  then show False by auto
qed

lemma isolated_pole_imp_neg_zorder:
  assumes f_iso:isolated_singularity_at f z
    and is_pole f z
  shows zorder f z<0
proof −
  obtain e where [simp]:e>0 and f_holo:f holomorphic_on ball z e − {z}
   using f_iso analytic_imp_holomorphic unfolding isolated_singularity_at_def
by blast
  show ?thesis
    using zorder_exist_pole[OF f_holo,simplified,OF ‹is_pole f z›]
    by auto
qed

lemma isolated_singularity_at_deriv[singularity_intros]:
  assumes isolated_singularity_at f x
  shows isolated_singularity_at (deriv f) x
proof −
  obtain r where r>0 f analytic_on ball x r − {x}
    using assms unfolding isolated_singularity_at_def by auto
  from analytic_deriv[OF this(2)]
  have deriv f analytic_on ball x r − {x} .
```

```
  with ‹r>0› show ?thesis
    unfolding isolated_singularity_at_def by auto
qed

lemma zorder_deriv_minus_1:
  fixes f g::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in at z. f w ≠ 0
      and f_ord:zorder f z ≠0
    shows zorder (deriv f) z = zorder f z − 1
proof −
  define P where P = zor_poly f z
  define n where n = zorder f z
  have n≠0 unfolding n_def using f_ord by auto

  obtain r where P z ≠ 0 r>0 and P_holo:P holomorphic_on cball z r
        and (∀ w∈cball z r − {z}. f w
                        = P w ∗ (w − z) powi n ∧ P w ≠ 0)
    using zorder_exist[OF f_iso f_ness f_nconst,folded P_def n_def] by auto
  from this(4)
  have f_eq:(∀ w∈cball z r − {z}. f w
                        = P w ∗ (w − z) powi n ∧ P w ≠ 0)
    using complex_powr_of_int f_ord n_def by presburger

  define D where D = (λw. (deriv P w ∗ (w − z) + of_int n ∗ P w)
                        ∗ (w − z) powi (n − 1))

  have deriv_f_eq:deriv f w = D w if w ∈ ball z r − {z} for w
  proof −
    have ev': eventually (λw. w ∈ ball z r − {z}) (nhds w)
      using that by (intro eventually_nhds_in_open) auto

    define wz where wz = w − z

    have wz ≠0 unfolding wz_def using that by auto
    moreover have (P has_field_derivative deriv P w) (at w)
      by (meson DiffD1 Elementary_Metric_Spaces.open_ball P_holo
          ball_subset_cball holomorphic_derivI holomorphic_on_subset that)
    ultimately have ((λw. P w ∗ (w − z) powi n) has_field_derivative D w) (at
w)
      unfolding D_def using that
      apply (auto intro!: derivative_eq_intros)
      apply (fold wz_def)
      by (auto simp:algebra_simps simp flip:power_int_add_1')
    also have ?this ⟷ (f has_field_derivative D w) (at w)
      using f_eq
      by (intro has_field_derivative_cong_ev refl eventually_mono[OF ev']) auto
    ultimately have (f has_field_derivative D w) (at w) by simp
```

```
    moreover have (f has_field_derivative deriv f w) (at w)
      by (metis DERIV_imp_deriv calculation)
    ultimately show ?thesis using DERIV_imp_deriv by blast
  qed

  show zorder (deriv f) z = n - 1
  proof (rule zorder_eqI)
    show open (ball z r) z ∈ ball z r
      using ‹r > 0› by auto
    define g where g=(λw. (deriv P w * (w - z) + of_int n * P w))
    show g holomorphic_on ball z r
      unfolding g_def using P_holo
      by (auto intro!:holomorphic_intros)
    show g z ≠ 0
      unfolding g_def using ‹P z ≠ 0› ‹n≠0› by auto
    show deriv f w =
        (deriv P w * (w - z) + of_int n * P w) * (w - z) powi (n - 1)
      if w ∈ ball z r w ≠ z for w
      using D_def deriv_f_eq that by blast
  qed
qed


lemma deriv_divide_is_pole: — Generalises ⟦is_pole ?f ?x; isolated_singularity_at
?f ?x⟧ ⟹ zorder (deriv ?f) ?x = zorder ?f ?x - 1
  fixes f g::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and fg_nconst: ∃\_F w in (at z). deriv f w *  f w ≠ 0
      and f_ord:zorder f z ≠0
    shows is_pole (λz. deriv f z / f z) z
proof (rule neg_zorder_imp_is_pole)
  define ff where ff=(λw. deriv f w / f w)
  show isolated_singularity_at ff z
    using f_iso f_ness unfolding ff_def
    by (auto intro:singularity_intros)
  show not_essential ff z
    unfolding ff_def using f_ness f_iso
    by (auto intro:singularity_intros)

  have zorder ff z =  zorder (deriv f) z - zorder f z
    unfolding ff_def using f_iso f_ness fg_nconst
    apply (rule_tac zorder_divide)
    by (auto intro:singularity_intros)
  moreover have zorder (deriv f) z = zorder f z - 1
  proof (rule zorder_deriv_minus_1)
    show  ∃\_F w in at z. f w ≠ 0
      using fg_nconst frequently_elim1 by fastforce
  qed (use f_iso f_ness f_ord in auto)
```

ultimately show *zorder ff z < 0* **by** *auto*

show $\exists_F$ *w in at z. ff w ≠ 0*
unfolding *ff_def* **using** *fg_nconst* **by** *auto*
qed

lemma *is_pole_deriv_divide_is_pole*:
fixes *f g*::*complex ⇒ complex* **and** *z*::*complex*
assumes *f_iso*:*isolated_singularity_at f z*
and *is_pole f z*
shows *is_pole* (*λz. deriv f z / f z*) *z*
proof (*rule deriv_divide_is_pole*[*OF f_iso*])
show *not_essential f z*
using ‹*is_pole f z*› **unfolding** *not_essential_def* **by** *auto*
show $\exists_F$ *w in at z. deriv f w * f w ≠ 0*
apply (*rule isolated_pole_imp_nzero_times*)
using *assms* **by** *auto*
show *zorder f z ≠ 0*
using *isolated_pole_imp_neg_zorder assms* **by** *fastforce*
qed

5.12 Isolated zeroes

definition *isolated_zero* :: (*complex ⇒ complex*) *⇒ complex ⇒ bool* **where**
isolated_zero f z ⟷ f z = 0 ∧ eventually (*λz. f z ≠ 0*) (*at z*)

lemma *isolated_zero_altdef*: *isolated_zero f z ⟷ f z = 0 ∧ ¬z islimpt* {*z. f z = 0*}
unfolding *isolated_zero_def eventually_at_filter eventually_nhds islimpt_def*
by *blast*

lemma *isolated_zero_mult1*:
assumes *isolated_zero f x isolated_zero g x*
shows *isolated_zero* (*λx. f x * g x*) *x*
proof −
have *eventually* (*λx. f x ≠ 0*) (*at x*) *eventually* (*λx. g x ≠ 0*) (*at x*)
using *assms* **unfolding** *isolated_zero_def* **by** *auto*
hence *eventually* (*λx. f x * g x ≠ 0*) (*at x*)
by *eventually_elim auto*
with *assms* **show** *?thesis*
by (*auto simp*: *isolated_zero_def*)
qed

lemma *isolated_zero_mult2*:
assumes *isolated_zero f x g x ≠ 0 g analytic_on* {*x*}
shows *isolated_zero* (*λx. f x * g x*) *x*
proof −
have *eventually* (*λx. f x ≠ 0*) (*at x*)
using *assms* **unfolding** *isolated_zero_def* **by** *auto*

```
  moreover have eventually (λx. g x ≠ 0) (at x)
    using analytic_at_neq_imp_eventually_neq[of g x 0] assms by auto
  ultimately have eventually (λx. f x * g x ≠ 0) (at x)
    by eventually_elim auto
  thus ?thesis
    using assms(1) by (auto simp: isolated_zero_def)
qed

lemma isolated_zero_mult3:
  assumes isolated_zero f x g x ≠ 0 g analytic_on {x}
  shows   isolated_zero (λx. g x * f x) x
  using isolated_zero_mult2[OF assms] by (simp add: mult_ac)

lemma isolated_zero_prod:
  assumes ⋀x. x ∈ I ⟹ isolated_zero (f x) z I ≠ {} finite I
  shows   isolated_zero (λy. ∏x∈I. f x y) z
 using assms(3,2,1) by (induction rule: finite_ne_induct) (auto intro: isolated_zero_mult1)

lemma non_isolated_zero':
  assumes isolated_singularity_at f z not_essential f z f z = 0 ¬isolated_zero f z
  shows   eventually (λz. f z = 0) (at z)
proof (rule not_essential_frequently_0_imp_eventually_0)
  from assms show frequently (λz. f z = 0) (at z)
    by (auto simp: frequently_def isolated_zero_def)
qed fact+

lemma non_isolated_zero:
  assumes ¬isolated_zero f z f analytic_on {z} f z = 0
  shows   eventually (λz. f z = 0) (nhds z)
proof -
  have eventually (λz. f z = 0) (at z)
    by (rule non_isolated_zero')
     (use assms in ‹auto intro: not_essential_analytic isolated_singularity_at_analytic›)
  with ‹f z = 0› show ?thesis
    unfolding eventually_at_filter by (auto elim!: eventually_mono)
qed

lemma not_essential_compose:
  assumes not_essential f (g z) g analytic_on {z}
  shows   not_essential (λx. f (g x)) z
proof (cases isolated_zero (λw. g w - g z) z)
  case False
  hence eventually (λw. g w - g z = 0) (nhds z)
    by (rule non_isolated_zero) (use assms in ‹auto intro!: analytic_intros›)
  hence not_essential (λx. f (g x)) z ⟷ not_essential (λ_. f (g z)) z
    by (intro not_essential_cong refl)
       (auto elim!: eventually_mono simp: eventually_at_filter)
  thus ?thesis
    by (simp add: not_essential_const)
```

next
case *True*
hence *ev*: *eventually* (λ*w*. *g w* ≠ *g z*) (*at z*)
by (*auto simp*: *isolated_zero_def*)
from *assms* **consider** *c* **where** *f* −*g z*→ *c* | *is_pole f* (*g z*)
by (*auto simp*: *not_essential_def*)
have *isCont g z*
by (*rule analytic_at_imp_isCont*) *fact*
hence *lim*: *g* −*z*→ *g z*
using *isContD* **by** *blast*

from *assms*(*1*) **consider** *c* **where** *f* −*g z*→ *c* | *is_pole f* (*g z*)
unfolding *not_essential_def* **by** *blast*
thus *?thesis*
proof *cases*
fix *c* **assume** *f* −*g z*→ *c*
hence (λ*x*. *f* (*g x*)) −*z*→ *c*
by (*rule filterlim_compose*) (*use lim ev* **in** ‹*auto simp*: *filterlim_at*›)
thus *?thesis*
by (*auto simp*: *not_essential_def*)
next
assume *is_pole f* (*g z*)
hence *is_pole* (λ*x*. *f* (*g x*)) *z*
by (*rule is_pole_compose*) *fact*+
thus *?thesis*
by (*auto simp*: *not_essential_def*)
qed
qed

5.13 Isolated points

definition *isolated_points_of* :: *complex set* ⇒ *complex set* **where**
isolated_points_of A = {*z*∈*A*. *eventually* (λ*w*. *w* ∉ *A*) (*at z*)}

lemma *isolated_points_of_altdef*: *isolated_points_of A* = {*z*∈*A*. ¬*z islimpt A*}
unfolding *isolated_points_of_def islimpt_def eventually_at_filter eventually_nhds*
by *blast*

lemma *isolated_points_of_empty* [*simp*]: *isolated_points_of* {} = {}
and *isolated_points_of_UNIV* [*simp*]: *isolated_points_of UNIV* = {}
by (*auto simp*: *isolated_points_of_def*)

lemma *isolated_points_of_open_is_empty* [*simp*]: *open A* ⟹ *isolated_points_of A* = {}
unfolding *isolated_points_of_altdef*
by (*simp add*: *interior_limit_point interior_open*)

lemma *isolated_points_of_subset*: *isolated_points_of A* ⊆ *A*
by (*auto simp*: *isolated_points_of_def*)

lemma *isolated_points_of_discrete*:
assumes *discrete A*
shows *isolated_points_of A = A*
using *assms* **by** (*auto simp*: *isolated_points_of_def discrete_altdef*)

lemmas *uniform_discreteI1 = uniformI1*
lemmas *uniform_discreteI2 = uniformI2*

lemma *isolated_singularity_at_compose*:
assumes *isolated_singularity_at f (g z) g analytic_on {z}*
shows *isolated_singularity_at (λx. f (g x)) z*
proof (*cases isolated_zero (λw. g w − g z) z*)
case *False*
hence *eventually (λw. g w − g z = 0) (nhds z)*
by (*rule non_isolated_zero*) (*use assms* **in** ‹*auto intro*!: *analytic_intros*›)
hence *isolated_singularity_at (λx. f (g x)) z ⟷ isolated_singularity_at (λ_. f (g z)) z*
by (*intro isolated_singularity_at_cong refl*)
(*auto elim*!: *eventually_mono simp*: *eventually_at_filter*)
thus *?thesis*
by (*simp add*: *isolated_singularity_at_const*)
next
case *True*
from *assms*(*1*) **obtain** *r* **where** *r*: *r > 0 f analytic_on ball (g z) r − {g z}*
by (*auto simp*: *isolated_singularity_at_def*)
hence *holo_f*: *f holomorphic_on ball (g z) r − {g z}*
by (*subst* (*asm*) *analytic_on_open*) *auto*
from *assms*(*2*) **obtain** *r′* **where** *r′*: *r′ > 0 g holomorphic_on ball z r′*
by (*auto simp*: *analytic_on_def*)

have *continuous_on (ball z r′) g*
using *holomorphic_on_imp_continuous_on r′* **by** *blast*
hence *isCont g z*
using *r′* **by** (*subst* (*asm*) *continuous_on_eq_continuous_at*) *auto*
hence *g −z→ g z*
using *isContD* **by** *blast*
hence *eventually (λw. g w ∈ ball (g z) r) (at z)*
using ‹*r > 0*› **unfolding** *tendsto_def* **by** *force*
moreover have *eventually (λw. g w ≠ g z) (at z)* **using** *True*
by (*auto simp*: *isolated_zero_def elim*!: *eventually_mono*)
ultimately have *eventually (λw. g w ∈ ball (g z) r − {g z}) (at z)*
by *eventually_elim auto*
then obtain *r″* **where** *r″*: *r″ > 0 ∀ w∈ball z r″−{z}. g w ∈ ball (g z) r − {g z}*
unfolding *eventually_at_filter eventually_nhds_metric ball_def*
by (*auto simp*: *dist_commute*)
have *f ∘ g holomorphic_on ball z (min r′ r″) − {z}*
proof (*rule holomorphic_on_compose_gen*)

```
    show g holomorphic_on ball z (min r' r'') - {z}
      by (rule holomorphic_on_subset[OF r'(2)]) auto
    show f holomorphic_on ball (g z) r - {g z}
      by fact
    show g ` (ball z (min r' r'') - {z}) ⊆ ball (g z) r - {g z}
      using r'' by force
  qed
  hence f ∘ g analytic_on ball z (min r' r'') - {z}
    by (subst analytic_on_open) auto
  thus ?thesis using ‹r' > 0› ‹r'' > 0›
    by (auto simp: isolated_singularity_at_def o_def intro!: exI[of _ min r' r''])
qed

lemma is_pole_power_int_0:
  assumes f analytic_on {x} isolated_zero f x n < 0
  shows   is_pole (λx. f x powi n) x
proof -
  have f -x→ f x
    using assms(1) by (simp add: analytic_at_imp_isCont isContD)
  with assms show ?thesis
    unfolding is_pole_def
   by (intro filterlim_power_int_neg_at_infinity) (auto simp: isolated_zero_def)
qed

lemma isolated_zero_imp_not_constant_on:
  assumes isolated_zero f x x ∈ A open A
  shows   ¬f constant_on A
proof
  assume f constant_on A
  then obtain c where c: ⋀x. x ∈ A ⟹ f x = c
    by (auto simp: constant_on_def)
  from assms and c[of x] have [simp]: c = 0
    by (auto simp: isolated_zero_def)
  have eventually (λx. f x ≠ 0) (at x)
    using assms by (auto simp: isolated_zero_def)
  moreover have eventually (λx. x ∈ A) (at x)
    using assms by (intro eventually_at_in_open') auto
  ultimately have eventually (λx. False) (at x)
    by eventually_elim (use c in auto)
  thus False
    by simp
qed

end
theory Complex_Residues
  imports Complex_Singularities
begin
```

5.14 Definition of residues

Wenda Li and LC Paulson (2016). A Formal Proof of Cauchy's Residue Theorem. Interactive Theorem Proving

definition *residue* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ *complex* **where**
residue f z = (*SOME int*. ∃ *e*>*0*. ∀ *ε*>*0*. *ε*<*e*
⟶ (*f has_contour_integral 2*∗*pi*∗ i ∗*int*) (*circlepath z ε*))

lemma *residue_cong*:
assumes *eq*: *eventually* (λ*z*. *f z* = *g z*) (*at z*) **and** *z* = *z′*
shows *residue f z* = *residue g z′*
proof −
from *assms* **have** *eq′*: *eventually* (λ*z*. *g z* = *f z*) (*at z*)
by (*simp add*: *eq_commute*)
let *?P* = λ*f c e*. (∀ *ε*>*0*. *ε* < *e* ⟶
(*f has_contour_integral of_real* (*2* ∗ *pi*) ∗ i ∗ *c*) (*circlepath z ε*))
have *residue f z* = *residue g z* **unfolding** *residue_def*
proof (*rule Eps_cong*)
fix *c* :: *complex*
have ∃ *e*>*0*. *?P g c e*
if ∃ *e*>*0*. *?P f c e* **and** *eventually* (λ*z*. *f z* = *g z*) (*at z*) **for** *f g*
proof −
from *that*(*1*) **obtain** *e* **where** *e*: *e* > *0* *?P f c e*
by *blast*
from *that*(*2*) **obtain** *e′* **where** *e′*: *e′* > *0* ⋀*z′*. *z′* ≠ *z* ⟹ *dist z′ z* < *e′* ⟹
f z′ = *g z′*
unfolding *eventually_at* **by** *blast*
have *?P g c* (*min e e′*)
proof (*intro allI exI impI*, *goal_cases*)
case (*1 ε*)
hence (*f has_contour_integral of_real* (*2* ∗ *pi*) ∗ i ∗ *c*) (*circlepath z ε*)
using *e*(*2*) **by** *auto*
thus *?case*
proof (*rule has_contour_integral_eq*)
fix *z′* **assume** *z′* ∈ *path_image* (*circlepath z ε*)
hence *dist z′ z* < *e′* **and** *z′* ≠ *z*
using *1* **by** (*auto simp*: *dist_commute*)
with *e′*(*2*)[*of z′*] **show** *f z′* = *g z′* **by** *simp*
qed
qed
moreover from *e* **and** *e′* **have** *min e e′* > *0* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed
from *this*[*OF* _ *eq*] **and** *this*[*OF* _ *eq′*]
show (∃ *e*>*0*. *?P f c e*) ⟷ (∃ *e*>*0*. *?P g c e*)
by *blast*
qed
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *contour_integral_circlepath_eq*:
assumes *open s* **and** *f_holo*:*f holomorphic_on* (*s*−{*z*}) **and** *0*<*e1 e1*≤*e2*
and *e2_cball*:*cball z e2* ⊆ *s*
shows
f contour_integrable_on circlepath z e1
f contour_integrable_on circlepath z e2
contour_integral (*circlepath z e2*) *f* = *contour_integral* (*circlepath z e1*) *f*
proof −
define *l* **where** *l* ≡ *linepath* (*z*+*e2*) (*z*+*e1*)
have [*simp*]:*valid_path l pathstart l*=*z*+*e2 pathfinish l*=*z*+*e1* **unfolding** *l_def*
by *auto*
have *e2*>*0* **using** ‹*e1*>*0*› ‹*e1*≤*e2*› **by** *auto*
have *zl_img*:*z*∉*path_image l*
proof
assume *z* ∈ *path_image l*
then have *e2* ≤ *cmod* (*e2* − *e1*)
using *segment_furthest_le*[*of z z*+*e2 z*+*e1 z*+*e2*,*simplified*] ‹*e1*>*0*› ‹*e2*>*0*›
unfolding *l_def*
by (*auto simp add*:*closed_segment_commute*)
thus *False* **using** ‹*e2*>*0*› ‹*e1*>*0*› ‹*e1*≤*e2*›
apply (*subst* (*asm*) *norm_of_real*)
by *auto*
qed
define *g* **where** *g* ≡ *circlepath z e2* +++ *l* +++ *reversepath* (*circlepath z e1*) +++ *reversepath l*
show [*simp*]: *f contour_integrable_on circlepath z e2 f contour_integrable_on* (*circlepath z e1*)
proof −
show *f contour_integrable_on circlepath z e2*
apply (*intro contour_integrable_continuous_circlepath*[*OF*
continuous_on_subset[*OF holomorphic_on_imp_continuous_on*[*OF f_holo*]]])
using ‹*e2*>*0*› *e2_cball* **by** *auto*
show *f contour_integrable_on* (*circlepath z e1*)
apply (*intro contour_integrable_continuous_circlepath*[*OF*
continuous_on_subset[*OF holomorphic_on_imp_continuous_on*[*OF f_holo*]]])
using ‹*e1*>*0*› ‹*e1*≤*e2*› *e2_cball* **by** *auto*
qed
have [*simp*]:*f contour_integrable_on l*
proof −
have *closed_segment* (*z* + *e2*) (*z* + *e1*) ⊆ *cball z e2* **using** ‹*e2*>*0*› ‹*e1*>*0*› ‹*e1*≤*e2*›
by (*intro closed_segment_subset*,*auto simp add*:*dist_norm*)
hence *closed_segment* (*z* + *e2*) (*z* + *e1*) ⊆ *s* − {*z*} **using** *zl_img e2_cball*
unfolding *l_def*
by *auto*
then show *f contour_integrable_on l* **unfolding** *l_def*

```
      apply (intro contour_integrable_continuous_linepath[OF
            continuous_on_subset[OF holomorphic_on_imp_continuous_on[OF
f_holo]]])
      by auto
    qed
  let ?ig=λg. contour_integral g f
  have (f has_contour_integral 0) g
    proof (rule Cauchy_theorem_global[OF _ f_holo])
      show open (s − {z}) using ‹open s› by auto
      show valid_path g unfolding g_def l_def by auto
      show pathfinish g = pathstart g unfolding g_def l_def by auto
    next
      have path_img:path_image g ⊆ cball z e2
        proof −
        have closed_segment (z + e2) (z + e1) ⊆ cball z e2 using ‹e2>0› ‹e1>0›
‹e1≤e2›
          by (intro closed_segment_subset,auto simp add:dist_norm)
        moreover have sphere z |e1| ⊆ cball z e2 using ‹e2>0› ‹e1≤e2› ‹e1>0›
by auto
        ultimately show ?thesis unfolding g_def l_def using ‹e2>0›
          by (simp add: path_image_join closed_segment_commute)
        qed
      show path_image g ⊆ s − {z}
        proof −
          have z∉path_image g using zl_img
        unfolding g_def l_def by (auto simp add: path_image_join closed_segment_commute)
          moreover note ‹cball z e2 ⊆ s› and path_img
          ultimately show ?thesis by auto
        qed
      show winding_number g w = 0 whenw ∉ s − {z} for w
        proof −
          have winding_number g w = 0 when w∉s using that e2_cball
            apply (intro winding_number_zero_outside[OF _ _ _ _ path_img])
            by (auto simp add:g_def l_def)
          moreover have winding_number g z=0
            proof −
              let ?Wz=λg. winding_number g z
                have ?Wz g = ?Wz (circlepath z e2) + ?Wz l + ?Wz (reversepath
(circlepath z e1))
                  + ?Wz (reversepath l)
                using ‹e2>0› ‹e1>0› zl_img unfolding g_def l_def
                   by (subst winding_number_join,auto simp add:path_image_join
closed_segment_commute)+
              also have ... = ?Wz (circlepath z e2) + ?Wz (reversepath (circlepath
z e1))
                using zl_img
                apply (subst (2) winding_number_reversepath)
                by (auto simp add:l_def closed_segment_commute)
              also have ... = 0
```

```
            proof −
              have ?Wz (circlepath z e2) = 1 using ‹e2>0›
                by (auto intro: winding_number_circlepath_centre)
               moreover have ?Wz (reversepath (circlepath z e1)) = −1 using
‹e1>0›
                apply (subst winding_number_reversepath)
                by (auto intro: winding_number_circlepath_centre)
              ultimately show ?thesis by auto
            qed
          finally show ?thesis .
        qed
      ultimately show ?thesis using that by auto
    qed
  qed
 then have 0 = ?ig g using contour_integral_unique by simp
 also have ... = ?ig (circlepath z e2) + ?ig l + ?ig (reversepath (circlepath z e1))
    + ?ig (reversepath l)
   unfolding g_def
   by (auto simp add:contour_integrable_reversepath_eq)
 also have ... = ?ig (circlepath z e2)  − ?ig (circlepath z e1)
   by (auto simp add:contour_integral_reversepath)
 finally show contour_integral (circlepath z e2) f = contour_integral (circlepath
z e1) f
   by simp
qed

lemma base_residue:
 assumes open s z∈s r>0 and f_holo:f holomorphic_on (s − {z})
   and r_cball:cball z r ⊆ s
 shows (f has_contour_integral 2 ∗ pi ∗ i ∗ (residue f z)) (circlepath z r)
proof −
 obtain e where e>0 and e_cball:cball z e ⊆ s
   using open_contains_cball[of s] ‹open s› ‹z∈s› by auto
 define c where c ≡ 2 ∗ pi ∗ i
 define i where i ≡ contour_integral (circlepath z e) f / c
 have (f has_contour_integral c∗i) (circlepath z ε) when ε>0 ε<e for ε
   proof −
     have contour_integral (circlepath z e) f = contour_integral (circlepath z ε) f
        f contour_integrable_on circlepath z ε
        f contour_integrable_on circlepath z e
       using ‹ε<e›
         by (intro contour_integral_circlepath_eq[OF ‹open s› f_holo ‹ε>0› _
e_cball],auto)+
     then show ?thesis unfolding i_def c_def
       by (auto intro:has_contour_integral_integral)
   qed
 then have ∃ e>0. ∀ε>0. ε<e ⟶ (f has_contour_integral c ∗ (residue f z))
(circlepath z ε)
   unfolding residue_def c_def
```

```
    apply (rule_tac someI[of _ i],intro  exI[where x=e])
    by (auto simp add:‹e>0› c_def)
  then obtain e' where e'>0
        and e'_def:∀ε>0. ε<e' ⟶ (f has_contour_integral c * (residue f z))
(circlepath z ε)
    by auto
  let ?int=λe. contour_integral (circlepath z e) f
  define  ε where ε ≡ Min {r,e'} / 2
  have ε>0 ε≤r ε<e' using ‹r>0› ‹e'>0› unfolding ε_def by auto
  have (f has_contour_integral c * (residue f z)) (circlepath z ε)
    using e'_def[rule_format,OF ‹ε>0› ‹ε<e'›] .
  then show ?thesis unfolding c_def
   using contour_integral_circlepath_eq[OF ‹open s› f_holo ‹ε>0› ‹ε≤r› r_cball]
    by (auto elim: has_contour_integral_eqpath[of _ _ circlepath z ε circlepath z
r])
qed

lemma residue_holo:
  assumes open s z ∈ s and f_holo: f holomorphic_on s
  shows residue f z = 0
proof −
  define c where c ≡ 2 * pi * i
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (f has_contour_integral c*residue f z) (circlepath z e)
    using f_holo
   by (auto intro: base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c_def])
  moreover have (f has_contour_integral 0) (circlepath z e)
    using f_holo e_cball ‹e>0›
    by (auto intro: Cauchy_theorem_convex_simple[of _ cball z e])
  ultimately have c*residue f z =0
    using has_contour_integral_unique by blast
  thus ?thesis unfolding c_def  by auto
qed

lemma residue_const:residue (λ_. c) z = 0
  by (intro residue_holo[of UNIV::complex set],auto intro:holomorphic_intros)

lemma residue_add:
  assumes open s z ∈ s and f_holo: f holomorphic_on s − {z}
     and g_holo:g holomorphic_on s − {z}
  shows residue (λz. f z + g z) z= residue f z + residue g z
proof −
  define c where c ≡ 2 * pi * i
  define fg where fg ≡ (λz. f z+g z)
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (fg has_contour_integral c * residue fg z) (circlepath z e)
    unfolding fg_def using f_holo g_holo
```

```
    apply (intro base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c_def])
    by (auto intro:holomorphic_intros)
  moreover have (fg has_contour_integral c*residue f z + c* residue g z) (circlepath
z e)
    unfolding fg_def using f_holo g_holo
     by (auto intro: has_contour_integral_add base_residue[OF ‹open s› ‹z∈s›
‹e>0› _ e_cball,folded c_def])
  ultimately have c*(residue f z + residue g z) = c * residue fg z
    using has_contour_integral_unique by (auto simp add:distrib_left)
  thus ?thesis unfolding fg_def
    by (auto simp add:c_def)
qed

lemma residue_lmul:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. c * (f z)) z= c * residue f z
proof (cases c=0)
  case True
  thus ?thesis using residue_const by auto
next
  case False
  define c' where c' ≡ 2 * pi * i
  define f' where f' ≡ (λz. c * (f z))
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (f' has_contour_integral c' * residue f' z) (circlepath z e)
    unfolding f'_def using f_holo
    apply (intro base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c'_def])
    by (auto intro:holomorphic_intros)
  moreover have (f' has_contour_integral c * (c' * residue f z)) (circlepath z e)
    unfolding f'_def using f_holo
    by (auto intro: has_contour_integral_lmul
      base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c'_def])
  ultimately have c' * residue f' z  = c * (c' * residue f z)
    using has_contour_integral_unique by auto
  thus ?thesis unfolding f'_def c'_def using False
    by (auto simp add:field_simps)
qed

lemma residue_rmul:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. (f z) * c) z= residue f z * c
using residue_lmul[OF assms,of c] by (auto simp add:algebra_simps)

lemma residue_div:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. (f z) / c) z= residue f z / c
using residue_lmul[OF assms,of 1/c] by (auto simp add:algebra_simps)
```

lemma *residue_neg*:
assumes *open s z* ∈ *s* **and** *f_holo*: *f holomorphic_on s* − {*z*}
shows *residue* (λ*z*. − (*f z*)) *z*= − *residue f z*
using *residue_lmul*[*OF assms*,*of* −*1*] **by** *auto*

lemma *residue_diff*:
assumes *open s z* ∈ *s* **and** *f_holo*: *f holomorphic_on s* − {*z*}
and *g_holo*:*g holomorphic_on s* − {*z*}
shows *residue* (λ*z*. *f z* − *g z*) *z*= *residue f z* − *residue g z*
using *residue_add*[*OF assms*(*1*,*2*,*3*),*of* λ*z*. − *g z*] *residue_neg*[*OF assms*(*1*,*2*,*4*)]
by (*auto intro*:*holomorphic_intros g_holo*)

lemma *residue_simple*:
assumes *open s z*∈*s* **and** *f_holo*:*f holomorphic_on s*
shows *residue* (λ*w*. *f w* / (*w* − *z*)) *z* = *f z*
proof −
define *c* **where** *c* ≡ *2* ∗ *pi* ∗ i
define *f*′ **where** *f*′ ≡ λ*w*. *f w* / (*w* − *z*)
obtain *e* **where** *e*>*0* **and** *e_cball*:*cball z e* ⊆ *s* **using** ‹*open s*› ‹*z*∈*s*›
using *open_contains_cball_eq* **by** *blast*
have (*f*′ *has_contour_integral c* ∗ *f z*) (*circlepath z e*)
unfolding *f*′*_def c_def* **using** ‹*e*>*0*› *f_holo e_cball*
by (*auto intro*!: *Cauchy_integral_circlepath_simple holomorphic_intros*)
moreover have (*f*′ *has_contour_integral c* ∗ *residue f*′ *z*) (*circlepath z e*)
unfolding *f*′*_def* **using** *f_holo*
apply (*intro base_residue*[*OF* ‹*open s*› ‹*z*∈*s*› ‹*e*>*0*› _ *e_cball*,*folded c_def*])
by (*auto intro*!:*holomorphic_intros*)
ultimately have *c* ∗ *f z* = *c* ∗ *residue f*′ *z*
using *has_contour_integral_unique* **by** *blast*
thus *?thesis* **unfolding** *c_def f*′*_def* **by** *auto*
qed

lemma *residue_simple*′:
assumes *s*: *open s z* ∈ *s* **and** *holo*: *f holomorphic_on* (*s* − {*z*})
and *lim*: ((λ*w*. *f w* ∗ (*w* − *z*)) ⟶ *c*) (*at z*)
shows *residue f z* = *c*
proof −
define *g* **where** *g* = (λ*w*. *if w* = *z then c else f w* ∗ (*w* − *z*))
from *holo* **have** (λ*w*. *f w* ∗ (*w* − *z*)) *holomorphic_on* (*s* − {*z*}) (**is** *?P*)
by (*force intro*: *holomorphic_intros*)
also have *?P* ⟷ *g holomorphic_on* (*s* − {*z*})
by (*intro holomorphic_cong refl*) (*simp_all add*: *g_def*)
finally have ∗: *g holomorphic_on* (*s* − {*z*}) **.**

note *lim*
also have (λ*w*. *f w* ∗ (*w* − *z*)) −*z*→ *c* ⟷ *g* −*z*→ *g z*
by (*intro filterlim_cong refl*) (*simp_all add*: *g_def* [*abs_def*] *eventually_at_filter*)
finally have ∗∗: *g* −*z*→ *g z* **.**

```
  have g_holo: g holomorphic_on s
    by (rule no_isolated_singularity'[where K = {z}])
       (insert assms * **, simp_all add: at_within_open_NO_MATCH)
  from s and this have residue (λw. g w / (w - z)) z = g z
    by (rule residue_simple)
  also have ∀F za in at z. g za / (za - z) = f za
    unfolding eventually_at by (auto intro!: exI[of _ 1] simp: field_simps g_def)
  hence residue (λw. g w / (w - z)) z = residue f z
    by (intro residue_cong refl)
  finally show ?thesis
    by (simp add: g_def)
qed

lemma residue_holomorphic_over_power:
  assumes open A z0 ∈ A f holomorphic_on A
  shows   residue (λz. f z / (z - z0) ^ Suc n) z0 = (deriv ^^ n) f z0 / fact n
proof -
  let ?f = λz. f z / (z - z0) ^ Suc n
  from assms(1,2) obtain r where r: r > 0 cball z0 r ⊆ A
    by (auto simp: open_contains_cball)
  have (?f has_contour_integral 2 * pi * i * residue ?f z0) (circlepath z0 r)
    using r assms by (intro base_residue[of A]) (auto intro!: holomorphic_intros)
  moreover have (?f has_contour_integral 2 * pi * i / fact n * (deriv ^^ n) f
z0) (circlepath z0 r)
    using assms r
    by (intro Cauchy_has_contour_integral_higher_derivative_circlepath)
     (auto intro!: holomorphic_on_subset[OF assms(3)] holomorphic_on_imp_continuous_on)
  ultimately have 2 * pi * i * residue ?f z0 = 2 * pi * i / fact n * (deriv ^^ n)
f z0
    by (rule has_contour_integral_unique)
  thus ?thesis by (simp add: field_simps)
qed

lemma residue_holomorphic_over_power':
  assumes open A 0 ∈ A f holomorphic_on A
  shows   residue (λz. f z / z ^ Suc n) 0 = (deriv ^^ n) f 0 / fact n
  using residue_holomorphic_over_power[OF assms] by simp

theorem residue_fps_expansion_over_power_at_0:
  assumes f has_fps_expansion F
  shows   residue (λz. f z / z ^ Suc n) 0 = fps_nth F n
proof -
  from has_fps_expansion_imp_holomorphic[OF assms] obtain s
    where open s 0 ∈ s f holomorphic_on s ⋀z. z ∈ s ⟹ f z = eval_fps F z
    by auto
  with assms have residue (λz. f z / (z - 0) ^ Suc n) 0 = (deriv ^^ n) f 0 / fact
n
    unfolding has_fps_expansion_def
    by (intro residue_holomorphic_over_power[of s]) (auto simp: zero_ereal_def)
```

```
  also from assms have ... = fps_nth F n
    by (subst fps_nth_fps_expansion) auto
  finally show ?thesis by simp
qed

lemma residue_pole_order:
  fixes f::complex ⇒ complex and z::complex
  defines n ≡ nat (− zorder f z) and h ≡ zor_poly f z
  assumes f_iso:isolated_singularity_at f z
    and pole:is_pole f z
  shows residue f z = ((deriv ^^ (n − 1)) h z / fact (n−1))
proof −
  define g where g ≡ λx. if x=z then 0 else inverse (f x)
  obtain e where [simp]:e>0 and f_holo:f holomorphic_on ball z e − {z}
   using f_iso analytic_imp_holomorphic unfolding isolated_singularity_at_def
by blast
  obtain r where 0 < n 0 < r and r_cball:cball z r ⊆ ball z e and h_holo: h
holomorphic_on cball z r
     and h_divide:(∀ w∈cball z r. (w≠z ⟶ f w = h w / (w − z) ^ n) ∧ h w ≠ 0)
  proof −
    obtain r where r:zorder f z < 0 h z ≠ 0 r>0 cball z r ⊆ ball z e h holomor-
phic_on cball z r
        (∀ w∈cball z r − {z}. f w = h w / (w − z) ^ n ∧ h w ≠ 0)
      using zorder_exist_pole[OF f_holo,simplified,OF ‹is_pole f z›,folded n_def
h_def] by auto
    have n>0 using ‹zorder f z < 0› unfolding n_def by simp
    moreover have (∀ w∈cball z r. (w≠z ⟶ f w = h w / (w − z) ^ n) ∧ h w ≠
0)
      using ‹h z≠0› r(6) by blast
    ultimately show ?thesis using r(3,4,5) that by blast
  qed
  have r_nonzero:⋀w. w ∈ ball z r − {z} ⟹ f w ≠ 0
    using h_divide by simp
  define c where c ≡ 2 ∗ pi ∗ i
  define der_f where der_f ≡ ((deriv ^^ (n − 1)) h z / fact (n−1))
  define h' where h' ≡ λu. h u / (u − z) ^ n
   have (h' has_contour_integral c / fact (n − 1) ∗ (deriv ^^ (n − 1)) h z)
(circlepath z r)
    unfolding h'_def
   proof (rule Cauchy_has_contour_integral_higher_derivative_circlepath[of z r
h z n−1,
       folded c_def Suc_pred'[OF ‹n>0›]])
    show continuous_on (cball z r) h using holomorphic_on_imp_continuous_on
h_holo by simp
      show h holomorphic_on ball z r using h_holo by auto
      show  z ∈ ball z r using ‹r>0› by auto
    qed
  then have (h' has_contour_integral c ∗ der_f) (circlepath z r) unfolding
der_f_def by auto
```

```
  then have (f has_contour_integral c * der_f) (circlepath z r)
    proof (elim has_contour_integral_eq)
      fix x assume x ∈ path_image (circlepath z r)
      hence x∈cball z r - {z} using ‹r>0› by auto
      then show h' x = f x using h_divide unfolding h'_def by auto
    qed
  moreover have (f has_contour_integral c * residue f z) (circlepath z r)
     using base_residue[of ‹ball z e› z,simplified,OF ‹r>0› f_holo r_cball,folded
c_def]
    unfolding c_def by simp
 ultimately have c * der_f =  c * residue f z using has_contour_integral_unique
by blast
  hence der_f = residue f z unfolding c_def by auto
  thus ?thesis unfolding der_f_def by auto
qed

lemma residue_simple_pole:
  assumes isolated_singularity_at f z0
  assumes is_pole f z0 zorder f z0 = - 1
  shows   residue f z0 = zor_poly f z0 z0
  using assms by (subst residue_pole_order) simp_all

lemma residue_simple_pole_limit:
  assumes isolated_singularity_at f z0
  assumes is_pole f z0 zorder f z0 = - 1
  assumes ((λx. f (g x) * (g x - z0)) ⟶ c) F
  assumes filterlim g (at z0) F F ≠ bot
  shows   residue f z0 = c
proof -
  have residue f z0 = zor_poly f z0 z0
    by (rule residue_simple_pole assms)+
  also have ... = c
    apply (rule zor_poly_pole_eqI)
    using assms by auto
  finally show ?thesis .
qed

lemma
  assumes f_holo:f holomorphic_on s and g_holo:g holomorphic_on s
        and open s connected s z ∈ s
  assumes g_deriv:(g has_field_derivative g') (at z)
  assumes f z ≠ 0 g z = 0 g' ≠ 0
  shows   porder_simple_pole_deriv: zorder (λw. f w / g w) z = - 1
    and   residue_simple_pole_deriv: residue (λw. f w / g w) z = f z / g'
proof -
  have [simp]:isolated_singularity_at f z isolated_singularity_at g z
   using isolated_singularity_at_holomorphic[OF _ ‹open s› ‹z∈s›] f_holo g_holo
    by (meson Diff_subset holomorphic_on_subset)+
  have [simp]:not_essential f z not_essential g z
```

```
  unfolding not_essential_def using f_holo g_holo assms(3,5)
 by (meson continuous_on_eq_continuous_at continuous_within holomorphic_on_imp_continuous_on)+
have g_nconst:∃⇩F w in at z. g w ≠0
proof (rule ccontr)
  assume ¬ (∃⇩F w in at z. g w ≠ 0)
  then have ∀⇩F w in nhds z. g w = 0
    unfolding eventually_at eventually_nhds frequently_at using ‹g z = 0›
    by (metis open_ball UNIV_I centre_in_ball dist_commute mem_ball)
  then have deriv g z = deriv (λ_. 0) z
    by (intro deriv_cong_ev) auto
  then have deriv g z = 0 by auto
  then have g' = 0 using g_deriv DERIV_imp_deriv by blast
  then show False using ‹g'≠0› by auto
qed

have zorder (λw. f w / g w) z = zorder f z − zorder g z
proof −
  have ∀⇩F w in at z. f w ≠0 ∧ w∈s
    apply (rule non_zero_neighbour_alt)
    using assms by auto
  with g_nconst have ∃⇩F w in at z. f w * g w ≠ 0
    by (elim frequently_rev_mp eventually_rev_mp,auto)
  then show ?thesis using zorder_divide[of f z g] by auto
qed
moreover have zorder f z=0
  apply (rule zorder_zero_eqI[OF f_holo ‹open s› ‹z∈s›])
  using ‹f z≠0› by auto
moreover have zorder g z=1
  apply (rule zorder_zero_eqI[OF g_holo ‹open s› ‹z∈s›])
  subgoal using assms(8) by auto
  subgoal using DERIV_imp_deriv assms(9) g_deriv by auto
  subgoal by simp
  done
ultimately show zorder (λw. f w / g w) z = − 1 by auto

show residue (λw. f w / g w) z = f z / g'
proof (rule residue_simple_pole_limit[where g=id and F=at z,simplified])
  show zorder (λw. f w / g w) z = − 1 by fact
  show isolated_singularity_at (λw. f w / g w) z
    by (auto intro: singularity_intros)
  show is_pole (λw. f w / g w) z
  proof (rule is_pole_divide)
    have ∀⇩F x in at z. g x ≠ 0
      apply (rule non_zero_neighbour)
      using g_nconst by auto
    moreover have g −z→ 0
      using DERIV_isCont assms(8) continuous_at g_deriv by force
    ultimately show filterlim g (at 0) (at z) unfolding filterlim_at by simp
    show isCont f z
```

using *assms*(*3*,*5*) *continuous_on_eq_continuous_at f_holo holomorphic_on_imp_continuous_on*
by *auto*
show $f\ z \neq 0$ **by** *fact*
qed
show *filterlim id* (*at z*) (*at z*) **by** (*simp add*: *filterlim_iff*)
have $((\lambda w.\ (f\ w * (w - z)) / g\ w) \longrightarrow f\ z / g')$ (*at z*)
proof (*rule lhopital_complex_simple*)
show $((\lambda w.\ f\ w * (w - z))$ *has_field_derivative f z*) (*at z*)
using *assms* **by** (*auto intro*!: *derivative_eq_intros holomorphic_derivI*[*OF f_holo*])
show (*g has_field_derivative g'*) (*at z*) **by** *fact*
qed (*insert assms*, *auto*)
then show $((\lambda w.\ (f\ w / g\ w) * (w - z)) \longrightarrow f\ z / g')$ (*at z*)
by (*simp add*: *field_split_simps*)
qed
qed

5.15 Poles and residues of some well-known functions

lemma *is_pole_Gamma*: *is_pole Gamma* (−*of_nat n*)
unfolding *is_pole_def* **using** *Gamma_poles* **.**

lemma *Gamma_residue*:
residue Gamma (−*of_nat n*) = (−*1*) ^ *n* / *fact n*
proof (*rule residue_simple'*)
show *open* (− ($\mathbb{Z}_{\leq 0}$ − {−*of_nat n*}) :: *complex set*)
by (*intro open_Compl closed_subset_Ints*) *auto*
show *Gamma holomorphic_on* (− ($\mathbb{Z}_{\leq 0}$ − {−*of_nat n*}) − {− *of_nat n*})
by (*rule holomorphic_Gamma*) *auto*
show (λ*w*. *Gamma w* ∗ (*w* − (−*of_nat n*))) −(−*of_nat n*)→ (− *1*) ^ *n* / *fact n*
using *Gamma_residues*[*of n*] **by** *simp*
qed *auto*

end

6 The Residue Theorem, the Argument Principle and Rouché's Theorem

theory *Residue_Theorem*
imports *Complex_Residues HOL−Library.Landau_Symbols*
begin

Several theorems that could be moved up, IF there were a previous theory importing both Landau Symbols and Elementary Metric Spaces

lemma *continuous_bounded_at_infinity_imp_bounded*:
fixes *f* :: *real* ⇒ *'a* :: *real_normed_field*
assumes *f* ∈ *O*[*at_bot*](λ_. *1*)

assumes $f \in O[at\_top](\lambda\_.\ 1)$
assumes *continuous_on UNIV f*
shows *bounded* (*range f*)
proof −
from *assms*(*1*) **obtain** *c1* **where** *eventually* ($\lambda x.$ *norm* (*f x*) ≤ *c1*) *at_bot*
by (*auto elim*!: *landau_o.bigE*)
then obtain *x1* **where** *x1*: $\bigwedge x.\ x \le x1 \Longrightarrow$ *norm* (*f x*) ≤ *c1*
by (*auto simp*: *eventually_at_bot_linorder*)
from *assms*(*2*) **obtain** *c2* **where** *eventually* ($\lambda x.$ *norm* (*f x*) ≤ *c2*) *at_top*
by (*auto elim*!: *landau_o.bigE*)
then obtain *x2* **where** *x2*: $\bigwedge x.\ x \ge x2 \Longrightarrow$ *norm* (*f x*) ≤ *c2*
by (*auto simp*: *eventually_at_top_linorder*)
have *compact* (*f* ‘ {*x1*..*x2*})
by (*intro compact_continuous_image continuous_on_subset*[*OF assms*(*3*)])
auto
hence *bounded* (*f* ‘ {*x1*..*x2*})
by (*rule compact_imp_bounded*)
then obtain *c3* **where** *c3*: $\bigwedge x.\ x \in \{x1..x2\} \Longrightarrow$ *norm* (*f x*) ≤ *c3*
unfolding *bounded_iff* **by** *fast*
have *norm* (*f x*) ≤ *Max* {*c1*, *c2*, *c3*} **for** *x*
by (*cases x* ≤ *x1*; *cases x* ≥ *x2*) (*use x1 x2 c3* **in** ‹*auto simp*: *le_max_iff_disj*›)
thus *?thesis*
unfolding *bounded_iff* **by** *blast*
qed

lemma *holomorphic_on_extend*:
assumes *f holomorphic_on S* − {ξ} ξ ∈ *interior S f* ∈ $O[at\ \xi](\lambda\_.\ 1)$
shows ($\exists g.$ *g holomorphic_on S* ∧ ($\forall z{\in}S - \{\xi\}.$ *g z* = *f z*))
by (*subst holomorphic_on_extend_bounded*) (*insert assms*, *auto elim*!: *landau_o.bigE*)

lemma *removable_singularities*:
assumes *finite X X* ⊆ *interior S f holomorphic_on* (*S* − *X*)
assumes $\bigwedge z.\ z \in X \Longrightarrow f \in O[at\ z](\lambda\_.\ 1)$
shows $\exists g.$ *g holomorphic_on S* ∧ ($\forall z{\in}S{-}X.$ *g z* = *f z*)
using *assms*
proof (*induction arbitrary*: *f rule*: *finite_induct*)
case *empty*
thus *?case* **by** *auto*
next
case (*insert z0 X f*)
from *insert.prems* **and** *insert.hyps* **have** *z0*: *z0* ∈ *interior* (*S* − *X*)
by (*auto simp*: *interior_diff finite_imp_closed*)
hence $\exists g.$ *g holomorphic_on* (*S* − *X*) ∧ ($\forall z{\in}S - X - \{z0\}.$ *g z* = *f z*)
using *insert.prems insert.hyps* **by** (*intro holomorphic_on_extend*) *auto*
then obtain *g* **where** *g*: *g holomorphic_on* (*S* − *X*) $\forall z{\in}S - X - \{z0\}.$ *g z* = *f z* **by** *blast*
have $\exists h.$ *h holomorphic_on S* ∧ ($\forall z{\in}S - X.$ *h z* = *g z*)
proof (*rule insert.IH*)
fix *z0′* **assume** *z0′*: *z0′* ∈ *X*

```
    hence eventually (λz. z ∈ interior S − (X − {z0'}) − {z0}) (nhds z0')
      using insert.prems insert.hyps
      by (intro eventually_nhds_in_open open_Diff finite_imp_closed) auto
    hence ev: eventually (λz. z ∈ S − X − {z0}) (at z0')
      unfolding eventually_at_filter
      by eventually_elim (insert z0' insert.hyps interior_subset[of S], auto)
    have g ∈ Θ[at z0'](f)
      by (intro bigthetaI_cong eventually_mono[OF ev]) (insert g, auto)
    also have f ∈ O[at z0'](λ_. 1)
      using z0' by (intro insert.prems) auto
    finally show g ∈ ... .
  qed (insert insert.prems g, auto)
  then obtain h where h holomorphic_on S ∀ z∈S − X. h z = g z by blast
  with g have h holomorphic_on S ∀ z∈S − insert z0 X. h z = f z by auto
  thus ?case by blast
qed

lemma continuous_imp_bigo_1:
  assumes continuous (at x within A) f
  shows   f ∈ O[at x within A](λ_. 1)
proof (rule bigoI_tendsto)
  from assms show ((λx. f x / 1) ⟶ f x) (at x within A)
    by (auto simp: continuous_within)
qed auto

lemma taylor_bigo_linear:
  assumes f field_differentiable at x0 within A
  shows   (λx. f x − f x0) ∈ O[at x0 within A](λx. x − x0)
proof −
  from assms obtain f' where (f has_field_derivative f') (at x0 within A)
    by (auto simp: field_differentiable_def)
  hence ((λx. (f x − f x0) / (x − x0)) ⟶ f') (at x0 within A)
    by (auto simp: has_field_derivative_iff)
 thus ?thesis by (intro bigoI_tendsto[where c = f'] ) (auto simp: eventually_at_filter)
qed
```

6.1 Cauchy's residue theorem

```
lemma get_integrable_path:
  assumes open s connected (s−pts) finite pts f holomorphic_on (s−pts)  a∈s−pts
b∈s−pts
  obtains g where valid_path g pathstart g = a pathfinish g = b
    path_image g ⊆ s−pts f contour_integrable_on g using assms
proof (induct arbitrary:s thesis a rule:finite_induct[OF ‹finite pts›])
  case 1
  obtain g where valid_path g path_image g ⊆ s pathstart g = a pathfinish g = b
    using connected_open_polynomial_connected[OF ‹open s›,of a b ] ‹connected
(s − {})›
      valid_path_polynomial_function 1.prems(6) 1.prems(7) by auto
```

```
  moreover have f contour_integrable_on g
    using contour_integrable_holomorphic_simple[OF _ ‹open s› ‹valid_path g›
‹path_image g ⊆ s›,of f]
      ‹f holomorphic_on s − {}›
    by auto
  ultimately show ?case using 1(1)[of g] by auto
next
  case idt:(2 p pts)
  obtain e where e>0 and e:∀ w∈ball a e. w ∈ s ∧ (w ≠ a ⟶ w ∉ insert p pts)
    using finite_ball_avoid[OF ‹open s› ‹finite (insert p pts)›, of a]
      ‹a ∈ s − insert p pts›
    by auto
  define a' where a' ≡ a+e/2
  have a'∈s−{p} −pts  using e[rule_format,of a+e/2] ‹e>0›
    by (auto simp add:dist_complex_def a'_def)
  then obtain g' where g'[simp]:valid_path g' pathstart g' = a' pathfinish g' = b
    path_image g' ⊆ s − {p} − pts f contour_integrable_on g'
    using idt.hyps(3)[of a' s−{p}] idt.prems idt.hyps(1)
    by (metis Diff_insert2 open_delete)
  define g where g ≡ linepath a a' +++ g'
  have valid_path g unfolding g_def by (auto intro: valid_path_join)
  moreover have pathstart g = a and  pathfinish g = b unfolding g_def by auto
  moreover have path_image g ⊆ s − insert p pts unfolding g_def
    proof (rule subset_path_image_join)
      have closed_segment a a' ⊆ ball a e using ‹e>0›
      by (auto dest!:segment_bound1 simp:a'_def dist_complex_def norm_minus_commute)
      then show path_image (linepath a a') ⊆ s − insert p pts using e idt(9)
        by auto
    next
      show path_image g' ⊆ s − insert p pts using g'(4) by blast
    qed
  moreover have f contour_integrable_on g
    proof −
      have closed_segment a a' ⊆ ball a e using ‹e>0›
      by (auto dest!:segment_bound1 simp:a'_def dist_complex_def norm_minus_commute)
      then have continuous_on (closed_segment a a') f
      using e idt.prems(6) holomorphic_on_imp_continuous_on[OF idt.prems(5)]
        apply (elim continuous_on_subset)
        by auto
      then have f contour_integrable_on linepath a a'
        using contour_integrable_continuous_linepath by auto
      then show ?thesis unfolding g_def
        apply (rule contour_integrable_joinI)
        by (auto simp add: ‹e>0›)
    qed
  ultimately show ?case using idt.prems(1)[of g] by auto
qed

lemma Cauchy_theorem_aux:
```

assumes *open s connected (s−pts) finite pts pts ⊆ s f holomorphic_on s−pts*
valid_path g pathfinish g = pathstart g path_image g ⊆ s−pts
∀ z. (z ∉ s) ⟶ winding_number g z = 0
∀ p∈s. h p>0 ∧ (∀ w∈cball p (h p). w∈s ∧ (w≠p ⟶ w ∉ pts))
shows *contour_integral g f = (∑ p∈pts. winding_number g p * contour_integral (circlepath p (h p)) f)*
using *assms*
proof (*induct arbitrary:s g rule:finite_induct[OF ‹finite pts›]*)
case *1*
then show *?case* **by** (*simp add: Cauchy_theorem_global contour_integral_unique*)
next
case (*2 p pts*)
note *fin[simp] = ‹finite (insert p pts)›*
and *connected = ‹connected (s − insert p pts)›*
and *valid[simp] = ‹valid_path g›*
and *g_loop[simp] = ‹pathfinish g = pathstart g›*
and *holo[simp]= ‹f holomorphic_on s − insert p pts›*
and *path_img = ‹path_image g ⊆ s − insert p pts›*
and *winding = ‹∀ z. z ∉ s ⟶ winding_number g z = 0›*
and *h = ‹∀ pa∈s. 0 < h pa ∧ (∀ w∈cball pa (h pa). w ∈ s ∧ (w ≠ pa ⟶ w ∉ insert p pts))›*
have *h p>0* **and** *p∈s*
and *h_p: ∀ w∈cball p (h p). w ∈ s ∧ (w ≠ p ⟶ w ∉ insert p pts)*
using *h ‹insert p pts ⊆ s›* **by** *auto*
obtain *pg* **where** *pg[simp]: valid_path pg pathstart pg = pathstart g pathfinish pg=p+h p*
path_image pg ⊆ s−insert p pts f contour_integrable_on pg
proof −
have *p + h p∈cball p (h p)* **using** *h[rule_format,of p]*
by (*simp add: ‹p ∈ s› dist_norm*)
then have *p + h p ∈ s − insert p pts* **using** *h[rule_format,of p] ‹insert p pts ⊆ s›*
by *fastforce*
moreover have *pathstart g ∈ s − insert p pts* **using** *path_img* **by** *auto*
ultimately show *?thesis*
using *get_integrable_path[OF ‹open s› connected fin holo,of pathstart g p+h p] that*
by *blast*
qed
obtain *n::int* **where** *n=winding_number g p*
using *integer_winding_number[OF _ g_loop,of p] valid path_img*
by (*metis DiffD2 Ints_cases insertI1 subset_eq valid_path_imp_path*)
define *p_circ* **where** *p_circ ≡ circlepath p (h p)*
define *p_circ_pt* **where** *p_circ_pt ≡ linepath (p+h p) (p+h p)*
define *n_circ* **where** *n_circ ≡ λn. ((+++) p_circ ^^ n) p_circ_pt*
define *cp* **where** *cp ≡ if n≥0 then reversepath (n_circ (nat n)) else n_circ (nat (− n))*
have *n_circ:valid_path (n_circ k)*
winding_number (n_circ k) p = k

```
    pathstart (n_circ k) = p + h p pathfinish (n_circ k) = p + h p
    path_image (n_circ k) =  (if k=0 then {p + h p} else sphere p (h p))
    p ∉ path_image (n_circ k)
     ⋀p'. p'∉s − pts ⟹ winding_number (n_circ k) p'=0 ∧ p'∉path_image
(n_circ k)
    f contour_integrable_on (n_circ k)
    contour_integral (n_circ k) f = k ∗  contour_integral p_circ f
    for k
  proof (induct k)
    case 0
    show valid_path (n_circ 0)
     and path_image (n_circ 0) =  (if 0=0 then {p + h p} else sphere p (h p))
     and winding_number (n_circ 0) p = of_nat 0
     and pathstart (n_circ 0) = p + h p
     and pathfinish (n_circ 0) = p + h p
     and p ∉ path_image (n_circ 0)
     unfolding n_circ_def p_circ_pt_def using ‹h p > 0›
     by (auto simp add: dist_norm)
    show winding_number (n_circ 0) p'=0 ∧ p'∉path_image (n_circ 0) when
p'∉s− pts for p'
     unfolding n_circ_def p_circ_pt_def
     apply (auto intro!:winding_number_trivial)
      by (metis Diff_iff pathfinish_in_path_image pg(3) pg(4) subsetCE sub-
set_insertI that)+
    show f contour_integrable_on (n_circ 0)
     unfolding n_circ_def p_circ_pt_def
    by (auto intro!:contour_integrable_continuous_linepath simp add:continuous_on_sing)
    show contour_integral (n_circ 0) f = of_nat 0 ∗ contour_integral p_circ f
     unfolding n_circ_def p_circ_pt_def by auto
  next
    case (Suc k)
    have n_Suc:n_circ (Suc k) = p_circ +++ n_circ k unfolding n_circ_def
by auto
     have pcirc:p ∉ path_image p_circ valid_path p_circ pathfinish p_circ =
pathstart (n_circ k)
      using Suc(3) unfolding p_circ_def using ‹h p > 0› by (auto simp add:
p_circ_def)
    have pcirc_image:path_image p_circ ⊆ s − insert p pts
     proof −
      have path_image p_circ ⊆ cball p (h p) using ‹0 < h p› p_circ_def by
auto
       then show ?thesis using h_p pcirc(1) by auto
     qed
    have pcirc_integrable:f contour_integrable_on p_circ
     by (auto simp add:p_circ_def intro!: pcirc_image[unfolded p_circ_def]
    contour_integrable_continuous_circlepath holomorphic_on_imp_continuous_on
      holomorphic_on_subset[OF holo])
    show valid_path (n_circ (Suc k))
     using valid_path_join[OF pcirc(2) Suc(1) pcirc(3)] unfolding n_circ_def
```

```
by auto
    show path_image (n_circ (Suc k))
       = (if Suc k = 0 then {p + complex_of_real (h p)} else sphere p (h p))
      proof −
       have path_image p_circ = sphere p (h p)
         unfolding p_circ_def using ‹0 < h p› by auto
       then show ?thesis unfolding n_Suc  using Suc.hyps(5)  ‹h p>0›
         by (auto simp add:  path_image_join[OF pcirc(3)]  dist_norm)
      qed
    then show p ∉ path_image (n_circ (Suc k)) using ‹h p>0› by auto
    show winding_number (n_circ (Suc k)) p = of_nat (Suc k)
      proof −
       have winding_number p_circ p = 1
        by (simp add: ‹h p > 0› p_circ_def winding_number_circlepath_centre)
         moreover have p ∉ path_image (n_circ k) using Suc(5) ‹h p>0› by
auto
       then have winding_number (p_circ +++ n_circ k) p
          = winding_number p_circ p + winding_number (n_circ k) p
         using  valid_path_imp_path Suc.hyps(1) Suc.hyps(2) pcirc
         apply (intro winding_number_join)
         by auto
       ultimately show ?thesis using Suc(2) unfolding n_circ_def
         by auto
      qed
    show pathstart (n_circ (Suc k)) = p + h p
      by (simp add: n_circ_def p_circ_def)
    show pathfinish (n_circ (Suc k)) = p + h p
      using Suc(4) unfolding n_circ_def by auto
     show winding_number (n_circ (Suc k)) p'=0 ∧  p'∉path_image (n_circ
(Suc k)) when p'∉s−pts for p'
      proof −
        have  p' ∉ path_image p_circ using ‹p ∈ s› h p_circ_def that using
pcirc_image by blast
       moreover have p' ∉ path_image (n_circ k)
         using Suc.hyps(7) that by blast
       moreover have winding_number p_circ p' = 0
         proof −
          have path_image p_circ ⊆ cball p (h p)
            using h unfolding p_circ_def using ‹p ∈ s› by fastforce
           moreover have p'∉cball p (h p) using ‹p ∈ s› h that 2.hyps(2) by
fastforce
          ultimately show ?thesis unfolding p_circ_def
           apply (intro winding_number_zero_outside)
           by auto
         qed
       ultimately show ?thesis
         unfolding n_Suc
         apply (subst winding_number_join)
      by (auto simp: valid_path_imp_path pcirc Suc that not_in_path_image_join
```

```
Suc.hyps(7)[OF that])
      qed
    show f contour_integrable_on (n_circ (Suc k))
      unfolding n_Suc
        by (rule contour_integrable_joinI[OF pcirc_integrable Suc(8) pcirc(2)
Suc(1)])
      show contour_integral (n_circ (Suc k)) f = (Suc k) *  contour_integral
p_circ f
      unfolding n_Suc
    by (auto simp add:contour_integral_join[OF pcirc_integrable Suc(8) pcirc(2)
Suc(1)]
       Suc(9) algebra_simps)
  qed
 have cp[simp]:pathstart cp = p + h p  pathfinish cp = p + h p
      valid_path cp path_image cp ⊆ s − insert p pts
      winding_number cp p = − n
      ⋀p'. p'∉s − pts ⟹ winding_number cp p'=0 ∧ p' ∉ path_image cp
      f contour_integrable_on cp
      contour_integral cp f = − n * contour_integral p_circ f
  proof −
   show pathstart cp = p + h p and pathfinish cp = p + h p and valid_path cp
     using n_circ unfolding cp_def by auto
  next
   have sphere p (h p) ⊆  s − insert p pts
     using h[rule_format,of p] ‹insert p pts ⊆ s› by force
   moreover  have p + complex_of_real (h p) ∈ s − insert p pts
     using pg(3) pg(4) by (metis pathfinish_in_path_image subsetCE)
   ultimately show path_image cp ⊆  s − insert p pts unfolding cp_def
     using n_circ(5)  by auto
  next
   show winding_number cp p = − n
     unfolding cp_def using winding_number_reversepath n_circ ‹h p>0›
     by (auto simp: valid_path_imp_path)
  next
   show winding_number cp p'=0 ∧ p' ∉ path_image cp when p'∉s − pts for
p'
     unfolding cp_def
     apply (auto)
     apply (subst winding_number_reversepath)
     by (auto simp add: valid_path_imp_path n_circ(7)[OF that] n_circ(1))
  next
   show f contour_integrable_on cp unfolding cp_def
     using contour_integrable_reversepath_eq n_circ(1,8) by auto
  next
   show contour_integral cp f = − n * contour_integral p_circ f
       unfolding cp_def using contour_integral_reversepath[OF n_circ(1)]
n_circ(9)
     by auto
  qed
```

```
  define g' where g' ≡ g +++ pg +++ cp +++ (reversepath pg)
  have contour_integral g' f = (∑ p∈pts. winding_number g' p * contour_integral
(circlepath p (h p)) f)
    proof (rule 2.hyps(3)[of s−{p} g',OF _ _ ‹finite pts› ])
      show connected (s − {p} − pts) using connected by (metis Diff_insert2)
      show open (s − {p}) using ‹open s› by auto
      show  pts ⊆ s − {p} using ‹insert p pts ⊆ s› ‹ p ∉ pts›  by blast
        show f holomorphic_on s − {p} − pts using holo ‹p ∉ pts› by (metis
Diff_insert2)
      show valid_path g'
        unfolding g'_def cp_def using n_circ valid pg g_loop
        by (auto intro!:valid_path_join )
      show pathfinish g' = pathstart g'
        unfolding g'_def cp_def using pg(2) by simp
      show path_image g' ⊆ s − {p} − pts
        proof −
          define s' where s' ≡ s − {p} − pts
          have s':s' = s−insert p pts  unfolding s'_def by auto
          then show ?thesis using path_img pg(4) cp(4)
            unfolding g'_def
            apply (fold s'_def s')
            apply (intro subset_path_image_join)
            by auto
        qed
      note path_join_imp[simp]
      show ∀ z. z ∉ s − {p} ⟶ winding_number g' z = 0
        proof clarify
          fix z assume z:z∉s − {p}
            have winding_number (g +++ pg +++ cp +++ reversepath pg) z =
winding_number g z
              + winding_number (pg +++ cp +++ (reversepath pg)) z
            proof (rule winding_number_join)
            show path g using ‹valid_path g› by (simp add: valid_path_imp_path)
              show z ∉ path_image g using z path_img by auto
              show path (pg +++ cp +++ reversepath pg) using pg(3) cp
                by (simp add: valid_path_imp_path)
            next
             have path_image (pg +++ cp +++ reversepath pg) ⊆ s − insert p pts
                using pg(4) cp(4) by (auto simp:subset_path_image_join)
              then show z ∉ path_image (pg +++ cp +++ reversepath pg) using
z by auto
            next
             show pathfinish g = pathstart (pg +++ cp +++ reversepath pg) using
g_loop by auto
            qed
          also have ... = winding_number g z + (winding_number pg z
              + winding_number (cp +++ (reversepath pg)) z)
            proof (subst add_left_cancel,rule winding_number_join)
              show path pg and path (cp +++ reversepath pg)
```

```
            and pathfinish pg = pathstart (cp +++ reversepath pg)
             by (auto simp add: valid_path_imp_path)
            show z ∉ path_image pg using pg(4) z by blast
            show z ∉ path_image (cp +++ reversepath pg) using z
            by (metis Diff_iff ‹z ∉ path_image pg› contra_subsetD cp(4) insertI1
                not_in_path_image_join path_image_reversepath singletonD)
          qed
        also have ... = winding_number g z + (winding_number pg z
          + (winding_number cp z + winding_number (reversepath pg) z))
          apply (auto intro!:winding_number_join simp: valid_path_imp_path)
          apply (metis Diff_iff contra_subsetD cp(4) insertI1 singletonD z)
          by (metis Diff_insert2 Diff_subset contra_subsetD pg(4) z)
        also have ... = winding_number g z + winding_number cp z
          apply (subst winding_number_reversepath)
          apply (auto simp: valid_path_imp_path)
          by (metis Diff_iff contra_subsetD insertI1 pg(4) singletonD z)
           finally have winding_number g' z = winding_number g z + wind-
ing_number cp z
          unfolding g'_def .
        moreover have winding_number g z + winding_number cp z = 0
          using winding z ‹n=winding_number g p› by auto
        ultimately show winding_number g' z = 0 unfolding g'_def by auto
      qed
    show ∀ pa∈s − {p}. 0 < h pa ∧ (∀ w∈cball pa (h pa). w ∈ s − {p} ∧ (w ≠
pa ⟶ w ∉ pts))
      using h by fastforce
  qed
 moreover have contour_integral g' f = contour_integral g f
    − winding_number g p ∗ contour_integral p_circ f
  proof −
    have contour_integral g' f =  contour_integral g f
      + contour_integral (pg +++ cp +++ reversepath pg) f
      unfolding g'_def
      apply (subst contour_integral_join)
      by (auto simp add:open_Diff[OF ‹open s›,OF finite_imp_closed[OF fin]]
        intro!: contour_integrable_holomorphic_simple[OF holo _ _ path_img]
        contour_integrable_reversepath)
    also have ... = contour_integral g f + contour_integral pg f
      + contour_integral (cp +++ reversepath pg) f
      apply (subst contour_integral_join)
      by (auto simp add:contour_integrable_reversepath)
    also have ... = contour_integral g f + contour_integral pg f
      + contour_integral cp f + contour_integral (reversepath pg) f
      apply (subst contour_integral_join)
      by (auto simp add:contour_integrable_reversepath)
    also have ... = contour_integral g f + contour_integral cp f
      using contour_integral_reversepath
      by (auto simp add:contour_integrable_reversepath)
   also have ... = contour_integral g f − winding_number g p ∗ contour_integral
```

```
p_circ f
      using ‹n=winding_number g p› by auto
    finally show ?thesis .
  qed
 moreover have winding_number g' p' = winding_number g p' when p'∈pts for
p'
  proof −
    have [simp]: p' ∉ path_image g p' ∉ path_image pg p'∉path_image cp
      using 2.prems(8) that
      apply blast
      apply (metis Diff_iff Diff_insert2 contra_subsetD pg(4) that)
      by (meson DiffD2 cp(4) rev_subsetD subset_insertI that)
    have winding_number g' p' = winding_number g p'
      + winding_number (pg +++ cp +++ reversepath pg) p' unfolding g'_def
      apply (subst winding_number_join)
      apply (simp_all add: valid_path_imp_path)
      apply (intro not_in_path_image_join)
      by auto
    also have ... = winding_number g p' + winding_number pg p'
      + winding_number (cp +++ reversepath pg) p'
      apply (subst winding_number_join)
      apply (simp_all add: valid_path_imp_path)
      apply (intro not_in_path_image_join)
      by auto
      also have ... = winding_number g p' + winding_number pg p'+ wind-
ing_number cp p'
      + winding_number (reversepath pg) p'
      apply (subst winding_number_join)
      by (simp_all add: valid_path_imp_path)
    also have ... = winding_number g p' + winding_number cp p'
      apply (subst winding_number_reversepath)
      by (simp_all add: valid_path_imp_path)
    also have ... = winding_number g p' using that by auto
    finally show ?thesis .
  qed
 ultimately show ?case unfolding p_circ_def
  apply (subst (asm) sum.cong[OF refl,
      of pts _ λp. winding_number g p * contour_integral (circlepath p (h p)) f])
  by (auto simp add:sum.insert[OF ‹finite pts› ‹p∉pts›] algebra_simps)
qed

lemma Cauchy_theorem_singularities:
 assumes open s connected s finite pts and
        holo:f holomorphic_on s−pts and
        valid_path g and
        loop:pathfinish g = pathstart g and
        path_image g ⊆ s−pts and
        homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0 and
        avoid:∀ p∈s. h p>0 ∧ (∀ w∈cball p (h p). w∈s ∧ (w≠p ⟶ w ∉ pts))
```

shows *contour_integral g f* = (∑ *p*∈*pts*. *winding_number g p* ∗ *contour_integral* (*circlepath p* (*h p*)) *f*)
(**is** *?L*=*?R*)
proof −
define *circ* **where** *circ* ≡ λ*p*. *winding_number g p* ∗ *contour_integral* (*circlepath p* (*h p*)) *f*
define *pts1* **where** *pts1* ≡ *pts* ∩ *s*
define *pts2* **where** *pts2* ≡ *pts* − *pts1*
have *pts*=*pts1* ∪ *pts2 pts1* ∩ *pts2* = {} *pts2* ∩ *s*={} *pts1*⊆*s*
unfolding *pts1_def pts2_def* **by** *auto*
have *contour_integral g f* = (∑ *p*∈*pts1*. *circ p*) **unfolding** *circ_def*
proof (*rule Cauchy_theorem_aux*[*OF* ‹*open s*› _ _ ‹*pts1*⊆*s*› _ ‹*valid_path g*› *loop* _ *homo*])
have *finite pts1* **unfolding** *pts1_def* **using** ‹*finite pts*› **by** *auto*
then show *connected* (*s* − *pts1*)
using ‹*open s*› ‹*connected s*› *connected_open_delete_finite*[*of s*] **by** *auto*
next
show *finite pts1* **using** ‹*pts* = *pts1* ∪ *pts2*› *assms*(*3*) **by** *auto*
show *f holomorphic_on s* − *pts1* **by** (*metis Diff_Int2 Int_absorb holo pts1_def*)
show *path_image g* ⊆ *s* − *pts1* **using** *assms*(*7*) *pts1_def* **by** *auto*
show ∀ *p*∈*s*. *0* < *h p* ∧ (∀ *w*∈*cball p* (*h p*). *w* ∈ *s* ∧ (*w* ≠ *p* ⟶ *w* ∉ *pts1*))
by (*simp add*: *avoid pts1_def*)
qed
moreover have *sum circ pts2*=*0*
proof −
have *winding_number g p*=*0* **when** *p*∈*pts2* **for** *p*
using ‹*pts2* ∩ *s*={}› *that homo*[*rule_format*,*of p*] **by** *auto*
thus *?thesis* **unfolding** *circ_def*
apply (*intro sum.neutral*)
by *auto*
qed
moreover have *?R*=*sum circ pts1* + *sum circ pts2*
unfolding *circ_def*
using *sum.union_disjoint*[*OF* _ _ ‹*pts1* ∩ *pts2* = {}›] ‹*finite pts*› ‹*pts*=*pts1* ∪ *pts2*›
by *blast*
ultimately show *?thesis*
apply (*fold circ_def*)
by *auto*
qed

theorem *Residue_theorem*:
fixes *s pts*::*complex set* **and** *f*::*complex* ⇒ *complex*
and *g*::*real* ⇒ *complex*
assumes *open s connected s finite pts* **and**
holo:*f holomorphic_on s*−*pts* **and**
valid_path g **and**
loop:*pathfinish g* = *pathstart g* **and**

```
        path_image g ⊆ s−pts and
        homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0
  shows contour_integral g f = 2 * pi * i *(∑ p∈pts. winding_number g p *
residue f p)
proof −
  define c where c ≡ 2 * pi * i
  obtain h where avoid:∀ p∈s. h p>0 ∧ (∀ w∈cball p (h p). w∈s ∧ (w≠p ⟶ w
∉ pts))
    using finite_cball_avoid[OF ‹open s› ‹finite pts›] by metis
  have contour_integral g f
     = (∑ p∈pts. winding_number g p * contour_integral (circlepath p (h p)) f)
    using Cauchy_theorem_singularities[OF assms avoid] .
  also have ... = (∑ p∈pts. c * winding_number g p * residue f p)
    proof (intro sum.cong)
      show pts = pts by simp
    next
      fix x assume x ∈ pts
      show winding_number g x * contour_integral (circlepath x (h x)) f
         = c * winding_number g x * residue f x
        proof (cases x∈s)
          case False
          then have winding_number g x=0 using homo by auto
          thus ?thesis by auto
        next
          case True
          have contour_integral (circlepath x (h x)) f = c* residue f x
            using ‹x∈pts› ‹finite pts› avoid[rule_format,OF True]
         apply (intro base_residue[of s−(pts−{x}),THEN contour_integral_unique,folded
c_def])
           by (auto intro:holomorphic_on_subset[OF holo] open_Diff[OF ‹open s›
finite_imp_closed])
          then show ?thesis by auto
        qed
    qed
  also have ... = c * (∑ p∈pts. winding_number g p * residue f p)
    by (simp add: sum_distrib_left algebra_simps)
  finally show ?thesis unfolding c_def .
qed
```

6.2 The argument principle

```
theorem argument_principle:
  fixes f::complex ⇒ complex and poles s:: complex set
  defines pz ≡ {w∈s. f w = 0 ∨ w ∈ poles} — pz is the set of poles and zeros
  assumes open s connected s and
        f_holo:f holomorphic_on s−poles and
        h_holo:h holomorphic_on s and
        valid_path g and
        loop:pathfinish g = pathstart g and
```

```
        path_img:path_image g ⊆ s − pz and
        homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0 and
        finite:finite pz and
        poles:∀ p∈s∩poles. is_pole f p
  shows contour_integral g (λx. deriv f x * h x / f x) = 2 * pi * i *
        (∑ p∈pz. winding_number g p * h p * zorder f p)
    (is ?L=?R)
proof −
  define c where c ≡ 2 * complex_of_real pi * i
  define ff where ff ≡ (λx. deriv f x * h x / f x)
  define cont where cont ≡ λff p e. (ff has_contour_integral c * zorder f p * h p
) (circlepath p e)
  define avoid where avoid ≡ λp e. ∀ w∈cball p e. w ∈ s ∧ (w ≠ p ⟶ w ∉ pz)

  have ∃ e>0. avoid p e ∧ (p∈pz ⟶ cont ff p e) when p∈s for p
  proof −
    obtain e1 where e1>0 and e1_avoid:avoid p e1
      using finite_cball_avoid[OF ‹open s› finite] ‹p∈s› unfolding avoid_def by
auto
    have ∃ e2>0. cball p e2 ⊆ ball p e1 ∧ cont ff p e2 when p∈pz
    proof −
      define po where po ≡ zorder f p
      define pp where pp ≡ zor_poly f p
      define f' where f' ≡ λw. pp w * (w − p) powi po
      define ff' where ff' ≡ (λx. deriv f' x * h x / f' x)
      obtain r where pp p≠0 r>0 and
        r<e1 and
        pp_holo:pp holomorphic_on cball p r and
        pp_po:(∀ w∈cball p r−{p}. f w = pp w * (w − p) powi po ∧ pp w ≠ 0)
      proof −
        have isolated_singularity_at f p
        proof −
          have f holomorphic_on ball p e1 − {p}
            apply (intro holomorphic_on_subset[OF f_holo])
            using e1_avoid ‹p∈pz› unfolding avoid_def pz_def by force
          then show ?thesis unfolding isolated_singularity_at_def
            using ‹e1>0› analytic_on_open open_delete by blast
        qed
        moreover have not_essential f p
        proof (cases is_pole f p)
          case True
          then show ?thesis unfolding not_essential_def by auto
        next
          case False
          then have p∈s−poles using ‹p∈s› poles unfolding pz_def by auto
          moreover have open (s−poles)
          proof −
            have closed (s ∩ poles)
              using finite by (simp add: pz_def finite_imp_closed rev_finite_subset
```

```
subset_eq)
          then show ?thesis
            by (metis Diff_Compl Diff_Diff_Int Diff_eq ‹open s› open_Diff)
        qed
        ultimately have isCont f p
              using holomorphic_on_imp_continuous_on[OF f_holo] continu-
ous_on_eq_continuous_at
          by auto
        then show ?thesis unfolding isCont_def not_essential_def by auto
      qed
      moreover have ∃F w in at p. f w ≠ 0
      proof (rule ccontr)
        assume ¬ (∃F w in at p. f w ≠ 0)
        then have ∀F w in at p. f w= 0 unfolding frequently_def by auto
        then obtain r1 where r1>0 and r1:∀w∈ball p r1 − {p}. f w =0
          unfolding eventually_at by (auto simp add:dist_commute)
        obtain r2 where r2>0 and r2: ball p r2 ⊆ s
          using ‹p∈s› ‹open s› openE by blast
        define rr where rr=min r1 r2

        from r1 r2
        have ball p rr − {p} ⊆ {w∈ s ∩ ball p rr−{p}. f w=0}
          unfolding rr_def by auto
        moreover have infinite (ball p rr − {p})
          using ‹r1>0› ‹r2>0› finite_imp_not_open
          unfolding rr_def by fastforce
          ultimately have infinite {w∈s ∩ ball p rr−{p}. f w=0} using infi-
nite_super by blast
        then have infinite pz
        unfolding pz_def by (smt (verit) infinite_super Collect_mono_iff DiffE
Int_iff)
        then show False using ‹finite pz› by auto
      qed
      ultimately obtain r where pp p ≠ 0 and r:r>0 pp holomorphic_on cball
p r
              (∀w∈cball p r − {p}. f w = pp w ∗ (w − p) powi po ∧ pp w ≠ 0)
        using zorder_exist[of f p,folded po_def pp_def] by auto
      define r1 where r1=min r e1 / 2
      have r1<e1 unfolding r1_def using ‹e1>0› ‹r>0› by auto
      moreover have r1>0 pp holomorphic_on cball p r1
            (∀w∈cball p r1 − {p}. f w = pp w ∗ (w − p) powi po ∧ pp w ≠ 0)
        unfolding r1_def using ‹e1>0› r by auto
      ultimately show ?thesis using that ‹pp p≠0› by auto
    qed

    define e2 where e2 ≡ r/2
    have e2>0 using ‹r>0› unfolding e2_def by auto
    define anal where anal ≡ λw. deriv pp w ∗ h w / pp w
    define prin where prin ≡ λw. po ∗ h w / (w − p)
```

```
    have ((λw.  prin w + anal w) has_contour_integral c * po * h p) (circlepath
p e2)
    proof (rule has_contour_integral_add[of _ _ _ _ 0,simplified])
      have ball p r ⊆ s
          using ‹r<e1› avoid_def ball_subset_cball e1_avoid by (simp add:
subset_eq)
      then have cball p e2 ⊆ s
        using ‹r>0› unfolding e2_def by auto
      then have (λw. po * h w) holomorphic_on cball p e2
        using h_holo by (auto intro!: holomorphic_intros)
      then show (prin has_contour_integral c * po * h p ) (circlepath p e2)
        using Cauchy_integral_circlepath_simple[folded c_def, of λw. po * h w]
‹e2>0›
        unfolding prin_def by (auto simp add: mult.assoc)
      have anal holomorphic_on ball p r unfolding anal_def
        using pp_holo h_holo pp_po ‹ball p r ⊆ s› ‹pp p≠0›
        by (auto intro!: holomorphic_intros)
      then show (anal has_contour_integral 0) (circlepath p e2)
        using e2_def ‹r>0›
        by (auto elim!: Cauchy_theorem_disc_simple)
    qed
    then have cont ff' p e2 unfolding cont_def po_def
    proof (elim has_contour_integral_eq)
      fix w assume w ∈ path_image (circlepath p e2)
      then have w∈ball p r and w≠p unfolding e2_def using ‹r>0› by auto
      define wp where wp ≡ w−p
      have wp≠0 and pp w ≠0
        unfolding wp_def using ‹w≠p› ‹w∈ball p r› pp_po by auto
      moreover have der_f':deriv f' w = po * pp w * (w−p) powi (po − 1) +
deriv pp w * (w−p) powi po
      proof (rule DERIV_imp_deriv)
        have (pp has_field_derivative (deriv pp w)) (at w)
          using DERIV_deriv_iff_has_field_derivative pp_holo ‹w≠p›
        by (meson open_ball ‹w ∈ ball p r› ball_subset_cball holomorphic_derivI
holomorphic_on_subset)
       then show  (f' has_field_derivative of_int po * pp w * (w − p) powi (po
− 1)
                + deriv pp w * (w − p) powi po) (at w)
          unfolding f'_def using ‹w≠p›
       by (auto intro!: derivative_eq_intros DERIV_cong[OF has_field_derivative_powr_of_int])
      qed
      ultimately show prin w + anal w = ff' w
        unfolding ff'_def prin_def anal_def
        apply simp
        apply (unfold f'_def)
        apply (fold wp_def)
        apply (auto simp add:field_simps)
        by (metis (no_types, lifting) mult.commute power_int_minus_mult)
    qed
```

```
    then have cont ff p e2 unfolding cont_def
    proof (elim has_contour_integral_eq)
      fix w assume w ∈ path_image (circlepath p e2)
      then have w∈ball p r and w≠p unfolding e2_def using ‹r>0› by auto
      have deriv f' w =  deriv f w
      proof (rule complex_derivative_transform_within_open[where s=ball p r
− {p}])
       show f' holomorphic_on ball p r − {p} unfolding f'_def using pp_holo
         by (auto intro!: holomorphic_intros)
       next
        have ball p e1 − {p} ⊆ s − poles
         using ball_subset_cball e1_avoid[unfolded avoid_def] unfolding pz_def
          by auto
        then have ball p r − {p} ⊆ s − poles
          apply (elim dual_order.trans)
          using ‹r<e1› by auto
        then show f holomorphic_on ball p r − {p} using f_holo
          by auto
       next
        show open (ball p r − {p}) by auto
        show w ∈ ball p r − {p} using ‹w∈ball p r› ‹w≠p› by auto
       next
        fix x assume x ∈ ball p r − {p}
        then show f' x = f x
          using pp_po unfolding f'_def by auto
       qed
       moreover have  f' w  =  f w
        using ‹w ∈ ball p r› ball_subset_cball subset_iff pp_po ‹w≠p›
        unfolding f'_def by auto
       ultimately show ff' w = ff w
        unfolding ff'_def ff_def by simp
      qed
      moreover have cball p e2 ⊆ ball p e1
       using ‹0 < r› ‹r<e1› e2_def by auto
      ultimately show ?thesis using ‹e2>0› by auto
    qed
    then obtain e2 where e2:p∈pz ⟶ e2>0 ∧ cball p e2 ⊆ ball p e1 ∧ cont ff
p e2
      by auto
    define e4 where e4 ≡ if p∈pz then e2 else  e1
    have e4>0 using e2 ‹e1>0› unfolding e4_def by auto
     moreover have avoid p e4 using e2 ‹e1>0› e1_avoid unfolding e4_def
avoid_def by auto
    moreover have p∈pz ⟶ cont ff p e4
      by (auto simp add: e2 e4_def)
    ultimately show ?thesis by auto
  qed
  then obtain get_e where get_e:∀ p∈s. get_e p>0 ∧ avoid p (get_e p)
     ∧ (p∈pz ⟶ cont ff p (get_e p))
```

```
    by metis
  define ci where ci ≡ λp. contour_integral (circlepath p (get_e p)) ff
  define w where w ≡ λp. winding_number g p
  have contour_integral g ff = (∑ p∈pz. w p * ci p) unfolding ci_def w_def
  proof (rule Cauchy_theorem_singularities[OF ‹open s› ‹connected s› finite _
‹valid_path g› loop
        path_img homo])
   have open (s - pz) using open_Diff[OF _ finite_imp_closed[OF finite]] ‹open
s› by auto
    then show ff holomorphic_on s - pz unfolding ff_def using f_holo h_holo
      by (auto intro!: holomorphic_intros simp add:pz_def)
  next
    show ∀ p∈s. 0 < get_e p ∧ (∀ w∈cball p (get_e p). w ∈ s ∧ (w ≠ p ⟶ w ∉
pz))
      using get_e using avoid_def by blast
  qed
  also have ... = (∑ p∈pz. c * w p * h p * zorder f p)
  proof (rule sum.cong[of pz pz,simplified])
    fix p assume p ∈ pz
    show w p * ci p = c * w p * h p * (zorder f p)
    proof (cases p∈s)
      assume p ∈ s
      have ci p = c * h p * (zorder f p) unfolding ci_def
        apply (rule contour_integral_unique)
          using get_e ‹p∈s› ‹p∈pz› unfolding cont_def by (metis mult.assoc
mult.commute)
      thus ?thesis by auto
    next
      assume p∉s
      then have w p=0 using homo unfolding w_def by auto
      then show ?thesis by auto
    qed
  qed
  also have ... = c*(∑ p∈pz. w p * h p * zorder f p)
    unfolding sum_distrib_left by (simp add:algebra_simps)
  finally have contour_integral g ff = c * (∑ p∈pz. w p * h p * of_int (zorder f
p)) .
  then show ?thesis unfolding ff_def c_def w_def by simp
qed
```

6.3 Coefficient asymptotics for generating functions

For a formal power series that has a meromorphic continuation on some disc in the context plane, we can use the Residue Theorem to extract precise asymptotic information from the residues at the poles. This can be used to derive the asymptotic behaviour of the coefficients ($a_n \sim \ldots$). If the function is meromorphic on the entire complex plane, one can even derive a full asymptotic expansion.

We will first show the relationship between the coefficients and the sum over the residues with an explicit remainder term (the contour integral along the circle used in the Residue theorem).

theorem
fixes $f :: complex \Rightarrow complex$ **and** $n :: nat$ **and** $r :: real$
defines $g \equiv (\lambda w.\ f\ w\ /\ w\ \hat{}\ Suc\ n)$ **and** $\gamma \equiv circlepath\ 0\ r$
assumes *open A connected A cball 0 r* $\subseteq A$ $r > 0$
assumes *f holomorphic_on A − S S* $\subseteq$ *ball 0 r finite S* $0 \notin S$
shows *fps_coeff_conv_residues*:
$(deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n =$
$contour\_integral\ \gamma\ g\ /\ (2 * pi * \mathrm{i}) - (\sum z{\in}S.\ residue\ g\ z)$ (**is** *?thesis1*)
and *fps_coeff_residues_bound*:
$(\bigwedge z.\ norm\ z = r \Longrightarrow z \notin k \Longrightarrow norm\ (f\ z) \leq C) \Longrightarrow C \geq 0 \Longrightarrow finite\ k \Longrightarrow$
$norm\ ((deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n + (\sum z{\in}S.\ residue\ g\ z)) \leq C\ /\ r\ \hat{}\ n$
proof −
have *holo*: *g holomorphic_on A − insert 0 S*
unfolding *g_def* **using** *assms* **by** (*auto intro*!: *holomorphic_intros*)
have $contour\_integral\ \gamma\ g = 2 * pi * \mathrm{i} * (\sum z{\in}insert\ 0\ S.\ winding\_number\ \gamma\ z * residue\ g\ z)$
proof (*rule Residue_theorem*)
show *g holomorphic_on A − insert 0 S* **by** *fact*
from *assms* **show** $\forall z.\ z \notin A \longrightarrow winding\_number\ \gamma\ z = 0$
unfolding *γ_def* **by** (*intro allI impI winding_number_zero_outside*[*of _ cball 0 r*]) *auto*
qed (*insert assms*, *auto simp*: *γ_def*)
also have *winding_number γ z = 1* **if** *z ∈ insert 0 S* **for** *z*
unfolding *γ_def* **using** *assms that* **by** (*intro winding_number_circlepath*) *auto*
hence $(\sum z{\in}insert\ 0\ S.\ winding\_number\ \gamma\ z * residue\ g\ z) = (\sum z{\in}insert\ 0\ S.\ residue\ g\ z)$
by (*intro sum.cong*) *simp_all*
also have $\ldots = residue\ g\ 0 + (\sum z{\in}S.\ residue\ g\ z)$
using ‹*0 ∉ S*› **and** ‹*finite S*› **by** (*subst sum.insert*) *auto*
also from ‹$r > 0$› **have** *0 ∈ cball 0 r* **by** *simp*
with *assms* **have** *0 ∈ A − S* **by** *blast*
with *assms* **have** $residue\ g\ 0 = (deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n$
unfolding *g_def* **by** (*subst residue_holomorphic_over_power'*[*of A − S*])
(*auto simp*: *finite_imp_closed*)
finally show *?thesis1*
by (*simp add*: *field_simps*)

assume *C*: $\bigwedge z.\ norm\ z = r \Longrightarrow z \notin k \Longrightarrow norm\ (f\ z) \leq C$ $C \geq 0$ **and** *k*: *finite k*
have $(deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n + (\sum z{\in}S.\ residue\ g\ z) = contour\_integral\ \gamma\ g\ /\ (2 * pi * \mathrm{i})$
using ‹*?thesis1*› **by** (*simp add*: *algebra_simps*)
also have $norm\ \ldots = norm\ (contour\_integral\ \gamma\ g)\ /\ (2 * pi)$
by (*simp add*: *norm_divide norm_mult*)
also have $norm\ (contour\_integral\ \gamma\ g) \leq C\ /\ r\ \hat{}\ Suc\ n * (2 * pi * r)$

proof (*rule has_contour_integral_bound_circlepath_strong*)
from ‹*open A*› **and** ‹*finite S*› **have** *open* (*A* − *insert 0 S*)
by (*blast intro*: *finite_imp_closed*)
with *assms* **show** (*g has_contour_integral contour_integral γ g*) (*circlepath 0 r*)
unfolding *γ_def*
by (*intro has_contour_integral_integral contour_integrable_holomorphic_simple* [*OF holo*]) *auto*
next
fix *z* **assume** *z*: *norm* (*z* − *0*) = *r z* ∉ *k*
hence *norm* (*g z*) = *norm* (*f z*) / *r* ^ *Suc n*
by (*simp add*: *norm_divide g_def norm_mult norm_power*)
also have ... ≤ *C* / *r* ^ *Suc n*
using *k* **and** ‹*r* > *0*› **and** *z* **by** (*intro divide_right_mono C zero_le_power*) *auto*
finally show *norm* (*g z*) ≤ *C* / *r* ^ *Suc n* **.**
qed (*insert C*(*2*) *k* ‹*r* > *0*›, *auto*)
also from ‹*r* > *0*› **have** *C* / *r* ^ *Suc n* ∗ (*2* ∗ *pi* ∗ *r*) / (*2* ∗ *pi*) = *C* / *r* ^ *n*
by *simp*
finally show *norm* ((*deriv* ^^ *n*) *f 0* / *fact n* + (∑ *z*∈*S*. *residue g z*)) ≤ ...
by − (*simp_all add*: *divide_right_mono*)
qed

Since the circle is fixed, we can get an upper bound on the values of the generating function on the circle and therefore show that the integral is $O(r^{-n})$.

corollary *fps_coeff_residues_bigo*:
fixes *f* :: *complex* ⇒ *complex* **and** *r* :: *real*
assumes *open A connected A cball 0 r* ⊆ *A r* > *0*
assumes *f holomorphic_on A* − *S S* ⊆ *ball 0 r finite S 0* ∉ *S*
assumes *g*: *eventually* (λ*n*. *g n* = −(∑ *z*∈*S*. *residue* (λ*z*. *f z* / *z* ^ *Suc n*) *z*)) *sequentially*
(**is** *eventually* (λ*n*. _ = −*?g*′ *n*) _)
shows (λ*n*. (*deriv* ^^ *n*) *f 0* / *fact n* − *g n*) ∈ *O*(λ*n*. *1* / *r* ^ *n*) (**is** (λ*n*. *?c n* − _) ∈ *O*(_))
proof −
from *assms* **have** *compact* (*f* ‘ *sphere 0 r*)
by (*intro compact_continuous_image holomorphic_on_imp_continuous_on holomorphic_on_subset*[*OF* ‹*f holomorphic_on A* − *S*›]) *auto*
hence *bounded* (*f* ‘ *sphere 0 r*) **by** (*rule compact_imp_bounded*)
then obtain *C* **where** *C*: ⋀*z*. *z* ∈ *sphere 0 r* ⟹ *norm* (*f z*) ≤ *C*
by (*auto simp*: *bounded_iff sphere_def*)
have *0* ≤ *norm* (*f* (*of_real r*)) **by** *simp*
also from *C*[*of of_real r*] **and** ‹*r* > *0*› **have** ... ≤ *C* **by** *simp*
finally have *C_nonneg*: *C* ≥ *0* **.**

have (λ*n*. *?c n* + *?g*′ *n*) ∈ *O*(λ*n*. *of_real* (*1* / *r* ^ *n*))
proof (*intro bigoI*[*of* _ *C*] *always_eventually allI*)
fix *n* :: *nat*

```
    from assms and C and C_nonneg have norm (?c n + ?g' n) ≤ C / r ^ n
      by (intro fps_coeff_residues_bound[where A = A and k = {}]) auto
    also have ... = C * norm (complex_of_real (1 / r ^ n))
      using ‹r > 0› by (simp add: norm_divide norm_power)
    finally show norm (?c n + ?g' n) ≤ ... .
  qed
  also have ?this ⟷ (λn. ?c n - g n) ∈ O(λn. of_real (1 / r ^ n))
    by (intro landau_o.big.in_cong eventually_mono[OF g]) simp_all
  finally show ?thesis .
qed

corollary fps_coeff_residues_bigo':
  fixes f :: complex ⇒ complex and r :: real
  assumes exp: f has_fps_expansion F
  assumes open A connected A cball 0 r ⊆ A r > 0
  assumes f holomorphic_on A - S S ⊆ ball 0 r finite S 0 ∉ S
   assumes eventually (λn. g n = -(∑z∈S. residue (λz. f z / z ^ Suc n) z))
sequentially
          (is eventually (λn. _ = -?g' n) _)
  shows   (λn. fps_nth F n - g n) ∈ O(λn. 1 / r ^ n) (is (λn. ?c n - _) ∈
O(_))
proof -
  have fps_nth F = (λn. (deriv ^^ n) f 0 / fact n)
    using fps_nth_fps_expansion[OF exp] by (intro ext) simp_all
  with fps_coeff_residues_bigo[OF assms(2-)] show ?thesis by simp
qed
```

6.4 Rouche's theorem

```
theorem Rouche_theorem:
  fixes f g::complex ⇒ complex and s:: complex set
  defines fg≡(λp. f p + g p)
  defines zeros_fg≡{p∈s. fg p = 0} and zeros_f≡{p∈s. f p = 0}
  assumes
    open s and connected s and
    finite zeros_fg and
    finite zeros_f and
    f_holo:f holomorphic_on s and
    g_holo:g holomorphic_on s and
    valid_path γ and
    loop:pathfinish γ = pathstart γ and
    path_img:path_image γ ⊆ s  and
    path_less:∀ z∈path_image γ. cmod(f z) > cmod(g z) and
    homo:∀ z. (z ∉ s) ⟶ winding_number γ z = 0
  shows (∑p∈zeros_fg. winding_number γ p * zorder fg p)
        = (∑p∈zeros_f. winding_number γ p * zorder f p)
proof -
  have path_fg:path_image γ ⊆ s - zeros_fg
  proof -
```

```
    have False when z∈path_image γ and f z + g z=0 for z
    proof −
      have cmod (f z) > cmod (g z) using ‹z∈path_image γ› path_less by auto
        moreover have f z = − g z  using ‹f z + g z =0› by (simp add:
eq_neg_iff_add_eq_0)
      then have cmod (f z) = cmod (g z) by auto
      ultimately show False by auto
    qed
    then show ?thesis unfolding zeros_fg_def fg_def using path_img by auto
  qed
  have path_f:path_image γ ⊆ s − zeros_f
  proof −
    have False when z∈path_image γ and f z =0 for z
    proof −
      have cmod (g z) < cmod (f z)  using ‹z∈path_image γ› path_less by auto
      then have cmod (g z) < 0 using ‹f z=0› by auto
      then show False by auto
    qed
    then show ?thesis unfolding zeros_f_def using path_img by auto
  qed
  define w where w ≡ λp. winding_number γ p
  define c where c ≡ 2 ∗ complex_of_real pi ∗ i
  define h where h ≡ λp. g p / f p + 1
  obtain spikes
    where finite spikes and spikes: ∀ x∈{0..1} − spikes. γ differentiable at x
    using ‹valid_path γ›
   by (auto simp: valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
  have h_contour:((λx. deriv h x / h x) has_contour_integral 0) γ
  proof −
    have outside_img:0 ∈ outside (path_image (h o γ))
    proof −
      have h p ∈ ball 1 1 when p∈path_image γ for p
      proof −
        have cmod (g p/f p) <1
              by (smt (verit) divide_less_eq_1_pos norm_divide norm_ge_zero
path_less that)
       then show ?thesis unfolding h_def by (auto simp add:dist_complex_def)
      qed
      then have path_image (h o γ) ⊆ ball 1 1
        by (simp add: image_subset_iff path_image_compose)
      moreover have  (0::complex) ∉ ball 1 1 by (simp add: dist_norm)
      ultimately show ?thesis
        using  convex_in_outside[of ball 1 1 0] outside_mono by blast
    qed
    have valid_h:valid_path (h ∘ γ)
   proof (rule valid_path_compose_holomorphic[OF ‹valid_path γ› _ _ path_f])
      show h holomorphic_on s − zeros_f
        unfolding h_def using f_holo g_holo
        by (auto intro!: holomorphic_intros simp add:zeros_f_def)
```

```
    next
      show open (s − zeros_f) using ‹finite zeros_f› ‹open s› finite_imp_closed
        by auto
    qed
    have ((λz. 1/z) has_contour_integral 0) (h ∘ γ)
    proof −
     have 0 ∉ path_image (h ∘ γ) using outside_img by (simp add: outside_def)
      then have ((λz. 1/z) has_contour_integral c ∗ winding_number (h ∘ γ) 0)
(h ∘ γ)
       using has_contour_integral_winding_number[of h o γ 0,simplified] valid_h
        unfolding c_def by auto
      moreover have winding_number (h o γ) 0 = 0
      proof −
        have 0 ∈ outside (path_image (h ∘ γ)) using outside_img .
        moreover have path (h o γ)
          using valid_h  by (simp add: valid_path_imp_path)
        moreover have pathfinish (h o γ) = pathstart (h o γ)
          by (simp add: loop pathfinish_compose pathstart_compose)
       ultimately show ?thesis using winding_number_zero_in_outside by auto
      qed
      ultimately show ?thesis by auto
    qed
    moreover have vector_derivative (h ∘ γ) (at x) = vector_derivative γ (at x)
∗ deriv h (γ x)
      when x∈{0..1} − spikes for x
    proof (rule vector_derivative_chain_at_general)
      show γ differentiable at x using that ‹valid_path γ› spikes by auto
    next
      define der where der ≡ λp. (deriv g p ∗ f p − g p ∗ deriv f p)/(f p ∗ f p)
      define t where t ≡ γ x
      have f t≠0 unfolding zeros_f_def t_def
        by (metis DiffD1 image_eqI norm_not_less_zero norm_zero path_defs(4)
path_less that)
      moreover have t∈s
        using contra_subsetD path_image_def path_fg t_def that by fastforce
      ultimately have (h has_field_derivative der t) (at t)
        unfolding h_def der_def using g_holo f_holo ‹open s›
        by (auto intro!: holomorphic_derivI derivative_eq_intros)
      then show h field_differentiable at (γ x)
        unfolding t_def field_differentiable_def by blast
    qed
    then have  ((/) 1 has_contour_integral 0) (h ∘ γ)
                = ((λx. deriv h x / h x) has_contour_integral 0) γ
      unfolding has_contour_integral
      apply (intro has_integral_spike_eq[OF negligible_finite, OF ‹finite spikes›])
      by auto
    ultimately show ?thesis by auto
  qed
  then have contour_integral γ (λx. deriv h x / h x) = 0
```

```
    using  contour_integral_unique by simp
  moreover have contour_integral γ (λx. deriv fg x / fg x) = contour_integral γ
(λx. deriv f x / f x)
      + contour_integral γ (λp. deriv h p / h p)
  proof −
    have (λp. deriv f p / f p) contour_integrable_on γ
     proof (rule contour_integrable_holomorphic_simple[OF _ _ ‹valid_path γ›
path_f])
      show open (s − zeros_f) using finite_imp_closed[OF ‹finite zeros_f›] ‹open
s›
        by auto
      then show (λp. deriv f p / f p) holomorphic_on s − zeros_f
        using f_holo
        by (auto intro!: holomorphic_intros simp add:zeros_f_def)
    qed
    moreover have (λp. deriv h p / h p) contour_integrable_on γ
      using h_contour
      by (simp add: has_contour_integral_integrable)
    ultimately have contour_integral γ (λx. deriv f x / f x + deriv h x / h x) =
                      contour_integral γ (λp. deriv f p / f p) + contour_integral γ
(λp. deriv h p / h p)
      using contour_integral_add[of (λp. deriv f p / f p) γ (λp. deriv h p / h p) ]
      by auto
    moreover have deriv fg p / fg p =  deriv f p / f p + deriv h p / h p
                  when p∈ path_image γ for p
    proof −
      have fg p≠0 and f p≠0 using path_f path_fg that unfolding zeros_f_def
zeros_fg_def
        by auto
      have h p≠0
      proof (rule ccontr)
        assume ¬ h p ≠ 0
       then have g p / f p= −1 unfolding h_def by (simp add: add_eq_0_iff2)
        then have cmod (g p/f p) = 1 by auto
        moreover have cmod (g p/f p) <1
          by (simp add: ‹f p ≠ 0› norm_divide path_less that)
        ultimately show False by auto
      qed
      have der_fg:deriv fg p =  deriv f p + deriv g p unfolding fg_def
        using f_holo g_holo holomorphic_on_imp_differentiable_at[OF _  ‹open
s›] path_img that
        by auto
      have der_h:deriv h p = (deriv g p ∗ f p − g p ∗ deriv f p)/(f p ∗ f p)
      proof −
        define der where der ≡ λp. (deriv g p ∗ f p − g p ∗ deriv f p)/(f p ∗ f p)
        have p∈s using path_img that by auto
        then have (h has_field_derivative der p) (at p)
          unfolding h_def der_def using g_holo f_holo ‹open s› ‹f p≠0›
          by (auto intro!: derivative_eq_intros holomorphic_derivI)
```

```
      then show ?thesis unfolding der_def using DERIV_imp_deriv by auto
    qed
    show ?thesis
      apply (simp only:der_fg der_h)
      apply (auto simp add:field_simps ‹h p≠0› ‹f p≠0› ‹fg p≠0›)
      by (auto simp add:field_simps h_def ‹f p≠0› fg_def)
  qed
  then have contour_integral γ (λp. deriv fg p / fg p)
              = contour_integral γ (λp. deriv f p / f p + deriv h p / h p)
    by (elim contour_integral_eq)
  ultimately show ?thesis by auto
 qed
 moreover have contour_integral γ (λx. deriv fg x / fg x) = c * (∑ p∈zeros_fg.
w p * zorder fg p)
 proof −
   have fg holomorphic_on s unfolding fg_def using f_holo g_holo holomor-
phic_on_add by auto
   moreover
    have path_image γ ⊆ s − {p∈s. fg p = 0} using path_fg unfolding ze-
ros_fg_def .
   moreover
   have finite {p∈s. fg p = 0} using ‹finite zeros_fg› unfolding zeros_fg_def .
   ultimately show ?thesis
     unfolding c_def zeros_fg_def w_def
    using argument_principle[OF ‹open s› ‹connected s› _ _ ‹valid_path γ› loop
_ homo, of _ {} λ_. 1]
     by simp
 qed
 moreover have contour_integral γ (λx. deriv f x / f x) = c * (∑ p∈zeros_f. w
p * zorder f p)
   unfolding c_def zeros_f_def w_def
 proof (rule argument_principle[OF ‹open s› ‹connected s› _ _ ‹valid_path γ›
loop _ homo
       , of _ {} λ_. 1,simplified])
   show f holomorphic_on s using f_holo g_holo holomorphic_on_add by auto
  show path_image γ ⊆ s − {p∈s. f p = 0} using path_f unfolding zeros_f_def
.
   show finite {p∈s. f p = 0} using ‹finite zeros_f› unfolding zeros_f_def .
 qed
 ultimately have c* (∑ p∈zeros_fg. w p * (zorder fg p)) = c* (∑ p∈zeros_f.
w p * (zorder f p))
   by auto
 then show ?thesis unfolding c_def using w_def by auto
qed

end
theory Laurent_Convergence
 imports HOL−Computational_Algebra.Formal_Laurent_Series HOL−Library.Landau_Symbols
        Residue_Theorem
```

begin

TODO: Better than *?f field_differentiable at (?c * ?z) ⟹ deriv (λw. ?f (?c * w)) ?z = ?c * deriv ?f (?c * ?z)*?

lemma *deriv_compose_linear'*:
assumes *f field_differentiable at (c * z+a)*
shows *deriv (λw. f (c * w+a)) z = c * deriv f (c * z+a)*
apply (*subst deriv_chain*[**where** *f=λw. c * w+a*,*unfolded comp_def*])
using *assms* **by** (*auto intro*:*derivative_intros*)

TODO: Better than ⟦*?f holomorphic_on ?T*; *open ?S*; *open ?T*; *?z ∈ ?S*; ⋀*w. w ∈ ?S ⟹ ?u * w ∈ ?T*⟧ ⟹ *deriv*$^{?n}$ *(λw. ?f (?u * w)) ?z = ?u*$^{?n}$ ** deriv*$^{?n}$ *?f (?u * ?z)*?

lemma *higher_deriv_compose_linear'*:
fixes *z*::*complex*
assumes *f*: *f holomorphic_on T* **and** *S*: *open S* **and** *T*: *open T* **and** *z*: *z ∈ S*
and *fg*: ⋀*w. w ∈ S ⟹ u * w+c ∈ T*
shows (*deriv* ^^ *n*) *(λw. f (u * w+c)) z = u^n * (deriv* ^^ *n) f (u * z+c)*
using *z*
proof (*induction n arbitrary*: *z*)
case *0* **then show** *?case* **by** *simp*
next
case (*Suc n z*)
have *holo0*: *f holomorphic_on (λw. u * w+c) ‘ S*
by (*meson fg f holomorphic_on_subset image_subset_iff*)
have *holo2*: (*deriv* ^^ *n*) *f holomorphic_on (λw. u * w+c) ‘ S*
by (*meson f fg holomorphic_higher_deriv holomorphic_on_subset image_subset_iff T*)
have *holo3*: *(λz. u ^ n * (deriv* ^^ *n) f (u * z+c)) holomorphic_on S*
by (*intro holo2 holomorphic_on_compose* [**where** *g=(deriv* ^^ *n) f*, *unfolded o_def*] *holomorphic_intros*)
have *(λw. u * w+c) holomorphic_on S f holomorphic_on (λw. u * w+c) ‘ S*
by (*rule holo0 holomorphic_intros*)+
then have *holo1*: *(λw. f (u * w+c)) holomorphic_on S*
by (*rule holomorphic_on_compose* [**where** *g=f*, *unfolded o_def*])
have *deriv ((deriv* ^^ *n) (λw. f (u * w+c))) z = deriv (λz. u^n * (deriv* ^^ *n) f (u*z+c)) z*
proof (*rule complex_derivative_transform_within_open* [*OF _ holo3 S Suc.prems*])
show *(deriv* ^^ *n) (λw. f (u * w+c)) holomorphic_on S*
by (*rule holomorphic_higher_deriv* [*OF holo1 S*])
qed (*simp add*: *Suc.IH*)
also have *… = u^n * deriv (λz. (deriv* ^^ *n) f (u * z+c)) z*
proof −
have *(deriv* ^^ *n) f analytic_on T*
by (*simp add*: *analytic_on_open f holomorphic_higher_deriv T*)
then have *(λw. (deriv* ^^ *n) f (u * w+c)) analytic_on S*
proof −
have *(deriv* ^^ *n) f ∘ (λw. u * w+c) holomorphic_on S*

```
        using holomorphic_on_compose[OF _ holo2] ‹(λw. u * w+c) holomorphic_on S›
        by simp
      then show ?thesis
        by (simp add: S analytic_on_open o_def)
    qed
    then show ?thesis
     by (intro deriv_cmult analytic_on_imp_differentiable_at [OF _ Suc.prems])
  qed
  also have ... = u * u ^ n * deriv ((deriv ^^ n) f) (u * z+c)
  proof −
    have (deriv ^^ n) f field_differentiable at (u * z+c)
    using Suc.prems T f fg holomorphic_higher_deriv holomorphic_on_imp_differentiable_at
by blast
    then show ?thesis
      by (simp add: deriv_compose_linear')
  qed
  finally show ?case
    by simp
qed

lemma fps_to_fls_numeral [simp]: fps_to_fls (numeral n) = numeral n
  by (metis fps_to_fls_of_nat of_nat_numeral)

lemma fls_const_power: fls_const (a ^ b) = fls_const a ^ b
  by (induction b) (auto simp flip: fls_const_mult_const)

lemma fls_deriv_numeral [simp]: fls_deriv (numeral n) = 0
  by (metis fls_deriv_of_int of_int_numeral)

lemma fls_const_numeral [simp]: fls_const (numeral n) = numeral n
  by (metis fls_of_nat of_nat_numeral)

lemma fls_mult_of_int_nth [simp]:
  shows fls_nth (numeral k * f) n = numeral k * fls_nth f n
  and   fls_nth (f * numeral k) n = fls_nth f n * numeral k
  by (metis fls_const_numeral fls_mult_const_nth)+

lemma fls_nth_numeral' [simp]:
  fls_nth (numeral n) 0 = numeral n k ≠ 0 ⟹ fls_nth (numeral n) k = 0
  by (subst fls_const_numeral [symmetric], subst fls_const_nth, simp)+

lemma fls_subdegree_prod:
  fixes F :: 'a ⇒ 'b :: field_char_0 fls
  assumes ⋀x. x ∈ I ⟹ F x ≠ 0
  shows   fls_subdegree (∏ x∈I. F x) = (∑ x∈I. fls_subdegree (F x))
  using assms by (induction I rule: infinite_finite_induct) auto

lemma fls_subdegree_prod':
```

```
  fixes F :: 'a ⇒ 'b :: field_char_0 fls
  assumes ⋀x. x ∈ I ⟹ fls_subdegree (F x) ≠ 0
  shows   fls_subdegree (∏ x∈I. F x) = (∑ x∈I. fls_subdegree (F x))
proof (intro fls_subdegree_prod)
  show F x ≠ 0 if x ∈ I for x
    using assms[OF that] by auto
qed

instance fps :: (semiring_char_0) semiring_char_0
proof
  show inj (of_nat :: nat ⇒ 'a fps)
  proof
    fix m n :: nat
    assume of_nat m = (of_nat n :: 'a fps)
    hence fps_nth (of_nat m) 0 = (fps_nth (of_nat n) 0 :: 'a)
      by (simp only: )
    thus m = n
      by simp
  qed
qed

instance fls :: (semiring_char_0) semiring_char_0
proof
  show inj (of_nat :: nat ⇒ 'a fls)
  proof
    fix m n :: nat
    assume of_nat m = (of_nat n :: 'a fls)
    hence fls_nth (of_nat m) 0 = (fls_nth (of_nat n) 0 :: 'a)
      by (simp only: )
    thus m = n
      by (simp add: fls_of_nat_nth)
  qed
qed

lemma fls_const_eq_0_iff [simp]: fls_const c = 0 ⟷ c = 0
  using fls_const_0 fls_const_nonzero by blast

lemma fls_subdegree_add_eq1:
  assumes f ≠ 0 fls_subdegree f < fls_subdegree g
  shows   fls_subdegree (f + g) = fls_subdegree f
proof (intro antisym)
  from assms have *: fls_nth (f + g) (fls_subdegree f) ≠ 0
    by auto
  from * show fls_subdegree (f + g) ≤ fls_subdegree f
    by (rule fls_subdegree_leI)
  from * have f + g ≠ 0
    using fls_nonzeroI by blast
  thus fls_subdegree f ≤ fls_subdegree (f + g)
    using assms(2) fls_plus_subdegree by force
```

qed

lemma *fls_subdegree_add_eq2*:
assumes $g \neq 0$ *fls_subdegree* $g <$ *fls_subdegree* f
shows *fls_subdegree* $(f + g) =$ *fls_subdegree* g
proof (*intro antisym*)
from *assms* **have** $*$: *fls_nth* $(f + g)$ (*fls_subdegree* g) $\neq 0$
by *auto*
from $*$ **show** *fls_subdegree* $(f + g) \leq$ *fls_subdegree* g
by (*rule fls_subdegree_leI*)
from $*$ **have** $f + g \neq 0$
using *fls_nonzeroI* **by** *blast*
thus *fls_subdegree* $g \leq$ *fls_subdegree* $(f + g)$
using *assms*(2) *fls_plus_subdegree* **by** *force*
qed

lemma *fls_subdegree_diff_eq1*:
assumes $f \neq 0$ *fls_subdegree* $f <$ *fls_subdegree* g
shows *fls_subdegree* $(f - g) =$ *fls_subdegree* f
using *fls_subdegree_add_eq1*[*of* f $-g$] *assms* **by** *simp*

lemma *fls_subdegree_diff_eq2*:
assumes $g \neq 0$ *fls_subdegree* $g <$ *fls_subdegree* f
shows *fls_subdegree* $(f - g) =$ *fls_subdegree* g
using *fls_subdegree_add_eq2*[*of* $-g$ f] *assms* **by** *simp*

lemma *nat_minus_fls_subdegree_plus_const_eq*:
nat $(-$*fls_subdegree* $(F +$ *fls_const* $c)) =$ *nat* $(-$*fls_subdegree* $F)$
proof (*cases fls_subdegree* $F < 0$)
case *True*
hence *fls_subdegree* $(F +$ *fls_const* $c) =$ *fls_subdegree* F
by (*intro fls_subdegree_add_eq1*) *auto*
thus *?thesis*
by *simp*
next
case *False*
thus *?thesis*
by (*auto simp*: *fls_subdegree_ge0I*)
qed

lemma *at_to_0'*: *NO_MATCH* 0 $z \Longrightarrow$ *at* $z =$ *filtermap* $(\lambda x.\ x + z)$ (*at* 0)
for z :: *'a*::*real_normed_vector*
by (*rule at_to_0*)

lemma *nhds_to_0*: *nhds* $(x ::$ *'a* :: *real_normed_vector*$) =$ *filtermap* $((+)\ x)$ (*nhds* 0)
proof −
have $(\lambda xa.\ xa - - x) = (+)\ x$
by *auto*

```
  thus ?thesis
    using filtermap_nhds_shift[of −x 0] by simp
qed

lemma nhds_to_0': NO_MATCH 0 x ⟹ nhds (x :: 'a :: real_normed_vector)
= filtermap ((+) x) (nhds 0)
  by (rule nhds_to_0)


definition fls_conv_radius :: complex fls ⇒ ereal where
  fls_conv_radius f = fps_conv_radius (fls_regpart f)

definition eval_fls :: complex fls ⇒ complex ⇒ complex where
  eval_fls F z = eval_fps (fls_base_factor_to_fps F) z ∗ z powi fls_subdegree F

definition
  has_laurent_expansion :: (complex ⇒ complex) ⇒ complex fls ⇒ bool
  (infixl has'_laurent'_expansion 60)
  where (f has_laurent_expansion F) ⟷
          fls_conv_radius F > 0 ∧ eventually (λz. eval_fls F z = f z) (at 0)

lemma has_laurent_expansion_schematicI:
  f has_laurent_expansion F ⟹ F = G ⟹ f has_laurent_expansion G
  by simp

lemma has_laurent_expansion_cong:
  assumes eventually (λx. f x = g x) (at 0) F = G
  shows   (f has_laurent_expansion F) ⟷ (g has_laurent_expansion G)
proof −
  have eventually (λz. eval_fls F z = g z) (at 0)
    if eventually (λz. eval_fls F z = f z) (at 0) eventually (λx. f x = g x) (at 0)
for f g
    using that by eventually_elim auto
  from this[of f g] this[of g f] show ?thesis
    using assms by (auto simp: eq_commute has_laurent_expansion_def)
qed

lemma has_laurent_expansion_cong':
  assumes eventually (λx. f x = g x) (at z) F = G z = z'
   shows    ((λx. f (z + x)) has_laurent_expansion F) ⟷ ((λx. g (z' + x))
has_laurent_expansion G)
  by (intro has_laurent_expansion_cong)
     (use assms in ‹auto simp: at_to_0' eventually_filtermap add_ac›)

lemma fls_conv_radius_altdef:
  fls_conv_radius F = fps_conv_radius (fls_base_factor_to_fps F)
proof −
  have conv_radius (λn. fls_nth F (int n)) = conv_radius (λn. fls_nth F (int n
+ fls_subdegree F))
```

```
  proof (cases fls_subdegree F ≥ 0)
    case True
    hence conv_radius (λn. fls_nth F (int n + fls_subdegree F)) =
          conv_radius (λn. fls_nth F (int (n + nat (fls_subdegree F))))
      by auto
    thus ?thesis
      by (subst (asm) conv_radius_shift) auto
  next
    case False
    hence conv_radius (λn. fls_nth F (int n)) =
        conv_radius (λn. fls_nth F (fls_subdegree F + int (n + nat (−fls_subdegree
F))))
      by auto
    thus ?thesis
      by (subst (asm) conv_radius_shift) (auto simp: add_ac)
  qed
  thus ?thesis
    by (simp add: fls_conv_radius_def fps_conv_radius_def)
qed

lemma eval_fps_of_nat [simp]: eval_fps (of_nat n) z = of_nat n
  and eval_fps_of_int [simp]: eval_fps (of_int m) z = of_int m
  by (simp_all flip: fps_of_nat fps_of_int)

lemma fls_subdegree_numeral [simp]: fls_subdegree (numeral n) = 0
  by (metis fls_subdegree_of_nat of_nat_numeral)

lemma fls_regpart_numeral [simp]: fls_regpart (numeral n) = numeral n
  by (metis fls_regpart_of_nat of_nat_numeral)

lemma fps_conv_radius_of_nat [simp]: fps_conv_radius (of_nat n) = ∞
  and fps_conv_radius_of_int [simp]: fps_conv_radius (of_int m) = ∞
  by (simp_all flip: fps_of_nat fps_of_int)

lemma fps_conv_radius_fls_regpart: fps_conv_radius (fls_regpart F) = fls_conv_radius
F
  by (simp add: fls_conv_radius_def)

lemma fls_conv_radius_0 [simp]: fls_conv_radius 0 = ∞
  and fls_conv_radius_1 [simp]: fls_conv_radius 1 = ∞
  and fls_conv_radius_const [simp]: fls_conv_radius (fls_const c) = ∞
  and fls_conv_radius_numeral [simp]: fls_conv_radius (numeral num) = ∞
  and fls_conv_radius_of_nat [simp]: fls_conv_radius (of_nat n) = ∞
  and fls_conv_radius_of_int [simp]: fls_conv_radius (of_int m) = ∞
  and fls_conv_radius_X [simp]: fls_conv_radius fls_X = ∞
  and fls_conv_radius_X_inv [simp]: fls_conv_radius fls_X_inv = ∞
   and fls_conv_radius_X_intpow [simp]: fls_conv_radius (fls_X_intpow m) =
∞
  by (simp_all add: fls_conv_radius_def fls_X_intpow_regpart)
```

lemma *fls_conv_radius_shift* [*simp*]: *fls_conv_radius* (*fls_shift n F*) = *fls_conv_radius F*
unfolding *fls_conv_radius_altdef* **by** (*subst fls_base_factor_to_fps_shift*) (*rule refl*)

lemma *fls_conv_radius_fps_to_fls* [*simp*]: *fls_conv_radius* (*fps_to_fls F*) = *fps_conv_radius F*
by (*simp add*: *fls_conv_radius_def*)

lemma *fls_conv_radius_deriv* [*simp*]: *fls_conv_radius* (*fls_deriv F*) ≥ *fls_conv_radius F*
proof −
have *fls_conv_radius* (*fls_deriv F*) = *fps_conv_radius* (*fls_regpart* (*fls_deriv F*))
by (*simp add*: *fls_conv_radius_def*)
also have *fls_regpart* (*fls_deriv F*) = *fps_deriv* (*fls_regpart F*)
by (*intro fps_ext*) (*auto simp*: *add_ac*)
also have *fps_conv_radius* ... ≥ *fls_conv_radius F*
by (*simp add*: *fls_conv_radius_def fps_conv_radius_deriv*)
finally show *?thesis* .
qed

lemma *fls_conv_radius_uminus* [*simp*]: *fls_conv_radius* (−*F*) = *fls_conv_radius F*
by (*simp add*: *fls_conv_radius_def*)

lemma *fls_conv_radius_add*: *fls_conv_radius* (*F* + *G*) ≥ *min* (*fls_conv_radius F*) (*fls_conv_radius G*)
by (*simp add*: *fls_conv_radius_def fps_conv_radius_add*)

lemma *fls_conv_radius_diff*: *fls_conv_radius* (*F* − *G*) ≥ *min* (*fls_conv_radius F*) (*fls_conv_radius G*)
by (*simp add*: *fls_conv_radius_def fps_conv_radius_diff*)

lemma *fls_conv_radius_mult*: *fls_conv_radius* (*F* ∗ *G*) ≥ *min* (*fls_conv_radius F*) (*fls_conv_radius G*)
proof (*cases F* = *0* ∨ *G* = *0*)
case *False*
hence [*simp*]: *F* ≠ *0* *G* ≠ *0*
by *auto*
have *fls_conv_radius* (*F* ∗ *G*) = *fps_conv_radius* (*fls_regpart* (*fls_shift* (*fls_subdegree F* + *fls_subdegree G*) (*F* ∗ *G*)))
by (*simp add*: *fls_conv_radius_altdef*)
also have *fls_regpart* (*fls_shift* (*fls_subdegree F* + *fls_subdegree G*) (*F* ∗ *G*)) =
fls_base_factor_to_fps F ∗ *fls_base_factor_to_fps G*
by (*simp add*: *fls_times_def*)
also have *fps_conv_radius* ... ≥ *min* (*fls_conv_radius F*) (*fls_conv_radius G*)

unfolding *fls_conv_radius_altdef* **by** (*rule fps_conv_radius_mult*)
finally show *?thesis* **.**
qed *auto*

lemma *fps_conv_radius_add_ge*:
fps_conv_radius F $\geq$ *r* $\Longrightarrow$ *fps_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fps_conv_radius* (*F* + *G*) $\geq$ *r*
using *fps_conv_radius_add*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fps_conv_radius_diff_ge*:
fps_conv_radius F $\geq$ *r* $\Longrightarrow$ *fps_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fps_conv_radius* (*F* − *G*) $\geq$ *r*
using *fps_conv_radius_diff*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fps_conv_radius_mult_ge*:
fps_conv_radius F $\geq$ *r* $\Longrightarrow$ *fps_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fps_conv_radius* (*F* ∗ *G*) $\geq$ *r*
using *fps_conv_radius_mult*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fls_conv_radius_add_ge*:
fls_conv_radius F $\geq$ *r* $\Longrightarrow$ *fls_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fls_conv_radius* (*F* + *G*) $\geq$ *r*
using *fls_conv_radius_add*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fls_conv_radius_diff_ge*:
fls_conv_radius F $\geq$ *r* $\Longrightarrow$ *fls_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fls_conv_radius* (*F* − *G*) $\geq$ *r*
using *fls_conv_radius_diff*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fls_conv_radius_mult_ge*:
fls_conv_radius F $\geq$ *r* $\Longrightarrow$ *fls_conv_radius G* $\geq$ *r* $\Longrightarrow$ *fls_conv_radius* (*F* ∗ *G*) $\geq$ *r*
using *fls_conv_radius_mult*[*of F G*] **by** (*simp add*: *min_def split*: *if_splits*)

lemma *fls_conv_radius_power*: *fls_conv_radius* (*F* ^ *n*) $\geq$ *fls_conv_radius F*
by (*induction n*) (*auto intro*!: *fls_conv_radius_mult_ge*)

lemma *eval_fls_0* [*simp*]: *eval_fls 0 z* = *0*
and *eval_fls_1* [*simp*]: *eval_fls 1 z* = *1*
and *eval_fls_const* [*simp*]: *eval_fls* (*fls_const c*) *z* = *c*
and *eval_fls_numeral* [*simp*]: *eval_fls* (*numeral num*) *z* = *numeral num*
and *eval_fls_of_nat* [*simp*]: *eval_fls* (*of_nat n*) *z* = *of_nat n*
and *eval_fls_of_int* [*simp*]: *eval_fls* (*of_int m*) *z* = *of_int m*
and *eval_fls_X* [*simp*]: *eval_fls fls_X z* = *z*
and *eval_fls_X_intpow* [*simp*]: *eval_fls* (*fls_X_intpow m*) *z* = *z powi m*
by (*simp_all add*: *eval_fls_def*)

lemma *eval_fls_at_0*: *eval_fls F 0* = (*if fls_subdegree F* $\geq$ *0 then fls_nth F 0 else 0*)

```
  by (cases fls_subdegree F = 0)
    (simp_all add: eval_fls_def fls_regpart_def eval_fps_at_0)

lemma eval_fps_to_fls:
  assumes norm z < fps_conv_radius F
  shows   eval_fls (fps_to_fls F) z = eval_fps F z
proof (cases F = 0)
  case [simp]: False
  have eval_fps F z = eval_fps (unit_factor F * normalize F) z
    by (metis unit_factor_mult_normalize)
  also have ... = eval_fps (unit_factor F * fps_X ^ subdegree F) z
    by simp
  also have ... = eval_fps (unit_factor F) z * z ^ subdegree F
    using assms by (subst eval_fps_mult) auto
  also have ... = eval_fls (fps_to_fls F) z
   unfolding eval_fls_def fls_base_factor_to_fps_to_fls fls_subdegree_fls_to_fps
             power_int_of_nat ..
  finally show ?thesis ..
qed auto

lemma eval_fls_shift:
  assumes [simp]: z ≠ 0
  shows   eval_fls (fls_shift n F) z = eval_fls F z * z powi -n
proof (cases F = 0)
  case [simp]: False
  show ?thesis
  unfolding eval_fls_def
  by (subst fls_base_factor_to_fps_shift, subst fls_shift_subdegree[OF ‹F ≠ 0›],
subst power_int_diff)
     (auto simp: power_int_minus divide_simps)
qed auto

lemma eval_fls_add:
  assumes ereal (norm z) < fls_conv_radius F ereal (norm z) < fls_conv_radius
G z ≠ 0
  shows   eval_fls (F + G) z = eval_fls F z + eval_fls G z
  using assms
proof (induction fls_subdegree F fls_subdegree G arbitrary: F G rule: linorder_wlog)
  case (sym F G)
  show ?case
    using sym(1)[of G F] sym(2-) by (simp add: add_ac)
next
  case (le F G)
  show ?case
  proof (cases F = 0 ∨ G = 0)
    case False
    hence [simp]: F ≠ 0 G ≠ 0
      by auto
    note [simp] = ‹z ≠ 0›
```

```
  define F' G' where F' = fls_base_factor_to_fps F G' = fls_base_factor_to_fps
G
    define m n where m = fls_subdegree F n = fls_subdegree G
    have m ≤ n
      using le by (auto simp: m_n_def)
  have conv1: ereal (cmod z) < fps_conv_radius F' ereal (cmod z) < fps_conv_radius
G'
      using assms le by (simp_all add: F'_G'_def fls_conv_radius_altdef)
    have conv2: ereal (cmod z) < fps_conv_radius (G' * fps_X ^ nat (n − m))
      using conv1 by (intro less_le_trans[OF _ fps_conv_radius_mult]) auto
    have conv3: ereal (cmod z) < fps_conv_radius (F' + G' * fps_X ^ nat (n −
m))
       using conv1 conv2 by (intro less_le_trans[OF _ fps_conv_radius_add])
auto

    have eval_fls F z + eval_fls G z = eval_fps F' z * z powi m + eval_fps G' z
* z powi n
      unfolding eval_fls_def m_n_def[symmetric] F'_G'_def[symmetric]
      by (simp add: power_int_add algebra_simps)
    also have ... = (eval_fps F' z + eval_fps G' z * z powi (n − m)) * z powi m
      by (simp add: algebra_simps power_int_diff)
    also have eval_fps G' z * z powi (n − m) = eval_fps (G' * fps_X ^ nat (n
− m)) z
    using assms ‹m ≤ n› conv1 by (subst eval_fps_mult) (auto simp: power_int_def)
    also have eval_fps F' z + ... = eval_fps (F' + G' * fps_X ^ nat (n − m)) z
      using conv1 conv2 by (subst eval_fps_add) auto
    also have ... = eval_fls (fps_to_fls (F' + G' * fps_X ^ nat (n − m))) z
      using conv3 by (subst eval_fps_to_fls) auto
    also have ... * z powi m = eval_fls (fls_shift (−m) (fps_to_fls (F' + G' *
fps_X ^ nat (n − m)))) z
      by (subst eval_fls_shift) auto
    also have fls_shift (−m) (fps_to_fls (F' + G' * fps_X ^ nat (n − m))) = F
+ G
      using ‹m ≤ n›
    by (simp add: fls_times_fps_to_fls fps_to_fls_power fls_X_power_conv_shift_1
                  fls_shifted_times_simps F'_G'_def m_n_def)
    finally show ?thesis ..
  qed auto
qed

lemma eval_fls_minus:
  assumes ereal (norm z) < fls_conv_radius F
  shows   eval_fls (−F) z = −eval_fls F z
 using assms by (simp add: eval_fls_def eval_fps_minus fls_conv_radius_altdef)

lemma eval_fls_diff:
  assumes ereal (norm z) < fls_conv_radius F ereal (norm z) < fls_conv_radius
G
     and [simp]: z ≠ 0
```

```
  shows   eval_fls (F − G) z = eval_fls F z − eval_fls G z
proof −
  have eval_fls (F + (−G)) z = eval_fls F z − eval_fls G z
    using assms by (subst eval_fls_add) (auto simp: eval_fls_minus)
  thus ?thesis
    by simp
qed

lemma eval_fls_mult:
  assumes ereal (norm z) < fls_conv_radius F ereal (norm z) < fls_conv_radius
G z ≠ 0
  shows   eval_fls (F ∗ G) z = eval_fls F z ∗ eval_fls G z
proof (cases F = 0 ∨ G = 0)
  case False
  hence [simp]: F ≠ 0 G ≠ 0
    by auto
  note [simp] = ‹z ≠ 0›
 define F′ G′ where F′ = fls_base_factor_to_fps F G′ = fls_base_factor_to_fps
G
  define m n where m = fls_subdegree F n = fls_subdegree G
  have eval_fls F z ∗ eval_fls G z = (eval_fps F′ z ∗ eval_fps G′ z) ∗ z powi (m
+ n)
    unfolding eval_fls_def m_n_def[symmetric] F′_G′_def[symmetric]
    by (simp add: power_int_add algebra_simps)
  also have ... = eval_fps (F′ ∗ G′) z ∗ z powi (m + n)
   using assms by (subst eval_fps_mult) (auto simp: F′_G′_def fls_conv_radius_altdef)
  also have ... = eval_fls (F ∗ G) z
    by (simp add: eval_fls_def F′_G′_def m_n_def) (simp add: fls_times_def)
  finally show ?thesis ..
qed auto

lemma eval_fls_power:
  assumes ereal (norm z) < fls_conv_radius F z ≠ 0
  shows   eval_fls (F ^ n) z = eval_fls F z ^ n
proof (induction n)
  case (Suc n)
  have eval_fls (F ^ Suc n) z = eval_fls (F ∗ F ^ n) z
    by simp
  also have ... = eval_fls F z ∗ eval_fls (F ^ n) z
   using assms by (subst eval_fls_mult) (auto intro!: less_le_trans[OF _ fls_conv_radius_power])
  finally show ?case
    using Suc by simp
qed auto

lemma norm_summable_fls:
  norm z < fls_conv_radius f ⟹ summable (λn. norm (fls_nth f n ∗ z ^ n))
 using norm_summable_fps[of z fls_regpart f] by (simp add: fls_conv_radius_def)

lemma norm_summable_fls′:
```

*norm z < fls_conv_radius f ⟹ summable (λn. norm (fls_nth f (n + fls_subdegree f) * z ^ n))*
using *norm_summable_fps[of z fls_base_factor_to_fps f]* **by** (*simp add*: *fls_conv_radius_altdef*)

lemma *summable_fls*:
*norm z < fls_conv_radius f ⟹ summable (λn. fls_nth f n * z ^ n)*
by (*rule summable_norm_cancel[OF norm_summable_fls]*)

theorem *sums_eval_fls*:
fixes *f*
defines *n ≡ fls_subdegree f*
assumes *norm z < fls_conv_radius f* **and** *z ≠ 0 ∨ n ≥ 0*
shows *(λk. fls_nth f (int k + n) * z powi (int k + n)) sums eval_fls f z*
proof (*cases z = 0*)
case [*simp*]: *False*
have *(λk. fps_nth (fls_base_factor_to_fps f) k * z ^ k * z powi n) sums (eval_fps (fls_base_factor_to_fps f) z * z powi n)*
using *assms(2)* **by** (*intro sums_eval_fps sums_mult2*) (*auto simp*: *fls_conv_radius_altdef*)
thus *?thesis*
by (*simp add*: *power_int_add n_def eval_fls_def mult_ac*)
next
case [*simp*]: *True*
with *assms* **have** *n ≥ 0*
by *auto*
have *(λk. fls_nth f (int k + n) * z powi (int k + n)) sums (∑k∈(if n ≤ 0 then {nat (−n)} else {}). fls_nth f (int k + n) * z powi (int k + n))*
by (*intro sums_finite*) (*auto split*: *if_splits*)
also have *... = eval_fls f z*
using ‹*n ≥ 0*› **by** (*auto simp*: *eval_fls_at_0 n_def not_le*)
finally show *?thesis* **.**
qed

lemma *holomorphic_on_eval_fls*:
fixes *f*
defines *n ≡ fls_subdegree f*
assumes *A ⊆ eball 0 (fls_conv_radius f) − (if n ≥ 0 then {} else {0})*
shows *eval_fls f holomorphic_on A*
proof (*cases n ≥ 0*)
case *True*
have *eval_fls f = (λz. eval_fps (fls_base_factor_to_fps f) z * z ^ nat n)*
using *True* **by** (*simp add*: *fun_eq_iff eval_fls_def power_int_def n_def*)
moreover have *... holomorphic_on A*
using *True assms(2)* **by** (*intro holomorphic_intros*) (*auto simp*: *fls_conv_radius_altdef*)
ultimately show *?thesis*
by *simp*
next
case *False*
show *?thesis* **using** *assms*

unfolding *eval_fls_def* **by** (*intro holomorphic_intros*) (*auto simp*: *fls_conv_radius_altdef*)
qed

lemma *holomorphic_on_eval_fls'* [*holomorphic_intros*]:
assumes *g holomorphic_on A*
assumes *g ‘ A* ⊆ *eball 0* (*fls_conv_radius f*) − (*if fls_subdegree f* ≥ *0 then* {} *else* {*0*})
shows (λ*x*. *eval_fls f* (*g x*)) *holomorphic_on A*
proof −
have *eval_fls f* ∘ *g holomorphic_on A*
by (*intro holomorphic_on_compose*[*OF assms*(*1*) *holomorphic_on_eval_fls*]) (*use assms* **in** *auto*)
thus *?thesis*
by (*simp add*: *o_def*)
qed

lemma *continuous_on_eval_fls*:
fixes *f*
defines *n* ≡ *fls_subdegree f*
assumes *A* ⊆ *eball 0* (*fls_conv_radius f*) − (*if n* ≥ *0 then* {} *else* {*0*})
shows *continuous_on A* (*eval_fls f*)
by (*intro holomorphic_on_imp_continuous_on holomorphic_on_eval_fls*) (*use assms* **in** *auto*)

lemma *continuous_on_eval_fls'* [*continuous_intros*]:
fixes *f*
defines *n* ≡ *fls_subdegree f*
assumes *g ‘ A* ⊆ *eball 0* (*fls_conv_radius f*) − (*if n* ≥ *0 then* {} *else* {*0*})
assumes *continuous_on A g*
shows *continuous_on A* (λ*x*. *eval_fls f* (*g x*))
using *assms*(*3*)
by (*intro continuous_on_compose2*[*OF continuous_on_eval_fls _ assms*(*2*)]) (*auto simp*: *n_def*)

lemmas *has_field_derivative_eval_fps'* [*derivative_intros*] =
DERIV_chain2[*OF has_field_derivative_eval_fps*]

lemma *fps_deriv_fls_regpart*: *fps_deriv* (*fls_regpart F*) = *fls_regpart* (*fls_deriv F*)
by (*intro fps_ext*) (*auto simp*: *add_ac*)

lemma *has_field_derivative_eval_fls*:
assumes *z* ∈ *eball 0* (*fls_conv_radius f*) − {*0*}
shows (*eval_fls f has_field_derivative eval_fls* (*fls_deriv f*) *z*) (*at z within A*)
proof −
define *g* **where** *g* = *fls_base_factor_to_fps f*
define *n* **where** *n* = *fls_subdegree f*
have [*simp*]: *fps_conv_radius g* = *fls_conv_radius f*

```
    by (simp add: fls_conv_radius_altdef g_def)
  have conv1: fps_conv_radius (fps_deriv g * fps_X) ≥ fls_conv_radius f
   by (intro fps_conv_radius_mult_ge order.trans[OF _ fps_conv_radius_deriv])
auto
  have conv2: fps_conv_radius (of_int n * g) ≥ fls_conv_radius f
    by (intro fps_conv_radius_mult_ge) auto
 have conv3: fps_conv_radius (fps_deriv g * fps_X + of_int n * g) ≥ fls_conv_radius
f
    by (intro fps_conv_radius_add_ge conv1 conv2)

  have [simp]: fps_conv_radius g = fls_conv_radius f
    by (simp add: g_def fls_conv_radius_altdef)
  have ((λz. eval_fps g z * z powi fls_subdegree f) has_field_derivative
        (eval_fps (fps_deriv g) z * z powi n + of_int n * z powi (n - 1) * eval_fps
g z))
         (at z within A)
    using assms by (auto intro!: derivative_eq_intros simp: n_def)
  also have (λz. eval_fps g z * z powi fls_subdegree f) = eval_fls f
    by (simp add: eval_fls_def g_def fun_eq_iff)
  also have eval_fps (fps_deriv g) z * z powi n + of_int n * z powi (n - 1) *
eval_fps g z =
            (z * eval_fps (fps_deriv g) z + of_int n * eval_fps g z) * z powi (n -
1)
    using assms by (auto simp: power_int_diff field_simps)
  also have (z * eval_fps (fps_deriv g) z + of_int n * eval_fps g z) =
             eval_fps (fps_deriv g * fps_X + of_int n * g) z
    using conv1 conv2 assms fps_conv_radius_deriv[of g]
    by (subst eval_fps_add) (auto simp: eval_fps_mult)
  also have ... = eval_fls (fps_to_fls (fps_deriv g * fps_X + of_int n * g)) z
    using conv3 assms by (subst eval_fps_to_fls) auto
   also have ... * z powi (n - 1) = eval_fls (fls_shift (1 - n) (fps_to_fls
(fps_deriv g * fps_X + of_int n * g))) z
    using assms by (subst eval_fls_shift) auto
  also have fls_shift (1 - n) (fps_to_fls (fps_deriv g * fps_X + of_int n * g))
= fls_deriv f
     by (intro fls_eqI) (auto simp: g_def n_def algebra_simps eq_commute[of _
fls_subdegree f])
  finally show ?thesis .
qed

lemma eval_fls_deriv:
  assumes z ∈ eball 0 (fls_conv_radius F) - {0}
  shows   eval_fls (fls_deriv F) z = deriv (eval_fls F) z
  by (rule sym, rule DERIV_imp_deriv, rule has_field_derivative_eval_fls, rule
assms)

lemma analytic_on_eval_fls:
  assumes A ⊆ eball 0 (fls_conv_radius f) - (if fls_subdegree f ≥ 0 then {} else
{0})
```

shows *eval_fls f analytic_on A*
proof (*rule analytic_on_subset* [*OF _ assms*])
show *eval_fls f analytic_on eball 0* (*fls_conv_radius f*) − (*if fls_subdegree f* ≥ *0 then* {} *else* {*0*})
using *holomorphic_on_eval_fls*[*OF order.refl*]
by (*subst analytic_on_open*) *auto*
qed

lemma *analytic_on_eval_fls'* [*analytic_intros*]:
assumes *g analytic_on A*
assumes *g ' A* ⊆ *eball 0* (*fls_conv_radius f*) − (*if fls_subdegree f* ≥ *0 then* {} *else* {*0*})
shows (λ*x*. *eval_fls f* (*g x*)) *analytic_on A*
proof −
have *eval_fls f* ∘ *g analytic_on A*
by (*intro analytic_on_compose*[*OF assms*(*1*) *analytic_on_eval_fls*]) (*use assms* **in** *auto*)
thus *?thesis*
by (*simp add*: *o_def*)
qed

lemma *continuous_eval_fls* [*continuous_intros*]:
assumes *z* ∈ *eball 0* (*fls_conv_radius F*) − (*if fls_subdegree F* ≥ *0 then* {} *else* {*0*})
shows *continuous* (*at z within A*) (*eval_fls F*)
proof −
have *isCont* (*eval_fls F*) *z*
using *continuous_on_eval_fls*[*OF order.refl*] *assms*
by (*subst* (*asm*) *continuous_on_eq_continuous_at*) *auto*
thus *?thesis*
using *continuous_at_imp_continuous_at_within* **by** *blast*
qed

named_theorems *laurent_expansion_intros*

lemma *has_laurent_expansion_imp_asymp_equiv_0*:
assumes *F*: *f has_laurent_expansion F*
defines *n* ≡ *fls_subdegree F*
shows *f* ∼[*at 0*] (λ*z*. *fls_nth F n* ∗ *z powi n*)
proof (*cases F* = *0*)
case *True*
thus *?thesis* **using** *assms*
by (*auto simp*: *has_laurent_expansion_def*)
next
case [*simp*]: *False*
define *G* **where** *G* = *fls_base_factor_to_fps F*

```
  have fls_conv_radius F > 0
    using F by (auto simp: has_laurent_expansion_def)
  hence isCont (eval_fps G) 0
    by (intro continuous_intros) (auto simp: G_def fps_conv_radius_fls_regpart
zero_ereal_def)
  hence lim: eval_fps G -0→ eval_fps G 0
    by (meson isContD)
  have [simp]: fps_nth G 0 ≠ 0
    by (auto simp: G_def)

  have f ∼[at 0] eval_fls F
   using F by (intro asymp_equiv_refl_ev) (auto simp: has_laurent_expansion_def
eq_commute)
  also have ... = (λz. eval_fps G z * z powi n)
    by (intro ext) (simp_all add: eval_fls_def G_def n_def)
  also have ... ∼[at 0] (λz. fps_nth G 0 * z powi n) using lim
    by (intro asymp_equiv_intros tendsto_imp_asymp_equiv_const) (auto simp:
eval_fps_at_0)
  also have fps_nth G 0 = fls_nth F n
    by (simp add: G_def n_def)
  finally show ?thesis
    by simp
qed

lemma has_laurent_expansion_imp_asymp_equiv:
  assumes F: (λw. f (z + w)) has_laurent_expansion F
  defines n ≡ fls_subdegree F
  shows   f ∼[at z] (λw. fls_nth F n * (w - z) powi n)
   using has_laurent_expansion_imp_asymp_equiv_0[OF assms(1)] unfolding
n_def
  by (simp add: at_to_0[of z] asymp_equiv_filtermap_iff add_ac)


lemmas [tendsto_intros del] = tendsto_power_int

lemma has_laurent_expansion_imp_tendsto_0:
  assumes F: f has_laurent_expansion F and fls_subdegree F ≥ 0
  shows   f -0→ fls_nth F 0
proof (rule asymp_equiv_tendsto_transfer)
  show (λz. fls_nth F (fls_subdegree F) * z powi fls_subdegree F) ∼[at 0] f
   by (rule asymp_equiv_symI, rule has_laurent_expansion_imp_asymp_equiv_0)
fact
  show (λz. fls_nth F (fls_subdegree F) * z powi fls_subdegree F) -0→ fls_nth
F 0
    by (rule tendsto_eq_intros refl | use assms(2) in simp)+
       (use assms(2) in ‹auto simp: power_int_0_left_If›)
qed

lemma has_laurent_expansion_imp_tendsto:
```

assumes F: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion* F **and** *fls_subdegree* $F \geq 0$
shows $f \ -z\to$ *fls_nth* $F\ 0$
using *has_laurent_expansion_imp_tendsto_0*[*OF assms*]
by (*simp add*: *at_to_0*[*of z*] *filterlim_filtermap add_ac*)

lemma *has_laurent_expansion_imp_filterlim_infinity_0*:
assumes F: *f has_laurent_expansion F* **and** *fls_subdegree* $F < 0$
shows *filterlim f at_infinity* (*at 0*)
proof (*rule asymp_equiv_at_infinity_transfer*)
have [*simp*]: $F \neq 0$
using *assms*(*2*) **by** *auto*
show $(\lambda z.$ *fls_nth F* (*fls_subdegree F*) $*$ *z powi fls_subdegree F*$) \sim[$*at 0*$]$ *f*
by (*rule asymp_equiv_symI*, *rule has_laurent_expansion_imp_asymp_equiv_0*) *fact*
show *filterlim* $(\lambda z.$ *fls_nth F* (*fls_subdegree F*) $*$ *z powi fls_subdegree F*) *at_infinity* (*at 0*)
by (*rule tendsto_mult_filterlim_at_infinity tendsto_const*
filterlim_power_int_neg_at_infinity | *use assms*(*2*) **in** *simp*)+
(*auto simp*: *eventually_at_filter*)
qed

lemma *has_laurent_expansion_imp_neg_fls_subdegree*:
assumes F: *f has_laurent_expansion F*
and *infy*:*filterlim f at_infinity* (*at 0*)
shows *fls_subdegree* $F < 0$
proof (*rule ccontr*)
assume *asm*:$\neg$ *fls_subdegree* $F < 0$
define *ff* **where** *ff*=$(\lambda z.$ *fls_nth F* (*fls_subdegree F*)
$*$ *z powi fls_subdegree F*)

have (*ff* $\longrightarrow$ (*if fls_subdegree F* $=0$ *then fls_nth F 0 else 0*)) (*at 0*)
using *asm* **unfolding** *ff_def*
by (*auto intro*!: *tendsto_eq_intros*)
moreover have *filterlim ff at_infinity* (*at 0*)
proof (*rule asymp_equiv_at_infinity_transfer*)
show *f* $\sim[$*at 0*$]$ *ff* **unfolding** *ff_def*
using *has_laurent_expansion_imp_asymp_equiv_0*[*OF F*] **unfolding** *ff_def*
.
show *filterlim f at_infinity* (*at 0*) **by** *fact*
qed
ultimately show *False*
using *not_tendsto_and_filterlim_at_infinity*[*of at* (*0*::*complex*)] **by** *auto*
qed

lemma *has_laurent_expansion_imp_filterlim_infinity*:
assumes F: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion* F **and** *fls_subdegree* $F < 0$
shows *filterlim f at_infinity* (*at z*)

using *has_laurent_expansion_imp_filterlim_infinity_0*[*OF assms*]
by (*simp add*: *at_to_0*[*of z*] *filterlim_filtermap add_ac*)

lemma *has_laurent_expansion_imp_is_pole_0*:
assumes *F*: *f has_laurent_expansion F* **and** *fls_subdegree F* < *0*
shows *is_pole f 0*
using *has_laurent_expansion_imp_filterlim_infinity_0*[*OF assms*]
by (*simp add*: *is_pole_def*)

lemma *is_pole_0_imp_neg_fls_subdegree*:
assumes *F*: *f has_laurent_expansion F* **and** *is_pole f 0*
shows *fls_subdegree F* < *0*
using *F assms*(*2*) *has_laurent_expansion_imp_neg_fls_subdegree is_pole_def*
by *blast*

lemma *has_laurent_expansion_imp_is_pole*:
assumes *F*: (λ*x*. *f* (*z* + *x*)) *has_laurent_expansion F* **and** *fls_subdegree F* < *0*
shows *is_pole f z*
using *has_laurent_expansion_imp_is_pole_0*[*OF assms*]
by (*simp add*: *is_pole_shift_0'*)

lemma *is_pole_imp_neg_fls_subdegree*:
assumes *F*: (λ*x*. *f* (*z* + *x*)) *has_laurent_expansion F* **and** *is_pole f z*
shows *fls_subdegree F* < *0*
apply (*rule is_pole_0_imp_neg_fls_subdegree*[*OF F*])
using *assms*(*2*) *is_pole_shift_0* **by** *blast*

lemma *is_pole_fls_subdegree_iff*:
assumes (λ*x*. *f* (*z* + *x*)) *has_laurent_expansion F*
shows *is_pole f z* ⟷ *fls_subdegree F* < *0*
using *assms is_pole_imp_neg_fls_subdegree has_laurent_expansion_imp_is_pole*
by *auto*

lemma
assumes *f has_laurent_expansion F*
shows *has_laurent_expansion_isolated_0*: *isolated_singularity_at f 0*
and *has_laurent_expansion_not_essential_0*: *not_essential f 0*
proof −
from *assms* **have** *eventually* (λ*z*. *eval_fls F z* = *f z*) (*at 0*)
by (*auto simp*: *has_laurent_expansion_def*)
then obtain *r* **where** *r*: *r* > *0* ⋀*z*. *z* ∈ *ball 0 r* − {*0*} ⟹ *eval_fls F z* = *f z*
by (*auto simp*: *eventually_at_filter ball_def eventually_nhds_metric*)

have *fls_conv_radius F* > *0*
using *assms* **by** (*auto simp*: *has_laurent_expansion_def*)
then obtain *R* :: *real* **where** *R*: *R* > *0 R* ≤ *min r* (*fls_conv_radius F*)
using ‹*r* > *0*› **by** (*metis dual_order.strict_implies_order ereal_dense2 ereal_less*(*2*) *min_def*)

```
  have eval_fls F holomorphic_on ball 0 R − {0}
    using r R by (intro holomorphic_intros ball_eball_mono Diff_mono) (auto
simp: ereal_le_less)
  also have ?this ⟷ f holomorphic_on ball 0 R − {0}
    using r R by (intro holomorphic_cong) auto
  also have ... ⟷ f analytic_on ball 0 R − {0}
    by (subst analytic_on_open) auto
  finally show isolated_singularity_at f 0
    unfolding isolated_singularity_at_def using ‹R > 0› by blast

  show not_essential f 0
  proof (cases fls_subdegree F ≥ 0)
    case True
    hence f −0→ fls_nth F 0
      by (intro has_laurent_expansion_imp_tendsto_0[OF assms])
    thus ?thesis
      by (auto simp: not_essential_def)
  next
    case False
    hence is_pole f 0
      by (intro has_laurent_expansion_imp_is_pole_0[OF assms]) auto
    thus ?thesis
      by (auto simp: not_essential_def)
  qed
qed

lemma
  assumes (λw. f (z + w)) has_laurent_expansion F
  shows   has_laurent_expansion_isolated: isolated_singularity_at f z
    and   has_laurent_expansion_not_essential: not_essential f z
 using has_laurent_expansion_isolated_0[OF assms] has_laurent_expansion_not_essential_0[OF
assms]
  by (simp_all add: isolated_singularity_at_shift_0 not_essential_shift_0)

lemma has_laurent_expansion_fps:
  assumes f has_fps_expansion F
  shows   f has_laurent_expansion fps_to_fls F
proof −
  from assms have radius: 0 < fps_conv_radius F and eval: ∀F z in nhds 0.
eval_fps F z = f z
    by (auto simp: has_fps_expansion_def)
  from eval have eval': ∀F z in at 0. eval_fps F z = f z
    using eventually_at_filter eventually_mono by fastforce
  moreover have eventually (λz. z ∈ eball 0 (fps_conv_radius F) − {0}) (at 0)
    using radius by (intro eventually_at_in_open) (auto simp: zero_ereal_def)
  ultimately have eventually (λz. eval_fls (fps_to_fls F) z = f z) (at 0)
    by eventually_elim (auto simp: eval_fps_to_fls)
  thus ?thesis using radius
    by (auto simp: has_laurent_expansion_def)
```

qed

lemma *has_laurent_expansion_const* [*simp*, *intro*, *laurent_expansion_intros*]:
(λ_. *c*) *has_laurent_expansion fls_const c*
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_0* [*simp*, *intro*, *laurent_expansion_intros*]:
(λ_. *0*) *has_laurent_expansion 0*
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_fps_expansion_0_iff*: *f has_fps_expansion 0* ⟷ *eventually* (λ*z*. *f z* = *0*) (*nhds 0*)
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_laurent_expansion_1* [*simp*, *intro*, *laurent_expansion_intros*]:
(λ_. *1*) *has_laurent_expansion 1*
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_numeral* [*simp*, *intro*, *laurent_expansion_intros*]:
(λ_. *numeral n*) *has_laurent_expansion numeral n*
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_fps_X_power* [*laurent_expansion_intros*]:
(λ*x*. *x* ^ *n*) *has_laurent_expansion* (*fls_X_intpow n*)
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_fps_X_power_int* [*laurent_expansion_intros*]:
(λ*x*. *x powi n*) *has_laurent_expansion* (*fls_X_intpow n*)
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_fps_X* [*laurent_expansion_intros*]:
(λ*x*. *x*) *has_laurent_expansion fls_X*
by (*auto simp*: *has_laurent_expansion_def*)

lemma *has_laurent_expansion_cmult_left* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows (λ*x*. *c* ∗ *f x*) *has_laurent_expansion fls_const c* ∗ *F*
proof −
from *assms* **have** *eventually* (λ*z*. *z* ∈ *eball 0* (*fls_conv_radius F*) − {*0*}) (*at 0*)
by (*intro eventually_at_in_open*) (*auto simp*: *has_laurent_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* (λ*z*. *eval_fls F z* = *f z*) (*at 0*)
by (*auto simp*: *has_laurent_expansion_def*)
ultimately have *eventually* (λ*z*. *eval_fls* (*fls_const c* ∗ *F*) *z* = *c* ∗ *f z*) (*at 0*)
by *eventually_elim* (*simp_all add*: *eval_fls_mult*)
with *assms* **show** *?thesis*
by (*auto simp*: *has_laurent_expansion_def intro*!: *less_le_trans*[*OF* _ *fls_conv_radius_mult*])
qed

lemma *has_laurent_expansion_cmult_right* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows (λx. *f x* ∗ *c*) *has_laurent_expansion F* ∗ *fls_const c*
proof −
have *F* ∗ *fls_const c* = *fls_const c* ∗ *F*
by (*intro fls_eqI*) (*auto simp*: *mult.commute*)
with *has_laurent_expansion_cmult_left* [*OF assms*] **show** *?thesis*
by (*simp add*: *mult.commute*)
qed

lemma *has_fps_expansion_scaleR* [*fps_expansion_intros*]:
fixes *F* :: *'a* :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*} *fps*
shows *f has_fps_expansion F* ⟹ (λx. *c* ∗*R f x*) *has_fps_expansion fps_const* (*of_real c*) ∗ *F*
unfolding *scaleR_conv_of_real* **by** (*intro fps_expansion_intros*)

lemma *has_laurent_expansion_scaleR* [*laurent_expansion_intros*]:
f has_laurent_expansion F ⟹ (λx. *c* ∗*R f x*) *has_laurent_expansion fls_const* (*of_real c*) ∗ *F*
unfolding *scaleR_conv_of_real* **by** (*intro laurent_expansion_intros*)

lemma *has_laurent_expansion_minus* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows (λx. − *f x*) *has_laurent_expansion* −*F*
proof −
from *assms* **have** *eventually* (λx. *x* ∈ *eball 0* (*fls_conv_radius F*) − {*0*}) (*at 0*)
by (*intro eventually_at_in_open*) (*auto simp*: *has_laurent_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* (λx. *eval_fls F x* = *f x*) (*at 0*)
by (*auto simp*: *has_laurent_expansion_def*)
ultimately have *eventually* (λx. *eval_fls* (−*F*) *x* = −*f x*) (*at 0*)
by *eventually_elim* (*auto simp*: *eval_fls_minus*)
thus *?thesis* **using** *assms* **by** (*auto simp*: *has_laurent_expansion_def*)
qed

lemma *has_laurent_expansion_add* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F g has_laurent_expansion G*
shows (λx. *f x* + *g x*) *has_laurent_expansion F* + *G*
proof −
from *assms* **have** *0* < *min* (*fls_conv_radius F*) (*fls_conv_radius G*)
by (*auto simp*: *has_laurent_expansion_def*)
also have ... ≤ *fls_conv_radius* (*F* + *G*)
by (*rule fls_conv_radius_add*)
finally have *radius*: ... > *0* **.**

from *assms* **have** *eventually* (λx. *x* ∈ *eball 0* (*fls_conv_radius F*) − {*0*}) (*at 0*)
eventually (λx. *x* ∈ *eball 0* (*fls_conv_radius G*) − {*0*}) (*at 0*)

by (*intro eventually_at_in_open*; *force simp*: *has_laurent_expansion_def zero_ereal_def*)+
moreover have *eventually* (λ*x*. *eval_fls F x* = *f x*) (*at 0*)
and *eventually* (λ*x*. *eval_fls G x* = *g x*) (*at 0*)
using *assms* **by** (*auto simp*: *has_laurent_expansion_def*)
ultimately have *eventually* (λ*x*. *eval_fls* (*F* + *G*) *x* = *f x* + *g x*) (*at 0*)
by *eventually_elim* (*auto simp*: *eval_fls_add*)
with *radius* **show** *?thesis* **by** (*auto simp*: *has_laurent_expansion_def*)
qed

lemma *has_laurent_expansion_diff* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F g has_laurent_expansion G*
shows (λ*x*. *f x* − *g x*) *has_laurent_expansion F* − *G*
using *has_laurent_expansion_add*[*of f F* λ*x*. − *g x* −*G*] *assms*
by (*simp add*: *has_laurent_expansion_minus*)

lemma *has_laurent_expansion_mult* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F g has_laurent_expansion G*
shows (λ*x*. *f x* ∗ *g x*) *has_laurent_expansion F* ∗ *G*
proof −
from *assms* **have** *0* < *min* (*fls_conv_radius F*) (*fls_conv_radius G*)
by (*auto simp*: *has_laurent_expansion_def*)
also have ... ≤ *fls_conv_radius* (*F* ∗ *G*)
by (*rule fls_conv_radius_mult*)
finally have *radius*: ... > *0* **.**

from *assms* **have** *eventually* (λ*x*. *x* ∈ *eball 0* (*fls_conv_radius F*) − {*0*}) (*at 0*)
eventually (λ*x*. *x* ∈ *eball 0* (*fls_conv_radius G*) − {*0*}) (*at 0*)
by (*intro eventually_at_in_open*; *force simp*: *has_laurent_expansion_def zero_ereal_def*)+
moreover have *eventually* (λ*x*. *eval_fls F x* = *f x*) (*at 0*)
and *eventually* (λ*x*. *eval_fls G x* = *g x*) (*at 0*)
using *assms* **by** (*auto simp*: *has_laurent_expansion_def*)
ultimately have *eventually* (λ*x*. *eval_fls* (*F* ∗ *G*) *x* = *f x* ∗ *g x*) (*at 0*)
by *eventually_elim* (*auto simp*: *eval_fls_mult*)
with *radius* **show** *?thesis* **by** (*auto simp*: *has_laurent_expansion_def*)
qed

lemma *has_fps_expansion_power* [*fps_expansion_intros*]:
fixes *F* :: *'a* :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*} *fps*
shows *f has_fps_expansion F* ⟹ (λ*x*. *f x* ^ *m*) *has_fps_expansion F* ^ *m*
by (*induction m*) (*auto intro*!: *fps_expansion_intros*)

lemma *has_laurent_expansion_power* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows (λ*x*. *f x* ^ *n*) *has_laurent_expansion F* ^ *n*
by (*induction n*) (*auto intro*!: *laurent_expansion_intros assms*)

lemma *has_laurent_expansion_sum* [*laurent_expansion_intros*]:
assumes ⋀*x*. *x* ∈ *I* ⟹ *f x has_laurent_expansion F x*

shows $(\lambda y.\ \sum x{\in}I.\ f\ x\ y)$ *has_laurent_expansion* $(\sum x{\in}I.\ F\ x)$
using *assms* **by** (*induction I rule*: *infinite_finite_induct*) (*auto intro*!: *laurent_expansion_intros*)

lemma *has_laurent_expansion_prod* [*laurent_expansion_intros*]:
assumes $\bigwedge x.\ x \in I \Longrightarrow f\ x$ *has_laurent_expansion* $F\ x$
shows $(\lambda y.\ \prod x{\in}I.\ f\ x\ y)$ *has_laurent_expansion* $(\prod x{\in}I.\ F\ x)$
using *assms* **by** (*induction I rule*: *infinite_finite_induct*) (*auto intro*!: *laurent_expansion_intros*)

lemma *has_laurent_expansion_deriv* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows *deriv f has_laurent_expansion fls_deriv F*
proof −
have *eventually* $(\lambda z.\ z \in$ *eball 0* (*fls_conv_radius F*) $-\ \{0\})$ (*at 0*)
using *assms* **by** (*intro eventually_at_in_open*)
(*auto simp*: *has_laurent_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* $(\lambda z.$ *eval_fls F z* $= f\ z)$ (*at 0*)
by (*auto simp*: *has_laurent_expansion_def*)
then obtain *s* **where** *open s 0* $\in$ *s* **and** *s*: $\bigwedge w.\ w \in s - \{0\} \Longrightarrow$ *eval_fls F w* $= f\ w$
by (*auto simp*: *eventually_nhds eventually_at_filter*)
hence *eventually* $(\lambda w.\ w \in s - \{0\})$ (*at 0*)
by (*intro eventually_at_in_open*) *auto*
ultimately have *eventually* $(\lambda z.$ *eval_fls* (*fls_deriv F*) $z =$ *deriv f z*) (*at 0*)
proof *eventually_elim*
case (*elim z*)
hence *eval_fls* (*fls_deriv F*) $z =$ *deriv* (*eval_fls F*) z
by (*simp add*: *eval_fls_deriv*)
also have *eventually* $(\lambda w.\ w \in s - \{0\})$ (*nhds z*)
using *elim* **and** ‹*open s*› **by** (*intro eventually_nhds_in_open*) *auto*
hence *eventually* $(\lambda w.$ *eval_fls F w* $= f\ w)$ (*nhds z*)
by *eventually_elim* (*use s* **in** *auto*)
hence *deriv* (*eval_fls F*) $z =$ *deriv f z*
by (*intro deriv_cong_ev refl*)
finally show *?case* **.**
qed
with *assms* **show** *?thesis*
by (*auto simp*: *has_laurent_expansion_def intro*!: *less_le_trans*[*OF* _ *fls_conv_radius_deriv*])
qed

lemma *has_laurent_expansion_shift* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows $(\lambda x.\ f\ x * x\ powi\ n)$ *has_laurent_expansion* (*fls_shift* $(-n)$ *F*)
proof −
have *eventually* $(\lambda x.\ x \in$ *eball 0* (*fls_conv_radius F*) $-\ \{0\})$ (*at 0*)
using *assms* **by** (*intro eventually_at_in_open*) (*auto simp*: *has_laurent_expansion_def zero_ereal_def*)
moreover have *eventually* $(\lambda x.$ *eval_fls F x* $= f\ x)$ (*at 0*)
using *assms* **by** (*auto simp*: *has_laurent_expansion_def*)
ultimately have *eventually* $(\lambda x.$ *eval_fls* (*fls_shift* $(-n)$ *F*) $x = f\ x * x\ powi$

```
n) (at 0)
    by eventually_elim (auto simp: eval_fls_shift assms)
  with assms show ?thesis by (auto simp: has_laurent_expansion_def)
qed

lemma has_laurent_expansion_shift' [laurent_expansion_intros]:
  assumes f has_laurent_expansion F
  shows   (λx. f x * x powi (−n)) has_laurent_expansion (fls_shift n F)
  using has_laurent_expansion_shift[OF assms, of −n] by simp


lemma has_laurent_expansion_deriv':
  assumes f has_laurent_expansion F
  assumes open A 0 ∈ A ⋀x. x ∈ A − {0} ⟹ (f has_field_derivative f' x) (at
x)
  shows   f' has_laurent_expansion fls_deriv F
proof −
  have deriv f has_laurent_expansion fls_deriv F
    by (intro laurent_expansion_intros assms)
  also have ?this ⟷ ?thesis
  proof (intro has_laurent_expansion_cong refl)
    have eventually (λz. z ∈ A − {0}) (at 0)
      by (intro eventually_at_in_open assms)
    thus eventually (λz. deriv f z = f' z) (at 0)
      by eventually_elim (auto intro!: DERIV_imp_deriv assms)
  qed
  finally show ?thesis .
qed

definition laurent_expansion :: (complex ⇒ complex) ⇒ complex ⇒ complex fls
where
  laurent_expansion f z =
     (if eventually (λz. f z = 0) (at z) then 0
      else fls_shift (−zorder f z) (fps_to_fls (fps_expansion (zor_poly f z) z)))

lemma laurent_expansion_cong:
  assumes eventually (λw. f w = g w) (at z) z = z'
  shows   laurent_expansion f z = laurent_expansion g z'
  unfolding laurent_expansion_def
  using zor_poly_cong[OF assms(1,2)] zorder_cong[OF assms] assms
  by (intro if_cong refl) (auto elim: eventually_elim2)

theorem not_essential_has_laurent_expansion_0:
  assumes isolated_singularity_at f 0 not_essential f 0
  shows   f has_laurent_expansion laurent_expansion f 0
proof (cases ∃ F w in at 0. f w ≠ 0)
  case False
  have (λ_. 0) has_laurent_expansion 0
    by simp
```

```
  also have ?this ⟷ f has_laurent_expansion 0
   using False by (intro has_laurent_expansion_cong) (auto simp: frequently_def)
  finally show ?thesis
    using False by (simp add: laurent_expansion_def frequently_def)
next
  case True
  define n where n = zorder f 0
  obtain r where r: zor_poly f 0 0 ≠ 0 zor_poly f 0 holomorphic_on cball 0 r r
> 0
                    ∀w∈cball 0 r - {0}. f w = zor_poly f 0 w * w powi n ∧
                                             zor_poly f 0 w ≠ 0
    using zorder_exist[OF assms True] unfolding n_def by auto
  have holo: zor_poly f 0 holomorphic_on ball 0 r
    by (rule holomorphic_on_subset[OF r(2)]) auto

  define F where F = fps_expansion (zor_poly f 0) 0
  have F: zor_poly f 0 has_fps_expansion F
    unfolding F_def by (rule has_fps_expansion_fps_expansion[OF _ _ holo])
(use ‹r > 0› in auto)
   have (λz. zor_poly f 0 z * z powi n) has_laurent_expansion fls_shift (-n)
(fps_to_fls F)
    by (intro laurent_expansion_intros has_laurent_expansion_fps[OF F])
  also have ?this ⟷ f has_laurent_expansion fls_shift (-n) (fps_to_fls F)
   by (intro has_laurent_expansion_cong refl eventually_mono[OF eventually_at_in_open[of
ball 0 r]])
      (use r in ‹auto simp: complex_powr_of_int›)
  finally show ?thesis using True
    by (simp add: laurent_expansion_def F_def n_def frequently_def)
qed

lemma not_essential_has_laurent_expansion:
  assumes isolated_singularity_at f z not_essential f z
  shows   (λx. f (z + x)) has_laurent_expansion laurent_expansion f z
proof -
  from assms(1) have iso:isolated_singularity_at (λx. f (z + x)) 0
    by (simp add: isolated_singularity_at_shift_0)
  moreover from assms(2) have ness:not_essential (λx. f (z + x)) 0
    by (simp add: not_essential_shift_0)
 ultimately have (λx. f (z + x)) has_laurent_expansion laurent_expansion (λx.
f (z + x)) 0
    by (rule not_essential_has_laurent_expansion_0)

  also have ... = laurent_expansion f z
  proof (cases ∃F w in at z. f w ≠ 0)
    case False
    then have ∀F w in at z. f w = 0 using not_frequently by force
    then have laurent_expansion (λx. f (z + x)) 0 = 0
      by (smt (verit, best) add.commute eventually_at_to_0 eventually_mono
          laurent_expansion_def)
```

```
  moreover have laurent_expansion f z = 0
    using ‹∀F w in at z. f w = 0› unfolding laurent_expansion_def by auto
  ultimately show ?thesis by auto
next
  case True
  define df where df=zor_poly (λx. f (z + x)) 0
  define g where g=(λu. u−z)

  have fps_expansion df 0
     = fps_expansion (df ∘ g) z
  proof −
    have ∃F w in at 0. f (z + w) ≠ 0 using True
      by (smt (verit, best) add.commute eventually_at_to_0
          eventually_mono not_frequently)
    from zorder_exist[OF iso ness this,folded df_def]
    obtain r where r>0 and df_holo:df holomorphic_on cball 0 r and df 0 ≠
0
        ∀w∈cball 0 r − {0}.
          f (z + w) = df w ∗ w powi (zorder (λw. f (z + w)) 0) ∧
          df w ≠ 0
      by auto
    then have df_nz:∀w∈ball 0 r. df w≠0 by auto

    have (deriv ^^ n) df 0 = (deriv ^^ n) (df ∘ g) z for n
      unfolding comp_def g_def
  proof (subst higher_deriv_compose_linear'[where u=1 and c=−z,simplified])
      show df holomorphic_on ball 0 r
        using df_holo by auto
      show open (ball z r) open (ball 0 r) z ∈ ball z r
        using ‹r>0› by auto
      show ⋀w. w ∈ ball z r ⟹ w − z ∈ ball 0 r
        by (simp add: dist_norm)
    qed auto
    then show ?thesis
      unfolding fps_expansion_def by auto
  qed
  also have ... = fps_expansion (zor_poly f z) z
  proof (rule fps_expansion_cong)
    have ∀F w in nhds z. zor_poly f z w
            = zor_poly (λu. f (z + u)) 0 (w − z)
      apply (rule zor_poly_shift)
      using True assms by auto
    then show ∀F w in nhds z. (df ∘ g) w = zor_poly f z w
      unfolding df_def g_def comp_def
      by (auto elim:eventually_mono)
  qed
  finally show ?thesis unfolding df_def
    by (auto simp: laurent_expansion_def at_to_0[of z]
        eventually_filtermap add_ac zorder_shift')
```

```
  qed
  finally show ?thesis .
qed

lemma has_fps_expansion_to_laurent:
  f has_fps_expansion F ⟷ f has_laurent_expansion fps_to_fls F ∧ f 0 =
fps_nth F 0
proof safe
  assume ∗: f has_laurent_expansion fps_to_fls F f 0 = fps_nth F 0
  have eventually (λz. z ∈ eball 0 (fps_conv_radius F)) (nhds 0)
   using ∗ by (intro eventually_nhds_in_open) (auto simp: has_laurent_expansion_def
zero_ereal_def)
  moreover have eventually (λz. z ≠ 0 ⟶ eval_fls (fps_to_fls F) z = f z) (nhds
0)
    using ∗ by (auto simp: has_laurent_expansion_def eventually_at_filter)
  ultimately have eventually (λz. f z = eval_fps F z) (nhds 0)
    by eventually_elim
      (auto simp: has_laurent_expansion_def eventually_at_filter eval_fps_at_0
eval_fps_to_fls ∗(2))
  thus f has_fps_expansion F
    using ∗ by (auto simp: has_fps_expansion_def has_laurent_expansion_def
eq_commute)
next
  assume f has_fps_expansion F
  thus f 0 = fps_nth F 0
    by (metis eval_fps_at_0 has_fps_expansion_imp_holomorphic)
qed (auto intro: has_laurent_expansion_fps)

lemma eval_fps_fls_base_factor [simp]:
  assumes z ≠ 0
  shows   eval_fps (fls_base_factor_to_fps F) z = eval_fls F z ∗ z powi −fls_subdegree
F
  using assms unfolding eval_fls_def by (simp add: power_int_minus field_simps)

lemma has_fps_expansion_imp_analytic_0:
  assumes f has_fps_expansion F
  shows   f analytic_on {0}
  by (meson analytic_at_two assms has_fps_expansion_imp_holomorphic)

lemma has_fps_expansion_imp_analytic:
  assumes (λx. f (z + x)) has_fps_expansion F
  shows   f analytic_on {z}
proof −
  have (λx. f (z + x)) analytic_on {0}
    by (rule has_fps_expansion_imp_analytic_0) fact
  hence (λx. f (z + x)) ∘ (λx. x − z) analytic_on {z}
    by (intro analytic_on_compose_gen analytic_intros) auto
  thus ?thesis
    by (simp add: o_def)
```

qed

lemma *is_pole_cong_asymp_equiv*:
assumes $f \sim[at\ z]\ g$ $z = z'$
shows *is_pole f z = is_pole g z'*
using *asymp_equiv_at_infinity_transfer*[*OF assms*(*1*)]
asymp_equiv_at_infinity_transfer[*OF asymp_equiv_symI*[*OF assms*(*1*)]]
assms(*2*)
unfolding *is_pole_def* **by** *auto*

lemma *not_is_pole_const* [*simp*]: ¬*is_pole* (λ_::'*a*::*perfect_space*. *c* :: *complex*) *z*
using *not_tendsto_and_filterlim_at_infinity*[*of at z* λ_::'*a*. *c c*] **by** (*auto simp*: *is_pole_def*)

lemma *has_laurent_expansion_imp_is_pole_iff*:
assumes *F*: (λ*x*. *f* (*z* + *x*)) *has_laurent_expansion F*
shows *is_pole f z* ⟷ *fls_subdegree F* < *0*
proof
assume *pole*: *is_pole f z*
have [*simp*]: *F* ≠ *0*
proof
assume *F* = *0*
hence *is_pole f z* ⟷ *is_pole* (λ_. *0* :: *complex*) *z* **using** *assms*
by (*intro is_pole_cong*)
(*auto simp*: *has_laurent_expansion_def at_to_0*[*of z*] *eventually_filtermap add_ac*)
with *pole* **show** *False*
by *simp*
qed

note *pole*
also have *is_pole f z* ⟷
is_pole (λ*w*. *fls_nth F* (*fls_subdegree F*) ∗ (*w* − *z*) *powi fls_subdegree F*) *z*
using *has_laurent_expansion_imp_asymp_equiv*[*OF F*] **by** (*intro is_pole_cong_asymp_equiv refl*)
also have ... ⟷ *is_pole* (λ*w*. (*w* − *z*) *powi fls_subdegree F*) *z*
by *simp*
finally have *pole'*: ... **.**

have *False* **if** *fls_subdegree F* ≥ *0*
proof −
have (λ*w*. (*w* − *z*) *powi fls_subdegree F*) *holomorphic_on UNIV*
using *that* **by** (*intro holomorphic_intros*) *auto*
hence ¬*is_pole* (λ*w*. (*w* − *z*) *powi fls_subdegree F*) *z*
by (*meson UNIV_I not_is_pole_holomorphic open_UNIV*)
with *pole'* **show** *False*
by *simp*

```
  qed
  thus fls_subdegree F < 0
    by force
qed (use has_laurent_expansion_imp_is_pole[OF assms] in auto)

lemma analytic_at_imp_has_fps_expansion_0:
  assumes f analytic_on {0}
  shows   f has_fps_expansion fps_expansion f 0
  using assms has_fps_expansion_fps_expansion analytic_at by fast

lemma deriv_shift_0: deriv f z = deriv (f ∘ (λx. z + x)) 0
proof −
  have ∗: (f ∘ (+) z has_field_derivative D) (at z')
    if (f has_field_derivative D) (at (z + z')) for D z z' and f :: 'a ⇒ 'a
  proof −
    have (f ∘ (+) z has_field_derivative D ∗ 1) (at z')
      by (rule DERIV_chain that derivative_eq_intros refl)+ auto
    thus ?thesis by simp
  qed
 have (λD. (f has_field_derivative D) (at z)) = (λ D. (f ∘ (+) z has_field_derivative
D) (at 0))
    using ∗[of f _ z 0] ∗[of f ∘ (+) z _ −z z] by (intro ext iffI) (auto simp: o_def)
  thus ?thesis
    by (simp add: deriv_def)
qed

lemma deriv_shift_0': NO_MATCH 0 z ⟹ deriv f z = deriv (f ∘ (λx. z + x))
0
  by (rule deriv_shift_0)

lemma higher_deriv_shift_0: (deriv ^^ n) f z = (deriv ^^ n) (f ∘ (λx. z + x)) 0
proof (induction n arbitrary: f)
  case (Suc n)
  have (deriv ^^ Suc n) f z = (deriv ^^ n) (deriv f) z
    by (subst funpow_Suc_right) auto
  also have ... = (deriv ^^ n) (λx. deriv f (z + x)) 0
    by (subst Suc) (auto simp: o_def)
  also have ... = (deriv ^^ n) (λx. deriv (λxa. f (z + x + xa)) 0) 0
    by (subst deriv_shift_0) (auto simp: o_def)
  also have (λx. deriv (λxa. f (z + x + xa)) 0) = deriv (λx. f (z + x))
    by (rule ext) (simp add: deriv_shift_0' o_def add_ac)
  also have (deriv ^^ n) ... 0 = (deriv ^^ Suc n) (f ∘ (λx. z + x)) 0
    by (subst funpow_Suc_right) (auto simp: o_def)
  finally show ?case .
qed auto

lemma higher_deriv_shift_0': NO_MATCH 0 z ⟹ (deriv ^^ n) f z = (deriv ^^
n) (f ∘ (λx. z + x)) 0
  by (rule higher_deriv_shift_0)
```

lemma *analytic_at_imp_has_fps_expansion*:
assumes *f analytic_on* {*z*}
shows (λ*x*. *f* (*z* + *x*)) *has_fps_expansion fps_expansion f z*
proof −
have *f* ∘ (λ*x*. *z* + *x*) *analytic_on* {*0*}
by (*intro analytic_on_compose_gen*[*OF* _ *assms*] *analytic_intros*) *auto*
hence (*f* ∘ (λ*x*. *z* + *x*)) *has_fps_expansion fps_expansion* (*f* ∘ (λ*x*. *z* + *x*)) *0*
unfolding *o_def* **by** (*intro analytic_at_imp_has_fps_expansion_0*) *auto*
also have ... = *fps_expansion f z*
by (*simp add*: *fps_expansion_def higher_deriv_shift_0'*)
finally show *?thesis* **by** (*simp add*: *add_ac*)
qed

lemma *has_laurent_expansion_zorder_0*:
assumes *f has_laurent_expansion F F* ≠ *0*
shows *zorder f 0* = *fls_subdegree F*
proof −
define *G* **where** *G* = *fls_base_factor_to_fps F*
from *assms* **obtain** *A* **where** *A*: *0* ∈ *A open A* ⋀*x*. *x* ∈ *A* − {*0*} ⟹ *eval_fls F x* = *f x*
unfolding *has_laurent_expansion_def eventually_at_filter eventually_nhds*
by *blast*

show *?thesis*
proof (*rule zorder_eqI*)
show *open* (*A* ∩ *eball 0* (*fls_conv_radius F*)) *0* ∈ *A* ∩ *eball 0* (*fls_conv_radius F*)
using *assms A* **by** (*auto simp*: *has_laurent_expansion_def zero_ereal_def*)
show *eval_fps G holomorphic_on A* ∩ *eball 0* (*fls_conv_radius F*)
by (*intro holomorphic_intros*) (*auto simp*: *fls_conv_radius_altdef G_def*)
show *eval_fps G 0* ≠ *0* **using** ‹*F* ≠ *0*›
by (*auto simp*: *eval_fps_at_0 G_def*)
next
fix *w* :: *complex* **assume** *w* ∈ *A* ∩ *eball 0* (*fls_conv_radius F*) *w* ≠ *0*
thus *f w* = *eval_fps G w* ∗ (*w* − *0*) *powi* (*fls_subdegree F*)
using *A* **unfolding** *G_def*
by (*subst eval_fps_fls_base_factor*)
(*auto simp*: *complex_powr_of_int power_int_minus field_simps*)
qed
qed

lemma *has_laurent_expansion_zorder*:
assumes (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F F* ≠ *0*
shows *zorder f z* = *fls_subdegree F*
using *has_laurent_expansion_zorder_0*[*OF assms*] **by** (*simp add*: *zorder_shift' add_ac*)

lemma *has_fps_expansion_zorder_0*:

assumes *f has_fps_expansion F* $F \neq 0$
shows *zorder f 0 = int (subdegree F)*
using *assms has_laurent_expansion_zorder_0*[*of f fps_to_fls F*]
by (*auto simp*: *has_fps_expansion_to_laurent fls_subdegree_fls_to_fps*)

lemma *has_fps_expansion_zorder*:
assumes (*λw. f (z + w)*) *has_fps_expansion F* $F \neq 0$
shows *zorder f z = int (subdegree F)*
using *has_fps_expansion_zorder_0*[*OF assms*]
by (*simp add*: *zorder_shift' add_ac*)

lemma *has_fps_expansion_fls_base_factor_to_fps*:
assumes *f has_laurent_expansion F*
defines *n* ≡ *fls_subdegree F*
defines *c* ≡ *fps_nth* (*fls_base_factor_to_fps F*) *0*
shows (*λz. if z = 0 then c else f z * z powi −n*) *has_fps_expansion fls_base_factor_to_fps F*
proof −
have (*λz. f z * z powi −n*) *has_laurent_expansion fls_shift* (*−*(*−n*)) *F*
by (*intro laurent_expansion_intros assms*)
also have *fls_shift* (*−*(*−n*)) *F* = *fps_to_fls* (*fls_base_factor_to_fps F*)
by (*simp add*: *n_def fls_shift_nonneg_subdegree*)
also have (*λz. f z * z powi − n*) *has_laurent_expansion fps_to_fls* (*fls_base_factor_to_fps F*) ⟷
(*λz. if z = 0 then c else f z * z powi −n*) *has_laurent_expansion fps_to_fls* (*fls_base_factor_to_fps F*)
by (*intro has_laurent_expansion_cong*) (*auto simp*: *eventually_at_filter*)
also have ... ⟷ (*λz. if z = 0 then c else f z * z powi −n*) *has_fps_expansion fls_base_factor_to_fps F*
by (*subst has_fps_expansion_to_laurent*) (*auto simp*: *c_def*)
finally show *?thesis* **.**
qed

lemma *zero_has_laurent_expansion_imp_eq_0*:
assumes (*λ_. 0*) *has_laurent_expansion F*
shows *F = 0*
proof −
have *at* (*0* :: *complex*) ≠ *bot*
by *auto*
moreover have (*λz. if z = 0 then fls_nth F* (*fls_subdegree F*) *else 0*) *has_fps_expansion fls_base_factor_to_fps F* (**is** *?f has_fps_expansion _*)
using *has_fps_expansion_fls_base_factor_to_fps*[*OF assms*] **by** (*simp cong*: *if_cong*)
hence *isCont ?f 0*
using *has_fps_expansion_imp_continuous* **by** *blast*
hence *?f −0→ fls_nth F* (*fls_subdegree F*)
by (*auto simp*: *isCont_def*)
moreover have *?f −0→ 0* ⟷ (*λ_*::*complex. 0* :: *complex*) *−0→ 0*
by (*intro filterlim_cong*) (*auto simp*: *eventually_at_filter*)

```
  hence ?f −0→ 0
    by simp
  ultimately have fls_nth F (fls_subdegree F) = 0
    by (rule tendsto_unique)
  thus ?thesis
    by (meson nth_fls_subdegree_nonzero)
qed

lemma has_laurent_expansion_unique:
  assumes f has_laurent_expansion F f has_laurent_expansion G
  shows   F = G
proof −
  from assms have (λx. f x − f x) has_laurent_expansion F − G
    by (intro laurent_expansion_intros)
  hence (λ_. 0) has_laurent_expansion F − G
    by simp
  hence F − G = 0
    by (rule zero_has_laurent_expansion_imp_eq_0)
  thus ?thesis
    by simp
qed

lemma laurent_expansion_eqI:
  assumes (λx. f (z + x)) has_laurent_expansion F
  shows   laurent_expansion f z = F
 using assms has_laurent_expansion_isolated has_laurent_expansion_not_essential
       has_laurent_expansion_unique not_essential_has_laurent_expansion by
blast

lemma laurent_expansion_0_eqI:
  assumes f has_laurent_expansion F
  shows   laurent_expansion f 0 = F
  using assms laurent_expansion_eqI[of f 0] by simp

lemma has_laurent_expansion_nonzero_imp_eventually_nonzero:
  assumes f has_laurent_expansion F F ≠ 0
  shows   eventually (λx. f x ≠ 0) (at 0)
proof (rule ccontr)
  assume ¬eventually (λx. f x ≠ 0) (at 0)
  with assms have eventually (λx. f x = 0) (at 0)
   by (intro not_essential_frequently_0_imp_eventually_0 has_laurent_expansion_isolated
            has_laurent_expansion_not_essential)
      (auto simp: frequently_def)
  hence (f has_laurent_expansion 0) ⟷ ((λ_. 0) has_laurent_expansion 0)
    by (intro has_laurent_expansion_cong) auto
  hence f has_laurent_expansion 0
    by simp
  with assms(1) have F = 0
    using has_laurent_expansion_unique by blast
```

```
  with ‹F ≠ 0› show False
    by contradiction
qed

lemma has_laurent_expansion_eventually_nonzero_iff':
  assumes f has_laurent_expansion F
  shows   eventually (λx. f x ≠ 0) (at 0) ⟷ F ≠ 0
proof
  assume ∀⇩F x in at 0. f x ≠ 0
  moreover have ¬ (∀⇩F x in at 0. f x ≠ 0) if F=0
  proof −
    have ∀⇩F x in at 0. f x = 0
      using assms that unfolding has_laurent_expansion_def by simp
    then show ?thesis unfolding not_eventually
      by (auto elim:eventually_frequentlyE)
  qed
  ultimately show F ≠ 0 by auto
qed (simp add:has_laurent_expansion_nonzero_imp_eventually_nonzero[OF assms])

lemma has_laurent_expansion_eventually_nonzero_iff:
  assumes (λw. f (z+w)) has_laurent_expansion F
  shows   eventually (λx. f x ≠ 0) (at z)  ⟷ F ≠ 0
  apply (subst eventually_at_to_0)
  apply (rule has_laurent_expansion_eventually_nonzero_iff')
  using assms by (simp add:add.commute)

lemma has_laurent_expansion_inverse [laurent_expansion_intros]:
  assumes f has_laurent_expansion F
  shows   (λx. inverse (f x)) has_laurent_expansion inverse F
proof (cases F = 0)
  case True
  thus ?thesis using assms
    by (auto simp: has_laurent_expansion_def)
next
  case False
  define G where G = laurent_expansion (λx. inverse (f x)) 0
  from False have ev: eventually (λz. f z ≠ 0) (at 0)
   by (intro has_laurent_expansion_nonzero_imp_eventually_nonzero[OF assms])

  have ∗: (λx. inverse (f x)) has_laurent_expansion G unfolding G_def
   by (intro not_essential_has_laurent_expansion_0 isolated_singularity_at_inverse
not_essential_inverse
        has_laurent_expansion_isolated_0[OF assms] has_laurent_expansion_not_essential_0[OF
assms])
  have (λx. f x ∗ inverse (f x)) has_laurent_expansion F ∗ G
    by (intro laurent_expansion_intros assms ∗)
  also have ?this ⟷ (λx. 1) has_laurent_expansion F ∗ G
    by (intro has_laurent_expansion_cong refl eventually_mono[OF ev]) auto
  finally have (λ_. 1) has_laurent_expansion F ∗ G .
```

```
  moreover have (λ_. 1) has_laurent_expansion 1
    by simp
  ultimately have F * G = 1
    using has_laurent_expansion_unique by blast
  hence G = inverse F
    using inverse_unique by blast
  with * show ?thesis
    by simp
qed

lemma has_laurent_expansion_power_int [laurent_expansion_intros]:
  f has_laurent_expansion F ⟹ (λx. f x powi n) has_laurent_expansion (F powi
n)
  by (auto simp: power_int_def intro!: laurent_expansion_intros)


lemma has_fps_expansion_0_analytic_continuation:
  assumes f has_fps_expansion 0 f holomorphic_on A
  assumes open A connected A 0 ∈ A x ∈ A
  shows   f x = 0
proof −
  have eventually (λz. z ∈ A ∧ f z = 0) (nhds 0) using assms
   by (intro eventually_conj eventually_nhds_in_open) (auto simp: has_fps_expansion_def)
  then obtain B where B: open B 0 ∈ B ∀z∈B. z ∈ A ∧ f z = 0
    unfolding eventually_nhds by blast
  show ?thesis
  proof (rule analytic_continuation_open[where f = f and g = λ_. 0])
    show B ≠ {}
      using ‹open B› B by auto
    show connected A
      using assms by auto
  qed (use assms B in auto)
qed

lemma has_laurent_expansion_0_analytic_continuation:
  assumes f has_laurent_expansion 0 f holomorphic_on A − {0}
  assumes open A connected A 0 ∈ A x ∈ A − {0}
  shows   f x = 0
proof −
  have eventually (λz. z ∈ A − {0} ∧ f z = 0) (at 0) using assms
   by (intro eventually_conj eventually_at_in_open) (auto simp: has_laurent_expansion_def)
  then obtain B where B: open B 0 ∈ B ∀z∈B − {0}. z ∈ A − {0} ∧ f z = 0
    unfolding eventually_at_filter eventually_nhds by blast
  show ?thesis
  proof (rule analytic_continuation_open[where f = f and g = λ_. 0])
    show B − {0} ≠ {}
      using ‹open B› ‹0 ∈ B› by (metis insert_Diff not_open_singleton)
    show connected (A − {0})
      using assms by (intro connected_open_delete) auto
```

qed (*use assms B* **in** *auto*)
qed

lemma *has_fps_expansion_cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) (*nhds 0*) *F* = *G*
shows *f has_fps_expansion F* ⟷ *g has_fps_expansion G*
using *assms*(*2*) **by** (*auto simp*: *has_fps_expansion_def elim*!: *eventually_elim2*[*OF assms*(*1*)])

lemma *zor_poly_has_fps_expansion*:
assumes *f has_laurent_expansion F F* ≠ *0*
shows *zor_poly f 0 has_fps_expansion fls_base_factor_to_fps F*
proof −
note [*simp*] = ‹*F* ≠ *0*›
have *eventually* (λ*z*. *f z* ≠ *0*) (*at 0*)
by (*rule has_laurent_expansion_nonzero_imp_eventually_nonzero*[*OF assms*])
hence *freq*: *frequently* (λ*z*. *f z* ≠ *0*) (*at 0*)
by (*rule eventually_frequently*[*rotated*]) *auto*

have ∗: *isolated_singularity_at f 0 not_essential f 0*
using *has_laurent_expansion_isolated_0*[*OF assms*(*1*)] *has_laurent_expansion_not_essential_0*[*OF assms*(*1*)]
by *auto*

define *G* **where** *G* = *fls_base_factor_to_fps F*
define *n* **where** *n* = *zorder f 0*
have *n_altdef*: *n* = *fls_subdegree F*
using *has_laurent_expansion_zorder_0* [*OF assms*(*1*)] **by** (*simp add*: *n_def*)
obtain *r* **where** *r*: *zor_poly f 0 0* ≠ *0 zor_poly f 0 holomorphic_on cball 0 r r* > *0*
∀ *w*∈*cball 0 r* − {*0*}. *f w* = *zor_poly f 0 w* ∗ *w powi n* ∧
zor_poly f 0 w ≠ *0*
using *zorder_exist*[*OF* ∗ *freq*] **unfolding** *n_def* **by** *auto*
obtain *r*′ **where** *r*′: *r*′ > *0* ∀ *x*∈*ball 0 r*′−{*0*}. *eval_fls F x* = *f x*
using *assms*(*1*) **unfolding** *has_laurent_expansion_def eventually_at_filter eventually_nhds_metric ball_def*
by (*auto simp*: *dist_commute*)
have *holo*: *zor_poly f 0 holomorphic_on ball 0 r*
by (*rule holomorphic_on_subset*[*OF r*(*2*)]) *auto*

have (λ*z*. *if z* = *0 then fps_nth G 0 else f z* ∗ *z powi* −*n*) *has_fps_expansion G*
unfolding *G_def n_altdef* **by** (*intro has_fps_expansion_fls_base_factor_to_fps assms*)
also have *?this* ⟷ *zor_poly f 0 has_fps_expansion G*
proof (*intro has_fps_expansion_cong*)
have *eventually* (λ*z*. *z* ∈ *ball 0* (*min r r*′)) (*nhds 0*)
using ‹*r* > *0*› ‹*r*′ > *0*› **by** (*intro eventually_nhds_in_open*) *auto*
thus ∀ *F x in nhds 0*. (*if x* = *0 then G* $ *0 else f x* ∗ *x powi* − *n*) = *zor_poly f 0 x*

```
    proof eventually_elim
      case (elim w)
      have w: w ∈ ball 0 r w ∈ ball 0 r'
        using elim by auto
      show ?case
      proof (cases w = 0)
        case False
        hence f w = zor_poly f 0 w * w powi n
          using r w by auto
        thus ?thesis using False
          by (simp add: powr_minus complex_powr_of_int power_int_minus)
      next
        case [simp]: True
        obtain R where R: R > 0 R ≤ r R ≤ r' R ≤ fls_conv_radius F
          using ‹r > 0› ‹r' > 0› assms(1) unfolding has_laurent_expansion_def
          by (smt (verit, ccfv_SIG) ereal_dense2 ereal_less(2) less_ereal.simps(1)
order.strict_implies_order order_trans)
        have eval_fps G 0 = zor_poly f 0 0
          proof (rule analytic_continuation_open[where f = eval_fps G and g =
zor_poly f 0])
          show connected (ball 0 R :: complex set)
            by auto
          have of_real R / 2 ∈ ball 0 R − {0 :: complex}
            using R by auto
          thus ball 0 R − {0 :: complex} ≠ {}
            by blast
          show eval_fps G holomorphic_on ball 0 R
            using R less_le_trans[OF _ R(4)] unfolding G_def
            by (intro holomorphic_intros) (auto simp: fls_conv_radius_altdef)
          show zor_poly f 0 holomorphic_on ball 0 R
            by (rule holomorphic_on_subset[OF holo]) (use R in auto)
          show eval_fps G z = zor_poly f 0 z if z ∈ ball 0 R − {0} for z
            using that r r' R n_altdef unfolding G_def
            by (subst eval_fps_fls_base_factor)
                  (auto simp: complex_powr_of_int field_simps power_int_minus
n_def)
        qed (use R in auto)
        hence zor_poly f 0 0 = fps_nth G 0
          by (simp add: eval_fps_at_0)
        thus ?thesis by simp
      qed
    qed
  qed (use r' in auto)
  finally show ?thesis
    by (simp add: G_def)
qed

lemma zorder_geI_0:
  assumes f analytic_on {0} f holomorphic_on A open A connected A 0 ∈ A z ∈
```

```
A f z ≠ 0
  assumes ⋀k. k < n ⟹ (deriv ^^ k) f 0 = 0
  shows   zorder f 0 ≥ n
proof −
  define F where F = fps_expansion f 0
  from assms have f has_fps_expansion F
    unfolding F_def using analytic_at_imp_has_fps_expansion_0 by blast
  hence laurent: f has_laurent_expansion fps_to_fls F and [simp]: f 0 = fps_nth
F 0
    by (simp_all add: has_fps_expansion_to_laurent)

  have [simp]: F ≠ 0
  proof
    assume [simp]: F = 0
    hence f z = 0
    proof (cases z = 0)
      case False
      have f has_laurent_expansion 0
        using laurent by simp
      thus ?thesis
      proof (rule has_laurent_expansion_0_analytic_continuation)
        show f holomorphic_on A − {0}
          using assms(2) by (rule holomorphic_on_subset) auto
      qed (use assms False in auto)
    qed auto
    with ‹f z ≠ 0› show False by contradiction
  qed

  have zorder f 0 = int (subdegree F)
   using has_laurent_expansion_zorder_0[OF laurent] by (simp add: fls_subdegree_fls_to_fps)
  also have subdegree F ≥ n
   using assms by (intro subdegree_geI ‹F ≠ 0›) (auto simp: F_def fps_expansion_def)
  hence int (subdegree F) ≥ int n
    by simp
  finally show ?thesis .
qed

lemma zorder_geI:
  assumes f analytic_on {x} f holomorphic_on A open A connected A x ∈ A z ∈
A f z ≠ 0
  assumes ⋀k. k < n ⟹ (deriv ^^ k) f x = 0
  shows   zorder f x ≥ n
proof −
  have zorder f x = zorder (f ∘ (λu. u + x)) 0
    by (subst zorder_shift) (auto simp: o_def)
  also have ... ≥ n
  proof (rule zorder_geI_0)
    show (f ∘ (λu. u + x)) analytic_on {0}
      by (intro analytic_on_compose_gen[OF _ assms(1)] analytic_intros) auto
```

```
    show f ∘ (λu. u + x) holomorphic_on ((+) (−x)) ‘ A
    by (intro holomorphic_on_compose_gen[OF _ assms(2)] holomorphic_intros)
auto
    show connected ((+) (− x) ‘ A)
      by (intro connected_continuous_image continuous_intros assms)
    show open ((+) (− x) ‘ A)
      by (intro open_translation assms)
    show z − x ∈ (+) (− x) ‘ A
      using ‹z ∈ A› by auto
    show 0 ∈ (+) (− x) ‘ A
      using ‹x ∈ A› by auto
    show (f ∘ (λu. u + x)) (z − x) ≠ 0
      using ‹f z ≠ 0› by auto
  next
    fix k :: nat assume k < n
    hence (deriv ^^ k) f x = 0
      using assms by simp
    also have (deriv ^^ k) f x = (deriv ^^ k) (f ∘ (+) x) 0
      by (subst higher_deriv_shift_0) auto
    finally show (deriv ^^ k) (f ∘ (λu. u + x)) 0 = 0
      by (subst add.commute) auto
  qed
  finally show ?thesis .
qed

lemma has_laurent_expansion_divide [laurent_expansion_intros]:
  assumes f has_laurent_expansion F and g has_laurent_expansion G
  shows   (λx. f x / g x) has_laurent_expansion (F / G)
proof −
  have (λx. f x * inverse (g x)) has_laurent_expansion (F * inverse G)
    by (intro laurent_expansion_intros assms)
  thus ?thesis
    by (simp add: field_simps)
qed

lemma vector_derivative_translate [simp]:
  vector_derivative ((+) z ∘ g) (at x within A) = vector_derivative g (at x within
A)
proof −
  have (((+) z ∘ g) has_vector_derivative g′) (at x within A)
    if (g has_vector_derivative g′) (at x within A) for g :: real ⇒ 'a and z g′
    unfolding o_def using that by (auto intro!: derivative_eq_intros)
  from this[of g _ z] this[of λx. z + g x _ −z] show ?thesis
    unfolding vector_derivative_def
    by (intro arg_cong[where f = Eps] ext) (auto simp: o_def algebra_simps)
qed

lemma has_contour_integral_translate:
  (f has_contour_integral I) ((+) z ∘ g) ⟷ ((λx. f (x + z)) has_contour_integral
```

```
I) g
  by (simp add: has_contour_integral_def add_ac)

lemma contour_integrable_translate:
  f contour_integrable_on ((+) z ∘ g) ⟷ (λx. f (x + z)) contour_integrable_on
g
  by (simp add: contour_integrable_on_def has_contour_integral_translate)

lemma contour_integral_translate:
  contour_integral ((+) z ∘ g) f = contour_integral g (λx. f (x + z))
 by (simp add: contour_integral_def contour_integrable_translate has_contour_integral_translate)

lemma residue_shift_0: residue f z = residue (λx. f (z + x)) 0
proof −
  define Q where
     Q = (λr f z ε. (f has_contour_integral complex_of_real (2 * pi) * i * r)
(circlepath z ε))
  define P where
    P = (λr f z. ∃e>0. ∀ε>0. ε < e ⟶ Q r f z ε)
  have path_eq: circlepath (z − w) ε = (+) (−w) ∘ circlepath z ε for z w ε
    by (simp add: circlepath_def o_def part_circlepath_def algebra_simps)
  have ∗: P r f z if P r (λx. f (x + w)) (z − w) for r w f z
   using that by (auto simp: P_def Q_def path_eq has_contour_integral_translate)
  have (SOME r. P r f z) = (SOME r. P r (λx. f (z + x)) 0)
    using ∗[of _ f z z] ∗[of _ λx. f (z + x) −z]
    by (intro arg_cong[where f = Eps] ext iffI) (simp_all add: add_ac)
  thus ?thesis
    by (simp add: residue_def P_def Q_def)
qed

lemma residue_shift_0': NO_MATCH 0 z ⟹ residue f z = residue (λx. f (z +
x)) 0
  by (rule residue_shift_0)

lemma has_laurent_expansion_residue_0:
  assumes f has_laurent_expansion F
  shows   residue f 0 = fls_residue F
proof (cases fls_subdegree F ≥ 0)
  case True
  have residue f 0 = residue (eval_fls F) 0
    using assms by (intro residue_cong) (auto simp: has_laurent_expansion_def
eq_commute)
  also have ... = 0
    by (rule residue_holo[OF _ _ holomorphic_on_eval_fls[OF order.refl]])
      (use True assms in ‹auto simp: has_laurent_expansion_def zero_ereal_def›)
  also have ... = fls_residue F
    using True by simp
  finally show ?thesis .
next
```

```
  case False
  hence F ≠ 0
    by auto
  have ∗: zor_poly f 0 has_fps_expansion fls_base_factor_to_fps F
    by (intro zor_poly_has_fps_expansion False assms ‹F ≠ 0›)

  have residue f 0 = (deriv ^^ (nat (−zorder f 0) − 1)) (zor_poly f 0) 0 / fact
(nat (− zorder f 0) − 1)
    by (intro residue_pole_order has_laurent_expansion_isolated_0[OF assms]
          has_laurent_expansion_imp_is_pole_0[OF assms]) (use False in auto)
  also have ... = fls_residue F
    using has_laurent_expansion_zorder_0[OF assms ‹F ≠ 0›] False
    by (subst fps_nth_fps_expansion [OF ∗, symmetric]) (auto simp: of_nat_diff)
  finally show ?thesis .
qed

lemma has_laurent_expansion_residue:
  assumes (λx. f (z + x)) has_laurent_expansion F
  shows   residue f z = fls_residue F
  using has_laurent_expansion_residue_0[OF assms] by (simp add: residue_shift_0')

lemma eval_fls_has_laurent_expansion [laurent_expansion_intros]:
  assumes fls_conv_radius F > 0
  shows   eval_fls F has_laurent_expansion F
  using assms by (auto simp: has_laurent_expansion_def)

lemma fps_expansion_unique_complex:
  fixes F G :: complex fps
  assumes f has_fps_expansion F f has_fps_expansion G
  shows   F = G
  using assms unfolding fps_eq_iff by (auto simp: fps_eq_iff fps_nth_fps_expansion)

lemma fps_expansion_eqI:
  assumes f has_fps_expansion F
  shows   fps_expansion f 0 = F
  using assms unfolding fps_eq_iff
  by (auto simp: fps_eq_iff fps_nth_fps_expansion fps_expansion_def)

lemma has_fps_expansion_imp_eval_fps_eq:
  assumes f has_fps_expansion F norm z < r
  assumes f holomorphic_on ball 0 r
  shows   eval_fps F z = f z
proof −
  have [simp]: fps_expansion f 0 = F
    by (rule fps_expansion_eqI) fact
  have ∗: f holomorphic_on eball 0 (ereal r)
    using assms by simp
  from conv_radius_fps_expansion[OF ∗] have fps_conv_radius F ≥ ereal r
    by simp
```

```
  have eval_fps (fps_expansion f 0) z = f (0 + z)
    by (rule eval_fps_expansion'[OF *]) (use assms in auto)
  thus ?thesis
    by simp
qed

lemma fls_conv_radius_ge:
  assumes f has_laurent_expansion F
  assumes f holomorphic_on eball 0 r - {0}
  shows   fls_conv_radius F ≥ r
proof -
  define n where n = fls_subdegree F
  define G where G = fls_base_factor_to_fps F
  define g where g = (λz. if z = 0 then fps_nth G 0 else f z * z powi -n)
  have G: g has_fps_expansion G
    unfolding G_def g_def n_def
    by (intro has_fps_expansion_fls_base_factor_to_fps assms)
  have (λz. f z * z powi -n) holomorphic_on eball 0 r - {0}
    by (intro holomorphic_intros assms) auto
  also have ?this ⟷ g holomorphic_on eball 0 r - {0}
    by (intro holomorphic_cong) (auto simp: g_def)
  finally have g analytic_on eball 0 r - {0}
    by (subst analytic_on_open) auto
  moreover have g analytic_on {0}
    using G has_fps_expansion_imp_analytic_0 by auto
  ultimately have g analytic_on (eball 0 r - {0} ∪ {0})
    by (subst analytic_on_Un) auto
  hence g analytic_on eball 0 r
    by (rule analytic_on_subset) auto
  hence g holomorphic_on eball 0 r
    by (subst (asm) analytic_on_open) auto
  hence fps_conv_radius (fps_expansion g 0) ≥ r
    by (intro conv_radius_fps_expansion)
  also have fps_expansion g 0 = G
    using G by (intro fps_expansion_eqI)
  finally show ?thesis
    by (simp add: fls_conv_radius_altdef G_def)
qed

lemma connected_eball [intro]: connected (eball (z :: 'a :: real_normed_vector) r)
  by (cases r) auto

lemma eval_fls_eqI:
  assumes f has_laurent_expansion F f holomorphic_on eball 0 r - {0}
  assumes z ∈ eball 0 r - {0}
  shows   eval_fls F z = f z
proof -
  have conv: fls_conv_radius F ≥ r
    by (intro fls_conv_radius_ge[OF assms(1,2)])
```

```
  have (λz. eval_fls F z − f z) has_laurent_expansion F − F
   using assms by (intro laurent_expansion_intros assms) (auto simp: has_laurent_expansion_def)
  hence (λz. eval_fls F z − f z) has_laurent_expansion 0
    by simp
  hence eval_fls F z − f z = 0
  proof (rule has_laurent_expansion_0_analytic_continuation)
    have ereal 0 ≤ ereal (norm z)
      by simp
    also have norm z < r
      using assms by auto
    finally have r > 0
      by (simp add: zero_ereal_def)
    thus open (eball 0 r :: complex set) connected (eball 0 r :: complex set)
        0 ∈ eball 0 r z ∈ eball 0 r − {0}
      using assms by (auto simp: zero_ereal_def)
   qed (auto intro!: holomorphic_intros assms less_le_trans[OF _ conv] split:
if_splits)
  thus ?thesis by simp
qed

lemma fls_nth_as_contour_integral:
  assumes F: f has_laurent_expansion F
  assumes holo: f holomorphic_on ball 0 r − {0}
  assumes R: 0 < R R < r
  shows ((λz. f z * z powi (−(n+1))) has_contour_integral
          complex_of_real (2 * pi) * ı * fls_nth F n) (circlepath 0 R)
proof −
  define I where I = (λz. f z * z powi (−(n+1)))
   have (I has_contour_integral complex_of_real (2 * pi) * ı * residue I 0)
(circlepath 0 R)
  proof (rule base_residue)
    show open (ball (0::complex) r) 0 ∈ ball (0::complex) r
      using R F by (auto simp: has_laurent_expansion_def zero_ereal_def)
  qed (use R in ‹auto intro!: holomorphic_intros holomorphic_on_subset[OF holo]
                  simp: I_def split: if_splits›)
  also have residue I 0 = fls_residue (fls_shift (n + 1) F)
   unfolding I_def by (intro has_laurent_expansion_residue_0 laurent_expansion_intros
F)
  also have ... = fls_nth F n
    by simp
  finally show ?thesis
    by (simp add: I_def)
qed

lemma tendsto_0_subdegree_iff_0:
  assumes F:f has_laurent_expansion F and F≠0
  shows (f −0→0) ⟷ fls_subdegree F > 0
proof −
  have ?thesis if is_pole f 0
```

```
  proof −
    have fls_subdegree F <0
      using is_pole_0_imp_neg_fls_subdegree[OF F that] .
    moreover then have ¬ f −0→0
      using ‹is_pole f 0› F at_neq_bot
        has_laurent_expansion_imp_filterlim_infinity_0
        not_tendsto_and_filterlim_at_infinity that
      by blast
    ultimately show ?thesis by auto
  qed
  moreover have ?thesis if ¬is_pole f 0 ∃ x. f −0→x
  proof −
    have fls_subdegree F ≥0
      using has_laurent_expansion_imp_is_pole_0[OF F] that(1)
      by linarith
    have f −0→0 if fls_subdegree F > 0
      using fls_eq0_below_subdegree[OF that]
     by (metis F ‹0 ≤ fls_subdegree F› has_laurent_expansion_imp_tendsto_0)
    moreover have fls_subdegree F > 0 if f −0→0
    proof −
      have False if fls_subdegree F = 0
      proof −
        have f −0→ fls_nth F 0
          using has_laurent_expansion_imp_tendsto_0
              [OF F ‹fls_subdegree F ≥0›] .
        then have fls_nth F 0 = 0 using ‹f −0→0›
          using LIM_unique by blast
        then have F = 0
          using nth_fls_subdegree_zero_iff ‹fls_subdegree F = 0›
          by metis
        with ‹F≠0› show False by auto
      qed
      with ‹fls_subdegree F ≥0›
      show ?thesis by fastforce
    qed
    ultimately show ?thesis by auto
  qed
  moreover have is_pole f 0 ∨ (∃ x. f −0→x)
  proof −
    have not_essential f 0
      using F has_laurent_expansion_not_essential_0 by auto
    then show ?thesis unfolding not_essential_def
      by auto
  qed
  ultimately show ?thesis by auto
qed

lemma tendsto_0_subdegree_iff:
  assumes F:(λw. f (z+w)) has_laurent_expansion F and F≠0
```

```
  shows (f −z→0) ⟷ fls_subdegree F > 0
  apply (subst Lim_at_zero)
  apply (rule tendsto_0_subdegree_iff_0)
  using assms by auto

lemma is_pole_0_deriv_divide_iff:
  assumes F:f has_laurent_expansion F and F≠0
  shows is_pole (λx. deriv f x / f x) 0 ⟷ is_pole f 0 ∨ (f −0→0)
proof −
  have (λx. deriv f x / f x) has_laurent_expansion fls_deriv F / F
    using F by (auto intro:laurent_expansion_intros)

  have is_pole (λx. deriv f x / f x) 0 ⟷
          fls_subdegree (fls_deriv F / F) < 0
    apply (rule is_pole_fls_subdegree_iff)
    using F by (auto intro:laurent_expansion_intros)
  also have ... ⟷ is_pole f 0 ∨ (f −0→0)
  proof (cases fls_subdegree F = 0)
    case True
    then have fls_subdegree (fls_deriv F / F) ≥ 0
      by (metis diff_zero div_0 ‹F≠0› fls_deriv_subdegree0
          fls_divide_subdegree)
    moreover then have ¬ is_pole f 0
      by (metis F True is_pole_0_imp_neg_fls_subdegree less_le)
    moreover have ¬ (f −0→0)
      using tendsto_0_subdegree_iff_0[OF F ‹F≠0›] True by auto
    ultimately show ?thesis by auto
  next
    case False
    then have fls_deriv F ≠ 0
      by (metis fls_const_subdegree fls_deriv_eq_0_iff)
    then have fls_subdegree (fls_deriv F / F) =
            fls_subdegree (fls_deriv F) − fls_subdegree F
      by (rule fls_divide_subdegree[OF _ ‹F≠0›])
    moreover have fls_subdegree (fls_deriv F) = fls_subdegree F − 1
      using fls_subdegree_deriv[OF False] .
    ultimately have fls_subdegree (fls_deriv F / F) < 0 by auto
    moreover have f −0→ 0 = (0 < fls_subdegree F)
      using tendsto_0_subdegree_iff_0[OF F ‹F ≠ 0›] .
    moreover have is_pole f 0 = (fls_subdegree F < 0)
      using is_pole_fls_subdegree_iff F by auto
    ultimately show ?thesis using False by auto
  qed
  finally show ?thesis .
qed

lemma is_pole_deriv_divide_iff:
  assumes F:(λw. f (z+w)) has_laurent_expansion F and F≠0
  shows is_pole (λx. deriv f x / f x) z ⟷ is_pole f z ∨ (f −z→0)
```

```
proof −
  define ff df where ff=(λw. f (z+w)) and df=(λw. deriv f (z + w))
  have is_pole (λx. deriv f x / f x) z
        ⟷ is_pole (λx. deriv ff x / ff x) 0
    unfolding ff_def df_def
    by (simp add:deriv_shift_0' is_pole_shift_0' comp_def algebra_simps)
  moreover have is_pole f z ⟷ is_pole ff 0
    unfolding ff_def by (auto simp:is_pole_shift_0')
  moreover have (f −z→0) ⟷ (ff −0→0)
    unfolding ff_def by (simp add: LIM_offset_zero_iff)
  moreover have is_pole (λx. deriv ff x / ff x) 0 = (is_pole ff 0 ∨ ff −0→ 0)
    apply (rule is_pole_0_deriv_divide_iff)
    using F ff_def ‹F≠0› by blast+
  ultimately show ?thesis by auto
qed

lemma subdegree_imp_eventually_deriv_nzero_0:
  assumes F:f has_laurent_expansion F and fls_subdegree F≠0
  shows eventually (λz. deriv f z ≠ 0) (at 0)
proof −
  have deriv f has_laurent_expansion fls_deriv F
    using has_laurent_expansion_deriv[OF F] .
  moreover have fls_deriv F≠0
    using ‹fls_subdegree F≠0›
    by (metis fls_const_subdegree fls_deriv_eq_0_iff)
  ultimately show ?thesis
    using has_laurent_expansion_eventually_nonzero_iff' by blast
qed

lemma subdegree_imp_eventually_deriv_nzero:
  assumes F:(λw. f (z+w)) has_laurent_expansion F
     and fls_subdegree F≠0
  shows eventually (λw. deriv f w ≠ 0) (at z)
proof −
  have ∀F x in at 0. deriv (λw. f (z + w)) x ≠ 0
    using subdegree_imp_eventually_deriv_nzero_0 assms by auto
  then show ?thesis
    apply (subst eventually_at_to_0)
    by (simp add:deriv_shift_0' comp_def algebra_simps)
qed

lemma has_fps_expansion_imp_asymp_equiv_0:
  fixes f :: complex ⇒ complex
  assumes F: f has_fps_expansion F
  defines n ≡ subdegree F
  shows   f ∼[nhds 0] (λz. fps_nth F n * z ^ n)
proof −
  have F': f has_laurent_expansion fps_to_fls F
    using F has_laurent_expansion_fps by blast
```

```
  have f ∼[at 0] (λz. fps_nth F n * z ^ n)
    using has_laurent_expansion_imp_asymp_equiv_0[OF F']
    by (simp add: fls_subdegree_fls_to_fps n_def)
  moreover have f 0 = fps_nth F n * 0 ^ n
    using F by (auto simp: n_def has_fps_expansion_to_laurent power_0_left)
  ultimately show ?thesis
    by (auto simp: asymp_equiv_nhds_iff)
qed

lemma has_fps_expansion_imp_tendsto_0:
  fixes f :: complex ⇒ complex
  assumes f has_fps_expansion F
  shows   (f ⟶ fps_nth F 0) (nhds 0)
proof (rule asymp_equiv_tendsto_transfer)
  show (λz. fps_nth F (subdegree F) * z ^ subdegree F) ∼[nhds 0] f
    by (rule asymp_equiv_symI, rule has_fps_expansion_imp_asymp_equiv_0)
fact
  have ((λz. F $ subdegree F * z ^ subdegree F) ⟶ F $ 0) (at 0)
    by (rule tendsto_eq_intros refl | simp)+ (auto simp: power_0_left)
  thus ((λz. F $ subdegree F * z ^ subdegree F) ⟶ F $ 0) (nhds 0)
    by (auto simp: tendsto_nhds_iff power_0_left)
qed

lemma has_fps_expansion_imp_0_eq_fps_nth_0:
  assumes f has_fps_expansion F
  shows   f 0 = fps_nth F 0
proof −
  have eventually (λx. f x = eval_fps F x) (nhds 0)
    using assms by (auto simp: has_fps_expansion_def eq_commute)
  then obtain A where open A 0 ∈ A ∀x∈A. f x = eval_fps F x
    unfolding eventually_nhds by blast
  hence f 0 = eval_fps F 0
    by blast
  thus ?thesis
    by (simp add: eval_fps_at_0)
qed

lemma fls_nth_compose_aux:
  assumes f has_fps_expansion F
  assumes G: g has_fps_expansion G fps_nth G 0 = 0 fps_deriv G ≠ 0
  assumes (f ∘ g) has_laurent_expansion H
  shows   fls_nth H (int n) = fps_nth (fps_compose F G) n
  using assms(1,5)
proof (induction n arbitrary: f F H rule: less_induct)
  case (less n f F H)
  have [simp]: g 0 = 0
    using has_fps_expansion_imp_0_eq_fps_nth_0[OF G(1)] G(2) by simp
  have ana_f: f analytic_on {0}
```

```
    using less.prems by (meson has_fps_expansion_imp_analytic_0)
  have ana_g: g analytic_on {0}
    using G by (meson has_fps_expansion_imp_analytic_0)
  have (f ∘ g) has_laurent_expansion fps_to_fls (fps_expansion (f ∘ g) 0)
   by (rule analytic_at_imp_has_fps_expansion_0 analytic_intros has_laurent_expansion_fps
          analytic_on_compose_gen ana_f ana_g)+ auto
  with less.prems have H = fps_to_fls (fps_expansion (f ∘ g) 0)
    using has_laurent_expansion_unique by blast
  also have fls_subdegree ... ≥ 0
    by (simp add: fls_subdegree_fls_to_fps)
  finally have subdeg: fls_subdegree H ≥ 0 .

  show ?case
  proof (cases n = 0)
    case [simp]: True
    have lim_g: g −0→ 0
      using has_laurent_expansion_imp_tendsto_0[of g fps_to_fls G] G
     by (auto simp: fls_subdegree_fls_to_fps_gt0 has_fps_expansion_to_laurent)
    have lim_f: (f ⟶ fps_nth F 0) (nhds 0)
      by (intro has_fps_expansion_imp_tendsto_0 less.prems)
    have (λx. f (g x)) −0→ fps_nth F 0
      by (rule filterlim_compose[OF lim_f lim_g])
    moreover have (f ∘ g) −0→ fls_nth H 0
      by (intro has_laurent_expansion_imp_tendsto_0 less.prems subdeg)
    ultimately have fps_nth F 0 = fls_nth H 0
      using tendsto_unique by (force simp: o_def)
    thus ?thesis
      by simp
  next
    case n: False
    define GH where GH = (fls_deriv H / fls_deriv (fps_to_fls G))
    define GH' where GH' = fls_regpart GH

    have (λx. deriv (f ∘ g) x / deriv g x) has_laurent_expansion
         fls_deriv H / fls_deriv (fps_to_fls G)
      by (intro laurent_expansion_intros less.prems has_laurent_expansion_fps[of
_ G] G)
      also have ?this ⟷ (deriv f ∘ g) has_laurent_expansion fls_deriv H /
fls_deriv (fps_to_fls G)
    proof (rule has_laurent_expansion_cong)
      from ana_f obtain r1 where r1: r1 > 0 f holomorphic_on ball 0 r1
        unfolding analytic_on_def by blast
      from ana_g obtain r2 where r2: r2 > 0 g holomorphic_on ball 0 r2
        unfolding analytic_on_def by blast
      have lim_g: g −0→ 0
        using has_laurent_expansion_imp_tendsto_0[of g fps_to_fls G] G
      by (auto simp: fls_subdegree_fls_to_fps_gt0 has_fps_expansion_to_laurent)
      moreover have open (ball 0 r1) 0 ∈ ball 0 r1
        using r1 by auto
```

```
      ultimately have eventually (λx. g x ∈ ball 0 r1) (at 0)
        unfolding tendsto_def by blast
      moreover have eventually (λx. deriv g x ≠ 0) (at 0)
      using G fps_to_fls_eq_0_iff has_fps_expansion_deriv has_fps_expansion_to_laurent
            has_laurent_expansion_nonzero_imp_eventually_nonzero by blast
      moreover have eventually (λx. x ∈ ball 0 (min r1 r2) - {0}) (at 0)
        by (intro eventually_at_in_open) (use r1 r2 in auto)
      ultimately show eventually (λx. deriv (f ∘ g) x / deriv g x = (deriv f ∘ g)
x) (at 0)
      proof eventually_elim
        case (elim x)
        thus ?case using r1 r2
          by (subst deriv_chain)
             (auto simp: field_simps holomorphic_on_def at_within_open[of _ ball
_ _])
      qed
    qed auto
    finally have GH: (deriv f ∘ g) has_laurent_expansion GH
      unfolding GH_def .

    have (deriv f ∘ g) has_laurent_expansion fps_to_fls (fps_expansion (deriv f
∘ g) 0)
    by (rule analytic_at_imp_has_fps_expansion_0 analytic_intros has_laurent_expansion_fps
          analytic_on_compose_gen ana_f ana_g)+ auto
    with GH have GH = fps_to_fls (fps_expansion (deriv f ∘ g) 0)
      using has_laurent_expansion_unique by blast
    also have fls_subdegree ... ≥ 0
      by (simp add: fls_subdegree_fls_to_fps)
    finally have subdeg': fls_subdegree GH ≥ 0 .

    have deriv f has_fps_expansion fps_deriv F
      by (intro fps_expansion_intros less.prems)
     from this and GH have IH: fls_nth GH (int k) = fps_nth (fps_compose
(fps_deriv F) G) k
      if k < n for k
      by (intro less.IH that)

    have fps_nth (fps_compose (fps_deriv F) G) n = (∑ i=0..n. of_nat (Suc i)
* F $ Suc i * G ^ i $ n)
      by (simp add: fps_compose_nth)

    have fps_nth (fps_compose F G) n =
          fps_nth (fps_deriv (fps_compose F G)) (n - 1) / of_nat n
      using n by (cases n) (auto simp del: of_nat_Suc)
     also have fps_deriv (fps_compose F G) = fps_compose (fps_deriv F) G *
fps_deriv G
      using G by (subst fps_compose_deriv) auto
     also have fps_nth ... (n - 1) = (∑ i=0..n-1. (fps_deriv F oo G) $ i *
fps_deriv G $ (n - 1 - i))
```

```
    unfolding fps_mult_nth ..
  also have ... = (∑ i=0..n-1. fps_nth GH' i * of_nat (n - i) * G $ (n -
i))
    using n by (intro sum.cong) (auto simp: IH Suc_diff_Suc GH'_def)
  also have ... = (∑ i=0..n. fps_nth GH' i * of_nat (n - i) * G $ (n - i))
    by (intro sum.mono_neutral_left) auto
  also have ... = fps_nth (GH' * Abs_fps (λi. of_nat i * fps_nth G i)) n
    by (simp add: fps_mult_nth mult_ac)
  also have Abs_fps (λi. of_nat i * fps_nth G i) = fps_X * fps_deriv G
    by (simp add: fps_mult_fps_X_deriv_shift)
  also have fps_nth (GH' * (fps_X * fps_deriv G)) n =
          fls_nth (fps_to_fls (GH' * (fps_X * fps_deriv G))) (int n)
    by simp
  also have fps_to_fls (GH' * (fps_X * fps_deriv G)) =
            GH * fps_to_fls (fps_deriv G) * fls_X
    using subdeg' by (simp add: mult_ac fls_times_fps_to_fls GH'_def)
  also have GH * fps_to_fls (fps_deriv G) = fls_deriv H
    unfolding GH_def using G by (simp add: fls_deriv_fps_to_fls)
  also have fls_deriv H * fls_X = fls_shift (-1) (fls_deriv H)
    using fls_X_times_conv_shift(2) by blast
  finally show ?thesis
    using n by simp
 qed
qed

lemma has_fps_expansion_compose [fps_expansion_intros]:
 fixes f g :: complex ⇒ complex
 assumes F: f has_fps_expansion F
 assumes G: g has_fps_expansion G fps_nth G 0 = 0
 shows   (f ∘ g) has_fps_expansion fps_compose F G
proof (cases fps_deriv G = 0)
 case False
 have [simp]: g 0 = 0
   using has_fps_expansion_imp_0_eq_fps_nth_0[OF G(1)] G(2) False by
simp
 have ana_f: f analytic_on {0}
   using F by (meson has_fps_expansion_imp_analytic_0)
 have ana_g: g analytic_on {0}
   using G by (meson has_fps_expansion_imp_analytic_0)
 have fg: (f ∘ g) has_fps_expansion fps_expansion (f ∘ g) 0
   by (rule analytic_at_imp_has_fps_expansion_0 analytic_intros
       analytic_on_compose_gen ana_f ana_g)+ auto

 have fls_nth (fps_to_fls (fps_expansion (f ∘ g) 0)) (int n) = fps_nth (fps_compose
F G) n for n
   by (rule fls_nth_compose_aux has_laurent_expansion_fps F G False fg)+
 hence fps_expansion (f ∘ g) 0 = fps_compose F G
   by (simp add: fps_eq_iff)
 thus ?thesis using fg
```

```
    by simp
next
  case True
  have [simp]: f 0 = fps_nth F 0
    using F by (auto dest: has_fps_expansion_imp_0_eq_fps_nth_0)
  from True have fps_nth G n = 0 for n
    using G(2) by (cases n) (auto simp del: of_nat_Suc)
  hence [simp]: G = 0
    by (auto simp: fps_eq_iff)
  have (λ_. f 0) has_fps_expansion fps_const (f 0)
    by (intro fps_expansion_intros)
  also have eventually (λx. g x = 0) (nhds 0)
    using G by (auto simp: has_fps_expansion_def)
 hence (λ_. f 0) has_fps_expansion fps_const (f 0) ⟷ (f ∘ g) has_fps_expansion
fps_const (f 0)
    by (intro has_fps_expansion_cong) (auto elim!: eventually_mono)
  thus ?thesis
    by simp
qed

hide_const (open) fls_compose_fps

definition fls_compose_fps :: 'a :: field fls ⇒ 'a fps ⇒ 'a fls where
  fls_compose_fps F G =
     fps_to_fls (fps_compose (fls_base_factor_to_fps F) G) * fps_to_fls G powi
fls_subdegree F

lemma fps_compose_of_nat [simp]: fps_compose (of_nat n :: 'a :: comm_ring_1
fps) H = of_nat n
  and fps_compose_of_int [simp]: fps_compose (of_int i) H = of_int i
  unfolding fps_of_nat [symmetric] fps_of_int [symmetric] numeral_fps_const
  by (rule fps_const_compose)+

lemmas [simp] = fps_to_fls_of_nat fps_to_fls_of_int

lemma fls_compose_fps_0 [simp]: fls_compose_fps 0 H = 0
  and fls_compose_fps_1 [simp]: fls_compose_fps 1 H = 1
 and fls_compose_fps_const [simp]: fls_compose_fps (fls_const c) H = fls_const
c
  and fls_compose_fps_of_nat [simp]: fls_compose_fps (of_nat n) H = of_nat
n
  and fls_compose_fps_of_int [simp]: fls_compose_fps (of_int i) H = of_int i
  and fls_compose_fps_X [simp]: fls_compose_fps fls_X F = fps_to_fls F
  by (simp_all add: fls_compose_fps_def)

lemma fls_compose_fps_0_right:
  fls_compose_fps F 0 = (if fls_subdegree F ≥ 0 then fls_const (fls_nth F 0) else
0)
  by (cases fls_subdegree F = 0) (simp_all add: fls_compose_fps_def)
```

lemma *fls_compose_fps_shift*:
assumes $H \neq 0$
shows *fls_compose_fps* (*fls_shift n F*) *H* = *fls_compose_fps F H* ∗ *fps_to_fls H powi* (−*n*)
proof (*cases F* = *0*)
case *False*
thus *?thesis*
using *assms* **by** (*simp add*: *fls_compose_fps_def power_int_diff power_int_minus field_simps*)
qed *auto*

lemma *fls_compose_fps_to_fls* [*simp*]:
assumes [*simp*]: *G* ≠ *0 fps_nth G 0* = *0*
shows *fls_compose_fps* (*fps_to_fls F*) *G* = *fps_to_fls* (*fps_compose F G*)
proof (*cases F* = *0*)
case *False*
define *n* **where** *n* = *subdegree F*
define *F*′ **where** *F*′ = *fps_shift n F*
have [*simp*]: *F*′ ≠ *0 subdegree F*′ = *0*
using *False* **by** (*auto simp*: *F*′*_def n_def*)
have *F_eq*: *F* = *F*′ ∗ *fps_X* ^ *n*
unfolding *F*′*_def n_def* **using** *subdegree_decompose* **by** *blast*
have *fls_compose_fps* (*fps_to_fls F*) *G* =
fps_to_fls (*fps_shift n* (*fls_regpart* (*fps_to_fls F*′ ∗ *fls_X_intpow* (*int n*))) *oo G*) ∗ *fps_to_fls* (*G* ^ *n*)
unfolding *F_eq fls_compose_fps_def*
by (*simp add*: *fls_times_fps_to_fls fls_X_power_conv_shift_1 power_int_add fls_subdegree_fls_to_fps fps_to_fls_power fls_regpart_shift_conv_fps_shift flip*: *fls_times_both_shifted_simp*)
also have *fps_to_fls F*′ ∗ *fls_X_intpow* (*int n*) = *fps_to_fls F*
by (*simp add*: *F_eq fls_times_fps_to_fls fps_to_fls_power fls_X_power_conv_shift_1*)
also have *fps_to_fls* (*fps_shift n* (*fls_regpart* (*fps_to_fls F*)) *oo G*) ∗ *fps_to_fls* (*G* ^ *n*) =
fps_to_fls ((*fps_shift n* (*fls_regpart* (*fps_to_fls F*)) ∗ *fps_X* ^ *n*) *oo G*)
by (*simp add*: *fls_times_fps_to_fls flip*: *fps_compose_power add*: *fps_compose_mult_distrib*)
also have *fps_shift n* (*fls_regpart* (*fps_to_fls F*)) ∗ *fps_X* ^ *n* = *F*
by (*simp add*: *F_eq*)
finally show *?thesis* .
qed (*auto simp*: *fls_compose_fps_def*)

lemma *fls_compose_fps_mult*:
assumes [*simp*]: *H* ≠ *0 fps_nth H 0* = *0*
shows *fls_compose_fps* (*F* ∗ *G*) *H* = *fls_compose_fps F H* ∗ *fls_compose_fps G H*
using *assms*
proof (*cases F* ∗ *G* = *0*)
case *False*

```
  hence [simp]: F ≠ 0 G ≠ 0
    by auto
  define n m where n = fls_subdegree F m = fls_subdegree G
  define F' where F' = fls_regpart (fls_shift n F)
  define G' where G' = fls_regpart (fls_shift m G)
  have F_eq: F = fls_shift (−n) (fps_to_fls F') and G_eq: G = fls_shift (−m)
(fps_to_fls G')
    by (simp_all add: F'_def G'_def n_m_def)
  have fls_compose_fps (F * G) H = fls_compose_fps (fls_shift (−(n + m))
(fps_to_fls (F' * G'))) H
    by (simp add: fls_times_fps_to_fls F_eq G_eq fls_shifted_times_simps)
  also have ... = fps_to_fls ((F' oo H) * (G' oo H)) * fps_to_fls H powi (m +
n)
    by (simp add: fls_compose_fps_shift fps_compose_mult_distrib)
  also have ... = fls_compose_fps F H * fls_compose_fps G H
   by (simp add: F_eq G_eq fls_compose_fps_shift fls_times_fps_to_fls power_int_add)
  finally show ?thesis .
qed auto

lemma fls_compose_fps_power:
  assumes [simp]: G ≠ 0 fps_nth G 0 = 0
  shows   fls_compose_fps (F ^ n) G = fls_compose_fps F G ^ n
  by (induction n) (auto simp: fls_compose_fps_mult)

lemma fls_compose_fps_add:
  assumes [simp]: H ≠ 0 fps_nth H 0 = 0
  shows  fls_compose_fps (F + G) H = fls_compose_fps F H + fls_compose_fps
G H
proof (cases F = 0 ∨ G = 0)
  case False
  hence [simp]: F ≠ 0 G ≠ 0
    by auto
  define n where n = min (fls_subdegree F) (fls_subdegree G)
  define F' where F' = fls_regpart (fls_shift n F)
  define G' where G' = fls_regpart (fls_shift n G)
  have F_eq: F = fls_shift (−n) (fps_to_fls F') and G_eq: G = fls_shift (−n)
(fps_to_fls G')
    unfolding n_def by (simp_all add: F'_def G'_def n_def)
  have F + G = fls_shift (−n) (fps_to_fls (F' + G'))
    by (simp add: F_eq G_eq)
  also have fls_compose_fps ... H = fls_compose_fps (fps_to_fls (F' + G')) H
* fps_to_fls H powi n
    by (subst fls_compose_fps_shift) auto
  also have ... = fps_to_fls (fps_compose (F' + G') H) * fps_to_fls H powi n
    by (subst fls_compose_fps_to_fls) auto
  also have ... = fls_compose_fps F H + fls_compose_fps G H
     by (simp add: F_eq G_eq fls_compose_fps_shift fps_compose_add_distrib
algebra_simps)
  finally show ?thesis .
```

qed *auto*

lemma *fls_compose_fps_uminus* [*simp*]: *fls_compose_fps* (−*F*) *H* = −*fls_compose_fps F H*
by (*simp add*: *fls_compose_fps_def fps_compose_uminus*)

lemma *fls_compose_fps_diff*:
assumes [*simp*]: *H* ≠ *0 fps_nth H 0* = *0*
shows *fls_compose_fps* (*F* − *G*) *H* = *fls_compose_fps F H* − *fls_compose_fps G H*
using *fls_compose_fps_add*[*of H F* −*G*] **by** *simp*

lemma *fps_compose_eq_0_iff*:
fixes *F G* :: *'a* :: *idom fps*
assumes *fps_nth G 0* = *0*
shows *fps_compose F G* = *0* ⟷ *F* = *0* ∨ (*G* = *0* ∧ *fps_nth F 0* = *0*)
proof *safe*
assume ∗: *fps_compose F G* = *0 F* ≠ *0*
have *fps_nth* (*fps_compose F G*) *0* = *fps_nth F 0*
by *simp*
also have *fps_compose F G* = *0*
by (*simp add*: ∗)
finally show *fps_nth F 0* = *0*
by *simp*
show *G* = *0*
proof (*rule ccontr*)
assume *G* ≠ *0*
hence *subdegree G* > *0* **using** *assms*
using *subdegree_eq_0_iff* **by** *blast*
define *N* **where** *N* = *subdegree F* ∗ *subdegree G*
have *fps_nth* (*fps_compose F G*) *N* = (∑ *i* = *0*..*N*. *fps_nth F i* ∗ *fps_nth* (*G* ^ *i*) *N*)
unfolding *fps_compose_def* **by** (*simp add*: *N_def*)
also have ... = (∑ *i*∈{*subdegree F*}. *fps_nth F i* ∗ *fps_nth* (*G* ^ *i*) *N*)
proof (*intro sum.mono_neutral_right ballI*)
fix *i* **assume** *i*: *i* ∈ {*0*..*N*} − {*subdegree F*}
show *fps_nth F i* ∗ *fps_nth* (*G* ^ *i*) *N* = *0*
proof (*cases i subdegree F rule*: *linorder_cases*)
assume *i* > *subdegree F*
hence *fps_nth* (*G* ^ *i*) *N* = *0*
using *i* ‹*subdegree G* > *0*› **by** (*intro fps_pow_nth_below_subdegree*) (*auto simp*: *N_def*)
thus *?thesis* **by** *simp*
qed (*use i* **in** ‹*auto simp*: *N_def*›)
qed (*use* ‹*subdegree G* > *0*› **in** ‹*auto simp*: *N_def*›)
also have ... = *fps_nth F* (*subdegree F*) ∗ *fps_nth* (*G* ^ *subdegree F*) *N*
by *simp*
also have ... ≠ *0*
using ‹*G* ≠ *0*› ‹*F* ≠ *0*› **by** (*auto simp*: *N_def*)

```
    finally show False using ∗ by auto
  qed
qed auto

lemma fls_compose_fps_eq_0_iff:
  assumes H ≠ 0 fps_nth H 0 = 0
  shows   fls_compose_fps F H = 0 ⟷ F = 0
  using assms fls_base_factor_to_fps_nonzero[of F]
  by (cases F = 0) (auto simp: fls_compose_fps_def fps_compose_eq_0_iff)

lemma fls_compose_fps_inverse:
  assumes [simp]: H ≠ 0 fps_nth H 0 = 0
  shows   fls_compose_fps (inverse F) H = inverse (fls_compose_fps F H)
proof (cases F = 0)
  case False
  have fls_compose_fps (inverse F) H ∗ fls_compose_fps F H =
       fls_compose_fps (inverse F ∗ F) H
    by (subst fls_compose_fps_mult) auto
  also have inverse F ∗ F = 1
    using False by simp
  finally show ?thesis
    using False by (simp add: field_simps fls_compose_fps_eq_0_iff)
qed auto

lemma fls_compose_fps_divide:
  assumes [simp]: H ≠ 0 fps_nth H 0 = 0
  shows   fls_compose_fps (F / G) H = fls_compose_fps F H / fls_compose_fps
G H
  using fls_compose_fps_mult[of H F inverse G] fls_compose_fps_inverse[of H
G]
  by (simp add: field_simps)

lemma fls_compose_fps_powi:
  assumes [simp]: H ≠ 0 fps_nth H 0 = 0
  shows   fls_compose_fps (F powi n) H = fls_compose_fps F H powi n
  by (simp add: power_int_def fls_compose_fps_power fls_compose_fps_inverse)

lemma fls_compose_fps_assoc:
  assumes [simp]: G ≠ 0 fps_nth G 0 = 0 H ≠ 0 fps_nth H 0 = 0
  shows fls_compose_fps (fls_compose_fps F G) H = fls_compose_fps F (fps_compose
G H)
proof (cases F = 0)
  case [simp]: False
  define n where n = fls_subdegree F
  define F' where F' = fls_regpart (fls_shift n F)
  have F_eq: F = fls_shift (−n) (fps_to_fls F')
    by (simp add: F'_def n_def)
  show ?thesis
   by (simp add: F_eq fls_compose_fps_shift fls_compose_fps_mult fls_compose_fps_powi
```

```
                fps_compose_eq_0_iff fps_compose_assoc)
qed auto

lemma subdegree_pos_iff: subdegree F > 0 ⟷ F ≠ 0 ∧ fps_nth F 0 = 0
  using subdegree_eq_0_iff[of F] by auto

lemma has_fps_expansion_fps_to_fls:
  assumes f has_laurent_expansion fps_to_fls F
  shows   (λz. if z = 0 then fps_nth F 0 else f z) has_fps_expansion F
  (is ?f' has_fps_expansion _)
proof −
  have f has_laurent_expansion fps_to_fls F ⟷ ?f' has_laurent_expansion
fps_to_fls F
    by (intro has_laurent_expansion_cong) (auto simp: eventually_at_filter)
  with assms show ?thesis
    by (auto simp: has_fps_expansion_to_laurent)
qed


lemma has_laurent_expansion_compose [laurent_expansion_intros]:
  fixes f g :: complex ⇒ complex
  assumes F: f has_laurent_expansion F
  assumes G: g has_laurent_expansion fps_to_fls G fps_nth G 0 = 0 G ≠ 0
  shows   (f ∘ g) has_laurent_expansion fls_compose_fps F G
proof −
  from assms have lim_g: g −0→ 0
    by (subst tendsto_0_subdegree_iff_0[OF G(1)])
       (auto simp: fls_subdegree_fls_to_fps subdegree_pos_iff)
  have ev1: eventually (λz. g z ≠ 0) (at 0)
    using ‹G ≠ 0› G(1) fps_to_fls_eq_0_iff has_laurent_expansion_fps
          has_laurent_expansion_nonzero_imp_eventually_nonzero by blast
  moreover have eventually (λz. z ≠ 0) (at (0 :: complex))
    by (auto simp: eventually_at_filter)
  ultimately have ev: eventually (λz. z ≠ 0 ∧ g z ≠ 0) (at 0)
    by eventually_elim blast
  from ev1 and lim_g have lim_g': filterlim g (at 0) (at 0)
    by (auto simp: filterlim_at)
  define g' where g' = (λz. if z = 0 then fps_nth G 0 else g z)

  show ?thesis
  proof (cases F = 0)
    assume [simp]: F = 0
    have eventually (λz. f z = 0) (at 0)
      using F by (auto simp: has_laurent_expansion_def)
    hence eventually (λz. f (g z) = 0) (at 0)
      using lim_g' by (rule eventually_compose_filterlim)
    thus ?thesis
      by (auto simp: has_laurent_expansion_def)
  next
```

```
    assume [simp]: F ≠ 0
    define n where n = fls_subdegree F
    define f' where
     f' = (λz. if z = 0 then fps_nth (fls_base_factor_to_fps F) 0 else f z * z powi
−n)
    have ((λz. (f' ∘ g') z * g z powi n)) has_laurent_expansion fls_compose_fps
F G
      unfolding f'_def n_def fls_compose_fps_def g'_def
    by (intro fps_expansion_intros laurent_expansion_intros has_fps_expansion_fps_to_fls
          has_fps_expansion_fls_base_factor_to_fps assms has_laurent_expansion_fps)
    also have ?this ⟷ ?thesis
      by (intro has_laurent_expansion_cong eventually_mono[OF ev])
        (auto simp: f'_def power_int_minus g'_def)
    finally show ?thesis .
  qed
qed

lemma has_laurent_expansion_fls_X_inv [laurent_expansion_intros]:
  inverse has_laurent_expansion fls_X_inv
  using has_laurent_expansion_inverse[OF has_laurent_expansion_fps_X]
  by (simp add: fls_inverse_X)

lemma fls_X_power_int [simp]: fls_X powi n = (fls_X_intpow n :: 'a :: divi-
sion_ring fls)
  by (auto simp: power_int_def fls_X_power_conv_shift_1 fls_inverse_X fls_inverse_shift
        simp flip: fls_inverse_X_power)

lemma fls_const_power_int: fls_const (c powi n) = fls_const (c :: 'a :: divi-
sion_ring) powi n
  by (auto simp: power_int_def fls_const_power fls_inverse_const)

lemma fls_nth_fls_compose_fps_linear:
  fixes c :: 'a :: field
  assumes [simp]: c ≠ 0
  shows fls_nth (fls_compose_fps F (fps_const c * fps_X)) n = fls_nth F n * c
powi n
proof −
  {
    assume *: n ≥ fls_subdegree F
    hence c ^ nat (n − fls_subdegree F) = c powi int (nat (n − fls_subdegree F))
      by (simp add: power_int_def)
    also have ... * c powi fls_subdegree F = c powi (int (nat (n − fls_subdegree
F)) + fls_subdegree F)
      using * by (subst power_int_add) auto
    also have ... = c powi n
      using * by simp
    finally have c ^ nat (n − fls_subdegree F) * c powi fls_subdegree F = c powi
n .
  }
```

```
  thus ?thesis
    by (simp add: fls_compose_fps_def fps_compose_linear fls_times_fps_to_fls
power_int_mult_distrib
                  fls_shifted_times_simps
            flip: fls_const_power_int)
qed

lemma zorder_times_analytic:
  assumes f analytic_on {z} g analytic_on {z}
  assumes eventually (λz. f z * g z ≠ 0) (at z)
  shows   zorder (λz. f z * g z) z = zorder f z + zorder g z
proof −
  have ∗: (λw. f (z + w)) has_fps_expansion fps_expansion f z
          (λw. g (z + w)) has_fps_expansion fps_expansion g z
            (λw. f (z + w) * g (z + w)) has_fps_expansion fps_expansion f z *
fps_expansion g z
    by (intro fps_expansion_intros analytic_at_imp_has_fps_expansion assms)+
  have [simp]: fps_expansion f z ≠ 0
  proof
    assume fps_expansion f z = 0
    hence eventually (λz. f z * g z = 0) (at z) using ∗(1)
      by (auto simp: has_fps_expansion_0_iff nhds_to_0' eventually_filtermap
eventually_at_filter
               elim: eventually_mono)
    with assms(3) have eventually (λz. False) (at z)
      by eventually_elim auto
    thus False by simp
  qed
  have [simp]: fps_expansion g z ≠ 0
  proof
    assume fps_expansion g z = 0
    hence eventually (λz. f z * g z = 0) (at z) using ∗(2)
      by (auto simp: has_fps_expansion_0_iff nhds_to_0' eventually_filtermap
eventually_at_filter
               elim: eventually_mono)
    with assms(3) have eventually (λz. False) (at z)
      by eventually_elim auto
    thus False by simp
  qed
  from ∗[THEN has_fps_expansion_zorder] show ?thesis
    by auto
qed

lemma analytic_on_prod [analytic_intros]:
  assumes ⋀x. x ∈ A ⟹ f x analytic_on B
  shows   (λz. ∏ x∈A. f x z) analytic_on B
  using assms by (induction A rule: infinite_finite_induct) (auto intro!: ana-
lytic_intros)
```

lemma *zorder_const* [*simp*]: $c \neq 0 \Longrightarrow$ *zorder* $(\lambda\_.\ c)\ z = 0$
by (*intro zorder_eqI*[**where** $s = UNIV$]) *auto*

lemma *zorder_prod_analytic*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x$ *analytic_on* $\{z\}$
assumes *eventually* $(\lambda z.\ (\prod x{\in}A.\ f\ x\ z) \neq 0)$ (*at* z)
shows *zorder* $(\lambda z.\ \prod x{\in}A.\ f\ x\ z)\ z = (\sum x{\in}A.\ zorder\ (f\ x)\ z)$
using *assms*
proof (*induction A rule*: *infinite_finite_induct*)
case (*insert x A*)
have *zorder* $(\lambda z.\ f\ x\ z * (\prod x{\in}A.\ f\ x\ z))\ z = zorder\ (f\ x)\ z + zorder\ (\lambda z.\ \prod x{\in}A.\ f\ x\ z)\ z$
using *insert.prems insert.hyps* **by** (*intro zorder_times_analytic analytic_intros*) *auto*
also have *zorder* $(\lambda z.\ \prod x{\in}A.\ f\ x\ z)\ z = (\sum x{\in}A.\ zorder\ (f\ x)\ z)$
using *insert.prems insert.hyps* **by** (*intro insert.IH*) (*auto elim*!: *eventually_mono*)
finally show *?case* **using** *insert*
by *simp*
qed *auto*

lemma *zorder_eq_0I*:
assumes *g analytic_on* $\{z\}$ *g z* $\neq 0$
shows *zorder g z* $= 0$
proof −
from *assms* **obtain** *r* **where** *r*: $r > 0$ *g holomorphic_on ball z r*
unfolding *analytic_on_def* **by** *blast*
thus *?thesis* **using** *assms*
by (*intro zorder_eqI*[*of ball z r _ g*]) *auto*
qed

lemma *zorder_pos_iff*:
assumes *f holomorphic_on A open A z* $\in$ *A frequently* $(\lambda z.\ f\ z \neq 0)$ (*at z*)
shows *zorder f z* $> 0 \longleftrightarrow f\ z = 0$
proof −
have *f analytic_on* $\{z\}$
using *assms analytic_at* **by** *blast*
hence $*$: $(\lambda w.\ f\ (z + w))$ *has_fps_expansion fps_expansion f z*
using *analytic_at_imp_has_fps_expansion* **by** *blast*
have *nz*: *fps_expansion f z* $\neq 0$
proof
assume *fps_expansion f z* $= 0$
hence *eventually* $(\lambda z.\ f\ z = 0)$ (*nhds z*)
using $*$ **by** (*auto simp*: *has_fps_expansion_def nhds_to_0' eventually_filtermap add_ac*)
hence *eventually* $(\lambda z.\ f\ z = 0)$ (*at z*)
by (*auto simp*: *eventually_at_filter elim*: *eventually_mono*)
with *assms* **show** *False*
by (*auto simp*: *frequently_def*)
qed

from *has_fps_expansion_zorder*[*OF* ∗ *this*] **have** *eq*: *zorder f z* = *int* (*subdegree* (*fps_expansion f z*))
by *auto*
moreover have *subdegree* (*fps_expansion f z*) = *0* ⟷ *fps_nth* (*fps_expansion f z*) *0* ≠ *0*
using *nz* **by** (*auto simp*: *subdegree_eq_0_iff*)
moreover have *fps_nth* (*fps_expansion f z*) *0* = *f z*
by (*auto simp*: *fps_expansion_def*)
ultimately show *?thesis*
by *auto*
qed

lemma *zorder_pos_iff'*:
assumes *f analytic_on* {*z*} *frequently* (λ*z*. *f z* ≠ *0*) (*at z*)
shows *zorder f z* > *0* ⟷ *f z* = *0*
proof −
from *assms*(*1*) **obtain** *A* **where** *A*: *open A* {*z*} ⊆ *A f holomorphic_on A*
unfolding *analytic_on_holomorphic* **by** *auto*
with *zorder_pos_iff* [*OF A*(*3*,*1*), *of z*] *assms* **show** *?thesis*
by *auto*
qed

lemma *zorder_ge_0*:
assumes *f analytic_on* {*z*} *frequently* (λ*z*. *f z* ≠ *0*) (*at z*)
shows *zorder f z* ≥ *0*
proof −
have ∗: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion fps_to_fls* (*fps_expansion f z*)
using *assms* **by** (*simp add*: *analytic_at_imp_has_fps_expansion has_laurent_expansion_fps*)
from ∗ *assms*(*2*) **have** *fps_to_fls* (*fps_expansion f z*) ≠ *0*
by (*auto simp*: *has_laurent_expansion_def frequently_def at_to_0' eventually_filtermap add_ac*)
with *has_laurent_expansion_zorder*[*OF* ∗] **show** *?thesis*
by (*simp add*: *fls_subdegree_fls_to_fps*)
qed

lemma *zorder_eq_0_iff*:
assumes *f analytic_on* {*z*} *frequently* (λ*w*. *f w* ≠ *0*) (*at z*)
shows *zorder f z* = *0* ⟷ *f z* ≠ *0*
proof
assume *f z* ≠ *0*
thus *zorder f z* = *0*
using *assms zorder_eq_0I* **by** *blast*
next
assume *zorder f z* = *0*
thus *f z* ≠ *0*
using *assms zorder_pos_iff'* **by** *fastforce*
qed

lemma *dist_mult_left*:

```
  dist (a * b) (a * c :: 'a :: real_normed_field) = norm a * dist b c
  unfolding dist_norm right_diff_distrib [symmetric] norm_mult by simp

lemma dist_mult_right:
  dist (b * a) (c * a :: 'a :: real_normed_field) = norm a * dist b c
  using dist_mult_left[of a b c] by (simp add: mult_ac)

lemma zorder_scale:
  assumes f analytic_on {a * z} eventually (λw. f w ≠ 0) (at (a * z)) a ≠ 0
  shows zorder (λw. f (a * w)) z = zorder f (a * z)
proof −
  from assms(1) obtain r where r: r > 0 f holomorphic_on ball (a * z) r
    by (auto simp: analytic_on_def)
  have *: open (ball (a * z) r) connected (ball (a * z) r) a * z ∈ ball (a * z) r
    using r ‹a ≠ 0› by (auto simp: dist_norm)
  from assms(2) have eventually (λw. f w ≠ 0 ∧ w ∈ ball (a * z) r − {a * z})
(at (a * z))
    using ‹r > 0› by (intro eventually_conj eventually_at_in_open) auto
  then obtain z0 where f z0 ≠ 0 ∧ z0 ∈ ball (a * z) r − {a * z}
    using eventually_happens[of _ at (a * z)] by force
  hence **: ∃ w∈ball (a * z) r. f w ≠ 0
    by blast

  define n where n = nat (zorder f (a * z))
  obtain r' where r':
     (if f (a * z) = 0 then 0 < zorder f (a * z) else zorder f (a * z) = 0)
      r' > 0 cball (a * z) r' ⊆ ball (a * z) r zor_poly f (a * z) holomorphic_on
cball (a * z) r'
     ⋀w. w ∈ cball (a * z) r' ⟹
       f w = zor_poly f (a * z) w * (w − a * z) ^ n ∧ zor_poly f (a * z) w ≠ 0
    unfolding n_def using zorder_exist_zero[OF r(2) * **] by blast

  show ?thesis
  proof (rule zorder_eqI)
    show open (ball z (r' / norm a)) z ∈ ball z (r' / norm a)
      using r ‹r' > 0› ‹a ≠ 0› by auto
    have (*) a ‘ ball z (r' / cmod a) ⊆ cball (a * z) r'
    proof safe
      fix w assume w ∈ ball z (r' / cmod a)
      thus a * w ∈ cball (a * z) r'
      using dist_mult_left[of a z w] ‹a ≠ 0› by (auto simp: divide_simps mult_ac)
    qed
    thus (λw. a ^ n * (zor_poly f (a * z) ∘ (λw. a * w)) w) holomorphic_on ball
z (r' / norm a)
      using ‹a ≠ 0› by (intro holomorphic_on_compose_gen[OF _ r'(4)] holo-
morphic_intros) auto
    show a ^ n * (zor_poly f (a * z) ∘ (λw. a * w)) z ≠ 0
      using r' ‹a ≠ 0› by auto
   show f (a * w) = a ^ n * (zor_poly f (a * z) ∘ (*) a) w * (w − z) powi (zorder
```

```
f (a * z))
      if w ∈ ball z (r' / norm a) w ≠ z for w
    proof −
      have f (a * w) = zor_poly f (a * z) (a * w) * (a * (w − z)) ^ n
        using that r'(5)[of a * w] dist_mult_left[of a z w] ‹a ≠ 0› unfolding
ring_distribs
        by (auto simp: divide_simps mult_ac)
      also have ... = a ^ n * zor_poly f (a * z) (a * w) * (w − z) ^ n
        by (subst power_mult_distrib) (auto simp: mult_ac)
      also have (w − z) ^ n = (w − z) powi of_nat n
        by simp
      also have of_nat n = zorder f (a * z)
        using r'(1) by (auto simp: n_def split: if_splits)
      finally show ?thesis
        unfolding o_def n_def .
    qed
  qed
qed

lemma subdegree_fps_compose [simp]:
  fixes F G :: 'a :: idom fps
  assumes [simp]: fps_nth G 0 = 0
  shows subdegree (fps_compose F G) = subdegree F * subdegree G
proof (cases G = 0; cases F = 0)
  assume [simp]: G ≠ 0 F ≠ 0
  define m where m = subdegree F
  define F' where F' = fps_shift m F
  have F_eq: F = F' * fps_X ^ m
    unfolding F'_def by (simp add: fps_shift_times_fps_X_power m_def)
  have [simp]: F' ≠ 0
    using ‹F ≠ 0› unfolding F_eq by auto
   have subdegree (fps_compose F G) = subdegree (fps_compose F' G) + m *
subdegree G
     by (simp add: F_eq fps_compose_mult_distrib fps_compose_eq_0_iff flip:
fps_compose_power)
  also have subdegree (fps_compose F' G) = 0
    by (intro subdegree_eq_0) (auto simp: F'_def m_def)
  finally show ?thesis by (simp add: m_def)
qed auto

lemma fls_subdegree_power_int [simp]:
  fixes   F :: 'a :: field fls
  shows fls_subdegree (F powi n) = n * fls_subdegree F
  by (auto simp: power_int_def fls_subdegree_pow)

lemma subdegree_fls_compose_fps [simp]:
  fixes G :: 'a :: field fps
  assumes [simp]: fps_nth G 0 = 0
  shows fls_subdegree (fls_compose_fps F G) = fls_subdegree F * subdegree G
```

proof (*cases* $F = 0$; *cases* $G = 0$)
assume [*simp*]: $G \neq 0$ $F \neq 0$
have *nz1*: *fls_base_factor_to_fps* $F \neq 0$
using ‹$F \neq 0$› *fls_base_factor_to_fps_nonzero* **by** *blast*
show *?thesis*
unfolding *fls_compose_fps_def* **using** *nz1*
by (*subst fls_subdegree_mult*) (*simp_all add*: *fps_compose_eq_0_iff fls_subdegree_fls_to_fps*)
qed (*auto simp*: *fls_compose_fps_0_right*)

lemma *zorder_compose_aux*:
assumes *isolated_singularity_at f 0 not_essential f 0*
assumes *G*: *g has_fps_expansion G G* $\neq 0$ *g 0* $= 0$
assumes *eventually* (λw. *f w* $\neq 0$) (*at 0*)
shows *zorder* (*f* $\circ$ *g*) *0* $=$ *zorder f 0* $*$ *subdegree G*
proof −
obtain *F* **where** *F*: *f has_laurent_expansion F*
using *not_essential_has_laurent_expansion_0*[*OF assms*(*1*,*2*)] **by** *blast*
have [*simp*]: *fps_nth G 0* $= 0$
using *G* ‹*g 0* $= 0$› **by** (*simp add*: *has_fps_expansion_imp_0_eq_fps_nth_0*)
note [*simp*] = ‹$G \neq 0$› ‹*g 0* $= 0$›
have [*simp*]: $F \neq 0$
using *has_laurent_expansion_eventually_nonzero_iff*[*of f 0 F*] *F assms* **by**
simp
have *FG*: (*f* $\circ$ *g*) *has_laurent_expansion fls_compose_fps F G*
by (*intro has_laurent_expansion_compose has_laurent_expansion_fps F G*)
auto

have *zorder* (*f* $\circ$ *g*) *0* $=$ *fls_subdegree* (*fls_compose_fps F G*)
using *has_laurent_expansion_zorder_0* [*OF FG*] **by** (*auto simp*: *fls_compose_fps_eq_0_iff*)
also have $\ldots$ $=$ *fls_subdegree F* $*$ *int* (*subdegree G*)
by *simp*
also have *fls_subdegree F* $=$ *zorder f 0*
using *has_laurent_expansion_zorder_0* [*OF F*] **by** *auto*
finally show *?thesis* **.**
qed

lemma *zorder_compose*:
assumes *isolated_singularity_at f* (*g z*) *not_essential f* (*g z*)
assumes *G*: (λx. *g* (*z* $+$ *x*) $-$ *g z*) *has_fps_expansion G G* $\neq 0$
assumes *eventually* (λw. *f w* $\neq 0$) (*at* (*g z*))
shows *zorder* (*f* $\circ$ *g*) *z* $=$ *zorder f* (*g z*) $*$ *subdegree G*
proof −
define f' **where** $f' = (\lambda w.\ f\ (g\ z + w))$
define g' **where** $g' = (\lambda w.\ g\ (z + w) - g\ z)$
have *zorder f* (*g z*) $=$ *zorder* f' *0*
by (*simp add*: f'*_def zorder_shift*$'$ *add_ac*)
have *zorder* (λx. *g x* $-$ *g z*) *z* $=$ *zorder* g' *0*
by (*simp add*: g'*_def zorder_shift*$'$ *add_ac*)
have *zorder* (*f* $\circ$ *g*) *z* $=$ *zorder* (f' $\circ$ g') *0*

```
    by (simp add: zorder_shift' f'_def g'_def add_ac o_def)
  also have ... = zorder f' 0 * int (subdegree G)
  proof (rule zorder_compose_aux)
    show isolated_singularity_at f' 0 unfolding f'_def
     using assms has_laurent_expansion_isolated_0 not_essential_has_laurent_expansion
by blast
    show not_essential f' 0 unfolding f'_def
     using assms has_laurent_expansion_not_essential_0 not_essential_has_laurent_expansion
by blast
   qed (use assms in ‹auto simp: f'_def g'_def at_to_0' eventually_filtermap
add_ac›)
  also have zorder f' 0 = zorder f (g z)
    by (simp add: f'_def zorder_shift' add_ac)
  finally show ?thesis .
qed

lemma fps_to_fls_eq_fls_const_iff [simp]: fps_to_fls F = fls_const c ⟷ F =
fps_const c
proof
  assume F = fps_const c
  thus fps_to_fls F = fls_const c
    by simp
next
  assume fps_to_fls F = fls_const c
  thus F = fps_const c
    by (metis fls_regpart_const fls_regpart_fps_trivial)
qed

lemma zorder_compose':
  assumes isolated_singularity_at f (g z) not_essential f (g z)
  assumes g analytic_on {z}
  assumes eventually (λw. f w ≠ 0) (at (g z))
  assumes eventually (λw. g w ≠ g z) (at z)
  shows   zorder (f ∘ g) z = zorder f (g z) * zorder (λx. g x - g z) z
proof -
  obtain G where G [fps_expansion_intros]: (λx. g (z + x)) has_fps_expansion
G
    using assms analytic_at_imp_has_fps_expansion by blast
  have G': (λx. g (z + x) - g z) has_fps_expansion G - fps_const (g z)
    by (intro fps_expansion_intros)
   hence G'': (λx. g (z + x) - g z) has_laurent_expansion fps_to_fls (G -
fps_const (g z))
    using has_laurent_expansion_fps by blast
  have nz: G - fps_const (g z) ≠ 0
    using has_laurent_expansion_eventually_nonzero_iff[OF G''] assms by auto
  have zorder (f ∘ g) z = zorder f (g z) * subdegree (G - fps_const (g z))
  proof (rule zorder_compose)
    show (λx. g (z + x) - g z) has_fps_expansion G - fps_const (g z)
      by (intro fps_expansion_intros)
```

```
  qed (use assms nz in auto)
  also have int (subdegree (G − fps_const (g z))) = fls_subdegree (fps_to_fls G
− fls_const (g z))
    by (simp flip: fls_subdegree_fls_to_fps)
  also have ... = zorder (λx. g x − g z) z
    using has_laurent_expansion_zorder [OF G''] nz by auto
  finally show ?thesis .
qed

lemma analytic_at_cong:
  assumes eventually (λx. f x = g x) (nhds x) x = y
  shows f analytic_on {x} ⟷ g analytic_on {y}
proof −
  have g analytic_on {x} if f analytic_on {x} eventually (λx. f x = g x) (nhds x)
for f g
  proof −
    have (λy. f (x + y)) has_fps_expansion fps_expansion f x
      by (rule analytic_at_imp_has_fps_expansion) fact
    also have ?this ⟷ (λy. g (x + y)) has_fps_expansion fps_expansion f x
      using that by (intro has_fps_expansion_cong refl) (auto simp: nhds_to_0'
eventually_filtermap)
    finally show ?thesis
      by (rule has_fps_expansion_imp_analytic)
  qed
  from this[of f g] this[of g f] show ?thesis using assms
    by (auto simp: eq_commute)
qed


lemma has_laurent_expansion_sin' [laurent_expansion_intros]:
  sin has_laurent_expansion fps_to_fls (fps_sin 1)
  using has_fps_expansion_sin' has_fps_expansion_to_laurent by blast

lemma has_laurent_expansion_cos' [laurent_expansion_intros]:
  cos has_laurent_expansion fps_to_fls (fps_cos 1)
  using has_fps_expansion_cos' has_fps_expansion_to_laurent by blast

lemma has_laurent_expansion_sin [laurent_expansion_intros]:
  (λz. sin (c ∗ z)) has_laurent_expansion fps_to_fls (fps_sin c)
  by (intro has_laurent_expansion_fps has_fps_expansion_sin)

lemma has_laurent_expansion_cos [laurent_expansion_intros]:
  (λz. cos (c ∗ z)) has_laurent_expansion fps_to_fls (fps_cos c)
  by (intro has_laurent_expansion_fps has_fps_expansion_cos)

lemma has_laurent_expansion_tan' [laurent_expansion_intros]:
  tan has_laurent_expansion fps_to_fls (fps_tan 1)
  using has_fps_expansion_tan' has_fps_expansion_to_laurent by blast
```

lemma *has_laurent_expansion_tan* [*laurent_expansion_intros*]:
(λ*z*. *tan* (*c* ∗ *z*)) *has_laurent_expansion fps_to_fls* (*fps_tan c*)
by (*intro has_laurent_expansion_fps has_fps_expansion_tan*)

end

7 The Great Picard Theorem and its Applications

Ported from HOL Light (cauchy.ml) by L C Paulson, 2017

theory *Great_Picard*
imports *Conformal_Mappings*
begin

7.1 Schottky's theorem

lemma *Schottky_lemma0*:
assumes *holf*: *f holomorphic_on S* **and** *cons*: *contractible S* **and** *a* ∈ *S*
and *f*: ⋀*z*. *z* ∈ *S* ⟹ *f z* ≠ *1* ∧ *f z* ≠ −*1*
obtains *g* **where** *g holomorphic_on S*
norm(*g a*) ≤ *1* + *norm*(*f a*) / *3*
⋀*z*. *z* ∈ *S* ⟹ *f z* = *cos*(*of_real pi* ∗ *g z*)
proof −
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *g*: *norm*(*g a*) ≤ *pi* + *norm*(*f a*)
and *f_eq_cos*: ⋀*z*. *z* ∈ *S* ⟹ *f z* = *cos*(*g z*)
using *contractible_imp_holomorphic_arccos_bounded* [*OF assms*]
by *blast*
show *?thesis*
proof
show (λ*z*. *g z* / *pi*) *holomorphic_on S*
by (*auto intro*: *holomorphic_intros holg*)
have *3* ≤ *pi*
using *pi_approx* **by** *force*
have *3* ∗ *norm*(*g a*) ≤ *3* ∗ (*pi* + *norm*(*f a*))
using *g* **by** *auto*
also have ... ≤ *pi* ∗ *3* + *pi* ∗ *cmod* (*f a*)
using ‹*3* ≤ *pi*› **by** (*simp add*: *mult_right_mono algebra_simps*)
finally show *cmod* (*g a* / *complex_of_real pi*) ≤ *1* + *cmod* (*f a*) / *3*
by (*simp add*: *field_simps norm_divide*)
show ⋀*z*. *z* ∈ *S* ⟹ *f z* = *cos* (*complex_of_real pi* ∗ (*g z* / *complex_of_real pi*))
by (*simp add*: *f_eq_cos*)
qed
qed

lemma *Schottky_lemma1*:
fixes *n*::*nat*
assumes *0* < *n*

```
  shows 0 < n + sqrt(real n ^ 2 - 1)
proof -
  have 0 < n * n
    by (simp add: assms)
  then show ?thesis
   by (metis add.commute add.right_neutral add_pos_nonneg assms diff_ge_0_iff_ge
nat_less_real_le of_nat_0 of_nat_0_less_iff of_nat_power power2_eq_square
real_sqrt_ge_0_iff)
qed


lemma Schottky_lemma2:
  fixes x::real
  assumes 0 ≤ x
  obtains n where 0 < n |x - ln (real n + sqrt ((real n)^2 - 1)) / pi| < 1/2
proof -
  obtain n0::nat where 0 < n0 ln(n0 + sqrt(real n0 ^ 2 - 1)) / pi ≤ x
  proof
    show ln(real 1 + sqrt(real 1 ^ 2 - 1))/pi ≤ x
      by (auto simp: assms)
  qed auto
  moreover
  obtain M::nat where ⋀n. ⟦0 < n; ln(n + sqrt(real n ^ 2 - 1)) / pi ≤ x⟧ ⟹
n ≤ M
  proof
    fix n::nat
    assume 0 < n ln (n + sqrt ((real n)^2 - 1)) / pi ≤ x
    then have ln (n + sqrt ((real n)^2 - 1)) ≤ x * pi
      by (simp add: field_split_simps)
    then have *: exp (ln (n + sqrt ((real n)^2 - 1))) ≤ exp (x * pi)
      by blast
    have 0: 0 ≤ sqrt ((real n)^2 - 1)
      using ‹0 < n› by auto
    have n + sqrt ((real n)^2 - 1) = exp (ln (n + sqrt ((real n)^2 - 1)))
      by (simp add: Suc_leI ‹0 < n› add_pos_nonneg real_of_nat_ge_one_iff)
    also have ... ≤ exp (x * pi)
      using * by blast
    finally have real n ≤ exp (x * pi)
      using 0 by linarith
    then show n ≤ nat (ceiling (exp(x * pi)))
      by linarith
  qed
  ultimately obtain n where
     0 < n and le_x: ln(n + sqrt(real n ^ 2 - 1)) / pi ≤ x
           and le_n: ⋀k. ⟦0 < k; ln(k + sqrt(real k ^ 2 - 1)) / pi ≤ x⟧ ⟹ k ≤
n
    using bounded_Max_nat [of λn. 0<n ∧ ln (n + sqrt ((real n)^2 - 1)) / pi ≤
x] by metis
  define a where a ≡ ln(n + sqrt(real n ^ 2 - 1)) / pi
```

```
  define b where b ≡ ln (1 + real n + sqrt ((1 + real n)^2 - 1)) / pi
  have le_xa: a ≤ x
   and le_na: ⋀k. ⟦0 < k; ln(k + sqrt(real k ^ 2 - 1)) / pi ≤ x⟧ ⟹ k ≤ n
    using le_x le_n by (auto simp: a_def)
  moreover have x < b
    using le_n [of Suc n] by (force simp: b_def)
  moreover have b - a < 1
  proof -
    have ln (1 + real n + sqrt ((1 + real n)^2 - 1)) - ln (real n + sqrt ((real n)^2
- 1)) =
          ln ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real n)^2 -
1)))
      by (simp add: ‹0 < n› Schottky_lemma1 add_pos_nonneg ln_div [symmetric])
    also have ... ≤ 3
    proof (cases n = 1)
      case True
      have sqrt 3 ≤ 2
        by (simp add: real_le_lsqrt)
      then have (2 + sqrt 3) ≤ 4
        by simp
      also have ... ≤ exp 3
        using exp_ge_add_one_self [of 3::real] by simp
      finally have ln (2 + sqrt 3) ≤ 3
          by (metis add_nonneg_nonneg add_pos_nonneg dbl_def dbl_inc_def
dbl_inc_simps(3)
           dbl_simps(3) exp_gt_zero ln_exp ln_le_cancel_iff real_sqrt_ge_0_iff
zero_le_one zero_less_one)
      then show ?thesis
        by (simp add: True)
    next
      case False with ‹0 < n› have 1 < n 2 ≤ n
        by linarith+
      then have 1: 1 ≤ real n * real n
      by (metis less_imp_le_nat one_le_power power2_eq_square real_of_nat_ge_one_iff)
      have *: 4 + (m+2) * 2 ≤ (m+2) * ((m+2) * 3) for m::nat
        by simp
      have 4 + n * 2 ≤ n * (n * 3)
        using * [of n-2]  ‹2 ≤ n›
        by (metis le_add_diff_inverse2)
      then have **: 4 + real n * 2 ≤ real n * (real n * 3)
      by (metis (mono_tags, opaque_lifting) of_nat_le_iff of_nat_add of_nat_mult
of_nat_numeral)
      have sqrt ((1 + real n)^2 - 1) ≤ 2 * sqrt ((real n)^2 - 1)
        by (auto simp: real_le_lsqrt power2_eq_square algebra_simps 1 **)
      then
      have ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real n)^2 -
1))) ≤ 2
        using Schottky_lemma1 ‹0 < n›  by (simp add: field_split_simps)
     then have ln ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real
```

```
n)^2 - 1))) ≤ ln 2
      using Schottky_lemma1 [of n] ‹0 < n›
      by (simp add: field_split_simps add_pos_nonneg)
    also have ... ≤ 3
      using ln_add_one_self_le_self [of 1] by auto
    finally show ?thesis .
  qed
  also have ... < pi
    using pi_approx by simp
  finally show ?thesis
    by (simp add: a_def b_def field_split_simps)
 qed
 ultimately have |x - a| < 1/2 ∨ |x - b| < 1/2
  by (auto simp: abs_if)
 then show thesis
 proof
  assume |x - a| < 1/2
  then show ?thesis
   by (rule_tac n=n in that) (auto simp: a_def ‹0 < n›)
 next
  assume |x - b| < 1/2
  then show ?thesis
   by (rule_tac n=Suc n in that) (auto simp: b_def ‹0 < n›)
 qed
qed


lemma Schottky_lemma3:
 fixes z::complex
 assumes z ∈ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n ^ 2
- 1)) / pi)})
          ∪ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (-ln(n + sqrt(real n ^ 2 -
1)) / pi)})
 shows cos(pi * cos(pi * z)) = 1 ∨ cos(pi * cos(pi * z)) = -1
proof -
 have sqrt2 [simp]: complex_of_real (sqrt x) * complex_of_real (sqrt x) = x if x
≥ 0 for x::real
   by (metis abs_of_nonneg of_real_mult real_sqrt_mult_self that)
 define plusi where plusi (e::complex) ≡ e + inverse e for e
 have 1: ∃k. plusi (exp (i * (of_int m * complex_of_real pi) - ln (real n + sqrt
((real n)^2 - 1)))) = of_int k * 2
         (is ∃k. ?Φ k)
       if n > 0 for m n
 proof -
  have eeq: e ≠ 0 ⟹ plusi e = n ⟷ (inverse e) ^ 2 = n/e - 1 for n e::complex
    by (auto simp: plusi_def field_simps power2_eq_square)
  have [simp]: 1 ≤ real n * real n
    using nat_0_less_mult_iff nat_less_real_le that by force
  consider odd m | even m
```

```
    by blast
  then have ∃k. ?Φ k
  proof cases
    case 1
    then have ?Φ (− n)
      using Schottky_lemma1 [OF that]
        by (simp add: eeq) (simp add: power2_eq_square exp_diff exp_Euler
exp_of_real algebra_simps sin_int_times_real cos_int_times_real)
    then show ?thesis ..
  next
    case 2
    then have ?Φ n
      using Schottky_lemma1 [OF that]
        by (simp add: eeq) (simp add: power2_eq_square exp_diff exp_Euler
exp_of_real algebra_simps)
    then show ?thesis ..
  qed
  then show ?thesis by blast
 qed
 have 2: ∃k. plusi (exp (i * (of_int m * complex_of_real pi) +
                  (ln (real n + sqrt ((real n)^2 − 1))))) = of_int k * 2
        (is ∃k. ?Φ k)
         if n > 0 for m n
 proof −
  have eeq: e ≠ 0 ⟹ plusi e = n ⟷ e^2 − n*e + 1 = 0 for n e::complex
    by (auto simp: plusi_def field_simps power2_eq_square)
  have [simp]: 1 ≤ real n * real n
   by (metis One_nat_def add.commute nat_less_real_le of_nat_1 of_nat_Suc
one_le_power power2_eq_square that)
  consider odd m | even m
    by blast
  then have ∃k. ?Φ k
  proof cases
    case 1
    then have ?Φ (− n)
      using Schottky_lemma1 [OF that]
        by (simp add: eeq) (simp add: power2_eq_square exp_add exp_Euler
exp_of_real algebra_simps sin_int_times_real cos_int_times_real)
    then show ?thesis ..
  next
    case 2
    then have ?Φ n
      using Schottky_lemma1 [OF that]
        by (simp add: eeq) (simp add: power2_eq_square exp_add exp_Euler
exp_of_real algebra_simps)
    then show ?thesis ..
  qed
  then show ?thesis by blast
 qed
```

```
  have ∃x. cos (complex_of_real pi * z) = of_int x
    using assms
     apply (auto simp: Ints_def cos_exp_eq exp_minus Complex_eq simp flip:
plusi_def)
     apply (auto simp: algebra_simps dest: 1 2)
    done
  then have sin(pi * cos(pi * z)) ^ 2 = 0
    by (simp add: Complex_Transcendental.sin_eq_0)
  then have 1 - cos(pi * cos(pi * z)) ^ 2 = 0
    by (simp add: sin_squared_eq)
  then show ?thesis
    using power2_eq_1_iff by auto
qed


theorem Schottky:
  assumes holf: f holomorphic_on cball 0 1
      and nof0: norm(f 0) ≤ r
      and not01: ⋀z. z ∈ cball 0 1 ⟹ ¬(f z = 0 ∨ f z = 1)
      and 0 < t t < 1 norm z ≤ t
    shows norm(f z) ≤ exp(pi * exp(pi * (2 + 2 * r + 12 * t / (1 - t))))
proof -
  obtain h where holf: h holomorphic_on cball 0 1
            and nh0: norm (h 0) ≤ 1 + norm(2 * f 0 - 1) / 3
            and h:  ⋀z. z ∈ cball 0 1 ⟹ 2 * f z - 1 = cos(of_real pi * h z)
  proof (rule Schottky_lemma0 [of λz. 2 * f z - 1 cball 0 1 0])
    show (λz. 2 * f z - 1) holomorphic_on cball 0 1
      by (intro holomorphic_intros holf)
    show contractible (cball (0::complex) 1)
      by (auto simp: convex_imp_contractible)
    show ⋀z. z ∈ cball 0 1 ⟹ 2 * f z - 1 ≠ 1 ∧ 2 * f z - 1 ≠ - 1
      using not01 by force
  qed auto
  obtain g where holg: g holomorphic_on cball 0 1
            and ng0:  norm(g 0) ≤ 1 + norm(h 0) / 3
            and g:    ⋀z. z ∈ cball 0 1 ⟹ h z = cos(of_real pi * g z)
  proof (rule Schottky_lemma0 [OF holf convex_imp_contractible, of 0])
    show ⋀z. z ∈ cball 0 1 ⟹ h z ≠ 1 ∧ h z ≠ - 1
      using h not01 by fastforce+
  qed auto
  have g0_2_f0: norm(g 0) ≤ 2 + norm(f 0)
  proof -
    have cmod (2 * f 0 - 1) ≤ cmod (2 * f 0) + 1
      by (metis norm_one norm_triangle_ineq4)
    also have ... ≤ 6 + 9 * cmod (f 0)
      by auto
    finally have 1 + norm(2 * f 0 - 1) / 3 ≤ (2 + norm(f 0) - 1) * 3
      by (simp add: divide_simps)
    with nh0 have norm(h 0) ≤ (2 + norm(f 0) - 1) * 3
```

```
    by linarith
  then have 1 + norm(h 0) / 3 ≤ 2 + norm(f 0)
    by simp
  with ng0 show ?thesis
    by auto
 qed
 have z ∈ ball 0 1
  using assms by auto
 have norm_g_12: norm(g z − g 0) ≤ (12 ∗ t) / (1 − t)
 proof −
  obtain g′ where g′: ⋀x. x ∈ cball 0 1 ⟹ (g has_field_derivative g′ x) (at x
within cball 0 1)
    using holg [unfolded holomorphic_on_def field_differentiable_def] by metis
  have int_g′: (g′ has_contour_integral g z − g 0) (linepath 0 z)
    using contour_integral_primitive [OF g′ valid_path_linepath, of 0 z]
    using ‹z ∈ ball 0 1› segment_bound1 by fastforce
  have cmod (g′ w) ≤ 12 / (1 − t) if w ∈ closed_segment 0 z for w
  proof −
    have w: w ∈ ball 0 1
      using segment_bound [OF that] ‹z ∈ ball 0 1› by simp
    have ∗: ⟦⋀b. (∃w ∈ T ∪ U. w ∈ ball b 1); ⋀x. x ∈ D ⟹ g x ∉ T ∪ U⟧
⟹ ∄b. ball b 1 ⊆ g ‘ D for T U D
      by force
    have ttt: 1 − t ≤ dist w u if cmod u = 1 for u
      using ‹norm z ≤ t› segment_bound1 [OF ‹w ∈ closed_segment 0 z›]
norm_triangle_ineq2 [of u w] that
      by (simp add: dist_norm norm_minus_commute)
    have ∄b. ball b 1 ⊆ g ‘ cball 0 1
    proof (rule ∗)
      show (∃w ∈ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n
^ 2 − 1)) / pi)}) ∪
                (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (−ln(n + sqrt(real n ^ 2
− 1)) / pi)}). w ∈ ball b 1) for b
      proof −
        obtain m where m: m ∈ ℤ |Re b − m| ≤ 1/2
          by (metis Ints_of_int abs_minus_commute of_int_round_abs_le)
        show ?thesis
        proof (cases 0::real Im b rule: le_cases)
          case le
          then obtain n where 0 < n and n: |Im b − ln (n + sqrt ((real n)^2 −
1)) / pi| < 1/2
            using Schottky_lemma2 [of Im b] by blast
          have dist b (Complex m (Im b)) ≤ 1/2
            by (metis cancel_comm_monoid_add_class.diff_cancel cmod_eq_Re
complex.sel(1) complex.sel(2) dist_norm m(2) minus_complex.code)
          moreover
           have dist (Complex m (Im b)) (Complex m (ln (n + sqrt ((real n)^2 −
1)) / pi)) < 1/2
             using n by (simp add: complex_norm cmod_eq_Re complex_diff
```

```
dist_norm del: Complex_eq)
          ultimately have dist b (Complex m (ln (real n + sqrt ((real n)^2 - 1))
/ pi)) < 1
           by (simp add: dist_triangle_lt [of b Complex m (Im b)] dist_commute)
          with le m ‹0 < n› show ?thesis
            apply (rule_tac x = Complex m (ln (real n + sqrt ((real n)^2 - 1)) /
pi) in bexI)
             by (force simp del: Complex_eq greaterThan_0)+
        next
          case ge
          then obtain n where 0 < n and n: |- Im b - ln (real n + sqrt ((real
n)^2 - 1)) / pi| < 1/2
            using Schottky_lemma2 [of - Im b] by auto
          have dist b (Complex m (Im b)) ≤ 1/2
            by (metis cancel_comm_monoid_add_class.diff_cancel cmod_eq_Re
complex.sel(1) complex.sel(2) dist_norm m(2) minus_complex.code)
          moreover
          have dist (Complex m (- ln (n + sqrt ((real n)^2 - 1)) / pi)) (Complex
m (Im b))
                = | - Im b - ln (real n + sqrt ((real n)^2 - 1)) / pi|
            by (simp add: complex_norm dist_norm cmod_eq_Re complex_diff)
           ultimately have dist b (Complex m (- ln (real n + sqrt ((real n)^2 -
1)) / pi)) < 1
              using n by (simp add: dist_triangle_lt [of b Complex m (Im b)]
dist_commute)
          with ge m ‹0 < n› show ?thesis
            by (rule_tac x = Complex m (- ln (real n + sqrt ((real n)^2 - 1)) /
pi) in bexI) auto
        qed
      qed
      show g v ∉ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n ^
2 - 1)) / pi)}) ∪
                 (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (-ln(n + sqrt(real n ^ 2
- 1)) / pi)})
          if v ∈ cball 0 1 for v
        using not01 [OF that]
          by (force simp: g [OF that, symmetric] h [OF that, symmetric] dest!:
Schottky_lemma3 [of g v])
    qed
    then have 12: (1 - t) * cmod (deriv g w) / 12 < 1
      using Bloch_general [OF holg _ ttt, of 1] w by force
    have g field_differentiable at w within cball 0 1
      using holg w by (simp add: holomorphic_on_def)
    then have g field_differentiable at w within ball 0 1
      using ball_subset_cball field_differentiable_within_subset by blast
    with w have der_gw: (g has_field_derivative deriv g w) (at w)
    by (simp add: field_differentiable_within_open [of _ ball 0 1] field_differentiable_derivI)
    with DERIV_unique [OF der_gw] g' w have deriv g w = g' w
    by (metis open_ball at_within_open ball_subset_cball has_field_derivative_subset
```

```
subsetCE)
      then show cmod (g' w) ≤ 12 / (1 - t)
        using g' 12 ‹t < 1› by (simp add: field_simps)
    qed
    then have cmod (g z - g 0) ≤ 12 / (1 - t) * cmod z
     using has_contour_integral_bound_linepath [OF int_g', of 12/(1 - t)] assms
      by simp
    with ‹cmod z ≤ t› ‹t < 1› show ?thesis
      by (simp add: field_split_simps)
  qed
  have fz: f z = (1 + cos(of_real pi * h z)) / 2
    using h ‹z ∈ ball 0 1› by (auto simp: field_simps)
  have cmod (f z) ≤ exp (cmod (complex_of_real pi * h z))
    by (simp add: fz mult.commute norm_cos_plus1_le)
  also have ... ≤ exp (pi * exp (pi * (2 + 2 * r + 12 * t / (1 - t))))
  proof (simp add: norm_mult)
    have cmod (g z - g 0) ≤ 12 * t / (1 - t)
      using norm_g_12 ‹t < 1› by (simp add: norm_mult)
    then have cmod (g z) - cmod (g 0) ≤ 12 * t / (1 - t)
      using norm_triangle_ineq2 order_trans by blast
    then have *: cmod (g z) ≤ 2 + 2 * r + 12 * t / (1 - t)
      using g0_2_f0 norm_ge_zero [of f 0] nof0
        by linarith
    have cmod (h z) ≤ exp (cmod (complex_of_real pi * g z))
      using ‹z ∈ ball 0 1› by (simp add: g norm_cos_le)
    also have ... ≤ exp (pi * (2 + 2 * r + 12 * t / (1 - t)))
      using ‹t < 1› nof0 * by (simp add: norm_mult)
    finally show cmod (h z) ≤ exp (pi * (2 + 2 * r + 12 * t / (1 - t))) .
  qed
  finally show ?thesis .
qed
```

7.2 The Little Picard Theorem

```
theorem Landau_Picard:
  obtains R
    where ⋀z. 0 < R z
          ⋀f. ⟦f holomorphic_on cball 0 (R(f 0));
               ⋀z. norm z ≤ R(f 0) ⟹ f z ≠ 0 ∧ f z ≠ 1⟧ ⟹ norm(deriv f 0)
< 1
proof -
  define R where R ≡ λz. 3 * exp(pi * exp(pi*(2 + 2 * cmod z + 12)))
  show ?thesis
  proof
    show Rpos: ⋀z. 0 < R z
      by (auto simp: R_def)
    show norm(deriv f 0) < 1
        if holf: f holomorphic_on cball 0 (R(f 0))
        and Rf: ⋀z. norm z ≤ R(f 0) ⟹ f z ≠ 0 ∧ f z ≠ 1 for f
```

```
    proof −
      let ?r = R(f 0)
      define g where g ≡ f ∘ (λz. of_real ?r ∗ z)
      have 0 < ?r
        using Rpos by blast
      have holg: g holomorphic_on cball 0 1
        unfolding g_def
     proof (intro holomorphic_intros holomorphic_on_compose holomorphic_on_subset
[OF holf])
        show (∗) (complex_of_real (R (f 0))) ‘ cball 0 1 ⊆ cball 0 (R (f 0))
          using Rpos by (auto simp: less_imp_le norm_mult)
      qed
      have ∗: norm(g z) ≤ exp(pi ∗ exp(pi ∗ (2 + 2 ∗ norm (f 0) + 12 ∗ t / (1
− t))))
          if 0 < t t < 1 norm z ≤ t for t z
      proof (rule Schottky [OF holg])
        show cmod (g 0) ≤ cmod (f 0)
          by (simp add: g_def)
        show ⋀z. z ∈ cball 0 1 ⟹ ¬ (g z = 0 ∨ g z = 1)
          using Rpos by (simp add: g_def less_imp_le norm_mult Rf)
      qed (auto simp: that)
      have C1: g holomorphic_on ball 0 (1/2)
        by (rule holomorphic_on_subset [OF holg]) auto
      have C2: continuous_on (cball 0 (1/2)) g
     by (meson cball_divide_subset_numeral holg holomorphic_on_imp_continuous_on
holomorphic_on_subset)
      have C3: cmod (g z) ≤ R (f 0) / 3 if cmod (0 − z) = 1/2 for z
      proof −
        have norm(g z) ≤ exp(pi ∗ exp(pi ∗ (2 + 2 ∗ norm (f 0) + 12)))
          using ∗ [of 1/2] that by simp
        also have ... = ?r / 3
          by (simp add: R_def)
        finally show ?thesis .
      qed
      then have cmod_g′_le: cmod (deriv g 0) ∗ 3 ≤ R (f 0) ∗ 2
        using Cauchy_inequality [OF C1 C2 _ C3, of 1] by simp
      have holf′: f holomorphic_on ball 0 (R(f 0))
        by (rule holomorphic_on_subset [OF holf]) auto
      then have fd0: f field_differentiable at 0
        by (rule holomorphic_on_imp_differentiable_at [OF _ open_ball])
          (auto simp: Rpos [of f 0])
      have g_eq: deriv g 0 = of_real ?r ∗ deriv f 0
        unfolding g_def
     by (metis DERIV_imp_deriv DERIV_chain DERIV_cmult_Id fd0 field_differentiable_derivI
mult.commute mult_zero_right)
      show ?thesis
        using cmod_g′_le Rpos [of f 0]  by (simp add: g_eq norm_mult)
    qed
  qed
```

qed

lemma *little_Picard_01*:
assumes *holf*: *f holomorphic_on UNIV* **and** *f01*: $\bigwedge z.\ f\ z \neq 0 \wedge f\ z \neq 1$
obtains *c* **where** $f = (\lambda x.\ c)$
proof −
obtain *R*
where *Rpos*: $\bigwedge z.\ 0 < R\ z$
and *R*: $\bigwedge h.\ [\![h\ holomorphic\_on\ cball\ 0\ (R(h\ 0)); \bigwedge z.\ norm\ z \leq R(h\ 0) \Longrightarrow h\ z \neq 0 \wedge h\ z \neq 1]\!] \Longrightarrow norm(deriv\ h\ 0) < 1$
using *Landau_Picard* **by** *metis*
have *contf*: *continuous_on UNIV f*
by (*simp add*: *holf holomorphic_on_imp_continuous_on*)
show *?thesis*
proof (*cases* $\forall x.\ deriv\ f\ x = 0$)
case *True*
have (*f has_field_derivative 0*) (*at x*) **for** *x*
by (*metis True UNIV_I holf holomorphic_derivI open_UNIV*)
then obtain *c* **where** $\bigwedge x.\ f(x) = c$
by (*meson UNIV_I DERIV_zero_connected_constant* [*OF connected_UNIV open_UNIV finite.emptyI contf*])
then show *?thesis*
using *that* **by** *auto*
next
case *False*
then obtain *w* **where** *w*: *deriv f w* $\neq$ *0* **by** *auto*
define *fw* **where** $fw \equiv (f \circ (\lambda z.\ w + z\ /\ deriv\ f\ w))$
have *norm_let1*: $norm(deriv\ fw\ 0) < 1$
proof (*rule R*)
show *fw holomorphic_on cball 0* (*R* (*fw 0*))
unfolding *fw_def*
by (*intro holomorphic_intros holomorphic_on_compose w holomorphic_on_subset* [*OF holf*] *subset_UNIV*)
show $fw\ z \neq 0 \wedge fw\ z \neq 1$ **if** *cmod z* $\leq$ *R* (*fw 0*) **for** *z*
using *f01* **by** (*simp add*: *fw_def*)
qed
have (*fw has_field_derivative deriv f w* $*$ *inverse* (*deriv f w*)) (*at 0*)
unfolding *fw_def*
apply (*intro DERIV_chain derivative_eq_intros w*)+
using *holf holomorphic_derivI* **by** (*force simp*: *field_simps*)+
then show *?thesis*
using *norm_let1 w* **by** (*simp add*: *DERIV_imp_deriv*)
qed
qed

theorem *little_Picard*:
assumes *holf*: *f holomorphic_on UNIV*

```
    and a ≠ b range f ∩ {a,b} = {}
  obtains c where f = (λx. c)
proof −
  let ?g = λx. 1/(b − a)∗(f x − b) + 1
  obtain c where ?g = (λx. c)
  proof (rule little_Picard_01)
    show ?g holomorphic_on UNIV
      by (intro holomorphic_intros holf)
    show ⋀z. ?g z ≠ 0 ∧ ?g z ≠ 1
      using assms by (auto simp: field_simps)
  qed auto
  then have ?g x = c for x
    by meson
  then have f x = c ∗ (b−a) + a for x
    using assms by (auto simp: field_simps)
  then show ?thesis
    using that by blast
qed
```

A couple of little applications of Little Picard

```
lemma holomorphic_periodic_fixpoint:
  assumes holf: f holomorphic_on UNIV
      and p ≠ 0 and per: ⋀z. f(z + p) = f z
  obtains x where f x = x
proof −
  have False if non: ⋀x. f x ≠ x
  proof −
    obtain c where (λz. f z − z) = (λz. c)
    proof (rule little_Picard)
      show (λz. f z − z) holomorphic_on UNIV
        by (simp add: holf holomorphic_on_diff)
      show range (λz. f z − z) ∩ {p,0} = {}
          using assms non by auto (metis add.commute diff_eq_eq)
      qed (auto simp: assms)
    with per show False
      by (metis add.commute add_cancel_left_left ‹p ≠ 0› diff_add_cancel)
  qed
  then show ?thesis
    using that by blast
qed


lemma holomorphic_involution_point:
  assumes holfU: f holomorphic_on UNIV and non: ⋀a. f ≠ (λx. a + x)
  obtains x where f(f x) = x
proof −
  { assume non_ff [simp]: ⋀x. f(f x) ≠ x
    then have non_fp [simp]: f z ≠ z for z
      by metis
```

```
  have holf: f holomorphic_on X for X
    using assms holomorphic_on_subset by blast
  obtain c where c: (λx. (f(f x) − x)/(f x − x)) = (λx. c)
  proof (rule little_Picard_01)
    show (λx. (f(f x) − x)/(f x − x)) holomorphic_on UNIV
      using non_fp
    by (intro holomorphic_intros holf holomorphic_on_compose [unfolded o_def,
OF holf]) auto
  qed auto
  then obtain c ≠ 0 c ≠ 1
  by (metis (no_types, opaque_lifting) non_ff diff_zero divide_eq_0_iff right_inverse_eq)
  have eq: f(f x) − c * f x = x*(1 − c) for x
    using fun_cong [OF c, of x] by (simp add: field_simps)
  have df_times_dff: deriv f z * (deriv f (f z) − c) = 1 * (1 − c) for z
  proof (rule DERIV_unique)
    show ((λx. f (f x) − c * f x) has_field_derivative
          deriv f z * (deriv f (f z) − c)) (at z)
      by (rule derivative_eq_intros holomorphic_derivI [OF holfU]
            DERIV_chain [unfolded o_def, where f=f and g=f] | simp add:
algebra_simps)+
    show ((λx. f (f x) − c * f x) has_field_derivative 1 * (1 − c)) (at z)
      by (simp add: eq mult_commute_abs)
  qed
  { fix z::complex
    obtain k where k: deriv f ∘ f = (λx. k)
    proof (rule little_Picard)
      show (deriv f ∘ f) holomorphic_on UNIV
        by (meson holfU holomorphic_deriv holomorphic_on_compose holomor-
phic_on_subset open_UNIV subset_UNIV)
      obtain deriv f (f x) ≠ 0 deriv f (f x) ≠ c for x
        using df_times_dff ‹c ≠ 1› eq_iff_diff_eq_0
        by (metis lambda_one mult_zero_left mult_zero_right)
      then show range (deriv f ∘ f) ∩ {0,c} = {}
        by force
    qed (use ‹c ≠ 0› in auto)
    have ¬ f constant_on UNIV
      by (meson UNIV_I non_ff constant_on_def)
    with holf open_mapping_thm have open(range f)
      by blast
    obtain l where l: ⋀x. f x − k * x = l
    proof (rule DERIV_zero_connected_constant [of UNIV {} λx. f x − k * x],
simp_all)
      have deriv f w − k = 0 for w
        proof (rule analytic_continuation [OF _ open_UNIV connected_UNIV
subset_UNIV, of λz. deriv f z − k f z range f w])
        show (λz. deriv f z − k) holomorphic_on UNIV
          by (intro holomorphic_intros holf open_UNIV)
        show f z islimpt range f
              by (metis (no_types, lifting) IntI UNIV_I ‹open (range f)› im-
```

```
age_eqI inf.absorb_iff2 inf_aci(1) islimpt_UNIV islimpt_eq_acc_point open_Int
top_greatest)
        show ⋀z. z ∈ range f ⟹ deriv f z − k = 0
          by (metis comp_def diff_self image_iff k)
      qed auto
      moreover
      have ((λx. f x − k ∗ x) has_field_derivative deriv f x − k) (at x) for x
           by (metis DERIV_cmult_Id Deriv.field_differentiable_diff UNIV_I
field_differentiable_derivI holf holomorphic_on_def)
      ultimately
      show ∀x. ((λx. f x − k ∗ x) has_field_derivative 0) (at x)
        by auto
      show continuous_on UNIV (λx. f x − k ∗ x)
     by (simp add: continuous_on_diff holf holomorphic_on_imp_continuous_on)
    qed (auto simp: connected_UNIV)
    have False
    proof (cases k=1)
      case True
      then have ∃x. k ∗ x + l ≠ a + x for a
        using l non [of a] ext [of f (+) a]
        by (metis add.commute diff_eq_eq)
      with True show ?thesis by auto
    next
      case False
      have ⋀x. (1 − k) ∗ x ≠ f 0
        using l [of 0]
          by (simp add: algebra_simps) (metis diff_add_cancel l mult.commute
non_fp)
      then show False
     by (metis False eq_iff_diff_eq_0 mult.commute nonzero_mult_div_cancel_right
times_divide_eq_right)
    qed
  }
 }
 then show thesis
   using that by blast
qed
```

7.3 The Arzelà–Ascoli theorem

```
lemma subsequence_diagonalization_lemma:
 fixes P :: nat ⇒ (nat ⇒ 'a) ⇒ bool
 assumes sub: ⋀i r. ∃k. strict_mono (k :: nat ⇒ nat) ∧ P i (r ∘ k)
    and P_P: ⋀i r::nat ⇒ 'a. ⋀k1 k2 N.
                ⟦P i (r ∘ k1); ⋀j. N ≤ j ⟹ ∃j'. j ≤ j' ∧ k2 j = k1 j'⟧ ⟹ P i (r
∘ k2)
  obtains k where strict_mono (k :: nat ⇒ nat) ⋀i. P i (r ∘ k)
proof −
 obtain kk where ⋀i r. strict_mono (kk i r :: nat ⇒ nat) ∧ P i (r ∘ (kk i r))
```

```
    using sub by metis
  then have sub_kk: ⋀i r. strict_mono (kk i r) and P_kk: ⋀i r. P i (r ∘ (kk i
r))
    by auto
  define rr where rr ≡ rec_nat (kk 0 r) (λn x. x ∘ kk (Suc n) (r ∘ x))
  then have [simp]: rr 0 = kk 0 r ⋀n. rr(Suc n) = rr n ∘ kk (Suc n) (r ∘ rr n)
    by auto
  show thesis
  proof
    have sub_rr: strict_mono (rr i) for i
      using sub_kk  by (induction i) (auto simp: strict_mono_def o_def)
    have P_rr: P i (r ∘ rr i) for i
      using P_kk  by (induction i) (auto simp: o_def)
    have i ≤ i+d ⟹ rr i n ≤ rr (i+d) n for d i n
    proof (induction d)
      case 0 then show ?case
        by simp
    next
      case (Suc d) then show ?case
        using seq_suble [OF sub_kk] strict_mono_less_eq [OF sub_rr]
        by (simp add: order_subst1)
    qed
    then have ⋀i j n. i ≤ j ⟹ rr i n ≤ rr j n
      by (metis le_iff_add)
    show strict_mono (λn. rr n n)
      unfolding strict_mono_Suc_iff
        by (simp add: Suc_le_lessD strict_monoD strict_mono_imp_increasing
sub_kk sub_rr)
    have ∃j. i ≤ j ∧ rr (n+d) i = rr n j for d n i
    proof (induction d arbitrary: i)
      case (Suc d)
      then show ?case
        using seq_suble [OF sub_kk] by simp (meson order_trans)
    qed auto
    then have ⋀m n i. n ≤ m ⟹ ∃j. i ≤ j ∧ rr m i = rr n j
      by (metis le_iff_add)
    then show P i (r ∘ (λn. rr n n)) for i
      by (meson P_rr P_P)
  qed
qed

lemma function_convergent_subsequence:
  fixes f :: [nat,'a] ⇒ 'b::{real_normed_vector,heine_borel}
  assumes countable S and M: ⋀n::nat. ⋀x. x ∈ S ⟹ norm(f n x) ≤ M
   obtains k where strict_mono (k::nat⇒nat) ⋀x. x ∈ S ⟹ ∃l. (λn. f (k n) x)
⟶ l
proof (cases S = {})
  case True
  then show ?thesis
```

```
    using strict_mono_id that by fastforce
next
  case False
  with ‹countable S› obtain σ :: nat ⇒ 'a where σ: S = range σ
    using uncountable_def by blast
  obtain k where strict_mono k and k: ⋀i. ∃ l. (λn. (f ∘ k) n (σ i)) ⟶ l
  proof (rule subsequence_diagonalization_lemma
      [of λi r. ∃ l. ((λn. (f ∘ r) n (σ i)) ⟶ l) sequentially id])
    show ∃ k::nat⇒nat. strict_mono k ∧ (∃ l. (λn. (f ∘ (r ∘ k)) n (σ i)) ⟶ l)
for i r
    proof −
      have f (r n) (σ i) ∈ cball 0 M for n
        by (simp add: σ M)
      then show ?thesis
        using compact_def [of cball (0::'b) M] by (force simp: o_def)
    qed
    show ∃ l. (λn. (f ∘ (r ∘ k2)) n (σ i)) ⟶ l
      if ∃ l. (λn. (f ∘ (r ∘ k1)) n (σ i)) ⟶ l ⋀j. N ≤ j ⟹ ∃ j'≥j. k2 j = k1 j'
      for i N and r k1 k2 :: nat⇒nat
      using that
      by (simp add: lim_sequentially) (metis (no_types, opaque_lifting) le_cases
order_trans)
  qed auto
  with σ that show ?thesis
    by force
qed


theorem Arzela_Ascoli:
  fixes ℱ :: [nat,'a::euclidean_space] ⇒ 'b::{real_normed_vector,heine_borel}
  assumes compact S
      and M: ⋀n x. x ∈ S ⟹ norm(ℱ n x) ≤ M
      and equicont:
          ⋀x e. ⟦x ∈ S; 0 < e⟧
                ⟹ ∃ d. 0 < d ∧ (∀ n y. y ∈ S ∧ norm(x − y) < d ⟶ norm(ℱ n
x − ℱ n y) < e)
  obtains g k where continuous_on S g strict_mono (k :: nat ⇒ nat)
                    ⋀e. 0 < e ⟹ ∃ N. ∀ n x. n ≥ N ∧ x ∈ S ⟶ norm(ℱ(k n) x −
g x) < e
proof −
  have UEQ: ⋀e. 0 < e ⟹ ∃ d. 0 < d ∧ (∀ n. ∀ x ∈ S. ∀ x' ∈ S. dist x' x < d
⟶ dist (ℱ n x') (ℱ n x) < e)
    apply (rule compact_uniformly_equicontinuous [OF ‹compact S›, of range ℱ])
    using equicont by (force simp: dist_commute dist_norm)+
  have continuous_on S g
      if ⋀e. 0 < e ⟹ ∃ N. ∀ n x. n ≥ N ∧ x ∈ S ⟶ norm(ℱ(r n) x − g x) < e
      for g:: 'a ⇒ 'b and r :: nat ⇒ nat
  proof (rule uniform_limit_theorem [of _ ℱ ∘ r])
    have continuous_on S (ℱ (r n)) for n
```

```
    using UEQ by (force simp: continuous_on_iff)
  then show ∀F n in sequentially. continuous_on S ((ℱ ∘ r) n)
    by (simp add: eventually_sequentially)
  show uniform_limit S (ℱ ∘ r) g sequentially
    using that by (metis (mono_tags, opaque_lifting) comp_apply dist_norm
uniform_limit_sequentially_iff)
 qed auto
 moreover
 obtain R where countable R R ⊆ S and SR: S ⊆ closure R
  by (metis separable that)
 obtain k where strict_mono k and k: ⋀x. x ∈ R ⟹ ∃l. (λn. ℱ (k n) x)
⟶ l
  using ‹R ⊆ S› by (force intro: function_convergent_subsequence [OF ‹countable
R› M])
 then have Cauchy: Cauchy ((λn. ℱ (k n) x)) if x ∈ R for x
  using convergent_eq_Cauchy that by blast
 have ∃N. ∀m n x. N ≤ m ∧ N ≤ n ∧ x ∈ S ⟶ dist ((ℱ ∘ k) m x) ((ℱ ∘ k)
n x) < e
  if 0 < e for e
 proof −
  obtain d where 0 < d
   and d: ⋀n. ∀x ∈ S. ∀x′ ∈ S. dist x′ x < d ⟶ dist (ℱ n x′) (ℱ n x) < e/3
   by (metis UEQ ‹0 < e› divide_pos_pos zero_less_numeral)
  obtain T where T ⊆ R and finite T and T: S ⊆ (⋃c∈T. ball c d)
  proof (rule compactE_image [OF ‹compact S›, of R (λx. ball x d)])
   have closure R ⊆ (⋃c∈R. ball c d)
    using ‹0 < d› by (auto simp: closure_approachable)
   with SR show S ⊆ (⋃c∈R. ball c d)
    by auto
  qed auto
  have ∃M. ∀m≥M. ∀n≥M. dist (ℱ (k m) x) (ℱ (k n) x) < e/3 if x ∈ R for x
   using Cauchy ‹0 < e› that unfolding Cauchy_def
   by (metis less_divide_eq_numeral1(1) mult_zero_left)
  then obtain MF where MF: ⋀x m n. ⟦x ∈ R; m ≥ MF x; n ≥ MF x⟧ ⟹
norm (ℱ (k m) x − ℱ (k n) x) < e/3
   using dist_norm by metis
  have dist ((ℱ ∘ k) m x) ((ℱ ∘ k) n x) < e
    if m: Max (MF ‘ T) ≤ m and n: Max (MF ‘ T) ≤ n x ∈ S for m n x
  proof −
   obtain t where t ∈ T and t: x ∈ ball t d
    using ‹x ∈ S› T by auto
   have norm(ℱ (k m) t − ℱ (k m) x) < e / 3
    by (metis ‹R ⊆ S› ‹T ⊆ R› ‹t ∈ T› d dist_norm mem_ball subset_iff t ‹x
∈ S›)
   moreover
   have norm(ℱ (k n) t − ℱ (k n) x) < e / 3
    by (metis ‹R ⊆ S› ‹T ⊆ R› ‹t ∈ T› subsetD d dist_norm mem_ball t ‹x
∈ S›)
   moreover
```

```
    have norm(F (k m) t - F (k n) t) < e / 3
    proof (rule MF)
      show t ∈ R
        using ‹T ⊆ R› ‹t ∈ T› by blast
      show MF t ≤ m MF t ≤ n
        by (meson Max_ge ‹finite T› ‹t ∈ T› finite_imageI imageI le_trans m
n)+
    qed
    ultimately
    show ?thesis
      unfolding dist_norm [symmetric] o_def
      by (metis dist_triangle_third dist_commute)
  qed
  then show ?thesis
    by force
 qed
 then obtain g where ∀ e>0. ∃ N. ∀ n x. N ≤ n ∧ x ∈ S ⟶ norm ((F ∘ k) n
x - g x) < e
   using uniformly_convergent_eq_cauchy [of λx. x ∈ S F ∘ k] by (auto simp
add: dist_norm)
 ultimately show thesis
   by (metis ‹strict_mono k› comp_apply that)
qed
```

7.3.1 Montel's theorem

a sequence of holomorphic functions uniformly bounded on compact subsets of an open set S has a subsequence that converges to a holomorphic function, and converges *uniformly* on compact subsets of S.

```
theorem Montel:
 fixes F :: [nat,complex] ⇒ complex
 assumes open S
    and H: ⋀h. h ∈ H ⟹ h holomorphic_on S
    and bounded: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ ∃ B. ∀ h ∈ H. ∀ z ∈ K. norm(h
z) ≤ B
    and rng_f: range F ⊆ H
 obtains g r
   where g holomorphic_on S strict_mono (r :: nat ⇒ nat)
       ⋀x. x ∈ S ⟹ ((λn. F (r n) x) ⟶⟶ g x) sequentially
        ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K (F ∘ r) g sequentially

proof -
 obtain K where comK: ⋀n. compact(K n) and KS: ⋀n::nat. K n ⊆ S
          and subK: ⋀X. ⟦compact X; X ⊆ S⟧ ⟹ ∃ N. ∀ n≥N. X ⊆ K n
   using open_Union_compact_subsets [OF ‹open S›] by metis
 then have ⋀i. ∃ B. ∀ h ∈ H. ∀ z ∈ K i. norm(h z) ≤ B
   by (simp add: bounded)
 then obtain B where B: ⋀i h z. ⟦h ∈ H; z ∈ K i⟧ ⟹ norm(h z) ≤ B i
   by metis
```

```
  have *: ∃r g. strict_mono (r::nat⇒nat) ∧ (∀e > 0. ∃N. ∀n≥N. ∀x ∈ K i.
norm((F ∘ r) n x − g x) < e)
       if ⋀n. F n ∈ H for F i
  proof −
    obtain g k where continuous_on (K i) g strict_mono (k::nat⇒nat)
                     ⋀e. 0 < e ⟹ ∃N. ∀n≥N. ∀x ∈ K i. norm(F(k n) x − g x) < e
    proof (rule Arzela_Ascoli [of K i F B i])
      show ∃d>0. ∀n y. y ∈ K i ∧ cmod (z − y) < d ⟶ cmod (F n z − F n
y) < e
           if z: z ∈ K i and 0 < e for z e
      proof −
        obtain r where 0 < r and r: cball z r ⊆ S
          using z KS [of i] ‹open S› by (force simp: open_contains_cball)
        have cball z (2/3 * r) ⊆ cball z r
          using ‹0 < r› by (simp add: cball_subset_cball_iff)
        then have z23S: cball z (2/3 * r) ⊆ S
          using r by blast
        obtain M where 0 < M and M: ⋀n w. dist z w ≤ 2/3 * r ⟹ norm(F
n w) ≤ M
        proof −
          obtain N where N: ∀n≥N. cball z (2/3 * r) ⊆ K n
            using subK compact_cball [of z (2/3 * r)] z23S by force
          have cmod (F n w) ≤ |B N| + 1 if dist z w ≤ 2/3 * r for n w
          proof −
            have w ∈ K N
              using N mem_cball that by blast
            then have cmod (F n w) ≤ B N
              using B ‹⋀n. F n ∈ H› by blast
            also have ... ≤ |B N| + 1
              by simp
            finally show ?thesis .
          qed
          then show ?thesis
            by (rule_tac M=|B N| + 1 in that) auto
        qed
        have cmod (F n z − F n y) < e
             if y ∈ K i and y_near_z: cmod (z − y) < r/3 cmod (z − y) < e * r
/ (6 * M)
             for n y
        proof −
          have ((λw. F n w / (w − ξ)) has_contour_integral
                  (2 * pi) * ı * winding_number (circlepath z (2/3 * r)) ξ * F n ξ)
                (circlepath z (2/3 * r))
            if dist ξ z < (2/3 * r) for ξ
          proof (rule Cauchy_integral_formula_convex_simple)
            have F n holomorphic_on S
              by (simp add: H ‹⋀n. F n ∈ H›)
            with z23S show F n holomorphic_on cball z (2/3 * r)
              using holomorphic_on_subset by blast
```

```
        qed (use that ‹0 < r› in ‹auto simp: dist_commute›)
        then have *: ((λw. ℱ n w / (w − ξ)) has_contour_integral (2 * pi) * i
* ℱ n ξ)
                     (circlepath z (2/3 * r))
          if dist ξ z < (2/3 * r) for ξ
         using that by (simp add: winding_number_circlepath dist_norm)
        have y: ((λw. ℱ n w / (w − y)) has_contour_integral (2 * pi) * i * ℱ
n y)
                    (circlepath z (2/3 * r))
         proof (rule *)
          show dist y z < 2/3 * r
           using that ‹0 < r› by (simp only: dist_norm norm_minus_commute)
         qed
        have z: ((λw. ℱ n w / (w − z)) has_contour_integral (2 * pi) * i * ℱ n
z)
                (circlepath z (2/3 * r))
          using ‹0 < r› by (force intro!: *)
         have le_er: cmod (ℱ n x / (x − y) − ℱ n x / (x − z)) ≤ e / r
            if cmod (x − z) = r/3 + r/3 for x
        proof −
          have ¬ (cmod (x − y) < r/3)
            using y_near_z(1) that ‹M > 0› ‹r > 0›
           by (metis (full_types) norm_diff_triangle_less norm_minus_commute
order_less_irrefl)
          then have r4_le_xy: r/4 ≤ cmod (x − y)
            using ‹r > 0› by simp
          then have neq: x ≠ y x ≠ z
        using that ‹r > 0› by (auto simp: field_split_simps norm_minus_commute)
          have leM: cmod (ℱ n x) ≤ M
            by (simp add: M dist_commute dist_norm that)
           have cmod (ℱ n x / (x − y) − ℱ n x / (x − z)) = cmod (ℱ n x) *
cmod (1 / (x − y) − 1 / (x − z))
         by (metis (no_types, lifting) divide_inverse mult.left_neutral norm_mult
right_diff_distrib')
          also have ... = cmod (ℱ n x) * cmod ((y − z) / ((x − y) * (x − z)))
            using neq by (simp add: field_split_simps)
          also have ... = cmod (ℱ n x) * (cmod (y − z) / (cmod(x − y) * (2/3
* r)))
            by (simp add: norm_mult norm_divide that)
          also have ... ≤ M * (cmod (y − z) / (cmod(x − y) * (2/3 * r)))
            using ‹r > 0› ‹M > 0› by (intro mult_mono [OF leM]) auto
          also have ... < M * ((e * r / (6 * M)) / (cmod(x − y) * (2/3 * r)))
            unfolding mult_less_cancel_left
            using y_near_z(2) ‹M > 0› ‹r > 0› neq
            by (simp add: field_simps mult_less_0_iff norm_minus_commute)
          also have ... ≤ e/r
            using ‹e > 0› ‹r > 0› r4_le_xy by (simp add: field_split_simps)
          finally show ?thesis by simp
        qed
```

```
        have (2 * pi) * cmod (\mathcal{F} n y - \mathcal{F} n z) = cmod ((2 * pi) * i * \mathcal{F} n y -
(2 * pi) * i * \mathcal{F} n z)
          by (simp add: right_diff_distrib [symmetric] norm_mult)
        also have cmod ((2 * pi) * i * \mathcal{F} n y - (2 * pi) * i * \mathcal{F} n z) \leq e / r *
(2 * pi * (2/3 * r))

      proof (rule has_contour_integral_bound_circlepath [OF has_contour_integral_diff
[OF y z]])
        show \bigwedgex. cmod (x - z) = 2/3 * r \Longrightarrow cmod (\mathcal{F} n x / (x - y) - \mathcal{F} n
x / (x - z)) \leq e / r
          using le_er by auto
        qed (use ‹e > 0› ‹r > 0› in auto)
        also have ... = (2 * pi) * e * ((2/3))
          using ‹r > 0› by (simp add: field_split_simps)
        finally have cmod (\mathcal{F} n y - \mathcal{F} n z) \leq e * (2/3)
          by simp
        also have ... < e
          using ‹e > 0› by simp
        finally show ?thesis by (simp add: norm_minus_commute)
      qed
      then show ?thesis
        apply (rule_tac x=min (r/3) ((e * r)/(6 * M)) in exI)
        using ‹0 < e› ‹0 < r› ‹0 < M› by simp
    qed
    show \bigwedgen x.  x \in K i \Longrightarrow cmod (\mathcal{F} n x) \leq B i
      using B ‹\bigwedgen. \mathcal{F} n \in \mathcal{H}› by blast
  next
    fix g :: complex \Rightarrow complex and k :: nat \Rightarrow nat
    assume *: \bigwedge(g::complex\Rightarrowcomplex) (k::nat\Rightarrownat). continuous_on (K i) g \Longrightarrow
              strict_mono k \Longrightarrow
              (\bigwedgee. 0 < e \Longrightarrow \exists N. \forall n\geqN. \forall x\inK i. cmod (\mathcal{F} (k n) x - g x) <
e) \Longrightarrow thesis
        continuous_on (K i) g
        strict_mono k
        \bigwedgee. 0 < e \Longrightarrow \exists N. \forall n x. N \leq n \wedge x \in K i \longrightarrow cmod (\mathcal{F} (k n) x - g
x) < e
    show ?thesis
      by (rule *(1)[OF *(2,3)], drule *(4)) auto
  qed (use comK in simp_all)
  then show ?thesis
    by auto
qed
define \Phi where \Phi \equiv \lambdag i r. \lambdak::nat\Rightarrownat. \forall e>0. \exists N. \forall n\geqN. \forall x\inK i. cmod
((\mathcal{F} \circ (r \circ k)) n x - g x) < e
obtain k :: nat \Rightarrow nat where strict_mono k and k: \bigwedgei. \exists g. \Phi g i id k
proof (rule subsequence_diagonalization_lemma [where r=id])
  show \exists g. \Phi g i id (r \circ k2)
    if ex: \exists g. \Phi g i id (r \circ k1) and \bigwedgej. N \leq j \Longrightarrow \exists j'\geqj. k2 j = k1 j'
    for i k1 k2 N and r::nat\Rightarrownat
```

```
    proof −
      obtain g where Φ g i id (r ∘ k1)
        using ex by blast
      then have Φ g i id (r ∘ k2)
        using that
        by (simp add: Φ_def) (metis (no_types, opaque_lifting) le_trans linear)
      then show ?thesis
        by metis
    qed
    have ∃k g. strict_mono (k::nat⇒nat) ∧ Φ g i id (r ∘ k) for i r
      unfolding Φ_def o_assoc using rng_f by (force intro!: ∗)
    then show ⋀i r. ∃k. strict_mono (k::nat⇒nat) ∧ (∃g. Φ g i id (r ∘ k))
      by force
  qed fastforce
  have ∃l. ∀e>0. ∃N. ∀n≥N. norm(F (k n) z − l) < e if z ∈ S for z
  proof −
    obtain G where G: ⋀i e. e > 0 ⟹ ∃M. ∀n≥M. ∀x∈K i. cmod ((F ∘ k) n
x − G i x) < e
      using k unfolding Φ_def by (metis id_comp)
    obtain N where ⋀n. n ≥ N ⟹ z ∈ K n
      using subK [of {z}] that ‹z ∈ S› by auto
    moreover have ⋀e. e > 0 ⟹ ∃M. ∀n≥M. ∀x∈K N. cmod ((F ∘ k) n x −
G N x) < e
      using G by auto
    ultimately show ?thesis
      by (metis comp_apply order_refl)
  qed
  then obtain g where g: ⋀z e. ⟦z ∈ S; e > 0⟧ ⟹ ∃N. ∀n≥N. norm(F (k n)
z − g z) < e
    by metis
  show ?thesis
  proof
    show g_lim: ⋀x. x ∈ S ⟹ (λn. F (k n) x) ⟶ g x
      by (simp add: lim_sequentially g dist_norm)
    have dg_le_e: ∃N. ∀n≥N. ∀x∈T. cmod (F (k n) x − g x) < e
      if T: compact T T ⊆ S and 0 < e for T e
    proof −
      obtain N where N: ⋀n. n ≥ N ⟹ T ⊆ K n
        using subK [OF T] by blast
      obtain h where h: ⋀e. e>0 ⟹ ∃M. ∀n≥M. ∀x∈K N. cmod ((F ∘ k) n x
− h x) < e
        using k unfolding Φ_def by (metis id_comp)
      have geq: g w = h w if w ∈ T for w
      proof (rule LIMSEQ_unique)
        show (λn. F (k n) w) ⟶ g w
          using ‹T ⊆ S› g_lim that by blast
        show (λn. F (k n) w) ⟶ h w
          using h N that by (force simp: lim_sequentially dist_norm)
      qed
```

```
    show ?thesis
      using T h N ‹0 < e› by (fastforce simp add: geq)
  qed
  then show ⋀K. ⟦compact K; K ⊆ S⟧
      ⟹ uniform_limit K (ℱ ∘ k) g sequentially
    by (simp add: uniform_limit_iff dist_norm eventually_sequentially)
  show g holomorphic_on S
  proof (rule holomorphic_uniform_sequence [OF ‹open S› ℋ])
    show ⋀n. (ℱ ∘ k) n ∈ ℋ
      by (simp add: range_subsetD rng_f)
     show ∃d>0. cball z d ⊆ S ∧ uniform_limit (cball z d) (λn. (ℱ ∘ k) n) g
sequentially
      if z ∈ S for z
    proof −
      obtain d where d: d>0 cball z d ⊆ S
        using ‹open S› ‹z ∈ S› open_contains_cball by blast
      then have uniform_limit (cball z d) (ℱ ∘ k) g sequentially
        using dg_le_e compact_cball by (auto simp: uniform_limit_iff eventu-
ally_sequentially dist_norm)
      with d show ?thesis by blast
    qed
  qed
 qed (auto simp: ‹strict_mono k›)
qed
```

7.4 Some simple but useful cases of Hurwitz's theorem

```
proposition Hurwitz_no_zeros:
 assumes S: open S connected S
    and holf: ⋀n::nat. ℱ n holomorphic_on S
    and holg: g holomorphic_on S
    and ul_g: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K ℱ g sequentially
    and nonconst: ¬ g constant_on S
    and nz: ⋀n z. z ∈ S ⟹ ℱ n z ≠ 0
    and z0 ∈ S
    shows g z0 ≠ 0
proof
 assume g0: g z0 = 0
 obtain h r m
   where 0 < m 0 < r and subS: ball z0 r ⊆ S
    and holh: h holomorphic_on ball z0 r
    and geq: ⋀w. w ∈ ball z0 r ⟹ g w = (w − z0)^m ∗ h w
    and hnz: ⋀w. w ∈ ball z0 r ⟹ h w ≠ 0
  by (blast intro: holomorphic_factor_zero_nonconstant [OF holg S ‹z0 ∈ S› g0
nonconst])
 then have holf0: ℱ n holomorphic_on ball z0 r for n
   by (meson holf holomorphic_on_subset)
 have ∗: ((λz. deriv (ℱ n) z / ℱ n z) has_contour_integral 0) (circlepath z0
(r/2)) for n
```

proof (*rule Cauchy_theorem_disc_simple*)
show (λ*z*. *deriv* (ℱ *n*) *z* / ℱ *n z*) *holomorphic_on ball z0 r*
by (*metis* (*no_types*) ‹*open S*› *holf holomorphic_deriv holomorphic_on_divide holomorphic_on_subset nz subS*)
qed (*use* ‹0 < *r*› **in** *auto*)
have *hol_dg*: *deriv g holomorphic_on S*
by (*simp add*: ‹*open S*› *holg holomorphic_deriv*)
have *continuous_on* (*sphere z0* (*r*/2)) (*deriv g*)
using ‹0 < *r*› *subS*
by (*intro holomorphic_on_imp_continuous_on holomorphic_on_subset* [*OF hol_dg*]) *auto*
then have *compact* (*deriv g* ‘ (*sphere z0* (*r*/2)))
by (*rule compact_continuous_image* [*OF _ compact_sphere*])
then have *bo_dg*: *bounded* (*deriv g* ‘ (*sphere z0* (*r*/2)))
using *compact_imp_bounded* **by** *blast*
have *continuous_on* (*sphere z0* (*r*/2)) (*cmod* ∘ *g*)
using ‹0 < *r*› *subS*
by (*intro continuous_intros holomorphic_on_imp_continuous_on holomorphic_on_subset* [*OF holg*]) *auto*
then have *compact* ((*cmod* ∘ *g*) ‘ *sphere z0* (*r*/2))
by (*rule compact_continuous_image* [*OF _ compact_sphere*])
moreover have (*cmod* ∘ *g*) ‘ *sphere z0* (*r*/2) ≠ {}
using ‹0 < *r*› **by** *auto*
ultimately obtain *b* **where** *b*: *b* ∈ (*cmod* ∘ *g*) ‘ *sphere z0* (*r*/2)
⋀*t*. *t* ∈ (*cmod* ∘ *g*) ‘ *sphere z0* (*r*/2) ⟹ *b* ≤ *t*
using *compact_attains_inf* [*of* (*norm* ∘ *g*) ‘ (*sphere z0* (*r*/2))] **by** *blast*
have (λ*n*. *contour_integral* (*circlepath z0* (*r*/2)) (λ*z*. *deriv* (ℱ *n*) *z* / ℱ *n z*))
⟶
contour_integral (*circlepath z0* (*r*/2)) (λ*z*. *deriv g z* / *g z*)
proof (*rule contour_integral_uniform_limit_circlepath*)
show ∀ *F n in sequentially*. (λ*z*. *deriv* (ℱ *n*) *z* / ℱ *n z*) *contour_integrable_on circlepath z0* (*r*/2)
using ∗ *contour_integrable_on_def eventually_sequentiallyI* **by** *meson*
show *uniform_limit* (*sphere z0* (*r*/2)) (λ*n z*. *deriv* (ℱ *n*) *z* / ℱ *n z*) (λ*z*. *deriv g z* / *g z*) *sequentially*
proof (*rule uniform_lim_divide* [*OF _ _ bo_dg*])
show *uniform_limit* (*sphere z0* (*r*/2)) (λ*a*. *deriv* (ℱ *a*)) (*deriv g*) *sequentially*
proof (*rule uniform_limitI*)
fix *e*::*real*
assume 0 < *e*

show ∀ *F n in sequentially*. ∀ *x* ∈ *sphere z0* (*r*/2). *dist* (*deriv* (ℱ *n*) *x*) (*deriv g x*) < *e*
proof −
have *dist* (*deriv* (ℱ *n*) *w*) (*deriv g w*) < *e*
if *e8*: ⋀*x*. *dist z0 x* ≤ 3 ∗ *r* / 4 ⟹ *dist* (ℱ *n x*) (*g x*) ∗ 8 < *r* ∗ *e*
and *w*: *w* ∈ *sphere z0* (*r*/2) **for** *n w*
proof −
have *ball w* (*r*/4) ⊆ *ball z0 r* *cball w* (*r*/4) ⊆ *ball z0 r*

```
        using ‹0 < r› w by (simp_all add: ball_subset_ball_iff cball_subset_ball_iff
dist_commute)
         with subS have wr4_sub: ball w (r/4) ⊆ S cball w (r/4) ⊆ S by force+
          moreover
          have (λz. ℱ n z − g z) holomorphic_on S
            by (intro holomorphic_intros holf holg)
          ultimately have hol: (λz. ℱ n z − g z) holomorphic_on ball w (r/4)
            and cont: continuous_on (cball w (r / 4)) (λz. ℱ n z − g z)
        using holomorphic_on_subset by (blast intro: holomorphic_on_imp_continuous_on)+
          have w ∈ S
            using ‹0 < r› wr4_sub by auto
          have dist z0 y ≤ 3 * r / 4 if dist w y < r/4 for y
          proof (rule dist_triangle_le [where z=w])
            show dist z0 w + dist y w ≤ 3 * r / 4
              using w that by (simp add: dist_commute)
          qed
          with e8 have in_ball: ⋀y. y ∈ ball w (r/4) ⟹ ℱ n y − g y ∈ ball 0
(r/4 * e/2)
            by (simp add: dist_norm [symmetric])
          have ℱ n field_differentiable at w
            by (metis holomorphic_on_imp_differentiable_at ‹w ∈ S› holf ‹open
S›)
          moreover
          have g field_differentiable at w
            using ‹w ∈ S› ‹open S› holg holomorphic_on_imp_differentiable_at
by auto
          moreover
          have cmod (deriv (λw. ℱ n w − g w) w) * 2 ≤ e
            using Cauchy_higher_deriv_bound [OF hol cont in_ball, of 1] ‹r >
0› by auto
          ultimately have dist (deriv (ℱ n) w) (deriv g w) ≤ e/2
            by (simp add: dist_norm)
          then show ?thesis
            using ‹e > 0› by auto
        qed
        moreover
        have cball z0 (3 * r / 4) ⊆ ball z0 r
          by (simp add: cball_subset_ball_iff ‹0 < r›)
        with subS have uniform_limit (cball z0 (3 * r/4)) ℱ g sequentially
          by (force intro: ul_g)
       then have ∀F n in sequentially. ∀x∈cball z0 (3 * r / 4). dist (ℱ n x) (g
x) < r / 4 * e / 2
          using ‹0 < e› ‹0 < r› by (force simp: intro!: uniform_limitD)
        ultimately show ?thesis
          by (force simp add: eventually_sequentially)
      qed
    qed
    show uniform_limit (sphere z0 (r/2)) ℱ g sequentially
    proof (rule uniform_limitI)
```

```
      fix e::real
      assume 0 < e
      have sphere z0 (r/2) ⊆ ball z0 r
        using ‹0 < r› by auto
      with subS have uniform_limit (sphere z0 (r/2)) ℱ g sequentially
        by (force intro: ul_g)
      then show ∀⇩F n in sequentially. ∀x ∈ sphere z0 (r/2). dist (ℱ n x) (g x)
< e
        using ‹0 < e› uniform_limit_iff by blast
    qed
    show b > 0 ⋀x. x ∈ sphere z0 (r/2) ⟹ b ≤ cmod (g x)
      using b ‹0 < r› by (fastforce simp: geq hnz)+
  qed
 qed (use ‹0 < r› in auto)
 then have (λn. 0) ⟶ contour_integral (circlepath z0 (r/2)) (λz. deriv g z
/ g z)
   by (simp add: contour_integral_unique [OF ∗])
 then have contour_integral (circlepath z0 (r/2)) (λz. deriv g z / g z) = 0
   by (simp add: LIMSEQ_const_iff)
 moreover
 have contour_integral (circlepath z0 (r/2)) (λz. deriv g z / g z) =
       contour_integral (circlepath z0 (r/2)) (λz. m / (z − z0) + deriv h z / h z)
 proof (rule contour_integral_eq, use ‹0 < r› in simp)
   fix w
   assume w: dist z0 w ∗ 2 = r
   then have w_inb: w ∈ ball z0 r
     using ‹0 < r› by auto
   have h_der: (h has_field_derivative deriv h w) (at w)
     using holh holomorphic_derivI w_inb by blast
   have deriv g w = ((of_nat m ∗ h w + deriv h w ∗ (w − z0)) ∗ (w − z0) ^ m)
/ (w − z0)
     if r = dist z0 w ∗ 2 w ≠ z0
   proof −
     have ((λw. (w − z0) ^ m ∗ h w) has_field_derivative
           (m ∗ h w + deriv h w ∗ (w − z0)) ∗ (w − z0) ^ m / (w − z0)) (at w)
       apply (rule derivative_eq_intros h_der refl)+
       using that ‹m > 0› ‹0 < r› apply (simp add: divide_simps distrib_right)
       by (metis Suc_pred mult.commute power_Suc)
     then show ?thesis
    proof (rule DERIV_imp_deriv [OF has_field_derivative_transform_within_open])
       show ⋀x. x ∈ ball z0 r ⟹ (x − z0) ^ m ∗ h x = g x
         by (simp add: hnz geq)
     qed (use that ‹m > 0› ‹0 < r› in auto)
   qed
   with ‹0 < r› ‹0 < m› w w_inb show deriv g w / g w = of_nat m / (w − z0)
+ deriv h w / h w
     by (auto simp: geq field_split_simps hnz)
 qed
 moreover
```

```
  have contour_integral (circlepath z0 (r/2)) (λz. m / (z − z0) + deriv h z / h
z) =
        2 ∗ of_real pi ∗ i ∗ m + 0
  proof (rule contour_integral_unique [OF has_contour_integral_add])
     show ((λx. m / (x − z0)) has_contour_integral 2 ∗ of_real pi ∗ i ∗ m)
(circlepath z0 (r/2))
       by (force simp: ‹0 < r› intro: Cauchy_integral_circlepath_simple)
    show ((λx. deriv h x / h x) has_contour_integral 0) (circlepath z0 (r/2))
      using hnz holh holomorphic_deriv holomorphic_on_divide ‹0 < r›
      by (fastforce intro!: Cauchy_theorem_disc_simple [of _ z0 r])
  qed
  ultimately show False using ‹0 < m› by auto
qed

corollary Hurwitz_injective:
  assumes S: open S connected S
      and holf: ⋀n::nat. F n holomorphic_on S
      and holg: g holomorphic_on S
      and ul_g: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K F g sequentially
      and nonconst: ¬ g constant_on S
      and inj: ⋀n. inj_on (F n) S
    shows inj_on g S
proof −
  have False if z12: z1 ∈ S z2 ∈ S z1 ≠ z2 g z2 = g z1 for z1 z2
  proof −
    obtain z0 where z0 ∈ S and z0: g z0 ≠ g z2
      using constant_on_def nonconst by blast
    have (λz. g z − g z1) holomorphic_on S
      by (intro holomorphic_intros holg)
    then obtain r where 0 < r ball z2 r ⊆ S ⋀z. dist z2 z < r ∧ z ≠ z2 ⟹ g
z ≠ g z1
      apply (rule isolated_zeros [of λz. g z − g z1 S z2 z0])
      using S ‹z0 ∈ S› z0 z12 by auto
    have g z2 − g z1 ≠ 0
    proof (rule Hurwitz_no_zeros [of S − {z1} λn z. F n z − F n z1 λz. g z −
g z1])
      show open (S − {z1})
        by (simp add: S open_delete)
      show connected (S − {z1})
        by (simp add: connected_open_delete [OF S])
      show ⋀n. (λz. F n z − F n z1) holomorphic_on S − {z1}
        by (intro holomorphic_intros holomorphic_on_subset [OF holf]) blast
      show (λz. g z − g z1) holomorphic_on S − {z1}
        by (intro holomorphic_intros holomorphic_on_subset [OF holg]) blast
      show uniform_limit K (λn z. F n z − F n z1) (λz. g z − g z1) sequentially
          if compact K K ⊆ S − {z1} for K
      proof (rule uniform_limitI)
        fix e::real
        assume e > 0
```

```
      have uniform_limit K F g sequentially
        using that ul_g by fastforce
      then have K: ∀F n in sequentially. ∀x∈K. dist (F n x) (g x) < e/2
        using ‹0 < e› by (force simp: intro!: uniform_limitD)
      have uniform_limit {z1} F g sequentially
        by (simp add: ul_g z12)
      then have ∀F n in sequentially. ∀x∈{z1}. dist (F n x) (g x) < e/2
        using ‹0 < e› by (force simp: intro!: uniform_limitD)
      then have z1: ∀F n in sequentially. dist (F n z1) (g z1) < e/2
        by simp
     show ∀F n in sequentially. ∀x∈K. dist (F n x − F n z1) (g x − g z1) < e
        apply (rule eventually_mono [OF  eventually_conj [OF K z1]])
     by (metis (no_types, opaque_lifting) diff_add_eq diff_diff_eq2 dist_commute
dist_norm dist_triangle_add_half)
    qed
    show ¬ (λz. g z − g z1) constant_on S − {z1}
      unfolding constant_on_def
      by (metis Diff_iff ‹z0 ∈ S› empty_iff insert_iff right_minus_eq z0 z12)
    show ⋀n z. z ∈ S − {z1} ⟹ F n z − F n z1 ≠ 0
      by (metis DiffD1 DiffD2 eq_iff_diff_eq_0 inj inj_onD insertI1 ‹z1 ∈ S›)
    show z2 ∈ S − {z1}
      using ‹z2 ∈ S› ‹z1 ≠ z2› by auto
  qed
  with z12 show False by auto
 qed
 then show ?thesis by (auto simp: inj_on_def)
qed
```

7.5 The Great Picard theorem

```
lemma GPicard1:
 assumes S: open S connected S and w ∈ S 0 < r Y ⊆ X
    and holX: ⋀h. h ∈ X ⟹ h holomorphic_on S
    and X01:  ⋀h z. ⟦h ∈ X; z ∈ S⟧ ⟹ h z ≠ 0 ∧ h z ≠ 1
    and r:    ⋀h. h ∈ Y ⟹ norm(h w) ≤ r
 obtains B Z where 0 < B open Z w ∈ Z Z ⊆ S ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹
norm(h z) ≤ B
proof −
 obtain e where e > 0 and e: cball w e ⊆ S
   using assms open_contains_cball_eq by blast
 show ?thesis
 proof
   show 0 < exp(pi * exp(pi * (2 + 2 * r + 12)))
     by simp
   show ball w (e / 2) ⊆ S
     using e ball_divide_subset_numeral ball_subset_cball by blast
   show cmod (h z) ≤ exp (pi * exp (pi * (2 + 2 * r + 12)))
       if h ∈ Y z ∈ ball w (e / 2) for h z
   proof −
```

```
    have h ∈ X
      using ‹Y ⊆ X› ‹h ∈ Y›  by blast
    have hol_h_o: (h ∘ (λz. (w + of_real e * z))) holomorphic_on cball 0 1
    proof (intro holomorphic_intros holomorphic_on_compose)
      have h holomorphic_on S
        using holX ‹h ∈ X› by auto
      then have h holomorphic_on cball w e
        by (metis e holomorphic_on_subset)
      moreover have (λz. w + complex_of_real e * z) ‘ cball 0 1 ⊆ cball w e
        using that ‹e > 0› by (auto simp: dist_norm norm_mult)
       ultimately show h holomorphic_on (λz. w + complex_of_real e * z) ‘
cball 0 1
        by (rule holomorphic_on_subset)
    qed
    have norm_le_r: cmod ((h ∘ (λz. w + complex_of_real e * z)) 0) ≤ r
      by (auto simp: r ‹h ∈ Y›)
    have le12: norm (of_real(inverse e) * (z − w)) ≤ 1/2
    using that ‹e > 0› by (simp add: inverse_eq_divide dist_norm norm_minus_commute
norm_divide)
     have non01: h (w + e * z) ≠ 0 ∧ h (w + e * z) ≠ 1 if z ∈ cball 0 1 for
z::complex
    proof (rule X01 [OF ‹h ∈ X›])
      have w + complex_of_real e * z ∈ cball w e
        using ‹0 < e› that by (auto simp: dist_norm norm_mult)
      then show w + complex_of_real e * z ∈ S
        by (rule subsetD [OF e])
    qed
    have cmod (h z) ≤ cmod (h (w + of_real e * (inverse e * (z − w))))
      using ‹0 < e› by (simp add: field_split_simps)
    also have ... ≤ exp (pi * exp (pi * (14 + 2 * r)))
     using r [OF ‹h ∈ Y›] Schottky [OF hol_h_o norm_le_r _ _ _ le12] non01
by auto
    finally
    show ?thesis by simp
  qed
 qed (use ‹e > 0› in auto)
qed

lemma GPicard2:
 assumes S ⊆ T connected T S ≠ {} open S ⋀x. ⟦x islimpt S; x ∈ T⟧ ⟹ x ∈ S
   shows S = T
 by (metis assms open_subset connected_clopen closedin_limpt)


lemma GPicard3:
 assumes S: open S connected S w ∈ S and Y ⊆ X
    and holX: ⋀h. h ∈ X ⟹ h holomorphic_on S
    and X01:  ⋀h z. ⟦h ∈ X; z ∈ S⟧ ⟹ h z ≠ 0 ∧ h z ≠ 1
    and no_hw_le1: ⋀h. h ∈ Y ⟹ norm(h w) ≤ 1
```

```
    and compact K K ⊆ S
  obtains B where ⋀h z. ⟦h ∈ Y; z ∈ K⟧ ⟹ norm(h z) ≤ B
proof −
  define U where U ≡ {z ∈ S. ∃B Z. 0 < B ∧ open Z ∧ z ∈ Z ∧ Z ⊆ S ∧
                          (∀h z'. h ∈ Y ∧ z' ∈ Z ⟶ norm(h z') ≤ B)}
  then have U ⊆ S by blast
  have U = S
  proof (rule GPicard2 [OF ‹U ⊆ S› ‹connected S›])
    show U ≠ {}
    proof −
      obtain B Z where 0 < B open Z w ∈ Z Z ⊆ S
        and  ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹ norm(h z) ≤ B
        using GPicard1 [OF S zero_less_one ‹Y ⊆ X› holX] X01 no_hw_le1 by
blast
      then show ?thesis
        unfolding U_def using ‹w ∈ S› by blast
    qed
    show open U
      unfolding open_subopen [of U] by (auto simp: U_def)
    fix v
    assume v: v islimpt U v ∈ S
    have ¬ (∀r>0. ∃h∈Y. r < cmod (h v))
    proof
      assume ∀r>0. ∃h∈Y. r < cmod (h v)
      then have ∀n. ∃h∈Y. Suc n < cmod (h v)
        by simp
      then obtain ℱ where FY: ⋀n. ℱ n ∈ Y and ltF: ⋀n. Suc n < cmod (ℱ
n v)
        by metis
      define 𝒢 where 𝒢 ≡ λn z. inverse(ℱ n z)
      have hol𝒢: 𝒢 n holomorphic_on S for n
      proof (simp add: 𝒢_def)
        show (λz. inverse (ℱ n z)) holomorphic_on S
          using FY X01 ‹Y ⊆ X› holX by (blast intro: holomorphic_on_inverse)
      qed
      have 𝒢not0: 𝒢 n z ≠ 0 and 𝒢not1: 𝒢 n z ≠ 1 if z ∈ S for n z
        using FY X01 ‹Y ⊆ X› that by (force simp: 𝒢_def)+
      have 𝒢_le1: cmod (𝒢 n v) ≤ 1 for n
        using less_le_trans linear ltF
        by (fastforce simp add: 𝒢_def norm_inverse inverse_le_1_iff)
      define W where W ≡ {h. h holomorphic_on S ∧ (∀z ∈ S. h z ≠ 0 ∧ h z
≠ 1)}
      obtain B Z where 0 < B open Z v ∈ Z Z ⊆ S
                  and B: ⋀h z. ⟦h ∈ range 𝒢; z ∈ Z⟧ ⟹ norm(h z) ≤ B
        apply (rule GPicard1 [OF ‹open S› ‹connected S› ‹v ∈ S› zero_less_one,
of range 𝒢 W])
        using hol𝒢 𝒢not0 𝒢not1 𝒢_le1 by (force simp: W_def)+
      then obtain e where e > 0 and e: ball v e ⊆ Z
        by (meson open_contains_ball)
```

```
    obtain h j where holh: h holomorphic_on ball v e and strict_mono j
                and lim:  ⋀x. x ∈ ball v e ⟹ (λn. 𝒢 (j n) x) ⟶ h x
                and ulim: ⋀K. ⟦compact K; K ⊆ ball v e⟧
                          ⟹ uniform_limit K (𝒢 ∘ j) h sequentially
    proof (rule Montel)
      show ⋀h. h ∈ range 𝒢 ⟹ h holomorphic_on ball v e
        by (metis ‹Z ⊆ S› e hol𝒢 holomorphic_on_subset imageE)
      show ⋀K. ⟦compact K; K ⊆ ball v e⟧ ⟹ ∃B. ∀h∈range 𝒢. ∀z∈K. cmod
(h z) ≤ B
        using B e by blast
    qed auto
    have h v = 0
    proof (rule LIMSEQ_unique)
      show (λn. 𝒢 (j n) v) ⟶ h v
        using ‹e > 0› lim by simp
      have lt_Fj: real x ≤ cmod (ℱ (j x) v) for x
       by (metis of_nat_Suc ltF ‹strict_mono j› add.commute less_eq_real_def
less_le_trans nat_le_real_less seq_suble)
      show (λn. 𝒢 (j n) v) ⟶ 0
    proof (rule Lim_null_comparison [OF eventually_sequentiallyI lim_inverse_n])
      show cmod (𝒢 (j x) v) ≤ inverse (real x) if 1 ≤ x for x
        using that by (simp add: 𝒢_def norm_inverse_le_norm [OF lt_Fj])
    qed
    qed
    have h v ≠ 0
    proof (rule Hurwitz_no_zeros [of ball v e 𝒢 ∘ j h])
      show ⋀n. (𝒢 ∘ j) n holomorphic_on ball v e
        using ‹Z ⊆ S› e hol𝒢 by force
      show ⋀n z. z ∈ ball v e ⟹ (𝒢 ∘ j) n z ≠ 0
        using 𝒢not0 ‹Z ⊆ S› e by fastforce
      show ¬ h constant_on ball v e
      proof (clarsimp simp: constant_on_def)
        fix c
        have False if ⋀z. dist v z < e ⟹ h z = c
        proof −
          have h v = c
            by (simp add: ‹0 < e› that)
          obtain y where y ∈ U y ≠ v and y: dist y v < e
            using v ‹e > 0› by (auto simp: islimpt_approachable)
          then obtain C T where y ∈ S C > 0 open T y ∈ T T ⊆ S
            and ⋀h z'. ⟦h ∈ Y; z' ∈ T⟧ ⟹ cmod (h z') ≤ C
            using ‹y ∈ U› by (auto simp: U_def)
          then have le_C: ⋀n. cmod (ℱ n y) ≤ C
            using FY by blast

          have ∀F n in sequentially. dist (𝒢 (j n) y) (h y) < inverse C
            using uniform_limitD [OF ulim [of {y}], of inverse C] ‹C > 0› y
            by (simp add: dist_commute)
          then obtain n where dist (𝒢 (j n) y) (h y) < inverse C
```

```
            by (meson eventually_at_top_linorder order_refl)
          moreover
          have h y = h v
            by (metis ‹h v = c› dist_commute that y)
          ultimately have cmod (inverse (F (j n) y)) < inverse C
            by (simp add: ‹h v = 0› G_def)
          then have C < norm (F (j n) y)
            by (metis G_def Gnot0 ‹y ∈ S› inverse_less_imp_less inverse_zero
norm_inverse zero_less_norm_iff)
          show False
            using ‹C < cmod (F (j n) y)› le_C not_less by blast
        qed
        then show ∃x∈ball v e. h x ≠ c by force
      qed
      show h holomorphic_on ball v e
        by (simp add: holh)
        show ⋀K. ⟦compact K; K ⊆ ball v e⟧ ⟹ uniform_limit K (G ∘ j) h
sequentially
        by (simp add: ulim)
    qed (use ‹e > 0› in auto)
    with ‹h v = 0› show False by blast
  qed
  then obtain r where 0 < r and r: ⋀h. h ∈ Y ⟹ cmod (h v) ≤ r
    by (metis not_le)
  moreover
   obtain B Z where 0 < B open Z v ∈ Z Z ⊆ S ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹
norm(h z) ≤ B
    using X01
    by (auto simp: r intro: GPicard1[OF ‹open S› ‹connected S› ‹v ∈ S› ‹r>0›
‹Y ⊆ X› holX] X01)
  ultimately show v ∈ U
    using v by (simp add: U_def) meson
qed
have ⋀x. x ∈ K ⟶ x ∈ U
  using ‹U = S› ‹K ⊆ S› by blast
then have ⋀x. x ∈ K ⟶ (∃B Z. 0 < B ∧ open Z ∧ x ∈ Z ∧
                        (∀h z'. h ∈ Y ∧ z' ∈ Z ⟶ norm(h z') ≤ B))
  unfolding U_def by blast
then obtain F Z where F: ⋀x. x ∈ K ⟹ open (Z x) ∧ x ∈ Z x ∧
                        (∀h z'. h ∈ Y ∧ z' ∈ Z x ⟶ norm(h z') ≤ F x)
  by metis
then obtain L where L ⊆ K finite L and L: K ⊆ (⋃c ∈ L. Z c)
  by (auto intro: compactE_image [OF ‹compact K›, of K Z])
then have ∗: ⋀x h z'. ⟦x ∈ L; h ∈ Y ∧ z' ∈ Z x⟧ ⟹ cmod (h z') ≤ F x
  using F by blast
have ∃B. ∀h z. h ∈ Y ∧ z ∈ K ⟶ norm(h z) ≤ B
proof (cases L = {})
  case True with L show ?thesis by simp
next
```

```
    case False
    then have ∀h z. h ∈ Y ∧ z ∈ K ⟶ (∃x∈L. cmod (h z) ≤ F x)
      by (metis ∗ L UN_E subset_iff)
    with False ‹finite L› show ?thesis
     by (rule_tac x = Max (F ‘ L) in exI) (simp add: linorder_class.Max_ge_iff)
  qed
  with that show ?thesis by metis
qed


lemma GPicard4:
  assumes 0 < k and holf: f holomorphic_on (ball 0 k − {0})
      and AE: ⋀e. ⟦0 < e; e < k⟧ ⟹ ∃d. 0 < d ∧ d < e ∧ (∀z ∈ sphere 0 d.
norm(f z) ≤ B)
  obtains ε where 0 < ε ε < k ⋀z. z ∈ ball 0 ε − {0} ⟹ norm(f z) ≤ B
proof −
  obtain ε where 0 < ε ε < k/2 and ε: ⋀z. norm z = ε ⟹ norm(f z) ≤ B
    using AE [of k/2] ‹0 < k› by auto
  show ?thesis
  proof
    show ε < k
      using ‹0 < k› ‹ε < k/2› by auto
    show cmod (f ξ) ≤ B if ξ: ξ ∈ ball 0 ε − {0} for ξ
    proof −
      obtain d where 0 < d d < norm ξ and d: ⋀z. norm z = d ⟹ norm(f z)
≤ B
        using AE [of norm ξ] ‹ε < k› ξ by auto
      have [simp]: closure (cball 0 ε − ball 0 d) = cball 0 ε − ball 0 d
        by (blast intro!: closure_closed)
      have [simp]: interior (cball 0 ε − ball 0 d) = ball 0 ε − cball (0::complex) d
        using ‹0 < ε› ‹0 < d› by (simp add: interior_diff)
      have ∗: norm(f w) ≤ B if w ∈ cball 0 ε − ball 0 d for w
      proof (rule maximum_modulus_frontier [of f cball 0 ε − ball 0 d])
        show f holomorphic_on interior (cball 0 ε − ball 0 d)
          using ‹ε < k› ‹0 < d› that by (auto intro: holomorphic_on_subset [OF
holf])
        show continuous_on (closure (cball 0 ε − ball 0 d)) f
       proof (intro holomorphic_on_imp_continuous_on holomorphic_on_subset
[OF holf])
          show closure (cball 0 ε − ball 0 d) ⊆ ball 0 k − {0}
            using ‹0 < d› ‹ε < k› by auto
        qed
        show ⋀z. z ∈ frontier (cball 0 ε − ball 0 d) ⟹ cmod (f z) ≤ B
          unfolding frontier_def
          using ε d less_eq_real_def by force
      qed (use that in auto)
      show ?thesis
        using ∗ ‹d < cmod ξ› that by auto
    qed
```

```
  qed (use ‹0 < ε› in auto)
qed


lemma GPicard5:
  assumes holf: f holomorphic_on (ball 0 1 − {0})
      and f01: ⋀z. z ∈ ball 0 1 − {0} ⟹ f z ≠ 0 ∧ f z ≠ 1
  obtains e B where 0 < e e < 1 0 < B
                    (∀ z ∈ ball 0 e − {0}. norm(f z) ≤ B) ∨
                    (∀ z ∈ ball 0 e − {0}. norm(f z) ≥ B)
proof −
  have [simp]: 1 + of_nat n ≠ (0::complex) for n
    using of_nat_eq_0_iff by fastforce
  have [simp]: cmod (1 + of_nat n) = 1 + of_nat n for n
    by (metis norm_of_nat of_nat_Suc)
  have ∗: (λx::complex. x / of_nat (Suc n)) ‘ (ball 0 1 − {0}) ⊆ ball 0 1 − {0}
for n
    by (auto simp: norm_divide field_split_simps split: if_split_asm)
  define h where h ≡ λn z::complex. f (z / (Suc n))
  have holh: (h n) holomorphic_on ball 0 1 − {0} for n
    unfolding h_def
  proof (rule holomorphic_on_compose_gen [unfolded o_def, OF _ holf ∗])
    show (λx. x / of_nat (Suc n)) holomorphic_on ball 0 1 − {0}
      by (intro holomorphic_intros) auto
  qed
  have h01: ⋀n z. z ∈ ball 0 1 − {0} ⟹ h n z ≠ 0 ∧ h n z ≠ 1
    unfolding h_def
    using ∗ by (force intro!: f01)
  obtain w where w: w ∈ ball 0 1 − {0::complex}
    by (rule_tac w = 1/2 in that) auto
  consider infinite {n. norm(h n w) ≤ 1} | infinite {n. 1 ≤ norm(h n w)}
   by (metis (mono_tags, lifting) infinite_nat_iff_unbounded_le le_cases mem_Collect_eq)
  then show ?thesis
  proof cases
    case 1
    with infinite_enumerate obtain r :: nat ⇒ nat
      where strict_mono r and r: ⋀n. r n ∈ {n. norm(h n w) ≤ 1}
      by blast
    obtain B where B: ⋀j z. ⟦norm z = 1/2; j ∈ range (h ∘ r)⟧ ⟹ norm(j z)
≤ B
    proof (rule GPicard3 [OF _ _ w, where K = sphere 0 (1/2)])
      show range (h ∘ r) ⊆
           {g. g holomorphic_on ball 0 1 − {0} ∧ (∀ z ∈ ball 0 1 − {0}. g z ≠ 0
∧ g z ≠ 1)}
        using h01 by (auto intro: holomorphic_intros holomorphic_on_compose
holh)
      show connected (ball 0 1 − {0::complex})
        by (simp add: connected_open_delete)
    qed (use r in auto)
```

```
    have normf_le_B: cmod(f z) ≤ B if norm z = 1 / (2 * (1 + of_nat (r n)))
for z n
    proof −
      have ∗: ⋀w. norm w = 1/2 ⟹ cmod((f (w / (1 + of_nat (r n))))) ≤ B
        using B by (auto simp: h_def o_def)
      have half: norm (z * (1 + of_nat (r n))) = 1/2
        by (simp add: norm_mult divide_simps that)
      show ?thesis
        using ∗ [OF half] by simp
    qed
    obtain ε where 0 < ε ε < 1 ⋀z. z ∈ ball 0 ε − {0} ⟹ cmod(f z) ≤ B
    proof (rule GPicard4 [OF zero_less_one holf, of B])
      fix e::real
      assume 0 < e e < 1
      obtain n where (1/e − 2) / 2 < real n
        using reals_Archimedean2 by blast
      also have ... ≤ r n
        using ‹strict_mono r› by (simp add: seq_suble)
      finally have (1/e − 2) / 2 < real (r n) .
      with ‹0 < e› have e: e > 1 / (2 + 2 * real (r n))
        by (simp add: field_simps)
      show ∃ d>0. d < e ∧ (∀ z∈sphere 0 d. cmod (f z) ≤ B)
        apply (rule_tac x=1 / (2 * (1 + of_nat (r n))) in exI)
        using normf_le_B by (simp add: e)
    qed blast
    then have ε: cmod (f z) ≤ |B| + 1 if cmod z < ε z ≠ 0 for z
      using that by fastforce
    have 0 < |B| + 1
      by simp
    then show ?thesis
      using ε by (force intro!: that [OF ‹0 < ε› ‹ε < 1›])
  next
    case 2
    with infinite_enumerate obtain r :: nat ⇒ nat
      where strict_mono r and r: ⋀n. r n ∈ {n. norm(h n w) ≥ 1}
      by blast
    obtain B where B: ⋀j z. ⟦norm z = 1/2; j ∈ range (λn. inverse ∘ h (r n))⟧
⟹ norm(j z) ≤ B
    proof (rule GPicard3 [OF _ _ w, where K = sphere 0 (1/2)])
      show range (λn. inverse ∘ h (r n)) ⊆
           {g. g holomorphic_on ball 0 1 − {0} ∧ (∀ z∈ball 0 1 − {0}. g z ≠ 0 ∧
g z ≠ 1)}
       using h01 by (auto intro!: holomorphic_intros holomorphic_on_compose_gen
[unfolded o_def, OF _ holh] holomorphic_on_compose)
      show connected (ball 0 1 − {0::complex})
        by (simp add: connected_open_delete)
      show ⋀j. j ∈ range (λn. inverse ∘ h (r n)) ⟹ cmod (j w) ≤ 1
        using r norm_inverse_le_norm by fastforce
    qed (use r in auto)
```

```
    have norm_if_le_B: cmod(inverse (f z)) ≤ B if norm z = 1 / (2 * (1 +
of_nat (r n))) for z n
    proof −
      have ∗: inverse (cmod((f (z / (1 + of_nat (r n)))))) ≤ B if norm z = 1/2
for z
        using B [OF that] by (force simp: norm_inverse h_def)
      have half: norm (z * (1 + of_nat (r n))) = 1/2
        by (simp add: norm_mult divide_simps that)
      show ?thesis
        using ∗ [OF half] by (simp add: norm_inverse)
    qed
    have hol_if: (inverse ∘ f) holomorphic_on (ball 0 1 − {0})
      by (metis (no_types, lifting) holf comp_apply f01 holomorphic_on_inverse
holomorphic_transform)
   obtain ε where 0 < ε ε < 1 and leB: ⋀z. z ∈ ball 0 ε − {0} ⟹ cmod((inverse
∘ f) z) ≤ B
    proof (rule GPicard4 [OF zero_less_one hol_if, of B])
      fix e::real
      assume 0 < e e < 1
      obtain n where (1/e − 2) / 2 < real n
        using reals_Archimedean2 by blast
      also have ... ≤ r n
        using ‹strict_mono r› by (simp add: seq_suble)
      finally have (1/e − 2) / 2 < real (r n) .
      with ‹0 < e› have e: e > 1 / (2 + 2 * real (r n))
        by (simp add: field_simps)
      show ∃ d>0. d < e ∧ (∀ z∈sphere 0 d. cmod ((inverse ∘ f) z) ≤ B)
        apply (rule_tac x=1 / (2 * (1 + of_nat (r n))) in exI)
        using norm_if_le_B by (simp add: e)
    qed blast
    have ε: cmod (f z) ≥ inverse B and B > 0 if cmod z < ε z ≠ 0 for z
    proof −
      have inverse (cmod (f z)) ≤ B
        using leB that by (simp add: norm_inverse)
      moreover
      have f z ≠ 0
        using ‹ε < 1› f01 that by auto
      ultimately show cmod (f z) ≥ inverse B
        by (simp add: norm_inverse inverse_le_imp_le)
      show B > 0
        using ‹f z ≠ 0› ‹inverse (cmod (f z)) ≤ B› not_le order.trans by fastforce
    qed
    then have B > 0
      by (metis ‹0 < ε› dense leI order.asym vector_choose_size)
    then have inverse B > 0
      by (simp add: field_split_simps)
    then show ?thesis
      using ε that [OF ‹0 < ε› ‹ε < 1›]
      by (metis Diff_iff dist_0_norm insert_iff mem_ball)
```

```
  qed
qed


lemma GPicard6:
  assumes open M z ∈ M a ≠ 0 and holf: f holomorphic_on (M − {z})
      and f0a: ⋀w. w ∈ M − {z} ⟹ f w ≠ 0 ∧ f w ≠ a
  obtains r where 0 < r ball z r ⊆ M
                  bounded(f ‘ (ball z r − {z})) ∨
                   bounded((inverse ∘ f) ‘ (ball z r − {z}))
proof −
  obtain r where 0 < r and r: ball z r ⊆ M
    using assms openE by blast
  let ?g = λw. f (z + of_real r ∗ w) / a
  obtain e B where 0 < e e < 1 0 < B
     and B: (∀z ∈ ball 0 e − {0}. norm(?g z) ≤ B) ∨ (∀z ∈ ball 0 e − {0}.
norm(?g z) ≥ B)
  proof (rule GPicard5)
    show ?g holomorphic_on ball 0 1 − {0}
   proof (intro holomorphic_intros holomorphic_on_compose_gen [unfolded o_def,
OF _ holf])
      show (λx. z + complex_of_real r ∗ x) ‘ (ball 0 1 − {0}) ⊆ M − {z}
        using ‹0 < r› r
        by (auto simp: dist_norm norm_mult subset_eq)
    qed (use ‹a ≠ 0› in auto)
    show ⋀w. w ∈ ball 0 1 − {0} ⟹ f (z + of_real r ∗ w) / a ≠ 0 ∧ f (z +
of_real r ∗ w) / a ≠ 1
      using f0a ‹0 < r› ‹a ≠ 0› r
      by (auto simp: field_split_simps dist_norm norm_mult subset_eq)
  qed
  show ?thesis
  proof
    show 0 < e∗r
      by (simp add: ‹0 < e› ‹0 < r›)
    have ball z (e ∗ r) ⊆ ball z r
      by (simp add: ‹0 < r› ‹e < 1› order.strict_implies_order subset_ball)
    then show ball z (e ∗ r) ⊆ M
      using r by blast
   consider ⋀z. z ∈ ball 0 e − {0} ⟹ norm(?g z) ≤ B | ⋀z. z ∈ ball 0 e − {0}
⟹ norm(?g z) ≥ B
      using B by blast
    then show bounded (f ‘ (ball z (e ∗ r) − {z})) ∨
          bounded ((inverse ∘ f) ‘ (ball z (e ∗ r) − {z}))
    proof cases
      case 1
      have ⟦dist z w < e ∗ r; w ≠ z⟧ ⟹ cmod (f w) ≤ B ∗ norm a for w
        using ‹a ≠ 0› ‹0 < r› 1 [of (w − z) / r]
        by (simp add: norm_divide dist_norm field_split_simps)
      then show ?thesis
```

```
      by (force simp: intro!: boundedI)
    next
      case 2
      have ⟦dist z w < e * r; w ≠ z⟧ ⟹ cmod (f w) ≥ B * norm a for w
        using ‹a ≠ 0› ‹0 < r› 2 [of (w - z) / r]
        by (simp add: norm_divide dist_norm field_split_simps)
      then have ⟦dist z w < e * r; w ≠ z⟧ ⟹ inverse (cmod (f w)) ≤ inverse (B
* norm a) for w
      by (metis ‹0 < B› ‹a ≠ 0› mult_pos_pos norm_inverse norm_inverse_le_norm
zero_less_norm_iff)
      then show ?thesis
        by (force simp: norm_inverse intro!: boundedI)
    qed
  qed
qed


theorem great_Picard:
  assumes open M z ∈ M a ≠ b and holf: f holomorphic_on (M - {z})
      and fab: ⋀w. w ∈ M - {z} ⟹ f w ≠ a ∧ f w ≠ b
  obtains l where (f ⟶ l) (at z) ∨ ((inverse ∘ f) ⟶ l) (at z)
proof -
  obtain r where 0 < r and zrM: ball z r ⊆ M
            and r: bounded((λz. f z - a) ` (ball z r - {z})) ∨
                   bounded((inverse ∘ (λz. f z - a)) ` (ball z r - {z}))
  proof (rule GPicard6 [OF ‹open M› ‹z ∈ M›])
    show b - a ≠ 0
      using assms by auto
    show (λz. f z - a) holomorphic_on M - {z}
      by (intro holomorphic_intros holf)
  qed (use fab in auto)
  have holfb: f holomorphic_on ball z r - {z}
    using zrM by (auto intro: holomorphic_on_subset [OF holf])
  have holfb_i: (λz. inverse(f z - a)) holomorphic_on ball z r - {z}
    using fab zrM by (fastforce intro!: holomorphic_intros holfb)
  show ?thesis
    using r
  proof
    assume bounded ((λz. f z - a) ` (ball z r - {z}))
    then obtain B where B: ⋀w. w ∈ (λz. f z - a) ` (ball z r - {z}) ⟹ norm
w ≤ B
      by (force simp: bounded_iff)
    then have ∀x. x ≠ z ∧ dist x z < r ⟶ cmod (f x - a) ≤ B
      by (simp add: dist_commute)
    with ‹0 < r› have ∀F w in at z. cmod (f w - a) ≤ B
      by (force simp add: eventually_at)
    moreover have ⋀x. cmod (f x - a) ≤ B ⟹ cmod (f x) ≤ B + cmod a
     by (metis add.commute add_le_cancel_right norm_triangle_sub order.trans)
    ultimately have ∃B. ∀F w in at z. cmod (f w) ≤ B
```

```
    by (metis (mono_tags, lifting) eventually_at)
  then obtain g where holg: g holomorphic_on ball z r and gf: ⋀w. w ∈ ball
z r − {z} ⟹ g w = f w
    using ‹0 < r› holomorphic_on_extend_bounded [OF holfb] by auto
  then have g −z→ g z
    unfolding continuous_at [symmetric]
    using ‹0 < r› centre_in_ball field_differentiable_imp_continuous_at
        holomorphic_on_imp_differentiable_at by blast
  then have (f ⟶ g z) (at z)
    using Lim_transform_within_open [of g g z z]
    using ‹0 < r› centre_in_ball gf by blast
  then show ?thesis
    using that by blast
 next
  assume bounded((inverse ∘ (λz. f z − a)) ‘ (ball z r − {z}))
  then obtain B where B: ⋀w. w ∈ (inverse ∘ (λz. f z − a)) ‘ (ball z r − {z})
⟹ norm w ≤ B
    by (force simp: bounded_iff)
  then have ∀ x. x ≠ z ∧ dist x z < r ⟶ cmod (inverse (f x − a)) ≤ B
    by (simp add: dist_commute)
  with ‹0 < r› have ∀ F w in at z. cmod (inverse (f w − a)) ≤ B
    by (auto simp add: eventually_at)
  then have ∃ B. ∀ F z in at z. cmod (inverse (f z − a)) ≤ B
    by blast
  then obtain g where holg: g holomorphic_on ball z r and gf: ⋀w. w ∈ ball
z r − {z} ⟹ g w = inverse (f w − a)
    using ‹0 < r› holomorphic_on_extend_bounded [OF holfb_i] by auto
  then have gz: g −z→ g z
    unfolding continuous_at [symmetric]
    using ‹0 < r› centre_in_ball field_differentiable_imp_continuous_at
        holomorphic_on_imp_differentiable_at by blast
  have gnz: ⋀w. w ∈ ball z r − {z} ⟹ g w ≠ 0
    using gf fab zrM by fastforce
  show ?thesis
  proof (cases g z = 0)
    case True
     have ∗: ⟦g ≠ 0; inverse g = f − a⟧ ⟹ g / (1 + a ∗ g) = inverse f for f
g::complex
      by (auto simp: field_simps)
    have (inverse ∘ f) −z→ 0
    proof (rule Lim_transform_within_open [of λw. g w / (1 + a ∗ g w) _ _
UNIV ball z r])
      show (λw. g w / (1 + a ∗ g w)) −z→ 0
        using True by (auto simp: intro!: tendsto_eq_intros gz)
      show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ g x / (1 + a ∗ g x) = (inverse ∘ f) x
        using ∗ gf gnz by simp
    qed (use ‹0 < r› in auto)
    with that show ?thesis by blast
  next
```

```
      case False
      show ?thesis
      proof (cases 1 + a * g z = 0)
        case True
        have (f ⟶ 0) (at z)
        proof (rule Lim_transform_within_open [of λw. (1 + a * g w) / g w _ _
_ ball z r])
          show (λw. (1 + a * g w) / g w) −z→ 0
            by (rule tendsto_eq_intros refl gz ‹g z ≠ 0› | simp add: True)+
          show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ (1 + a * g x) / g x = f x
            using fab fab zrM by (fastforce simp add: gf field_split_simps)
        qed (use ‹0 < r› in auto)
        then show ?thesis
          using that by blast
      next
        case False
        have *: ⟦g ≠ 0; inverse g = f − a⟧ ⟹ g / (1 + a * g) = inverse f for f
g::complex
          by (auto simp: field_simps)
        have (inverse ∘ f) −z→ g z / (1 + a * g z)
        proof (rule Lim_transform_within_open [of λw. g w / (1 + a * g w) _ _
UNIV ball z r])
          show (λw. g w / (1 + a * g w)) −z→ g z / (1 + a * g z)
            using False by (auto simp: False intro!: tendsto_eq_intros gz)
          show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ g x / (1 + a * g x) = (inverse ∘ f) x
            using * gf gnz by simp
        qed (use ‹0 < r› in auto)
        with that show ?thesis by blast
      qed
    qed
  qed
qed


corollary great_Picard_alt:
  assumes M: open M z ∈ M and holf: f holomorphic_on (M − {z})
    and non: ⋀l. ¬ (f ⟶ l) (at z) ⋀l. ¬ ((inverse ∘ f) ⟶ l) (at z)
  obtains a where − {a} ⊆ f ‘ (M − {z})
unfolding subset_iff image_iff
  by (metis great_Picard [OF M _ holf] non Compl_iff insertI1)


corollary great_Picard_infinite:
  assumes M: open M z ∈ M and holf: f holomorphic_on (M − {z})
    and non: ⋀l. ¬ (f ⟶ l) (at z) ⋀l. ¬ ((inverse ∘ f) ⟶ l) (at z)
  obtains a where ⋀w. w ≠ a ⟹ infinite {x. x ∈ M − {z} ∧ f x = w}
proof −
  have False if a ≠ b and ab: finite {x. x ∈ M − {z} ∧ f x = a} finite {x. x ∈ M
− {z} ∧ f x = b} for a b
```

```
  proof −
    have finab: finite {x. x ∈ M − {z} ∧ f x ∈ {a,b}}
      using finite_UnI [OF ab] unfolding mem_Collect_eq insert_iff empty_iff
      by (simp add: conj_disj_distribL)
    obtain r where 0 < r and zrM: ball z r ⊆ M and r: ⋀x. ⟦x ∈ M − {z}; f x
∈ {a,b}⟧ ⟹ x ∉ ball z r
    proof −
      obtain e where e > 0 and e: ball z e ⊆ M
        using assms openE by blast
      show ?thesis
      proof (cases {x ∈ M − {z}. f x ∈ {a, b}} = {})
        case True
        then show ?thesis
          using e ‹e > 0› that by fastforce
      next
        case False
        let ?r = min e (Min (dist z ‘ {x ∈ M − {z}. f x ∈ {a,b}}))
        show ?thesis
        proof
          show 0 < ?r
            using min_less_iff_conj Min_gr_iff finab False ‹0 < e› by auto
          have ball z ?r ⊆ ball z e
            by (simp add: subset_ball)
          with e show ball z ?r ⊆ M by blast
          show ⋀x. ⟦x ∈ M − {z}; f x ∈ {a, b}⟧ ⟹ x ∉ ball z ?r
            using min_less_iff_conj Min_gr_iff finab False ‹0 < e› by auto
        qed
      qed
    qed
    have holfb: f holomorphic_on (ball z r − {z})
      apply (rule holomorphic_on_subset [OF holf])
       using zrM by auto
     show ?thesis
      apply (rule great_Picard [OF open_ball _ ‹a ≠ b› holfb])
      using non ‹0 < r› r zrM by auto
  qed
  with that show thesis
    by meson
qed

theorem Casorati_Weierstrass:
  assumes open M z ∈ M f holomorphic_on (M − {z})
      and ⋀l. ¬ (f ⟶ l) (at z) ⋀l. ¬ ((inverse ∘ f) ⟶ l) (at z)
  shows closure(f ‘ (M − {z})) = UNIV
proof −
  obtain a where a: − {a} ⊆ f ‘ (M − {z})
    using great_Picard_alt [OF assms] .
  have UNIV = closure(− {a})
    by (simp add: closure_interior)
```

also have ... ⊆ *closure*(*f* ‘ (*M* − {*z*}))
by (*simp add*: *a closure_mono*)
finally show *?thesis*
by *blast*
qed

end

8 Moebius functions, Equivalents of Simply Connected Sets, Riemann Mapping Theorem

theory *Riemann_Mapping*
imports *Great_Picard*
begin

8.1 Moebius functions are biholomorphisms of the unit disc

definition *Moebius_function* :: [*real*,*complex*,*complex*] ⇒ *complex* **where**
Moebius_function ≡ λ*t w z*. *exp*(i ∗ *of_real t*) ∗ (*z* − *w*) / (*1* − *cnj w* ∗ *z*)

lemma *Moebius_function_simple*:
Moebius_function 0 w z = (*z* − *w*) / (*1* − *cnj w* ∗ *z*)
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_eq_zero*:
Moebius_function t w w = *0*
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_of_zero*:
Moebius_function t w 0 = − *exp*(i ∗ *of_real t*) ∗ *w*
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_norm_lt_1*:
assumes *w1*: *norm w* < *1* **and** *z1*: *norm z* < *1*
shows *norm* (*Moebius_function t w z*) < *1*
proof −
have *1* − *cnj w* ∗ *z* ≠ *0*
by (*metis complex_cnj_cnj complex_mod_sqrt_Re_mult_cnj mult.commute mult_less_cancel_right1 norm_ge_zero norm_mult norm_one order.asym right_minus_eq w1 z1*)
then have *VV*: *1* − *w* ∗ *cnj z* ≠ *0*
by (*metis complex_cnj_cnj complex_cnj_mult complex_cnj_one right_minus_eq*)
then have *1* − *norm* (*Moebius_function t w z*) ^ *2* =
((*1* − *norm w* ^ *2*) / (*norm* (*1* − *cnj w* ∗ *z*) ^ *2*)) ∗ (*1* − *norm z* ^ *2*)
apply (*cases w*)
apply (*cases z*)
apply (*simp add*: *Moebius_function_def divide_simps norm_divide norm_mult*)
apply (*simp add*: *complex_norm complex_diff complex_mult one_complex.code*

complex_cnj)
apply (*auto simp*: *algebra_simps power2_eq_square*)
done
then have $1 - (cmod\ (Moebius\_function\ t\ w\ z))^2 = (1 - cmod\ (w * w)) / (cmod\ (1 - cnj\ w * z))^2 * (1 - cmod\ (z * z))$
by (*simp add*: *norm_mult power2_eq_square*)
moreover have $0 < 1 - cmod\ (z * z)$
by (*metis* (*no_types*) *z1 diff_gt_0_iff_gt mult.left_neutral norm_mult_less*)
ultimately have $0 < 1 - norm\ (Moebius\_function\ t\ w\ z)\ \hat{}\ 2$
using ‹$1 - cnj\ w * z \neq 0$› *w1 norm_mult_less* **by** *fastforce*
then show *?thesis*
using *linorder_not_less* **by** *fastforce*
qed

lemma *Moebius_function_holomorphic*:
assumes $norm\ w < 1$
shows *Moebius_function t w holomorphic_on ball 0 1*
proof −
have $*$: $1 - z * w \neq 0$ **if** $norm\ z < 1$ **for** z
proof −
have $norm\ (1::complex) \neq norm\ (z * w)$
using *assms that norm_mult_less* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
show *?thesis*
apply (*simp add*: *Moebius_function_def*)
apply (*intro holomorphic_intros*)
using *assms* $*$
by (*metis complex_cnj_cnj complex_cnj_mult complex_cnj_one complex_mod_cnj mem_ball_0 mult.commute right_minus_eq*)
qed

lemma *Moebius_function_compose*:
assumes *meq*: $-w1 = w2$ **and** $norm\ w1 < 1\ norm\ z < 1$
shows $Moebius\_function\ 0\ w1\ (Moebius\_function\ 0\ w2\ z) = z$
proof −
have $norm\ w2 < 1$
using *assms* **by** *auto*
then have $-w1 = z$ **if** $cnj\ w2 * z = 1$
by (*metis assms*(*3*) *complex_mod_cnj less_irrefl mult.right_neutral norm_mult_less norm_one that*)
moreover have *z=0* **if** $1 - cnj\ w2 * z = cnj\ w1 * (z - w2)$
proof −
have $w2 * cnj\ w2 = 1$
using *that meq* **by** (*auto simp*: *algebra_simps*)
then show $z = 0$
by (*metis* (*no_types*) ‹$cmod\ w2 < 1$› *complex_mod_cnj less_irrefl mult.right_neutral norm_mult_less norm_one*)
qed

moreover have $z - w2 - w1 * (1 - cnj\ w2 * z) = z * (1 - cnj\ w2 * z - cnj\ w1 * (z - w2))$
using *meq* **by** (*fastforce simp*: *algebra_simps*)
ultimately
show *?thesis*
by (*simp add*: *Moebius_function_def divide_simps norm_divide norm_mult*)
qed

lemma *ball_biholomorphism_exists*:
assumes $a \in ball\ 0\ 1$
obtains $f\ g$ **where** $f\ a = 0$
f *holomorphic_on ball 0 1* f ‘ *ball 0 1* $\subseteq$ *ball 0 1*
g *holomorphic_on ball 0 1* g ‘ *ball 0 1* $\subseteq$ *ball 0 1*
$\bigwedge z.\ z \in ball\ 0\ 1 \Longrightarrow f\ (g\ z) = z$
$\bigwedge z.\ z \in ball\ 0\ 1 \Longrightarrow g\ (f\ z) = z$
proof
show *Moebius_function 0 a holomorphic_on ball 0 1 Moebius_function 0* $(-a)$ *holomorphic_on ball 0 1*
using *Moebius_function_holomorphic assms mem_ball_0* **by** *auto*
show *Moebius_function 0 a a* $= 0$
by (*simp add*: *Moebius_function_eq_zero*)
show *Moebius_function 0 a* ‘ *ball 0 1* $\subseteq$ *ball 0 1*
Moebius_function 0 $(-\ a)$ ‘ *ball 0 1* $\subseteq$ *ball 0 1*
using *Moebius_function_norm_lt_1 assms* **by** *auto*
show *Moebius_function 0 a* (*Moebius_function 0* $(-\ a)\ z) = z$
Moebius_function 0 $(-\ a)$ (*Moebius_function 0 a z*) $= z$ **if** $z \in ball\ 0\ 1$ **for** z
using *Moebius_function_compose assms that* **by** *auto*
qed

8.2 A big chain of equivalents of simple connectedness for an open set

lemma *biholomorphic_to_disc_aux*:
assumes *open S connected S* $0 \in S$ **and** *S01*: $S \subseteq ball\ 0\ 1$
and *prev*: $\bigwedge f.\ [\![f\ holomorphic\_on\ S;\ \bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0;\ inj\_on\ f\ S]\!]$
$\Longrightarrow \exists g.\ g\ holomorphic\_on\ S \wedge (\forall z \in S.\ f\ z = (g\ z)^2)$
shows $\exists f\ g.\ f\ holomorphic\_on\ S \wedge g\ holomorphic\_on\ ball\ 0\ 1 \wedge$
$(\forall z \in S.\ f\ z \in ball\ 0\ 1 \wedge g(f\ z) = z) \wedge$
$(\forall z \in ball\ 0\ 1.\ g\ z \in S \wedge f(g\ z) = z)$
proof −
define F **where** $F \equiv \{h.\ h\ holomorphic\_on\ S \wedge h$ ‘ $S \subseteq ball\ 0\ 1 \wedge h\ 0 = 0 \wedge inj\_on\ h\ S\}$
have *idF*: $id \in F$
using *S01* **by** (*auto simp*: *F_def*)
then have $F \neq \{\}$
by *blast*
have *imF_ne*: $((\lambda h.\ norm(deriv\ h\ 0))$ ‘ $F) \neq \{\}$
using *idF* **by** *auto*

```
  have holF: ⋀h. h ∈ F ⟹ h holomorphic_on S
    by (auto simp: F_def)
  obtain f where f ∈ F and normf: ⋀h. h ∈ F ⟹ norm(deriv h 0) ≤ norm(deriv
f 0)
  proof −
    obtain r where r > 0 and r: ball 0 r ⊆ S
      using ‹open S› ‹0 ∈ S› openE by auto
    have bdd: bdd_above ((λh. norm(deriv h 0)) ‘ F)
    proof (intro bdd_aboveI exI ballI, clarify)
      show norm (deriv f 0) ≤ 1 / r if f ∈ F for f
      proof −
        have r01: (∗) (complex_of_real r) ‘ ball 0 1 ⊆ S
          using that ‹r > 0› by (auto simp: norm_mult r [THEN subsetD])
        then have f holomorphic_on (∗) (complex_of_real r) ‘ ball 0 1
          using holomorphic_on_subset [OF holF] by (simp add: that)
        then have holf: f ∘ (λz. (r ∗ z)) holomorphic_on (ball 0 1)
          by (intro holomorphic_intros holomorphic_on_compose)
        have f0: (f ∘ (∗) (complex_of_real r)) 0 = 0
          using F_def that by auto
        have f ‘ S ⊆ ball 0 1
          using F_def that by blast
        with r01 have fr1: ⋀z. norm z < 1 ⟹ norm ((f ∘ (∗)(of_real r))z) < 1
          by force
        have ∗: ((λw. f (r ∗ w)) has_field_derivative deriv f (r ∗ z) ∗ r) (at z)
          if z ∈ ball 0 1 for z::complex
        proof (rule DERIV_chain' [where g=f])
          show (f has_field_derivative deriv f (of_real r ∗ z)) (at (of_real r ∗ z))
            apply (rule holomorphic_derivI [OF holF ‹open S›])
             apply (rule ‹f ∈ F›)
            by (meson imageI r01 subset_iff that)
        qed simp
        have df0: ((λw. f (r ∗ w)) has_field_derivative deriv f 0 ∗ r) (at 0)
          using ∗ [of 0] by simp
          have deq: deriv (λx. f (complex_of_real r ∗ x)) 0 = deriv f 0 ∗ com-
plex_of_real r
          using DERIV_imp_deriv df0 by blast
        have norm (deriv (f ∘ (∗) (complex_of_real r)) 0) ≤ 1
          by (auto intro: Schwarz_Lemma [OF holf f0 fr1, of 0])
        with ‹r > 0› show ?thesis
          by (simp add: deq norm_mult divide_simps o_def)
      qed
    qed
    define l where l ≡ SUP h∈F. norm (deriv h 0)
    have eql: norm (deriv f 0) = l if le: l ≤ norm (deriv f 0) and f ∈ F for f
      apply (rule order_antisym [OF _ le])
      using ‹f ∈ F› bdd cSUP_upper by (fastforce simp: l_def)
    obtain F where Fin: ⋀n. F n ∈ F and Flim: (λn. norm (deriv (F n) 0))
⟶ l
    proof −
```

```
    have ∃f. f ∈ F ∧ |norm (deriv f 0) − l| < 1 / (Suc n) for n
    proof −
      obtain f where f ∈ F and f: l < norm (deriv f 0) + 1/(Suc n)
        using cSup_least [OF imF_ne, of l − 1/(Suc n)] by (fastforce simp add:
l_def)
      then have |norm (deriv f 0) − l| < 1 / (Suc n)
        by (fastforce simp add: abs_if not_less eql)
      with ‹f ∈ F› show ?thesis
        by blast
    qed
    then obtain ℱ where fF: ⋀n. (ℱ n) ∈ F
      and fless: ⋀n. |norm (deriv (ℱ n) 0) − l| < 1 / (Suc n)
      by metis
    have (λn. norm (deriv (ℱ n) 0)) ⟶ l
    proof (rule metric_LIMSEQ_I)
      fix e::real
      assume e > 0
      then obtain N::nat where N: e > 1/(Suc N)
        using nat_approx_posE by blast
      show ∃N. ∀n≥N. dist (norm (deriv (ℱ n) 0)) l < e
      proof (intro exI allI impI)
        fix n assume N ≤ n
        have dist (norm (deriv (ℱ n) 0)) l < 1 / (Suc n)
          using fless by (simp add: dist_norm)
        also have ... < e
          using N ‹N ≤ n› inverse_of_nat_le le_less_trans by blast
        finally show dist (norm (deriv (ℱ n) 0)) l < e .
      qed
    qed
    with fF show ?thesis
      using that by blast
  qed
  have ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ ∃B. ∀h∈F. ∀z∈K. norm (h z) ≤ B
    by (rule_tac x=1 in exI) (force simp: F_def)
  moreover have range ℱ ⊆ F
    using ‹⋀n. ℱ n ∈ F› by blast
  ultimately obtain f and r :: nat ⇒ nat
    where holf: f holomorphic_on S and r: strict_mono r
      and limf: ⋀x. x ∈ S ⟹ (λn. ℱ (r n) x) ⟶ f x
        and ulimf: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K (ℱ ∘ r) f
sequentially
    using Montel [of S F ℱ, OF ‹open S› holF] by auto+
  have der: ⋀n x. x ∈ S ⟹ ((ℱ ∘ r) n has_field_derivative ((λn. deriv (ℱ n))
∘ r) n x) (at x)
    using ‹⋀n. ℱ n ∈ F› ‹open S› holF holomorphic_derivI by fastforce
  have ulim: ⋀x. x ∈ S ⟹ ∃d>0. cball x d ⊆ S ∧ uniform_limit (cball x d)
(ℱ ∘ r) f sequentially
    by (meson ulimf ‹open S› compact_cball open_contains_cball)
  obtain f' :: complex⇒complex where f': (f has_field_derivative f' 0) (at 0)
```

```
    and tof'0: (λn. ((λn. deriv (ℱ n)) ∘ r) n 0) ⟶ f' 0
    using has_complex_derivative_uniform_sequence [OF ‹open S› der ulim] ‹0
∈ S› by metis
  then have derf0: deriv f 0 = f' 0
    by (simp add: DERIV_imp_deriv)
  have f field_differentiable (at 0)
    using field_differentiable_def f' by blast
  have (λx. (norm (deriv (ℱ (r x)) 0))) ⟶ norm (deriv f 0)
   using isCont_tendsto_compose [OF continuous_norm [OF continuous_ident]
tof'0] derf0 by auto
  with LIMSEQ_subseq_LIMSEQ [OF ℱlim r] have no_df0: norm(deriv f 0)
= l
    by (force simp: o_def intro: tendsto_unique)
  have nonconstf: ¬ f constant_on S
  proof −
    have False if ⋀x. x ∈ S ⟹ f x = c for c
    proof −
      have deriv f 0 = 0
     by (metis that ‹open S› ‹0 ∈ S› DERIV_imp_deriv [OF has_field_derivative_transform_within_open
[OF DERIV_const]])
      with no_df0 have l = 0
        by auto
      with eql [OF _ idF] show False by auto
    qed
    then show ?thesis
      by (meson constant_on_def)
  qed
  show ?thesis
  proof
    show f ∈ F
      unfolding F_def
    proof (intro CollectI conjI holf)
      have norm(f z) ≤ 1 if z ∈ S for z
      proof (intro Lim_norm_ubound [OF _ limf] always_eventually allI that)
        fix n
        have ℱ (r n) ∈ F
          by (simp add: ℱin)
        then show norm (ℱ (r n) z) ≤ 1
          using that by (auto simp: F_def)
      qed simp
      then have fless1: norm(f z) < 1 if z ∈ S for z
       using maximum_modulus_principle [OF holf ‹open S› ‹connected S› ‹open
S›] nonconstf that
        by fastforce
      then show f ‘ S ⊆ ball 0 1
        by auto
      have (λn. ℱ (r n) 0) ⟶ 0
        using ℱin by (auto simp: F_def)
      then show f 0 = 0
```

```
        using tendsto_unique [OF _ limf ] ‹0 ∈ S› trivial_limit_sequentially by
blast
      show inj_on f S
      proof (rule Hurwitz_injective [OF ‹open S› ‹connected S› _ holf])
        show ⋀n. (ℱ ∘ r) n holomorphic_on S
          by (simp add: ℱin holF)
        show ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K (ℱ ∘ r) f sequentially
          by (metis ulimf)
        show ¬ f constant_on S
          using nonconstf by auto
        show ⋀n. inj_on ((ℱ ∘ r) n) S
          using ℱin by (auto simp: F_def)
      qed
    qed
    show ⋀h. h ∈ F ⟹ norm (deriv h 0) ≤ norm (deriv f 0)
      by (metis eql le_cases no_df0)
  qed
 qed
 have holf: f holomorphic_on S and injf: inj_on f S and f01: f ‘ S ⊆ ball 0 1
  using ‹f ∈ F› by (auto simp: F_def)
 obtain g where holg: g holomorphic_on (f ‘ S)
          and derg: ⋀z. z ∈ S ⟹ deriv f z * deriv g (f z) = 1
          and gf: ⋀z. z ∈ S ⟹ g(f z) = z
  using holomorphic_has_inverse [OF holf ‹open S› injf] by metis
 have ball 0 1 ⊆ f ‘ S
 proof
  fix a::complex
  assume a: a ∈ ball 0 1
  have False if ⋀x. x ∈ S ⟹ f x ≠ a
  proof −
    obtain h k where h a = 0
      and holh: h holomorphic_on ball 0 1 and h01: h ‘ ball 0 1 ⊆ ball 0 1
      and holk: k holomorphic_on ball 0 1 and k01: k ‘ ball 0 1 ⊆ ball 0 1
      and hk: ⋀z. z ∈ ball 0 1 ⟹ h (k z) = z
      and kh: ⋀z. z ∈ ball 0 1 ⟹ k (h z) = z
      using ball_biholomorphism_exists [OF a] by blast
    have nf1: ⋀z. z ∈ S ⟹ norm(f z) < 1
      using ‹f ∈ F› by (auto simp: F_def)
    have 1: h ∘ f holomorphic_on S
    using F_def ‹f ∈ F› holh holomorphic_on_compose holomorphic_on_subset
by blast
    have 2: ⋀z. z ∈ S ⟹ (h ∘ f) z ≠ 0
      by (metis ‹h a = 0› a comp_eq_dest_lhs nf1 kh mem_ball_0 that)
    have 3: inj_on (h ∘ f) S
     by (metis (no_types, lifting) F_def ‹f ∈ F› comp_inj_on inj_on_inverseI
injf kh mem_Collect_eq subset_inj_on)
    obtain ψ where holψ: ψ holomorphic_on ((h ∘ f) ‘ S)
      and ψ2: ⋀z. z ∈ S ⟹ ψ(h (f z)) ^ 2 = h(f z)
    proof (rule exE [OF prev [OF 1 2 3]], safe)
```

```
    fix ϑ
    assume holϑ: ϑ holomorphic_on S and ϑ2: (∀z∈S. (h ∘ f) z = (ϑ z)²)
    show thesis
    proof
      show (ϑ ∘ g ∘ k) holomorphic_on (h ∘ f) ‘ S
      proof (intro holomorphic_on_compose)
        show k holomorphic_on (h ∘ f) ‘ S
          apply (rule holomorphic_on_subset [OF holk])
          using f01 h01 by force
        show g holomorphic_on k ‘ (h ∘ f) ‘ S
          apply (rule holomorphic_on_subset [OF holg])
          by (auto simp: kh nf1)
        show ϑ holomorphic_on g ‘ k ‘ (h ∘ f) ‘ S
          apply (rule holomorphic_on_subset [OF holϑ])
          by (auto simp: gf kh nf1)
      qed
      show ((ϑ ∘ g ∘ k) (h (f z)))² = h (f z) if z ∈ S for z
      proof −
        have f z ∈ ball 0 1
          by (simp add: nf1 that)
        then have (ϑ (g (k (h (f z)))))² = (ϑ (g (f z)))²
          by (metis kh)
        also have ... = h (f z)
          using ϑ2 gf that by auto
        finally show ?thesis
          by (simp add: o_def)
      qed
    qed
    have normψ1: norm(ψ (h (f z))) < 1 if z ∈ S for z
    proof −
      have norm (ψ (h (f z)) ^ 2) < 1
          by (metis (no_types) that DIM_complex ψ2 h01 image_subset_iff
mem_ball_0 nf1)
      then show ?thesis
     by (metis le_less_trans mult_less_cancel_left2 norm_ge_zero norm_power
not_le power2_eq_square)
    qed
    then have ψ01: ψ (h (f 0)) ∈ ball 0 1
      by (simp add: ‹0 ∈ S›)
    obtain p q where p0: p (ψ (h (f 0))) = 0
      and holp: p holomorphic_on ball 0 1 and p01: p ‘ ball 0 1 ⊆ ball 0 1
      and holq: q holomorphic_on ball 0 1 and q01: q ‘ ball 0 1 ⊆ ball 0 1
      and pq: ⋀z. z ∈ ball 0 1 ⟹ p (q z) = z
      and qp: ⋀z. z ∈ ball 0 1 ⟹ q (p z) = z
      using ball_biholomorphism_exists [OF ψ01] by metis
    have p ∘ ψ ∘ h ∘ f ∈ F
      unfolding F_def
    proof (intro CollectI conjI holf)
```

```
      show p ∘ ψ ∘ h ∘ f holomorphic_on S
      proof (intro holomorphic_on_compose holf)
        show h holomorphic_on f ‘ S
          apply (rule holomorphic_on_subset [OF holh])
          using f01 by force
        show ψ holomorphic_on h ‘ f ‘ S
          apply (rule holomorphic_on_subset [OF holψ])
          by auto
        show p holomorphic_on ψ ‘ h ‘ f ‘ S
          apply (rule holomorphic_on_subset [OF holp])
          by (auto simp: normψ1)
      qed
      show (p ∘ ψ ∘ h ∘ f) ‘ S ⊆ ball 0 1
        apply clarsimp
        by (meson normψ1 p01 image_subset_iff mem_ball_0)
      show (p ∘ ψ ∘ h ∘ f) 0 = 0
        by (simp add: ‹p (ψ (h (f 0))) = 0›)
      show inj_on (p ∘ ψ ∘ h ∘ f) S
        unfolding inj_on_def o_def
        by (metis ψ2 dist_0_norm gf kh mem_ball nf1 normψ1 qp)
    qed
    then have le_norm_df0: norm (deriv (p ∘ ψ ∘ h ∘ f) 0) ≤ norm (deriv f 0)
      by (rule normf)
    have 1: k ∘ power2 ∘ q holomorphic_on ball 0 1
    proof (intro holomorphic_on_compose holq)
      show power2 holomorphic_on q ‘ ball 0 1
        using holomorphic_on_subset holomorphic_on_power
        by (blast intro: holomorphic_on_ident)
      show k holomorphic_on power2 ‘ q ‘ ball 0 1
        apply (rule holomorphic_on_subset [OF holk])
        using q01 by (auto simp: norm_power abs_square_less_1)
    qed
    have 2: (k ∘ power2 ∘ q) 0 = 0
      using p0 F_def ‹f ∈ F› ψ01 ψ2 ‹0 ∈ S› kh qp by force
    have 3: norm ((k ∘ power2 ∘ q) z) < 1 if norm z < 1 for z
    proof −
      have norm ((power2 ∘ q) z) < 1
        using that q01 by (force simp: norm_power abs_square_less_1)
      with k01 show ?thesis
        by fastforce
    qed
    have False if c: ∀ z. norm z < 1 ⟶ (k ∘ power2 ∘ q) z = c * z and norm
c = 1 for c
    proof −
      have c ≠ 0 using that by auto
      have norm (p(1/2)) < 1 norm (p(−1/2)) < 1
        using p01 by force+
        then have (k ∘ power2 ∘ q) (p(1/2)) = c * p(1/2) (k ∘ power2 ∘ q)
(p(−1/2)) = c * p(−1/2)
```

```
        using c by force+
      then have p (1/2) = p (- (1/2))
        by (auto simp: ‹c ≠ 0› qp o_def)
      then have q (p (1/2)) = q (p (- (1/2)))
        by simp
      then have 1/2 = - (1/2::complex)
        by (auto simp: qp)
      then show False
        by simp
    qed
    moreover
    have False if norm (deriv (k ∘ power2 ∘ q) 0) ≠ 1 norm (deriv (k ∘ power2
∘ q) 0) ≤ 1
      and le: ⋀ξ. norm ξ < 1 ⟹ norm ((k ∘ power2 ∘ q) ξ) ≤ norm ξ
    proof -
      have norm (deriv (k ∘ power2 ∘ q) 0) < 1
        using that by simp
      moreover have eq: deriv f 0 = deriv (k ∘ (λz. z ^ 2) ∘ q) 0 * deriv (p ∘
ψ ∘ h ∘ f) 0
    proof (intro DERIV_imp_deriv has_field_derivative_transform_within_open
[OF DERIV_chain])
        show (k ∘ power2 ∘ q has_field_derivative deriv (k ∘ power2 ∘ q) 0) (at
((p ∘ ψ ∘ h ∘ f) 0))
          using 1 holomorphic_derivI p0 by auto
        show (p ∘ ψ ∘ h ∘ f has_field_derivative deriv (p ∘ ψ ∘ h ∘ f) 0) (at 0)
          using ‹p ∘ ψ ∘ h ∘ f ∈ F› ‹open S› ‹0 ∈ S› holF holomorphic_derivI
by blast
        show ⋀x. x ∈ S ⟹ (k ∘ power2 ∘ q ∘ (p ∘ ψ ∘ h ∘ f)) x = f x
          using ψ2 f01 kh normψ1 qp by auto
      qed (use assms in simp_all)
      ultimately have cmod (deriv (p ∘ ψ ∘ h ∘ f) 0) ≤ 0
        using le_norm_df0
      by (metis linorder_not_le mult.commute mult_less_cancel_left2 norm_mult)
      moreover have 1 ≤ norm (deriv f 0)
        using normf [of id] by (simp add: idF)
      ultimately show False
        by (simp add: eq)
    qed
    ultimately show ?thesis
      using Schwarz_Lemma [OF 1 2 3] norm_one by blast
  qed
  then show a ∈ f ‘ S
    by blast
qed
then have f ‘ S = ball 0 1
  using F_def ‹f ∈ F› by blast
then show ?thesis
  apply (rule_tac x=f in exI)
  apply (rule_tac x=g in exI)
```

```
    using holf holg derg gf by safe force+
qed


locale SC_Chain =
  fixes S :: complex set
  assumes openS: open S
begin

lemma winding_number_zero:
  assumes simply_connected S
  shows connected S ∧
        (∀γ z. path γ ∧ path_image γ ⊆ S ∧
                pathfinish γ = pathstart γ ∧ z ∉ S ⟶ winding_number γ z = 0)
  using assms
 by (auto simp: simply_connected_imp_connected simply_connected_imp_winding_number_zero)

lemma contour_integral_zero:
  assumes valid_path g path_image g ⊆ S pathfinish g = pathstart g f holomor-
phic_on S
        ⋀γ z. ⟦path γ; path_image γ ⊆ S; pathfinish γ = pathstart γ; z ∉ S⟧ ⟹
winding_number γ z = 0
  shows (f has_contour_integral 0) g
  using assms by (meson Cauchy_theorem_global openS valid_path_imp_path)

lemma global_primitive:
  assumes connected S and holf: f holomorphic_on S
  and prev: ⋀γ f. ⟦valid_path γ; path_image γ ⊆ S; pathfinish γ = pathstart γ; f
holomorphic_on S⟧ ⟹ (f has_contour_integral 0) γ
  shows ∃h. ∀z ∈ S. (h has_field_derivative f z) (at z)
proof (cases S = {})
case True then show ?thesis
    by simp
next
  case False
  then obtain a where a ∈ S
    by blast
  show ?thesis
  proof (intro exI ballI)
    fix x assume x ∈ S
    then obtain d where d > 0 and d: cball x d ⊆ S
      using openS open_contains_cball_eq by blast
   let ?g = λz. (SOME g. polynomial_function g ∧ path_image g ⊆ S ∧ pathstart
g = a ∧ pathfinish g = z)
    show ((λz. contour_integral (?g z) f) has_field_derivative f x)
         (at x)
   proof (simp add: has_field_derivative_def has_derivative_at2 bounded_linear_mult_right,
rule Lim_transform)
      show (λy. inverse(norm(y − x)) *R (contour_integral(linepath x y) f − f x
```

```
∗ (y − x))) −x→ 0
      proof (clarsimp simp add: Lim_at)
       fix e::real assume e > 0
       moreover have continuous (at x) f
          using openS ‹x ∈ S› holf continuous_on_eq_continuous_at holomor-
phic_on_imp_continuous_on by auto
       ultimately obtain d1 where d1 > 0
          and d1: ⋀x'. dist x' x < d1 ⟹ dist (f x') (f x) < e/2
         unfolding continuous_at_eps_delta
         by (metis less_divide_eq_numeral1(1) mult_zero_left)
       obtain d2 where d2 > 0 and d2: ball x d2 ⊆ S
         using openS ‹x ∈ S› open_contains_ball_eq by blast
       have inverse (norm (y − x)) ∗ norm (contour_integral (linepath x y) f − f
x ∗ (y − x)) < e
         if 0 < d1 0 < d2 y ≠ x dist y x < d1 dist y x < d2 for y
       proof −
         have f contour_integrable_on linepath x y
       proof (rule contour_integrable_continuous_linepath [OF continuous_on_subset])
           show continuous_on S f
             by (simp add: holf holomorphic_on_imp_continuous_on)
           have closed_segment x y ⊆ ball x d2
             by (meson dist_commute_lessI dist_in_closed_segment le_less_trans
mem_ball subsetI that(5))
           with d2 show closed_segment x y ⊆ S
             by blast
         qed
         then obtain z where z: (f has_contour_integral z) (linepath x y)
           by (force simp: contour_integrable_on_def)
         have con: ((λw. f x) has_contour_integral f x ∗ (y − x)) (linepath x y)
           using has_contour_integral_const_linepath [of f x y x] by metis
         have norm (z − f x ∗ (y − x)) ≤ (e/2) ∗ norm (y − x)
         proof (rule has_contour_integral_bound_linepath)
          show ((λw. f w − f x) has_contour_integral z − f x ∗ (y − x)) (linepath
x y)
             by (rule has_contour_integral_diff [OF z con])
           show ⋀w. w ∈ closed_segment x y ⟹ norm (f w − f x) ≤ e/2
            by (metis d1 dist_norm less_le_trans not_less not_less_iff_gr_or_eq
segment_bound1 that(4))
         qed (use ‹e > 0› in auto)
          with ‹e > 0› have inverse (norm (y − x)) ∗ norm (z − f x ∗ (y − x))
≤ e/2
           by (simp add: field_split_simps)
         also have ... < e
           using ‹e > 0› by simp
         finally show ?thesis
           by (simp add: contour_integral_unique [OF z])
       qed
       with ‹d1 > 0› ‹d2 > 0›
       show ∃ d>0. ∀ z. z ≠ x ∧ dist z x < d ⟶
```

```
            inverse (norm (z − x)) ∗ norm (contour_integral (linepath x z) f −
f x ∗ (z − x)) < e
        by (rule_tac x=min d1 d2 in exI) auto
    qed
  next
    have ∗: (1 / norm (y − x)) ∗R (contour_integral (?g y) f −
          (contour_integral (?g x) f + f x ∗ (y − x))) =
          (contour_integral (linepath x y) f − f x ∗ (y − x)) /R norm (y − x)
      if 0 < d y ≠ x and yx: dist y x < d for y
    proof −
      have y ∈ S
        by (metis subsetD d dist_commute less_eq_real_def mem_cball yx)
      have gxy: polynomial_function (?g x) ∧ path_image (?g x) ⊆ S ∧ pathstart
(?g x) = a ∧ pathfinish (?g x) = x
            polynomial_function (?g y) ∧ path_image (?g y) ⊆ S ∧ pathstart
(?g y) = a ∧ pathfinish (?g y) = y
        using someI_ex [OF connected_open_polynomial_connected [OF openS
‹connected S› ‹a ∈ S›]] ‹x ∈ S› ‹y ∈ S›
        by meson+
      then have vp: valid_path (?g x) valid_path (?g y)
        by (simp_all add: valid_path_polynomial_function)
       have f0: (f has_contour_integral 0) ((?g x) +++ linepath x y +++
reversepath (?g y))
      proof (rule prev)
        show valid_path ((?g x) +++ linepath x y +++ reversepath (?g y))
          using gxy vp by (auto simp: valid_path_join)
        have closed_segment x y ⊆ cball x d
          using yx by (auto simp: dist_commute dest!: dist_in_closed_segment)
        then have closed_segment x y ⊆ S
          using d by blast
        then show path_image ((?g x) +++ linepath x y +++ reversepath (?g
y)) ⊆ S
          using gxy by (auto simp: path_image_join)
      qed (use gxy holf in auto)
      then have fintxy: f contour_integrable_on linepath x y
      by (metis (no_types, lifting) contour_integrable_joinD1 contour_integrable_joinD2
gxy(2) has_contour_integral_integrable pathfinish_linepath pathstart_reversepath
valid_path_imp_reverse valid_path_join valid_path_linepath vp(2))
      have fintgx: f contour_integrable_on (?g x) f contour_integrable_on (?g y)
       using openS contour_integrable_holomorphic_simple gxy holf vp by blast+
      show ?thesis
        apply (clarsimp simp add: divide_simps)
        using contour_integral_unique [OF f0]
      apply (simp add: fintxy gxy contour_integrable_reversepath contour_integral_reversepath
fintgx vp)
        apply (simp add: algebra_simps)
        done
    qed
    show (λz. (1 / norm (z − x)) ∗R
```

```
                (contour_integral (?g z) f − (contour_integral (?g x) f + f x ∗ (z
− x))) −
              (contour_integral (linepath x z) f − f x ∗ (z − x)) /R norm (z − x))
          −x→ 0
      apply (rule tendsto_eventually)
      apply (simp add: eventually_at)
      apply (rule_tac x=d in exI)
      using ‹d > 0› ∗ by simp
    qed
  qed
qed

lemma holomorphic_log:
  assumes connected S and holf: f holomorphic_on S and nz: ⋀z. z ∈ S ⟹ f z
≠ 0
  and prev: ⋀f. f holomorphic_on S ⟹ ∃ h. ∀ z ∈ S. (h has_field_derivative f z)
(at z)
  shows ∃ g. g holomorphic_on S ∧ (∀ z ∈ S. f z = exp(g z))
proof −
  have (λz. deriv f z / f z) holomorphic_on S
    by (simp add: openS holf holomorphic_deriv holomorphic_on_divide nz)
  then obtain g where g: ⋀z. z ∈ S ⟹ (g has_field_derivative deriv f z / f z)
(at z)
    using prev [of λz. deriv f z / f z] by metis
  have hfd: ⋀x. x ∈ S ⟹ ((λz. exp (g z) / f z) has_field_derivative 0) (at x)
    apply (rule derivative_eq_intros g| simp)+
      apply (subst DERIV_deriv_iff_field_differentiable)
    using openS holf holomorphic_on_imp_differentiable_at nz apply auto
    done
  obtain c where c: ⋀x. x ∈ S ⟹ exp (g x) / f x = c
 proof (rule DERIV_zero_connected_constant[OF ‹connected S› openS finite.emptyI])
    show continuous_on S (λz. exp (g z) / f z)
     by (metis (full_types) openS g continuous_on_divide continuous_on_exp holf
holomorphic_on_imp_continuous_on holomorphic_on_open nz)
    then show ∀ x∈S − {}. ((λz. exp (g z) / f z) has_field_derivative 0) (at x)
      using hfd by (blast intro: DERIV_zero_connected_constant [OF ‹connected
S› openS finite.emptyI, of λz. exp(g z) / f z])
  qed auto
  show ?thesis
  proof (intro exI ballI conjI)
    show (λz. Ln(inverse c) + g z) holomorphic_on S
      apply (intro holomorphic_intros)
      using openS g holomorphic_on_open by blast
    fix z :: complex
    assume z ∈ S
    then have exp (g z) / c = f z
    by (metis c divide_divide_eq_right exp_not_eq_zero nonzero_mult_div_cancel_left)
    moreover have 1 / c ≠ 0
      using ‹z ∈ S› c nz by fastforce
```

```
    ultimately show f z = exp (Ln (inverse c) + g z)
      by (simp add: exp_add inverse_eq_divide)
  qed
qed

lemma holomorphic_sqrt:
  assumes holf: f holomorphic_on S and nz: ⋀z. z ∈ S ⟹ f z ≠ 0
  and prev: ⋀f. ⟦f holomorphic_on S; ∀ z ∈ S. f z ≠ 0⟧ ⟹ ∃ g. g holomorphic_on
S ∧ (∀ z ∈ S. f z = exp(g z))
  shows ∃ g. g holomorphic_on S ∧ (∀ z ∈ S. f z = (g z)^2)
proof −
  obtain g where holg: g holomorphic_on S and g: ⋀z. z ∈ S ⟹ f z = exp (g
z)
    using prev [of f] holf nz by metis
  show ?thesis
  proof (intro exI ballI conjI)
    show (λz. exp(g z/2)) holomorphic_on S
      by (intro holomorphic_intros) (auto simp: holg)
    show ⋀z. z ∈ S ⟹ f z = (exp (g z/2))^2
    by (metis (no_types) g exp_double nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
  qed
qed

lemma biholomorphic_to_disc:
  assumes connected S and S: S ≠ {} S ≠ UNIV
  and prev: ⋀f. ⟦f holomorphic_on S; ∀ z ∈ S. f z ≠ 0⟧ ⟹ ∃ g. g holomorphic_on
S ∧ (∀ z ∈ S. f z = (g z)^2)
  shows ∃ f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧
                (∀ z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧
                (∀ z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)
proof −
  obtain a b where a ∈ S b ∉ S
    using S by blast
  then obtain δ where δ > 0 and δ: ball a δ ⊆ S
    using openS openE by blast
  obtain g where holg: g holomorphic_on S and eqg: ⋀z. z ∈ S ⟹ z − b = (g
z)^2
  proof (rule exE [OF prev [of λz. z − b]])
    show (λz. z − b) holomorphic_on S
      by (intro holomorphic_intros)
  qed (use ‹b ∉ S› in auto)
  have ¬ g constant_on S
  proof −
    have (a + δ/2) ∈ ball a δ a + (δ/2) ≠ a
      using ‹δ > 0› by (simp_all add: dist_norm)
    then show ?thesis
      unfolding constant_on_def
      using eqg [of a] eqg [of a + δ/2] ‹a ∈ S› δ
```

```
    by (metis diff_add_cancel subset_eq)
  qed
  then have open (g ‘ ball a δ)
    using open_mapping_thm [of g S ball a δ, OF holg openS ‹connected S›] δ by
blast
  then obtain r where r > 0 and r: ball (g a) r ⊆ (g ‘ ball a δ)
    by (metis ‹0 < δ› centre_in_ball imageI openE)
  have g_not_r: g z ∉ ball (−(g a)) r if z ∈ S for z
  proof
    assume g z ∈ ball (−(g a)) r
    then have − g z ∈ ball (g a) r
      by (metis add.inverse_inverse dist_minus mem_ball)
    with r have − g z ∈ (g ‘ ball a δ)
      by blast
    then obtain w where w: − g z = g w dist a w < δ
      by auto
    then have w ∈ ball a δ
      by simp
    then have w ∈ S
      using δ by blast
    then have w = z
      by (metis diff_add_cancel eqg power_minus_Bit0 that w(1))
    then have g z = 0
      using ‹− g z = g w› by auto
    with eqg [OF that] have z = b
      by auto
    with that ‹b ∉ S› show False
      by simp
  qed
  then have nz: ⋀z. z ∈ S ⟹ g z + g a ≠ 0
    by (metis ‹0 < r› add.commute add_diff_cancel_left' centre_in_ball diff_0)
  let ?f = λz. (r/3) / (g z + g a) − (r/3) / (g a + g a)
  obtain h where holh: h holomorphic_on S and h a = 0 and h01: h ‘ S ⊆ ball
0 1 and inj_on h S
  proof
    show ?f holomorphic_on S
      by (intro holomorphic_intros holg nz)
   have 3: ⟦norm x ≤ 1/3; norm y ≤ 1/3⟧ ⟹ norm(x − y) < 1 for x y::complex
      using norm_triangle_ineq4 [of x y] by simp
    have norm ((r/3) / (g z + g a) − (r/3) / (g a + g a)) < 1 if z ∈ S for z
      apply (rule 3)
      unfolding norm_divide
      using ‹r > 0› g_not_r [OF ‹z ∈ S›] g_not_r [OF ‹a ∈ S›]
      by (simp_all add: field_split_simps dist_commute dist_norm)
  then show ?f ‘ S ⊆ ball 0 1
    by auto
    show inj_on ?f S
      using ‹r > 0› eqg apply (clarsimp simp: inj_on_def)
      by (metis diff_add_cancel)
```

```
  qed auto
  obtain k where holk: k holomorphic_on (h ‘ S)
            and derk: ⋀z. z ∈ S ⟹ deriv h z * deriv k (h z) = 1
            and kh: ⋀z. z ∈ S ⟹ k(h z) = z
    using holomorphic_has_inverse [OF holh openS ‹inj_on h S›] by metis

  have 1: open (h ‘ S)
    by (simp add: ‹inj_on h S› holh openS open_mapping_thm3)
  have 2: connected (h ‘ S)
   by (simp add: connected_continuous_image ‹connected S› holh holomorphic_on_imp_continuous_on)
  have 3: 0 ∈ h ‘ S
    using ‹a ∈ S› ‹h a = 0› by auto
  have 4: ∃g. g holomorphic_on h ‘ S ∧ (∀z∈h ‘ S. f z = (g z)²)
    if holf: f holomorphic_on h ‘ S and nz: ⋀z. z ∈ h ‘ S ⟹ f z ≠ 0 inj_on f (h
‘ S) for f
  proof −
    obtain g where holg: g holomorphic_on S and eqg: ⋀z. z ∈ S ⟹ (f ∘ h) z
= (g z)²
    proof −
      have f ∘ h holomorphic_on S
        by (simp add: holh holomorphic_on_compose holf)
      moreover have ∀z∈S. (f ∘ h) z ≠ 0
        by (simp add: nz)
      ultimately show thesis
        using prev that by blast
    qed
    show ?thesis
    proof (intro exI conjI)
      show g ∘ k holomorphic_on h ‘ S
      proof −
        have k ‘ h ‘ S ⊆ S
          by (simp add: ‹⋀z. z ∈ S ⟹ k (h z) = z› image_subset_iff)
        then show ?thesis
          by (meson holg holk holomorphic_on_compose holomorphic_on_subset)
      qed
      show ∀z∈h ‘ S. f z = ((g ∘ k) z)²
        using eqg kh by auto
    qed
  qed
  obtain f g where f: f holomorphic_on h ‘ S and g: g holomorphic_on ball 0 1
      and gf: ∀z∈h ‘ S. f z ∈ ball 0 1 ∧ g (f z) = z and fg:∀z∈ball 0 1. g z ∈ h
‘ S ∧ f (g z) = z
    using biholomorphic_to_disc_aux [OF 1 2 3 h01 4] by blast
  show ?thesis
  proof (intro exI conjI)
    show f ∘ h holomorphic_on S
      by (simp add: f holh holomorphic_on_compose)
    show k ∘ g holomorphic_on ball 0 1
        by (metis holomorphic_on_subset image_subset_iff fg holk g holomor-
```

phic_on_compose)
qed (*use fg gf kh* **in** *auto*)
qed

lemma *homeomorphic_to_disc*:
assumes *S*: $S \neq \{\}$
and *prev*: $S = UNIV \vee$
$(\exists f\ g.\ f$ *holomorphic_on* $S \wedge g$ *holomorphic_on ball 0 1* $\wedge$
$(\forall z \in S.\ f\ z \in$ *ball 0 1* $\wedge\ g(f\ z) = z) \wedge$
$(\forall z \in$ *ball 0 1*$.\ g\ z \in S \wedge f(g\ z) = z))$ (**is** _ $\vee$ *?P*)
shows *S homeomorphic ball* (*0*::*complex*) *1*
using *prev*
proof
assume $S = UNIV$ **then show** *?thesis*
using *homeomorphic_ball01_UNIV homeomorphic_sym* **by** *blast*
next
assume *?P*
then show *?thesis*
unfolding *homeomorphic_minimal*
using *holomorphic_on_imp_continuous_on* **by** *blast*
qed

lemma *homeomorphic_to_disc_imp_simply_connected*:
assumes $S = \{\} \vee S$ *homeomorphic ball* (*0*::*complex*) *1*
shows *simply_connected S*
using *assms homeomorphic_simply_connected_eq convex_imp_simply_connected*
by *auto*

end

proposition
assumes *open S*
shows *simply_connected_eq_winding_number_zero*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$
$(\forall g\ z.$ *path g* $\wedge$ *path_image g* $\subseteq S \wedge$
pathfinish g = *pathstart g* $\wedge$ $(z \notin S)$
$\longrightarrow$ *winding_number g z* = *0*) (**is** *?wn0*)
and *simply_connected_eq_contour_integral_zero*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$
$(\forall g\ f.$ *valid_path g* $\wedge$ *path_image g* $\subseteq S \wedge$
pathfinish g = *pathstart g* $\wedge$ *f holomorphic_on S*
$\longrightarrow$ (*f has_contour_integral 0*) *g*) (**is** *?ci0*)
and *simply_connected_eq_global_primitive*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$
$(\forall f.$ *f holomorphic_on S* $\longrightarrow$
$(\exists h.\ \forall z.\ z \in S \longrightarrow$ (*h has_field_derivative f z*) (*at z*))) (**is** *?gp*)

```
    and simply_connected_eq_holomorphic_log:
        simply_connected S ⟷
          connected S ∧
          (∀f. f holomorphic_on S ∧ (∀z ∈ S. f z ≠ 0)
              ⟶ (∃g. g holomorphic_on S ∧ (∀z ∈ S. f z = exp(g z)))) (is ?log)
    and simply_connected_eq_holomorphic_sqrt:
        simply_connected S ⟷
          connected S ∧
          (∀f. f holomorphic_on S ∧ (∀z ∈ S. f z ≠ 0)
              ⟶ (∃g. g holomorphic_on S ∧ (∀z ∈ S.  f z = (g z)^2))) (is ?sqrt)
    and simply_connected_eq_biholomorphic_to_disc:
        simply_connected S ⟷
          S = {} ∨ S = UNIV ∨
          (∃f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧
                (∀z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧
                (∀z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)) (is ?bih)
    and simply_connected_eq_homeomorphic_to_disc:
         simply_connected S ⟷ S = {} ∨ S homeomorphic ball (0::complex) 1
(is ?disc)
proof −
  interpret SC_Chain
    using assms by (simp add: SC_Chain_def)
  have ?wn0 ∧ ?ci0 ∧ ?gp ∧ ?log ∧ ?sqrt ∧ ?bih ∧ ?disc
proof −
  have ∗: ⟦α ⟹ β; β ⟹ γ; γ ⟹ δ; δ ⟹ ζ; ζ ⟹ η; η ⟹ ϑ; ϑ ⟹ ξ; ξ ⟹
α⟧
       ⟹ (α ⟷ β) ∧ (α ⟷ γ) ∧ (α ⟷ δ) ∧ (α ⟷ ζ) ∧
          (α ⟷ η) ∧ (α ⟷ ϑ) ∧ (α ⟷ ξ) for α β γ δ ζ η ϑ ξ
    by blast
  show ?thesis
    apply (rule ∗)
    using winding_number_zero apply metis
    using contour_integral_zero apply metis
    using global_primitive apply metis
    using holomorphic_log apply metis
    using holomorphic_sqrt apply simp
    using biholomorphic_to_disc apply blast
    using homeomorphic_to_disc apply blast
    using homeomorphic_to_disc_imp_simply_connected apply blast
    done
qed
  then show ?wn0 ?ci0 ?gp ?log ?sqrt ?bih ?disc
    by safe
qed

corollary contractible_eq_simply_connected_2d:
  fixes S :: complex set
  shows open S ⟹ (contractible S ⟷ simply_connected S)
  apply safe
```

apply (*simp add*: *contractible_imp_simply_connected*)
using *convex_imp_contractible homeomorphic_contractible_eq simply_connected_eq_homeomorphic_to_di*
by *auto*

8.3 A further chain of equivalences about components of the complement of a simply connected set

(following 1.35 in Burckel'S book)

context *SC_Chain*
begin

lemma *frontier_properties*:
assumes *simply_connected S*
shows *if bounded S then connected*(*frontier S*)
else ∀ *C* ∈ *components*(*frontier S*). ¬ *bounded C*
proof −
have *S* = {} ∨ *S homeomorphic ball* (*0*::*complex*) *1*
using *simply_connected_eq_homeomorphic_to_disc assms openS* **by** *blast*
then show *?thesis*
proof
assume *S* = {}
then have *bounded S*
by *simp*
with ‹*S* = {}› **show** *?thesis*
by *simp*
next
assume *S01*: *S homeomorphic ball* (*0*::*complex*) *1*
then obtain *g f*
where *gim*: *g* ' *S* = *ball 0 1* **and** *fg*: ⋀*x*. *x* ∈ *S* ⟹ *f*(*g x*) = *x*
and *fim*: *f* ' *ball 0 1* = *S* **and** *gf*: ⋀*y*. *cmod y* < *1* ⟹ *g*(*f y*) = *y*
and *contg*: *continuous_on S g* **and** *contf*: *continuous_on* (*ball 0 1*) *f*
by (*fastforce simp*: *homeomorphism_def homeomorphic_def*)
define *D* **where** *D* ≡ λ*n*. *ball* (*0*::*complex*) (*1* − *1*/(*of_nat n* + *2*))
define *A* **where** *A* ≡ λ*n*. {*z*::*complex*. *1* − *1*/(*of_nat n* + *2*) < *norm z* ∧
norm z < *1*}
define *X* **where** *X* ≡ λ*n*::*nat*. *closure*(*f* ' *A n*)
have *D01*: *D n* ⊆ *ball 0 1* **for** *n*
by (*simp add*: *D_def ball_subset_ball_iff*)
have *A01*: *A n* ⊆ *ball 0 1* **for** *n*
by (*auto simp*: *A_def*)
have *cloX*: *closed*(*X n*) **for** *n*
by (*simp add*: *X_def*)
have *Xsubclo*: *X n* ⊆ *closure S* **for** *n*
unfolding *X_def* **by** (*metis A01 closure_mono fim image_mono*)
have *connX*: *connected*(*X n*) **for** *n*
unfolding *X_def*
apply (*rule connected_imp_connected_closure*)
apply (*rule connected_continuous_image*)
apply (*simp add*: *continuous_on_subset* [*OF contf A01*])

```
    using connected_annulus [of _ 0::complex] by (simp add: A_def)
  have nestX: X n ⊆ X m if m ≤ n for m n
  proof −
    have 1 − 1 / (real m + 2) ≤ 1 − 1 / (real n + 2)
      using that by (auto simp: field_simps)
    then show ?thesis
      by (auto simp: X_def A_def intro!: closure_mono)
  qed
  have closure S − S ⊆ (⋂ n. X n)
  proof
    fix x
    assume x ∈ closure S − S
    then have x ∈ closure S x ∉ S by auto
    show x ∈ (⋂ n. X n)
    proof
      fix n
      have ball 0 1 = closure (D n) ∪ A n
        by (auto simp: D_def A_def le_less_trans)
      with fim have Seq: S = f ‘ (closure (D n)) ∪ f ‘ (A n)
        by (simp add: image_Un)
      have continuous_on (closure (D n)) f
          by (simp add: D_def cball_subset_ball_iff continuous_on_subset [OF
contf])
      moreover have compact (closure (D n))
        by (simp add: D_def)
      ultimately have clo_fim: closed (f ‘ closure (D n))
        using compact_continuous_image compact_imp_closed by blast
      have ∗: (f ‘ cball 0 (1 − 1 / (real n + 2))) ⊆ S
        by (force simp: D_def Seq)
      show x ∈ X n
        using ‹x ∈ closure S› unfolding X_def Seq
        using ‹x ∉ S› ∗ D_def clo_fim by auto
    qed
  qed
  moreover have (⋂ n. X n) ⊆ closure S − S
  proof −
    have (⋂ n. X n) ⊆ closure S
    proof −
      have (⋂ n. X n) ⊆ X 0
        by blast
      also have ... ⊆ closure S
        apply (simp add: X_def fim [symmetric])
        apply (rule closure_mono)
        by (auto simp: A_def)
      finally show (⋂ n. X n) ⊆ closure S .
    qed
    moreover have (⋂ n. X n) ∩ S ⊆ {}
    proof (clarify, clarsimp simp: X_def fim [symmetric])
      fix x assume x [rule_format]: ∀ n. f x ∈ closure (f ‘ A n) and cmod x < 1
```

```
      then obtain n where n: 1 / (1 - norm x) < of_nat n
        using reals_Archimedean2 by blast
      with ‹cmod x < 1› gr0I have XX: 1 / of_nat n < 1 - norm x and n > 0
        by (fastforce simp: field_split_simps algebra_simps)+
      have f x ∈ f ‘ (D n)
        using n ‹cmod x < 1› by (auto simp: field_split_simps algebra_simps
D_def)
      moreover have  f ‘ D n ∩ closure (f ‘ A n) = {}
      proof -
        have op_fDn: open(f ‘ (D n))
        proof (rule invariance_of_domain)
          show continuous_on (D n) f
            by (rule continuous_on_subset [OF contf D01])
          show open (D n)
            by (simp add: D_def)
          show inj_on f (D n)
          unfolding inj_on_def using D01 by (metis gf mem_ball_0 subsetCE)
        qed
        have injf: inj_on f (ball 0 1)
          by (metis mem_ball_0 inj_on_def gf)
        have D n ∪ A n ⊆ ball 0 1
          using D01 A01 by simp
        moreover have D n ∩ A n = {}
          by (auto simp: D_def A_def)
        ultimately have f ‘ D n ∩ f ‘ A n = {}
          by (metis A01 D01 image_is_empty inj_on_image_Int injf)
        then show ?thesis
          by (simp add: open_Int_closure_eq_empty [OF op_fDn])
      qed
      ultimately show False
        using x [of n] by blast
    qed
    ultimately
    show (⋂n. X n) ⊆ closure S - S
      using closure_subset disjoint_iff_not_equal by blast
  qed
  ultimately have closure S - S = (⋂n. X n) by blast
  then have frontierS: frontier S = (⋂n. X n)
    by (simp add: frontier_def openS interior_open)
  show ?thesis
  proof (cases bounded S)
    case True
    have bouX: bounded (X n) for n
      apply (simp add: X_def)
      apply (rule bounded_closure)
      by (metis A01 fim image_mono bounded_subset [OF True])
    have compaX: compact (X n) for n
      apply (simp add: compact_eq_bounded_closed bouX)
      apply (auto simp: X_def)
```

```
      done
    have connected (⋂n. X n)
      by (metis nestX compaX connX connected_nest)
    then show ?thesis
      by (simp add: True ‹frontier S = (⋂n. X n)›)
  next
    case False
    have unboundedX: ¬ bounded(X n) for n
    proof
      assume bXn: bounded(X n)
      have continuous_on (cball 0 (1 - 1 / (2 + real n))) f
        by (simp add: cball_subset_ball_iff continuous_on_subset [OF contf])
      then have bounded (f ‘ cball 0 (1 - 1 / (2 + real n)))
        by (simp add: compact_imp_bounded [OF compact_continuous_image])
      moreover have bounded (f ‘ A n)
        by (auto simp: X_def closure_subset image_subset_iff bounded_subset
[OF bXn])
      ultimately have bounded (f ‘ (cball 0 (1 - 1/(2 + real n)) ∪ A n))
        by (simp add: image_Un)
      then have bounded (f ‘ ball 0 1)
        apply (rule bounded_subset)
        apply (auto simp: A_def algebra_simps)
        done
      then show False
        using False by (simp add: fim [symmetric])
    qed
    have clo_INTX: closed(⋂(range X))
      by (metis cloX closed_INT)
    then have lcX: locally compact (⋂(range X))
      by (metis closed_imp_locally_compact)
    have False if C: C ∈ components (frontier S) and boC: bounded C for C
    proof -
      have closed C
        by (metis C closed_components frontier_closed)
      then have compact C
        by (metis boC compact_eq_bounded_closed)
      have Cco: C ∈ components (⋂(range X))
        by (metis frontierS C)
      obtain K where C ⊆ K compact K
                and Ksub: K ⊆ ⋂(range X) and clo: closed(⋂(range X) - K)
      proof (cases {k. C ⊆ k ∧ compact k ∧ openin (top_of_set (⋂(range X)))
k} = {})
        case True
        then show ?thesis
          using Sura_Bura [OF lcX Cco ‹compact C›] boC
          by (simp add: True)
      next
        case False
        then obtain L where compact L C ⊆ L and K: openin (top_of_set
```

```
(⋂x. X x)) L
          by blast
        show ?thesis
        proof
          show L ⊆ ⋂(range X)
            by (metis K openin_imp_subset)
          show closed (⋂(range X) − L)
              by (metis closedin_diff closedin_self closedin_closed_trans [OF _
clo_INTX] K)
        qed (use ‹compact L› ‹C ⊆ L› in auto)
      qed
      obtain U V where open U and compact (closure U) and open V K ⊆ U
                  and V: ⋂(range X) − K ⊆ V and U ∩ V = {}
        using separation_normal_compact [OF ‹compact K› clo] by blast
      then have U ∩ (⋂ (range X) − K) = {}
        by blast
      have (closure U − U) ∩ (⋂n. X n ∩ closure U) ≠ {}
      proof (rule compact_imp_fip)
        show compact (closure U − U)
          by (metis ‹compact (closure U)› ‹open U› compact_diff)
        show ⋀T. T ∈ range (λn. X n ∩ closure U) ⟹ closed T
          by clarify (metis cloX closed_Int closed_closure)
        show (closure U − U) ∩ ⋂ℱ ≠ {}
          if finite ℱ and ℱ: ℱ ⊆ range (λn. X n ∩ closure U) for ℱ
        proof
          assume empty: (closure U − U) ∩ ⋂ℱ = {}
          obtain J where finite J and J: ℱ = (λn. X n ∩ closure U) ‘ J
            using finite_subset_image [OF ‹finite ℱ› ℱ] by auto
          show False
          proof (cases J = {})
            case True
            with J empty have closed U
              by (simp add: closure_subset_eq)
            have C ≠ {}
              using C in_components_nonempty by blast
            then have U ≠ {}
              using ‹K ⊆ U› ‹C ⊆ K› by blast
            moreover have U ≠ UNIV
              using ‹compact (closure U)› by auto
            ultimately show False
              using ‹open U› ‹closed U› clopen by blast
          next
            case False
            define j where j ≡ Max J
            have j ∈ J
              by (simp add: False ‹finite J› j_def)
            have jmax: ⋀m. m ∈ J ⟹ m ≤ j
              by (simp add: j_def ‹finite J›)
            have ⋂ ((λn. X n ∩ closure U) ‘ J) = X j ∩ closure U
```

```
            using False jmax nestX ‹j ∈ J› by auto
          then have X j ∩ closure U = X j ∩ U
            apply safe
            using DiffI J empty apply auto[1]
            using closure_subset by blast
          then have openin (top_of_set (X j)) (X j ∩ closure U)
            by (simp add: openin_open_Int ‹open U›)
          moreover have closedin (top_of_set (X j)) (X j ∩ closure U)
            by (simp add: closedin_closed_Int)
          moreover have X j ∩ closure U ≠ X j
                by (metis unboundedX ‹compact (closure U)› bounded_subset
compact_eq_bounded_closed inf.order_iff)
          moreover have X j ∩ closure U ≠ {}
          proof −
            have C ≠ {}
              using C in_components_nonempty by blast
            moreover have C ⊆ X j ∩ closure U
              using ‹C ⊆ K› ‹K ⊆ U› Ksub closure_subset by blast
            ultimately show ?thesis by blast
          qed
          ultimately show False
            using connX [of j] by (force simp: connected_clopen)
        qed
      qed
    qed
    moreover have (⋂n. X n ∩ closure U) = (⋂n. X n) ∩ closure U
      by blast
    moreover have x ∈ U if ⋀n. x ∈ X n x ∈ closure U for x
    proof −
      have x ∉ V
       using ‹U ∩ V = {}› ‹open V› closure_iff_nhds_not_empty that(2) by
blast
      then show ?thesis
      by (metis (no_types) Diff_iff INT_I V ‹K ⊆ U› contra_subsetD that(1))
    qed
    ultimately show False
      by (auto simp: open_Int_closure_eq_empty [OF ‹open V›, of U])
  qed
  then show ?thesis
    by (auto simp: False)
 qed
qed


lemma unbounded_complement_components:
  assumes C: C ∈ components (− S) and S: connected S
    and prev: if bounded S then connected(frontier S)
              else ∀ C ∈ components(frontier S). ¬ bounded C
```

```
  shows ¬ bounded C
proof (cases bounded S)
  case True
  with prev have S ≠ UNIV and confr: connected(frontier S)
    by auto
  obtain w where C_ccsw: C = connected_component_set (− S) w and w ∉ S
    using C by (auto simp: components_def)
  show ?thesis
  proof (cases S = {})
    case True with C show ?thesis by auto
  next
    case False
    show ?thesis
    proof
      assume bounded C
      then have outside C ≠ {}
        using outside_bounded_nonempty by metis
       then obtain z where z: ¬ bounded (connected_component_set (− C) z)
and z ∉ C
        by (auto simp: outside_def)
      have clo_ccs: closed (connected_component_set (− S) x) for x
        by (simp add: closed_Compl closed_connected_component openS)
     have connected_component_set (− S) w = connected_component_set (− S)
z
      proof (rule joinable_connected_component_eq [OF confr])
        show frontier S ⊆ − S
          using openS by (auto simp: frontier_def interior_open)
        have False if connected_component_set (− S) w ∩ frontier (− S) = {}
        proof −
          have C ∩ frontier S = {}
            using that by (simp add: C_ccsw)
          then show False
            by (metis C_ccsw ComplI Compl_eq_Compl_iff Diff_subset False ‹w
∉ S› clo_ccs closure_closed compl_bot_eq connected_component_eq_UNIV con-
nected_component_eq_empty empty_subsetI frontier_complement frontier_def fron-
tier_not_empty frontier_of_connected_component_subset le_inf_iff subset_antisym)
        qed
        then show connected_component_set (− S) w ∩ frontier S ≠ {}
          by auto
        have ∗: ⟦frontier C ⊆ C; frontier C ⊆ F; frontier C ≠ {}⟧ ⟹ C ∩ F ≠
{} for C F::complex set
          by blast
        have connected_component_set (− S) z ∩ frontier (− S) ≠ {}
        proof (rule ∗)
       show frontier (connected_component_set (− S) z) ⊆ connected_component_set
(− S) z
          by (auto simp: closed_Compl closed_connected_component frontier_def
openS)
          show frontier (connected_component_set (− S) z) ⊆ frontier (− S)
```

```
          using frontier_of_connected_component_subset by fastforce
        have ¬ bounded (−S)
          by (simp add: True cobounded_imp_unbounded)
        then have connected_component_set (− S) z ≠ {}
          apply (simp only: connected_component_eq_empty)
          using confr openS ‹bounded C› ‹w ∉ S›
          apply (simp add: frontier_def interior_open C_ccsw)
        by (metis ComplI Compl_eq_Diff_UNIV connected_UNIV closed_closure
closure_subset connected_component_eq_self
                  connected_diff_open_from_closed subset_UNIV)
        then show frontier (connected_component_set (− S) z) ≠ {}
          apply (simp add: frontier_eq_empty connected_component_eq_UNIV)
          apply (metis False compl_top_eq double_compl)
          done
      qed
      then show connected_component_set (− S) z ∩ frontier S ≠ {}
        by auto
    qed
    then show False
    by (metis C_ccsw Compl_iff ‹w ∉ S› ‹z ∉ C› connected_component_eq_empty
connected_component_idemp)
  qed
 qed
next
 case False
 obtain w where C_ccsw: C = connected_component_set (− S) w and w ∉ S
  using C by (auto simp: components_def)
 have frontier (connected_component_set (− S) w) ⊆ connected_component_set
(− S) w
  by (simp add: closed_Compl closed_connected_component frontier_subset_eq
openS)
 moreover have frontier (connected_component_set (− S) w) ⊆ frontier S
  using frontier_complement frontier_of_connected_component_subset by blast
 moreover have frontier (connected_component_set (− S) w) ≠ {}
  by (metis C C_ccsw False bounded_empty compl_top_eq connected_component_eq_UNIV
double_compl frontier_not_empty in_components_nonempty)
 ultimately obtain z where zin: z ∈ frontier S and z: z ∈ connected_component_set
(− S) w
  by blast
 have ∗: connected_component_set (frontier S) z ∈ components(frontier S)
  by (simp add: ‹z ∈ frontier S› componentsI)
 with prev False have ¬ bounded (connected_component_set (frontier S) z)
  by simp
 moreover have connected_component (− S) w = connected_component (− S)
z
  using connected_component_eq [OF z] by force
 ultimately show ?thesis
   by (metis C_ccsw ∗ zin bounded_subset closed_Compl closure_closed con-
nected_component_maximal
```

connected_component_refl connected_connected_component frontier_closures in_components_subset le_inf_iff mem_Collect_eq openS)
qed

lemma *empty_inside*:
assumes *connected S* $\bigwedge C$. $C \in$ *components* $(-S) \Longrightarrow \neg$ *bounded C*
shows *inside* $S = \{\}$
using *assms* **by** (*auto simp*: *components_def inside_def*)

lemma *empty_inside_imp_simply_connected*:
$[\![$*connected S*; *inside* $S = \{\}]\!] \Longrightarrow$ *simply_connected S*
by (*metis ComplI inside_Un_outside openS outside_mono simply_connected_eq_winding_number_zero subsetCE sup_bot.left_neutral winding_number_zero_in_outside*)

end

proposition
fixes *S* :: *complex set*
assumes *open S*
shows *simply_connected_eq_frontier_properties*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$
(*if bounded S then connected*(*frontier S*)
else $(\forall\, C \in$ *components*(*frontier S*). $\neg$*bounded C*)) (**is** *?fp*)
and *simply_connected_eq_unbounded_complement_components*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$ $(\forall\, C \in$ *components*$(-S)$. $\neg$*bounded C*) (**is** *?ucc*)
and *simply_connected_eq_empty_inside*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$ *inside* $S = \{\}$ (**is** *?ei*)
proof −
interpret *SC_Chain*
using *assms* **by** (*simp add*: *SC_Chain_def*)
have *?fp* $\wedge$ *?ucc* $\wedge$ *?ei*
using *empty_inside empty_inside_imp_simply_connected frontier_properties unbounded_complement_components winding_number_zero* **by** *blast*
then show *?fp ?ucc ?ei*
by *blast*+
qed

lemma *simply_connected_iff_simple*:
fixes *S* :: *complex set*
assumes *open S bounded S*
shows *simply_connected S* $\longleftrightarrow$ *connected S* $\wedge$ *connected*$(-S)$
apply (*simp add*: *simply_connected_eq_unbounded_complement_components assms*, *safe*)
apply (*metis DIM_complex assms*(*2*) *cobounded_has_bounded_component double_compl order_refl*)
by (*meson assms inside_bounded_complement_connected_empty simply_connected_eq_empty_inside*

simply_connected_eq_unbounded_complement_components)

lemma *subset_simply_connected_imp_inside_subset*:
fixes $A :: complex\ set$
assumes *simply_connected A open A* $B \subseteq A$
shows *inside B* $\subseteq A$
by (*metis assms Diff_eq_empty_iff inside_mono subset_empty simply_connected_eq_empty_inside*)

8.4 Further equivalences based on continuous logs and sqrts

context *SC_Chain*
begin

lemma *continuous_log*:
fixes $f :: complex \Rightarrow complex$
assumes *S*: *simply_connected S*
and *contf*: *continuous_on S f* **and** *nz*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
shows $\exists g.$ *continuous_on S g* $\wedge$ $(\forall z \in S.\ f\ z = exp(g\ z))$
proof −
consider $S = \{\}$ | *S homeomorphic ball* $(0::complex)$ *1*
using *simply_connected_eq_homeomorphic_to_disc* [*OF openS*] *S* **by** *metis*
then show *?thesis*
proof *cases*
case *1* **then show** *?thesis*
by *simp*
next
case *2*
then obtain *h k* :: $complex \Rightarrow complex$
where *kh*: $\bigwedge x.\ x \in S \Longrightarrow k(h\ x) = x$ **and** *him*: *h ' S = ball 0 1*
and *conth*: *continuous_on S h*
and *hk*: $\bigwedge y.\ y \in ball\ 0\ 1 \Longrightarrow h(k\ y) = y$ **and** *kim*: *k ' ball 0 1 = S*
and *contk*: *continuous_on* (*ball 0 1*) *k*
unfolding *homeomorphism_def homeomorphic_def* **by** *metis*
obtain *g* **where** *contg*: *continuous_on* (*ball 0 1*) *g*
and *expg*: $\bigwedge z.\ z \in ball\ 0\ 1 \Longrightarrow (f \circ k)\ z = exp\ (g\ z)$
proof (*rule continuous_logarithm_on_ball*)
show *continuous_on* (*ball 0 1*) $(f \circ k)$
using *contf continuous_on_compose contk kim* **by** *blast*
show $\bigwedge z.\ z \in ball\ 0\ 1 \Longrightarrow (f \circ k)\ z \neq 0$
using *kim nz* **by** *auto*
qed *auto*
then show *?thesis*
by (*metis comp_apply conth continuous_on_compose him imageI kh*)
qed
qed

lemma *continuous_sqrt*:
fixes $f :: complex \Rightarrow complex$
assumes *contf*: *continuous_on S f* **and** *nz*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$

```
  and prev: ⋀f::complex⇒complex.
              ⟦continuous_on S f; ⋀z. z ∈ S ⟹ f z ≠ 0⟧
                ⟹ ∃g. continuous_on S g ∧ (∀z ∈ S. f z = exp(g z))
  shows ∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2)
proof −
  obtain g where contg: continuous_on S g and geq: ⋀z. z ∈ S ⟹ f z = exp(g
z)
    using contf nz prev by metis
  show ?thesis
proof (intro exI ballI conjI)
  show continuous_on S (λz. exp(g z/2))
     by (intro continuous_intros) (auto simp: contg)
    show ⋀z. z ∈ S ⟹ f z = (exp (g z/2))^2
     by (metis (no_types, lifting) divide_inverse exp_double geq mult.left_commute
mult.right_neutral right_inverse zero_neq_numeral)
  qed
qed

lemma continuous_sqrt_imp_simply_connected:
  assumes connected S
    and prev: ⋀f::complex⇒complex. ⟦continuous_on S f; ∀z ∈ S. f z ≠ 0⟧
              ⟹ ∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2)
  shows simply_connected S
proof (clarsimp simp add: simply_connected_eq_holomorphic_sqrt [OF openS]
‹connected S›)
  fix f
  assume f holomorphic_on S and nz: ∀z∈S. f z ≠ 0
  then obtain g where contg: continuous_on S g and geq: ⋀z. z ∈ S ⟹ f z =
(g z)^2
    by (metis holomorphic_on_imp_continuous_on prev)
  show ∃g. g holomorphic_on S ∧ (∀z∈S. f z = (g z)^2)
  proof (intro exI ballI conjI)
    show g holomorphic_on S
    proof (clarsimp simp add: holomorphic_on_open [OF openS])
      fix z
      assume z ∈ S
      with nz geq have g z ≠ 0
        by auto
       obtain δ where 0 < δ ⋀w. ⟦w ∈ S; dist w z < δ⟧ ⟹ dist (g w) (g z) <
cmod (g z)
         using contg [unfolded continuous_on_iff] by (metis ‹g z ≠ 0› ‹z ∈ S›
zero_less_norm_iff)
      then have δ: ⋀w. ⟦w ∈ S; w ∈ ball z δ⟧ ⟹ g w + g z ≠ 0
        apply (clarsimp simp: dist_norm)
          by (metis ‹g z ≠ 0› add_diff_cancel_left' diff_0_right norm_eq_zero
norm_increases_online norm_minus_commute norm_not_less_zero not_less_iff_gr_or_eq)
      have ∗: (λx. (f x − f z) / (x − z) / (g x + g z)) −z→ deriv f z / (g z + g z)
        apply (intro tendsto_intros)
        using SC_Chain.openS SC_Chain_axioms ‹f holomorphic_on S› ‹z ∈ S›
```

```
has_field_derivativeD holomorphic_derivI apply fastforce
       using ‹z ∈ S› contg continuous_on_eq_continuous_at isCont_def openS
apply blast
      by (simp add: ‹g z ≠ 0›)
    then have (g has_field_derivative deriv f z / (g z + g z)) (at z)
      unfolding has_field_derivative_iff
    proof (rule Lim_transform_within_open)
      show open (ball z δ ∩ S)
        by (simp add: openS open_Int)
      show z ∈ ball z δ ∩ S
        using ‹z ∈ S› ‹0 < δ› by simp
      show ⋀x. ⟦x ∈ ball z δ ∩ S; x ≠ z⟧
             ⟹ (f x − f z) / (x − z) / (g x + g z) = (g x − g z) / (x − z)
        using δ
        apply (simp add: geq ‹z ∈ S› divide_simps)
        apply (auto simp: algebra_simps power2_eq_square)
        done
    qed
    then show ∃f'. (g has_field_derivative f') (at z) ..
  qed
 qed (use geq in auto)
qed

end

proposition
 fixes S :: complex set
 assumes open S
 shows simply_connected_eq_continuous_log:
       simply_connected S ⟷
        connected S ∧
        (∀f::complex⇒complex. continuous_on S f ∧ (∀z ∈ S. f z ≠ 0)
          ⟶ (∃g. continuous_on S g ∧ (∀z ∈ S. f z = exp (g z)))) (is ?log)
   and simply_connected_eq_continuous_sqrt:
       simply_connected S ⟷
        connected S ∧
        (∀f::complex⇒complex. continuous_on S f ∧ (∀z ∈ S. f z ≠ 0)
          ⟶ (∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2))) (is ?sqrt)
proof −
 interpret SC_Chain
   using assms by (simp add: SC_Chain_def)
 have ?log ∧ ?sqrt
proof −
 have ∗: ⟦α ⟹ β; β ⟹ γ; γ ⟹ α⟧
         ⟹ (α ⟷ β) ∧ (α ⟷ γ) for α β γ
   by blast
 show ?thesis
   apply (rule ∗)
   apply (simp add: local.continuous_log winding_number_zero)
```

apply (*simp add*: *continuous_sqrt*)
apply (*simp add*: *continuous_sqrt_imp_simply_connected*)
done
qed
then show *?log ?sqrt*
by *safe*
qed

8.5 More Borsukian results

lemma *Borsukian_componentwise_eq*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *S*: *locally connected S* ∨ *compact S*
shows *Borsukian S* ⟷ (∀ *C* ∈ *components S*. *Borsukian C*)
proof −
have ∗: *ANR*(−{*0*::*complex*})
by (*simp add*: *ANR_delete open_Compl open_imp_ANR*)
show *?thesis*
using *cohomotopically_trivial_on_components* [*OF assms* ∗] **by** (*auto simp*: *Borsukian_alt*)
qed

lemma *Borsukian_componentwise*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *locally connected S* ∨ *compact S* ⋀*C*. *C* ∈ *components S* ⟹ *Borsukian C*
shows *Borsukian S*
by (*metis Borsukian_componentwise_eq assms*)

lemma *simply_connected_eq_Borsukian*:
fixes *S* :: *complex set*
shows *open S* ⟹ (*simply_connected S* ⟷ *connected S* ∧ *Borsukian S*)
by (*auto simp*: *simply_connected_eq_continuous_log Borsukian_continuous_logarithm*)

lemma *Borsukian_eq_simply_connected*:
fixes *S* :: *complex set*
shows *open S* ⟹ *Borsukian S* ⟷ (∀ *C* ∈ *components S*. *simply_connected C*)
by (*meson Borsukian_componentwise_eq in_components_connected open_components open_imp_locally_connected simply_connected_eq_Borsukian*)

lemma *Borsukian_separation_open_closed*:
fixes *S* :: *complex set*
assumes *S*: *open S* ∨ *closed S* **and** *bounded S*
shows *Borsukian S* ⟷ *connected*(− *S*)
using *S*
proof
assume *open S*
show *?thesis*
unfolding *Borsukian_eq_simply_connected* [*OF* ‹*open S*›]

by (*metis* ‹*open S*› ‹*bounded S*› *bounded_subset in_components_maximal non-separation_by_component_eq open_components simply_connected_iff_simple*)
next
assume *closed S*
with ‹*bounded S*› **show** *?thesis*
by (*simp add*: *Borsukian_separation_compact compact_eq_bounded_closed*)
qed

8.6 Finally, the Riemann Mapping Theorem

theorem *Riemann_mapping_theorem*:
open S ∧ *simply_connected S* ⟷
S = {} ∨ *S* = *UNIV* ∨
(∃*f g*. *f holomorphic_on S* ∧ *g holomorphic_on ball 0 1* ∧
(∀*z* ∈ *S*. *f z* ∈ *ball 0 1* ∧ *g*(*f z*) = *z*) ∧
(∀*z* ∈ *ball 0 1*. *g z* ∈ *S* ∧ *f*(*g z*) = *z*))
(**is** *_* = *?rhs*)
proof −
have *simply_connected S* ⟷ *?rhs* **if** *open S*
by (*simp add*: *simply_connected_eq_biholomorphic_to_disc that*)
moreover have *open S* **if** *?rhs*
proof −
{ fix *f g*
assume *g*: *g holomorphic_on ball 0 1* ∀*z*∈*ball 0 1*. *g z* ∈ *S* ∧ *f* (*g z*) = *z*
and ∀*z*∈*S*. *cmod* (*f z*) < *1* ∧ *g* (*f z*) = *z*
then have *S* = *g* ‘ (*ball 0 1*)
by (*force simp*:)
then have *open S*
by (*metis open_ball g inj_on_def open_mapping_thm3*)
}
with *that* **show** *open S* **by** *auto*
qed
ultimately show *?thesis* **by** *metis*
qed

8.7 Applications to Winding Numbers

lemma *simply_connected_inside_simple_path*:
fixes *p* :: *real* ⇒ *complex*
shows *simple_path p* ⟹ *simply_connected*(*inside*(*path_image p*))
using *Jordan_inside_outside connected_simple_path_image inside_simple_curve_imp_closed simply_connected_eq_frontier_properties*
by *fastforce*

lemma *simply_connected_Int*:
fixes *S* :: *complex set*
assumes *open S open T simply_connected S simply_connected T connected* (*S* ∩ *T*)
shows *simply_connected* (*S* ∩ *T*)
using *assms* **by** (*force simp*: *simply_connected_eq_winding_number_zero open_Int*)

8.8 The winding number defines a continuous logarithm for the path itself

lemma *winding_number_as_continuous_log*:
assumes *path p* **and** ζ: $\zeta \notin$ *path_image p*
obtains *q* **where** *path q*
*pathfinish q − pathstart q = 2 * of_real pi * i * winding_number p ζ*
$\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ t = \zeta + exp(q\ t)$
proof −
let *?q* = $\lambda t.$ *2 * of_real pi * i * winding_number(subpath 0 t p) ζ + Ln(pathstart p − ζ)*
show *?thesis*
proof
have $*$: *continuous (at t within {0..1}) (λx. winding_number (subpath 0 x p) ζ)*
if *t*: $t \in \{0..1\}$ **for** *t*
proof −
let *?B = ball (p t) (norm(p t − ζ))*
have *p t ≠ ζ*
using *path_image_def that ζ* **by** *blast*
then have *simply_connected ?B*
by (*simp add*: *convex_imp_simply_connected*)
then have $\bigwedge f$::*complex⇒complex. continuous_on ?B f ∧ (∀ζ ∈ ?B. f ζ ≠ 0)*
$\longrightarrow$ *(∃ g. continuous_on ?B g ∧ (∀ζ ∈ ?B. f ζ = exp (g ζ)))*
by (*simp add*: *simply_connected_eq_continuous_log*)
moreover have *continuous_on ?B (λw. w − ζ)*
by (*intro continuous_intros*)
moreover have *(∀ z ∈ ?B. z − ζ ≠ 0)*
by (*auto simp*: *dist_norm*)
ultimately obtain *g* **where** *contg*: *continuous_on ?B g*
and *geq*: $\bigwedge z.\ z \in$ *?B* $\Longrightarrow$ *z − ζ = exp (g z)* **by** *blast*
obtain *d* **where** *0 < d* **and** *d*:
$\bigwedge x.\ [\![x \in \{0..1\};$ *dist x t < d*$]\!] \Longrightarrow$ *dist (p x) (p t) < cmod (p t − ζ)*
using ‹*path p*› *t* **unfolding** *path_def continuous_on_iff*
by (*metis* ‹*p t ≠ ζ*› *right_minus_eq zero_less_norm_iff*)
have *((λx. winding_number (λw. subpath 0 x p w − ζ) 0 −*
winding_number (λw. subpath 0 t p w − ζ) 0) ⟶ 0)
(at t within {0..1})
proof (*rule Lim_transform_within* [*OF _* ‹*d > 0*›])
have *continuous (at t within {0..1}) (g o p)*
proof (*rule continuous_within_compose*)
show *continuous (at t within {0..1}) p*
using ‹*path p*› *continuous_on_eq_continuous_within path_def that* **by** *blast*
show *continuous (at (p t) within p ‘ {0..1}) g*
by (*metis* (*no_types*, *lifting*) *open_ball UNIV_I* ‹*p t ≠ ζ*› *centre_in_ball contg continuous_on_eq_continuous_at continuous_within_topological right_minus_eq zero_less_norm_iff*)
qed

```
      with LIM_zero have ((λu. (g (subpath t u p 1) − g (subpath t u p 0)))
⟶ 0) (at t within {0..1})
        by (auto simp: subpath_def continuous_within o_def)
      then show ((λu. (g (subpath t u p 1) − g (subpath t u p 0)) / (2 * of_real
pi * i)) ⟶ 0)
         (at t within {0..1})
        by (simp add: tendsto_divide_zero)
      show (g (subpath t u p 1) − g (subpath t u p 0)) / (2 * of_real pi * i) =
          winding_number (λw. subpath 0 u p w − ζ) 0 − winding_number (λw.
subpath 0 t p w − ζ) 0
        if u ∈ {0..1} 0 < dist u t dist u t < d for u
      proof −
        have closed_segment t u ⊆ {0..1}
          using closed_segment_eq_real_ivl t that by auto
        then have piB: path_image(subpath t u p) ⊆ ?B
          apply (clarsimp simp add: path_image_subpath_gen)
            by (metis subsetD le_less_trans ‹dist u t < d› d dist_commute
dist_in_closed_segment)
        have *: path (g ∘ subpath t u p)
          apply (rule path_continuous_image)
          using ‹path p› t that apply auto[1]
          using piB contg continuous_on_subset by blast
        have (g (subpath t u p 1) − g (subpath t u p 0)) / (2 * of_real pi * i)
           =  winding_number (exp ∘ g ∘ subpath t u p) 0
          using winding_number_compose_exp [OF *]
          by (simp add: pathfinish_def pathstart_def o_assoc)
        also have ... = winding_number (λw. subpath t u p w − ζ) 0
        proof (rule winding_number_cong)
          have exp(g y) = y − ζ if y ∈ (subpath t u p) ‘ {0..1} for y
            by (metis that geq path_image_def piB subset_eq)
         then show ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ (exp ∘ g ∘ subpath t u p) x = subpath
t u p x − ζ
            by auto
        qed
        also have ... = winding_number (λw. subpath 0 u p w − ζ) 0 −
                        winding_number (λw. subpath 0 t p w − ζ) 0
          apply (simp add: winding_number_offset [symmetric])
          using winding_number_subpath_combine [OF ‹path p› ζ, of 0 t u] ‹t ∈
{0..1}› ‹u ∈ {0..1}›
          by (simp add: add.commute eq_diff_eq)
        finally show ?thesis .
      qed
    qed
    then show ?thesis
    by (subst winding_number_offset) (simp add: continuous_within LIM_zero_iff)
  qed
  show path ?q
    unfolding path_def
  by (intro continuous_intros) (simp add: continuous_on_eq_continuous_within
```

*)

have $\zeta \neq p\ 0$
by (*metis* ζ *pathstart_def pathstart_in_path_image*)
then show *pathfinish ?q − pathstart ?q = 2 ∗ of_real pi ∗* i *∗ winding_number p* ζ
by (*simp add*: *pathfinish_def pathstart_def*)
show $p\ t = \zeta + exp\ (?q\ t)$ **if** $t \in \{0..1\}$ **for** t
proof −
have *path* (*subpath 0 t p*)
using ‹*path p*› *that* **by** *auto*
moreover
have $\zeta \notin$ *path_image* (*subpath 0 t p*)
using ζ [*unfolded path_image_def*] *that* **by** (*auto simp*: *path_image_subpath*)
ultimately show *?thesis*
using *winding_number_exp_2pi* [*of subpath 0 t p* ζ] ‹$\zeta \neq p\ 0$›
by (*auto simp*: *exp_add algebra_simps pathfinish_def pathstart_def subpath_def*)
qed
qed
qed

8.9 Winding number equality is the same as path/loop homotopy in C - 0

lemma *winding_number_homotopic_loops_null_eq*:
assumes *path p* **and** ζ: $\zeta \notin$ *path_image p*
shows *winding_number p* $\zeta = 0 \longleftrightarrow (\exists a.\ homotopic\_loops\ (-\{\zeta\})\ p\ (\lambda t.\ a))$
(**is** *?lhs = ?rhs*)
proof
assume [*simp*]: *?lhs*
obtain *q* **where** *path q*
and *qeq*: *pathfinish q − pathstart q = 2 ∗ of_real pi ∗* i *∗ winding_number p* ζ
and *peq*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ t = \zeta + exp(q\ t)$
using *winding_number_as_continuous_log* [*OF assms*] **by** *blast*
have ∗: *homotopic_with_canon* $(\lambda r.\ pathfinish\ r = pathstart\ r)$
$\{0..1\}$ $(-\{\zeta\})$ $((\lambda w.\ \zeta + exp\ w) \circ q)$ $((\lambda w.\ \zeta + exp\ w) \circ (\lambda t.\ 0))$
proof (*rule homotopic_with_compose_continuous_left*)
show *homotopic_with_canon* $(\lambda f.\ pathfinish\ ((\lambda w.\ \zeta + exp\ w) \circ f) = pathstart\ ((\lambda w.\ \zeta + exp\ w) \circ f))$
$\{0..1\}$ *UNIV q* $(\lambda t.\ 0)$
proof (*rule homotopic_with_mono*, *simp_all add*: *pathfinish_def pathstart_def*)
have *homotopic_loops UNIV q* $(\lambda t.\ 0)$
by (*rule homotopic_loops_linear*) (*use qeq* ‹*path q*› **in** ‹*auto simp*: *path_defs*›)
then have *homotopic_with* $(\lambda r.\ r\ 1 = r\ 0)$ (*top_of_set* $\{0..1\}$) *euclidean q* $(\lambda t.\ 0)$
by (*simp add*: *homotopic_loops_def pathfinish_def pathstart_def*)
then show *homotopic_with* $(\lambda h.\ exp\ (h\ 1) = exp\ (h\ 0))$ (*top_of_set* $\{0..1\}$)

euclidean q (*λt. 0*)
by (*rule homotopic_with_mono*) *simp*
qed
show *continuous_on UNIV* (*λw. ζ + exp w*)
by (*rule continuous_intros*)+
show (*λw. ζ + exp w*) ∈ *UNIV* → −{*ζ*}
by *auto*
qed
then have *homotopic_with_canon* (*λr. pathfinish r = pathstart r*) {*0..1*} (−{*ζ*}) *p* (*λx. ζ + 1*)
by (*rule homotopic_with_eq*) (*auto simp*: *o_def peq pathfinish_def pathstart_def*)
then have *homotopic_loops* (−{*ζ*}) *p* (*λt. ζ + 1*)
by (*simp add*: *homotopic_loops_def*)
then show *?rhs* **..**
next
assume *?rhs*
then obtain *a* **where** *homotopic_loops* (−{*ζ*}) *p* (*λt. a*) **..**
then have *winding_number p ζ = winding_number* (*λt. a*) *ζ a* ≠ *ζ*
using *winding_number_homotopic_loops homotopic_loops_imp_subset* **by** (*force simp*:)+
moreover have *winding_number* (*λt. a*) *ζ = 0*
by (*metis winding_number_zero_const* ‹*a* ≠ *ζ*›)
ultimately show *?lhs* **by** *metis*
qed

lemma *winding_number_homotopic_paths_null_explicit_eq*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
shows *winding_number p ζ = 0* ⟷ *homotopic_paths* (−{*ζ*}) *p* (*linepath* (*pathstart p*) (*pathstart p*))
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
using *homotopic_loops_imp_homotopic_paths_null*
by (*force simp add*: *linepath_refl winding_number_homotopic_loops_null_eq* [*OF assms*])
next
assume *?rhs*
then show *?lhs*
by (*metis ζ pathstart_in_path_image winding_number_homotopic_paths winding_number_trivial*)
qed

lemma *winding_number_homotopic_paths_null_eq*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
shows *winding_number p ζ = 0* ⟷ (∃ *a. homotopic_paths* (−{*ζ*}) *p* (*λt. a*))
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*

then show *?rhs*
by (*auto simp*: *winding_number_homotopic_paths_null_explicit_eq* [*OF assms*] *linepath_refl*)
next
assume *?rhs*
then show *?lhs*
by (*metis ζ homotopic_paths_imp_pathfinish pathfinish_def pathfinish_in_path_image winding_number_homotopic_paths winding_number_zero_const*)
qed

proposition *winding_number_homotopic_paths_eq*:
assumes *path p* **and** *ζp*: *ζ* ∉ *path_image p*
and *path q* **and** *ζq*: *ζ* ∉ *path_image q*
and *qp*: *pathstart q* = *pathstart p pathfinish q* = *pathfinish p*
shows *winding_number p ζ* = *winding_number q ζ* ⟷ *homotopic_paths* (−{*ζ*}) *p q*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *winding_number* (*p* +++ *reversepath q*) *ζ* = *0*
using *assms* **by** (*simp add*: *winding_number_join winding_number_reversepath*)
moreover
have *path* (*p* +++ *reversepath q*) *ζ* ∉ *path_image* (*p* +++ *reversepath q*)
using *assms* **by** (*auto simp*: *not_in_path_image_join*)
ultimately obtain *a* **where** *homotopic_paths* (− {*ζ*}) (*p* +++ *reversepath q*) (*linepath a a*)
using *winding_number_homotopic_paths_null_explicit_eq* **by** *blast*
then show *?rhs*
using *homotopic_paths_imp_pathstart assms*
by (*fastforce simp add*: *dest*: *homotopic_paths_imp_homotopic_loops homotopic_paths_loop_parts*)
qed (*simp add*: *winding_number_homotopic_paths*)

lemma *winding_number_homotopic_loops_eq*:
assumes *path p* **and** *ζp*: *ζ* ∉ *path_image p*
and *path q* **and** *ζq*: *ζ* ∉ *path_image q*
and *loops*: *pathfinish p* = *pathstart p pathfinish q* = *pathstart q*
shows *winding_number p ζ* = *winding_number q ζ* ⟷ *homotopic_loops* (−{*ζ*}) *p q*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *pathstart p* ≠ *ζ pathstart q* ≠ *ζ*
using *ζp ζq* **by** *blast+*
moreover have *path_connected* (−{*ζ*})
by (*simp add*: *path_connected_punctured_universe*)
ultimately obtain *r* **where** *path r* **and** *rim*: *path_image r* ⊆ −{*ζ*}
and *pas*: *pathstart r* = *pathstart p* **and** *paf*: *pathfinish r* = *pathstart q*

```
    by (auto simp: path_connected_def)
  then have pathstart r ≠ ζ by blast
  have homotopic_loops (− {ζ}) p (r +++ q +++ reversepath r)
  proof (rule homotopic_paths_imp_homotopic_loops)
    have path (r +++ q +++ reversepath r)
      by (simp add: ‹path r› ‹path q› loops paf)
    moreover have ζ ∉ path_image (r +++ q +++ reversepath r)
      by (metis ζq not_in_path_image_join path_image_reversepath rim sub-
set_Compl_singleton)
    moreover have homotopic_loops (− {ζ}) (r +++ q +++ reversepath r) q
      using ‹path q› ‹path r› ζq homotopic_loops_conjugate loops(2) paf rim by
blast
    ultimately show homotopic_paths (− {ζ}) p (r +++ q +++ reversepath r)
      using loops pathfinish_join pathfinish_reversepath pathstart_join
    by (metis L ζp ‹path p› pas winding_number_homotopic_loops winding_number_homotopic_paths_eq)
  qed (use loops pas in auto)
  moreover have homotopic_loops (− {ζ}) (r +++ q +++ reversepath r) q
    using rim ζq by (auto simp: homotopic_loops_conjugate paf ‹path q› ‹path r›
loops)
  ultimately show ?rhs
    using homotopic_loops_trans by metis
qed (simp add: winding_number_homotopic_loops)

end
theory Meromorphic
  imports Laurent_Convergence Riemann_Mapping
begin

lemma analytic_at_cong:
  assumes eventually (λx. f x = g x) (nhds x) x = y
  shows f analytic_on {x} ⟷ g analytic_on {y}
proof −
  have g analytic_on {x} if f analytic_on {x} eventually (λx. f x = g x) (nhds x)
for f g
  proof −
    have (λy. f (x + y)) has_fps_expansion fps_expansion f x
      by (rule analytic_at_imp_has_fps_expansion) fact
    also have ?this ⟷ (λy. g (x + y)) has_fps_expansion fps_expansion f x
      using that by (intro has_fps_expansion_cong refl) (auto simp: nhds_to_0'
eventually_filtermap)
    finally show ?thesis
      by (rule has_fps_expansion_imp_analytic)
  qed
  from this[of f g] this[of g f] show ?thesis using assms
    by (auto simp: eq_commute)
qed

definition remove_sings :: (complex ⇒ complex) ⇒ complex ⇒ complex where
  remove_sings f z = (if ∃ c. f −z→ c then Lim (at z) f else 0)
```

lemma *remove_sings_eqI* [*intro*]:
assumes $f \xrightarrow{z} c$
shows *remove_sings f z = c*
using *assms* **unfolding** *remove_sings_def* **by** (*auto simp*: *tendsto_Lim*)

lemma *remove_sings_at_analytic* [*simp*]:
assumes *f analytic_on* {*z*}
shows *remove_sings f z = f z*
using *assms* **by** (*intro remove_sings_eqI*) (*simp add*: *analytic_at_imp_isCont isContD*)

lemma *remove_sings_at_pole* [*simp*]:
assumes *is_pole f z*
shows *remove_sings f z = 0*
using *assms* **unfolding** *remove_sings_def is_pole_def*
by (*meson at_neq_bot not_tendsto_and_filterlim_at_infinity*)

lemma *eventually_remove_sings_eq_at*:
assumes *isolated_singularity_at f z*
shows *eventually* (λ*w*. *remove_sings f w = f w*) (*at z*)
proof −
from *assms* **obtain** *r* **where** *r*: *r > 0 f analytic_on ball z r − {z}*
by (*auto simp*: *isolated_singularity_at_def*)
hence ∗: *f analytic_on* {*w*} **if** *w ∈ ball z r − {z}* **for** *w*
using *r that* **by** (*auto intro*: *analytic_on_subset*)
have *eventually* (λ*w*. *w ∈ ball z r − {z}*) (*at z*)
using *r* **by** (*intro eventually_at_in_open*) *auto*
thus *?thesis*
by *eventually_elim* (*auto simp*: *remove_sings_at_analytic* ∗)
qed

lemma *eventually_remove_sings_eq_nhds*:
assumes *f analytic_on* {*z*}
shows *eventually* (λ*w*. *remove_sings f w = f w*) (*nhds z*)
proof −
from *assms* **obtain** *A* **where** *A*: *open A z ∈ A f holomorphic_on A*
by (*auto simp*: *analytic_at*)
have *eventually* (λ*z*. *z ∈ A*) (*nhds z*)
by (*intro eventually_nhds_in_open A*)
thus *?thesis*
proof *eventually_elim*
case (*elim w*)
from *elim* **have** *f analytic_on* {*w*}
using *A analytic_at* **by** *blast*
thus *?case* **by** *auto*
qed
qed

lemma *remove_sings_compose*:
assumes *filtermap g* (*at z*) = *at z′*
shows *remove_sings* (*f* ∘ *g*) *z* = *remove_sings f z′*
proof (*cases* ∃ *c*. *f* −*z′*→ *c*)
case *True*
then obtain *c* **where** *c*: *f* −*z′*→ *c*
by *auto*
from *c* **have** *remove_sings f z′* = *c*
by *blast*
moreover from *c* **have** *remove_sings* (*f* ∘ *g*) *z* = *c*
using *c* **by** (*intro remove_sings_eqI*) (*auto simp*: *filterlim_def filtermap_compose assms*)
ultimately show *?thesis*
by *simp*
next
case *False*
hence ¬(∃ *c*. (*f* ∘ *g*) −*z*→ *c*)
by (*auto simp*: *filterlim_def filtermap_compose assms*)
with *False* **show** *?thesis*
by (*auto simp*: *remove_sings_def*)
qed

lemma *remove_sings_cong*:
assumes *eventually* (λ*x*. *f x* = *g x*) (*at z*) *z* = *z′*
shows *remove_sings f z* = *remove_sings g z′*
proof (*cases* ∃ *c*. *f* −*z*→ *c*)
case *True*
then obtain *c* **where** *c*: *f* −*z*→ *c* **by** *blast*
hence *remove_sings f z* = *c*
by *blast*
moreover have *f* −*z*→ *c* ⟷ *g* −*z′*→ *c*
using *assms* **by** (*intro filterlim_cong refl*) *auto*
with *c* **have** *remove_sings g z′* = *c*
by (*intro remove_sings_eqI*) *auto*
ultimately show *?thesis*
by *simp*
next
case *False*
have *f* −*z*→ *c* ⟷ *g* −*z′*→ *c* **for** *c*
using *assms* **by** (*intro filterlim_cong*) *auto*
with *False* **show** *?thesis*
by (*auto simp*: *remove_sings_def*)
qed

lemma *deriv_remove_sings_at_analytic* [*simp*]:
assumes *f analytic_on* {*z*}
shows *deriv* (*remove_sings f*) *z* = *deriv f z*
apply (*rule deriv_cong_ev*)

apply (*rule eventually_remove_sings_eq_nhds*)
using *assms* **by** *auto*

lemma *isolated_singularity_at_remove_sings* [*simp*, *intro*]:
assumes *isolated_singularity_at f z*
shows *isolated_singularity_at* (*remove_sings f*) *z*
using *isolated_singularity_at_cong*[*OF eventually_remove_sings_eq_at*[*OF assms*] *refl*] *assms*
by *simp*

lemma *not_essential_remove_sings_iff* [*simp*]:
assumes *isolated_singularity_at f z*
shows *not_essential* (*remove_sings f*) *z* ⟷ *not_essential f z*
using *not_essential_cong*[*OF eventually_remove_sings_eq_at*[*OF assms*(*1*)] *refl*]
by *simp*

lemma *not_essential_remove_sings* [*intro*]:
assumes *isolated_singularity_at f z not_essential f z*
shows *not_essential* (*remove_sings f*) *z*
by (*subst not_essential_remove_sings_iff*) (*use assms* **in** *auto*)

lemma
assumes *isolated_singularity_at f z*
shows *is_pole_remove_sings_iff* [*simp*]:
is_pole (*remove_sings f*) *z* ⟷ *is_pole f z*
and *zorder_remove_sings* [*simp*]:
zorder (*remove_sings f*) *z* = *zorder f z*
and *zor_poly_remove_sings* [*simp*]:
zor_poly (*remove_sings f*) *z* = *zor_poly f z*
and *has_laurent_expansion_remove_sings_iff* [*simp*]:
(λ*w*. *remove_sings f* (*z* + *w*)) *has_laurent_expansion F* ⟷
(λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
and *tendsto_remove_sings_iff* [*simp*]:
remove_sings f −*z*→ *c* ⟷ *f* −*z*→ *c*
by (*intro is_pole_cong eventually_remove_sings_eq_at refl zorder_cong*
zor_poly_cong has_laurent_expansion_cong′ tendsto_cong assms)+

lemma *get_all_poles_from_remove_sings*:
fixes *f*:: *complex* ⇒ *complex*
defines *ff*≡*remove_sings f*
assumes *f_holo*:*f holomorphic_on s* − *pts* **and** *finite pts*
pts⊆*s open s* **and** *not_ess*:∀ *x*∈*pts*. *not_essential f x*
obtains *pts′* **where**
pts′ ⊆ *pts finite pts′ ff holomorphic_on s* − *pts′* ∀ *x*∈*pts′*. *is_pole ff x*
proof −
define *pts′* **where** *pts′* = {*x*∈*pts*. *is_pole f x*}

have *pts′* ⊆ *pts* **unfolding** *pts′_def* **by** *auto*

```
then have finite pts' using ‹finite pts›
  using rev_finite_subset by blast
then have open (s − pts') using ‹open s›
  by (simp add: finite_imp_closed open_Diff)

have isolated:isolated_singularity_at f z if z∈pts for z
proof (rule isolated_singularity_at_holomorphic)
  show f holomorphic_on (s−(pts−{z})) − {z}
    by (metis Diff_insert f_holo insert_Diff that)
  show  open (s − (pts − {z}))
    by (meson assms(3) assms(5) finite_Diff finite_imp_closed open_Diff)
  show z ∈ s − (pts − {z})
    using assms(4) that by auto
qed

have ff holomorphic_on s − pts'
proof (rule no_isolated_singularity')
  show (ff ⟶ ff z) (at z within s − pts') if z ∈ pts−pts' for z
  proof −
    have at z within s − pts' = at z
      apply (rule at_within_open)
      using ‹open (s − pts')› that ‹pts⊆s›  by auto
    moreover have ff −z→ ff z
      unfolding ff_def
    proof (subst tendsto_remove_sings_iff)
      show isolated_singularity_at f z
        apply (rule isolated)
        using that by auto
      have not_essential f z
        using not_ess that by auto
      moreover have ¬is_pole f z
        using that unfolding pts'_def by auto
      ultimately have ∃ c. f −z→ c
        unfolding not_essential_def by auto
      then show f −z→ remove_sings f z
        using remove_sings_eqI by blast
    qed
    ultimately show ?thesis by auto
  qed
  have ff holomorphic_on s − pts
    using f_holo
  proof (elim holomorphic_transform)
    fix x assume x ∈ s − pts
    then have f analytic_on {x}
      using assms(3) assms(5) f_holo
      by (meson finite_imp_closed
          holomorphic_on_imp_analytic_at open_Diff)
    from remove_sings_at_analytic[OF this]
    show f x = ff x unfolding ff_def by auto
```

```
    qed
    then show ff holomorphic_on s − pts' − (pts − pts')
      apply (elim holomorphic_on_subset)
      by blast
    show open (s − pts')
      by (simp add: ‹open (s − pts')›)
    show finite (pts − pts')
      by (simp add: assms(3))
  qed
  moreover have ∀x∈pts'. is_pole ff x
    unfolding pts'_def
    using ff_def is_pole_remove_sings_iff isolated by blast
  moreover note ‹pts' ⊆ pts› ‹finite pts'›
  ultimately show ?thesis using that by auto
qed

lemma remove_sings_eq_0_iff:
  assumes not_essential f w
  shows remove_sings f w = 0 ⟷ is_pole f w ∨ f −w→ 0
proof (cases is_pole f w)
  case True
  then show ?thesis by simp
next
  case False
  then obtain c where c:f −w→ c
    using ‹not_essential f w› unfolding not_essential_def by auto
  then show ?thesis
    using False remove_sings_eqI by auto
qed

definition meromorphic_on:: [complex ⇒ complex, complex set, complex set] ⇒
bool
  (_ (meromorphic'_on) _ _ [50,50,50]50) where
  f meromorphic_on D pts ≡
     open D ∧ pts ⊆ D ∧ (∀z∈pts. isolated_singularity_at f z ∧ not_essential f z)
∧
     (∀z∈D. ¬(z islimpt pts)) ∧ (f holomorphic_on D−pts)

lemma meromorphic_imp_holomorphic: f meromorphic_on D pts ⟹ f holomor-
phic_on (D − pts)
  unfolding meromorphic_on_def by auto

lemma meromorphic_imp_closedin_pts:
  assumes f meromorphic_on D pts
  shows closedin (top_of_set D) pts
  by (meson assms closedin_limpt meromorphic_on_def)

lemma meromorphic_imp_open_diff':
  assumes f meromorphic_on D pts pts' ⊆ pts
```

shows *open* (*D* − *pts'*)
proof −
have *D* − *pts'* = *D* − *closure pts'*
proof *safe*
fix *x* **assume** *x*: *x* ∈ *D x* ∈ *closure pts' x* ∉ *pts'*
hence *x islimpt pts'*
by (*subst islimpt_in_closure*) *auto*
hence *x islimpt pts*
by (*rule islimpt_subset*) *fact*
with *assms x* **show** *False*
by (*auto simp*: *meromorphic_on_def*)
qed (*use closure_subset* **in** *auto*)
then show *?thesis*
using *assms meromorphic_on_def* **by** *auto*
qed

lemma *meromorphic_imp_open_diff*: *f meromorphic_on D pts* ⟹ *open* (*D* − *pts*)
by (*erule meromorphic_imp_open_diff'*) *auto*

lemma *meromorphic_pole_subset*:
assumes *merf*: *f meromorphic_on D pts*
shows {*x*∈*D*. *is_pole f x*} ⊆ *pts*
by (*smt* (*verit*) *Diff_iff assms mem_Collect_eq meromorphic_imp_open_diff meromorphic_on_def not_is_pole_holomorphic subsetI*)

named_theorems *meromorphic_intros*

lemma *meromorphic_on_subset*:
assumes *f meromorphic_on A pts open B B* ⊆ *A pts'* = *pts* ∩ *B*
shows *f meromorphic_on B pts'*
unfolding *meromorphic_on_def*
proof (*intro ballI conjI*)
fix *z* **assume** *z* ∈ *B*
show ¬*z islimpt pts'*
proof
assume *z islimpt pts'*
hence *z islimpt pts*
by (*rule islimpt_subset*) (*use* ‹*pts'* = _› **in** *auto*)
thus *False* **using** ‹*z* ∈ *B*› ‹*B* ⊆ *A*› *assms*(*1*)
by (*auto simp*: *meromorphic_on_def*)
qed
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)

lemma *meromorphic_on_superset_pts*:
assumes *f meromorphic_on A pts pts* ⊆ *pts' pts'* ⊆ *A* ∀ *x*∈*A*. ¬*x islimpt pts'*
shows *f meromorphic_on A pts'*
unfolding *meromorphic_on_def*
proof (*intro conjI ballI impI*)

fix z **assume** $z \in pts'$
from *assms*(*1*) **have** *holo*: f *holomorphic_on* $A - pts$ **and** *open* A
unfolding *meromorphic_on_def* **by** *blast+*
have *open* $(A - pts)$
by (*intro meromorphic_imp_open_diff*[*OF assms*(*1*)])

show *isolated_singularity_at* $f\ z$
proof (*cases* $z \in pts$)
case *False*
thus *?thesis*
using ‹*open* $(A - pts)$› *assms* ‹$z \in pts'$›
by (*intro isolated_singularity_at_holomorphic*[*of* _ $A - pts$] *holomorphic_on_subset*[*OF holo*])
auto
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)

show *not_essential* $f\ z$
proof (*cases* $z \in pts$)
case *False*
thus *?thesis*
using ‹*open* $(A - pts)$› *assms* ‹$z \in pts'$›
by (*intro not_essential_holomorphic*[*of* _ $A - pts$] *holomorphic_on_subset*[*OF holo*])
auto
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)

lemma *meromorphic_on_no_singularities*: f *meromorphic_on* A $\{\} \longleftrightarrow f$ *holomorphic_on* $A \wedge$ *open* A
by (*auto simp*: *meromorphic_on_def*)

lemma *holomorphic_on_imp_meromorphic_on*:
f *holomorphic_on* $A \Longrightarrow pts \subseteq A \Longrightarrow$ *open* $A \Longrightarrow \forall x{\in}A.\ \neg x$ *islimpt* $pts \Longrightarrow f$ *meromorphic_on* $A\ pts$
by (*rule meromorphic_on_superset_pts*[**where** $pts = \{\}$])
(*auto simp*: *meromorphic_on_no_singularities*)

lemma *meromorphic_on_const* [*meromorphic_intros*]:
assumes *open* $A\ \forall x{\in}A.\ \neg x$ *islimpt* $pts\ pts \subseteq A$
shows $(\lambda\_.\ c)$ *meromorphic_on* $A\ pts$
by (*rule holomorphic_on_imp_meromorphic_on*) (*use assms* **in** *auto*)

lemma *meromorphic_on_ident* [*meromorphic_intros*]:
assumes *open* $A\ \forall x{\in}A.\ \neg x$ *islimpt* $pts\ pts \subseteq A$
shows $(\lambda x.\ x)$ *meromorphic_on* $A\ pts$
by (*rule holomorphic_on_imp_meromorphic_on*) (*use assms* **in** *auto*)

lemma *meromorphic_on_id* [*meromorphic_intros*]:
assumes *open* $A\ \forall x{\in}A.\ \neg x$ *islimpt* $pts\ pts \subseteq A$

shows *id meromorphic_on A pts*
using *meromorphic_on_ident assms* **unfolding** *id_def* **.**

lemma *not_essential_add* [*singularity_intros*]:
assumes *f_ness*: *not_essential f z* **and** *g_ness*: *not_essential g z*
assumes *f_iso*: *isolated_singularity_at f z* **and** *g_iso*: *isolated_singularity_at g z*
shows *not_essential* (λ*w*. *f w* + *g w*) *z*
proof −
have (λ*w*. *f* (*z* + *w*) + *g* (*z* + *w*)) *has_laurent_expansion laurent_expansion f z* + *laurent_expansion g z*
by (*intro not_essential_has_laurent_expansion laurent_expansion_intros assms*)
hence *not_essential* (λ*w*. *f* (*z* + *w*) + *g* (*z* + *w*)) *0*
using *has_laurent_expansion_not_essential_0* **by** *blast*
thus *?thesis*
by (*simp add*: *not_essential_shift_0*)
qed

lemma *meromorphic_on_uminus* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*z*. −*f z*) *meromorphic_on A pts*
unfolding *meromorphic_on_def*
by (*use assms* **in** ‹*auto simp*: *meromorphic_on_def intro*!: *holomorphic_intros singularity_intros*›)

lemma *meromorphic_on_add* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts g meromorphic_on A pts*
shows (λ*z*. *f z* + *g z*) *meromorphic_on A pts*
unfolding *meromorphic_on_def*
by (*use assms* **in** ‹*auto simp*: *meromorphic_on_def intro*!: *holomorphic_intros singularity_intros*›)

lemma *meromorphic_on_add′*:
assumes *f meromorphic_on A pts1 g meromorphic_on A pts2*
shows (λ*z*. *f z* + *g z*) *meromorphic_on A* (*pts1* ∪ *pts2*)
proof (*rule meromorphic_intros*)
show *f meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*1*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
show *g meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*2*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
qed

lemma *meromorphic_on_add_const* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*z*. *f z* + *c*) *meromorphic_on A pts*
unfolding *meromorphic_on_def*
by (*use assms* **in** ‹*auto simp*: *meromorphic_on_def intro*!: *holomorphic_intros*

singularity_intros›)

lemma *meromorphic_on_minus_const* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*z*. *f z* − *c*) *meromorphic_on A pts*
using *meromorphic_on_add_const*[*OF assms*,*of* −*c*] **by** *simp*

lemma *meromorphic_on_diff* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts g meromorphic_on A pts*
shows (λ*z*. *f z* − *g z*) *meromorphic_on A pts*
using *meromorphic_on_add*[*OF assms*(*1*) *meromorphic_on_uminus*[*OF assms*(*2*)]]
by *simp*

lemma *meromorphic_on_diff′*:
assumes *f meromorphic_on A pts1 g meromorphic_on A pts2*
shows (λ*z*. *f z* − *g z*) *meromorphic_on A* (*pts1* ∪ *pts2*)
proof (*rule meromorphic_intros*)
show *f meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*1*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
show *g meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*2*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
qed

lemma *meromorphic_on_mult* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts g meromorphic_on A pts*
shows (λ*z*. *f z* ∗ *g z*) *meromorphic_on A pts*
unfolding *meromorphic_on_def*
by (*use assms* **in** ‹*auto simp*: *meromorphic_on_def intro*!: *holomorphic_intros singularity_intros*›)

lemma *meromorphic_on_mult′*:
assumes *f meromorphic_on A pts1 g meromorphic_on A pts2*
shows (λ*z*. *f z* ∗ *g z*) *meromorphic_on A* (*pts1* ∪ *pts2*)
proof (*rule meromorphic_intros*)
show *f meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*1*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
show *g meromorphic_on A* (*pts1* ∪ *pts2*)
by (*rule meromorphic_on_superset_pts*[*OF assms*(*2*)])
(*use assms* **in** ‹*auto simp*: *meromorphic_on_def islimpt_Un*›)
qed

lemma *meromorphic_on_imp_not_essential*:
assumes *f meromorphic_on A pts z* ∈ *A*
shows *not_essential f z*

proof (*cases* $z \in pts$)
case *False*
thus *?thesis*
using *not_essential_holomorphic*[*of f A − pts z*] *meromorphic_imp_open_diff*[*OF assms*(*1*)] *assms*
by (*auto simp*: *meromorphic_on_def*)
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)

lemma *meromorphic_imp_analytic*: *f meromorphic_on D pts* ⟹ *f analytic_on* (*D − pts*)
unfolding *meromorphic_on_def*
apply (*subst analytic_on_open*)
using *meromorphic_imp_open_diff meromorphic_on_id* **apply** *blast*
apply *auto*
done

lemma *not_islimpt_isolated_zeros*:
assumes *mero*: *f meromorphic_on A pts* **and** $z \in A$
shows ¬*z islimpt* {*w*∈*A*. *isolated_zero f w*}
proof
assume *islimpt*: *z islimpt* {*w*∈*A*. *isolated_zero f w*}
have *holo*: *f holomorphic_on A − pts* **and** *open A*
using *assms* **by** (*auto simp*: *meromorphic_on_def*)
have *open'*: *open* (*A − pts* − {*z*}))
by (*intro meromorphic_imp_open_diff'*[*OF mero*]) *auto*
then obtain *r* **where** *r*: $r > 0$ *ball z r* ⊆ *A − pts* − {*z*})
using *meromorphic_imp_open_diff*[*OF mero*] ‹*z* ∈ *A*› *openE* **by** *blast*

have *not_essential f z*
using *assms* **by** (*rule meromorphic_on_imp_not_essential*)
then consider *c* **where** *f* −*z*→ *c* | *is_pole f z*
unfolding *not_essential_def* **by** *blast*
thus *False*
proof *cases*
assume *is_pole f z*
hence *eventually* (λ*w*. *f w* ≠ *0*) (*at z*)
by (*rule non_zero_neighbour_pole*)
hence ¬*z islimpt* {*w*. *f w* = *0*}
by (*simp add*: *islimpt_conv_frequently_at frequently_def*)
moreover have *z islimpt* {*w*. *f w* = *0*}
using *islimpt* **by** (*rule islimpt_subset*) (*auto simp*: *isolated_zero_def*)
ultimately show *False* **by** *contradiction*
next
fix *c* **assume** *c*: *f* −*z*→ *c*
define *g* **where** *g* = (λ*w*. *if w* = *z then c else f w*)
have *holo'*: *g holomorphic_on A* − (*pts* − {*z*}) **unfolding** *g_def*
by (*intro removable_singularity holomorphic_on_subset*[*OF holo*] *open' c*) *auto*

```
    have eq_zero: g w = 0 if w ∈ ball z r for w
    proof (rule analytic_continuation[where f = g])
      show open (ball z r) connected (ball z r) {w∈ball z r. isolated_zero f w} ⊆
ball z r
        by auto
      have z islimpt {w∈A. isolated_zero f w} ∩ ball z r
      using islimpt ‹r > 0› by (intro islimpt_Int_eventually eventually_at_in_open′)
auto
      also have ... = {w∈ball z r. isolated_zero f w}
        using r by auto
      finally show z islimpt {w∈ball z r. isolated_zero f w}
        by simp
    next
      fix w assume w: w ∈ {w∈ball z r. isolated_zero f w}
      show g w = 0
      proof (cases w = z)
        case False
        thus ?thesis using w by (auto simp: g_def isolated_zero_def)
      next
        case True
        have z islimpt {z. f z = 0}
          using islimpt by (rule islimpt_subset) (auto simp: isolated_zero_def)
        thus ?thesis
          using w by (simp add: isolated_zero_altdef True)
      qed
    qed (use r that in ‹auto intro!: holomorphic_on_subset[OF holo′] simp: iso-
lated_zero_def›)

    have infinite ({w∈A. isolated_zero f w} ∩ ball z r)
      using islimpt ‹r > 0› unfolding islimpt_eq_infinite_ball by blast
    hence {w∈A. isolated_zero f w} ∩ ball z r ≠ {}
      by force
    then obtain z0 where z0: z0 ∈ A isolated_zero f z0 z0 ∈ ball z r
      by blast
    have ∀F y in at z0. y ∈ ball z r − (if z = z0 then {} else {z}) − {z0}
      using r z0 by (intro eventually_at_in_open) auto
    hence eventually (λw. f w = 0) (at z0)
    proof eventually_elim
      case (elim w)
      show ?case
        using eq_zero[of w] elim by (auto simp: g_def split: if_splits)
    qed
    hence eventually (λw. f w = 0) (at z0)
        by (auto simp: g_def eventually_at_filter elim!: eventually_mono split:
if_splits)
    moreover from z0 have eventually (λw. f w ≠ 0) (at z0)
      by (simp add: isolated_zero_def)
    ultimately have eventually (λ_. False) (at z0)
      by eventually_elim auto
```

```
    thus False
      by simp
  qed
qed

lemma closedin_isolated_zeros:
  assumes f meromorphic_on A pts
  shows   closedin (top_of_set A) {z∈A. isolated_zero f z}
  unfolding closedin_limpt using not_islimpt_isolated_zeros[OF assms] by auto

lemma meromorphic_on_deriv':
  assumes f meromorphic_on A pts open A
  assumes ⋀x. x ∈ A - pts ⟹ (f has_field_derivative f' x) (at x)
  shows   f' meromorphic_on A pts
  unfolding meromorphic_on_def
proof (intro conjI ballI)
  have open (A - pts)
    by (intro meromorphic_imp_open_diff[OF assms(1)])
  thus f' holomorphic_on A - pts
    by (rule derivative_is_holomorphic) (use assms in auto)
next
  fix z assume z ∈ pts
  hence z ∈ A
    using assms(1) by (auto simp: meromorphic_on_def)
  from ‹z ∈ pts› obtain r where r: r > 0 f analytic_on ball z r - {z}
   using assms(1) by (auto simp: meromorphic_on_def isolated_singularity_at_def)

  have open (ball z r ∩ (A - (pts - {z})))
    by (intro open_Int assms meromorphic_imp_open_diff'[OF assms(1)]) auto
  then obtain r' where r': r' > 0 ball z r' ⊆ ball z r ∩ (A - (pts - {z}))
    using r ‹z ∈ A› by (subst (asm) open_contains_ball) fastforce

  have open (ball z r' - {z})
    by auto
  hence f' holomorphic_on ball z r' - {z}
    by (rule derivative_is_holomorphic[of _ f]) (use r' in ‹auto intro!: assms(3)›)
  moreover have open (ball z r' - {z})
    by auto
  ultimately show isolated_singularity_at f' z
    unfolding isolated_singularity_at_def using ‹r' > 0›
    by (auto simp: analytic_on_open intro!: exI[of _ r'])
next
  fix z assume z: z ∈ pts
  hence z': not_essential f z z ∈ A
    using assms by (auto simp: meromorphic_on_def)
  from z'(1) show not_essential f' z
  proof (rule not_essential_deriv')
    show z ∈ A - (pts - {z})
      using ‹z ∈ A› by blast
```

```
    show open (A − (pts − {z}))
      by (intro meromorphic_imp_open_diff'[OF assms(1)]) auto
  qed (use assms in auto)
qed (use assms in ‹auto simp: meromorphic_on_def›)

lemma meromorphic_on_deriv [meromorphic_intros]:
  assumes f meromorphic_on A pts open A
  shows   deriv f meromorphic_on A pts
proof (intro meromorphic_on_deriv'[OF assms(1)])
  have ∗: open (A − pts)
    by (intro meromorphic_imp_open_diff[OF assms(1)])
  show (f has_field_derivative deriv f x) (at x) if x ∈ A − pts for x
    using assms(1) by (intro holomorphic_derivI[OF _ ∗ that]) (auto simp: mero-
morphic_on_def)
qed fact

lemma meromorphic_on_imp_analytic_at:
  assumes f meromorphic_on A pts z ∈ A − pts
  shows   f analytic_on {z}
 using assms by (metis analytic_at meromorphic_imp_open_diff meromorphic_on_def)

lemma meromorphic_compact_finite_pts:
  assumes f meromorphic_on D pts compact S S ⊆ D
  shows finite (S ∩ pts)
proof −
  { assume infinite (S ∩ pts)
    then obtain z where z ∈ S and z: z islimpt (S ∩ pts)
      using assms by (metis compact_eq_Bolzano_Weierstrass inf_le1)
    then have False
      using assms by (meson in_mono inf_le2 islimpt_subset meromorphic_on_def)
}
  then show ?thesis by metis
qed

lemma meromorphic_imp_countable:
  assumes f meromorphic_on D pts
  shows countable pts
proof −
  obtain K :: nat ⇒ complex set where K: D = (⋃n. K n) ⋀n. compact (K n)
   using assms unfolding meromorphic_on_def by (metis open_Union_compact_subsets)
  then have pts = (⋃n. K n ∩ pts)
    using assms meromorphic_on_def by auto
  moreover have ⋀n. finite (K n ∩ pts)
   by (metis K(1) K(2) UN_I assms image_iff meromorphic_compact_finite_pts
rangeI subset_eq)
  ultimately show ?thesis
    by (metis countableI_type countable_UN countable_finite)
qed
```

lemma *meromorphic_imp_connected_diff'*:
assumes *f meromorphic_on D pts connected D pts'* ⊆ *pts*
shows *connected* (*D* − *pts'*)
proof (*rule connected_open_diff_countable*)
show *countable pts'*
by (*rule countable_subset* [*OF assms*(*3*)]) (*use assms*(*1*) **in** ‹*auto simp*: *meromorphic_imp_countable*›)
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)

lemma *meromorphic_imp_connected_diff*:
assumes *f meromorphic_on D pts connected D*
shows *connected* (*D* − *pts*)
using *meromorphic_imp_connected_diff'*[*OF assms order.refl*] **.**

lemma *meromorphic_on_compose* [*meromorphic_intros*]:
assumes *f*: *f meromorphic_on A pts* **and** *g*: *g holomorphic_on B*
assumes *open B* **and** *g* ‘ *B* ⊆ *A*
shows (λ*x*. *f* (*g x*)) *meromorphic_on B* (*isolated_points_of* (*g* −‘ *pts* ∩ *B*))
unfolding *meromorphic_on_def*
proof (*intro ballI conjI*)
fix *z* **assume** *z*: *z* ∈ *isolated_points_of* (*g* −‘ *pts* ∩ *B*)
hence *z'*: *z* ∈ *B g z* ∈ *pts*
using *isolated_points_of_subset* **by** *blast*+
have *g'*: *g analytic_on* {*z*}
using *g z'* ‹*open B*› *analytic_at* **by** *blast*

show *isolated_singularity_at* (λ*x*. *f* (*g x*)) *z*
by (*rule isolated_singularity_at_compose*[*OF _ g'*]) (*use f z'* **in** ‹*auto simp*: *meromorphic_on_def*›)
show *not_essential* (λ*x*. *f* (*g x*)) *z*
by (*rule not_essential_compose*[*OF _ g'*]) (*use f z'* **in** ‹*auto simp*: *meromorphic_on_def*›)
next
fix *z* **assume** *z*: *z* ∈ *B*
hence *g z* ∈ *A*
using *assms* **by** *auto*
hence ¬*g z islimpt pts*
using *f* **by** (*auto simp*: *meromorphic_on_def*)
hence *ev*: *eventually* (λ*w*. *w* ∉ *pts*) (*at* (*g z*))
by (*auto simp*: *islimpt_conv_frequently_at frequently_def*)
have *g'*: *g analytic_on* {*z*}
by (*rule holomorphic_on_imp_analytic_at*[*OF g*]) (*use assms z* **in** *auto*)

have *eventually* (λ*w*. *w* ∉ *isolated_points_of* (*g* −‘ *pts* ∩ *B*)) (*at z*)
proof (*cases isolated_zero* (λ*w*. *g w* − *g z*) *z*)
case *True*
have *eventually* (λ*w*. *w* ∉ *pts*) (*at* (*g z*))
using *ev* **by** (*auto simp*: *islimpt_conv_frequently_at frequently_def*)

```
    moreover have g −z→ g z
      using analytic_at_imp_isCont[OF g'] isContD by blast
    hence lim: filterlim g (at (g z)) (at z)
      using True by (auto simp: filterlim_at isolated_zero_def)
    have eventually (λw. g w ∉ pts) (at z)
      using ev lim by (rule eventually_compose_filterlim)
    thus ?thesis
      by eventually_elim (auto simp: isolated_points_of_def)
  next
    case False
    have eventually (λw. g w − g z = 0) (nhds z)
      using False by (rule non_isolated_zero) (auto intro!: analytic_intros g')
    hence eventually (λw. g w = g z ∧ w ∈ B) (nhds z)
      using eventually_nhds_in_open[OF ‹open B› ‹z ∈ B›]
      by eventually_elim auto
    then obtain X where X: open X z ∈ X X ⊆ B ∀x∈X. g x = g z
      unfolding eventually_nhds by blast

    have z0 ∉ isolated_points_of (g −‘ pts ∩ B) if z0 ∈ X for z0
    proof (cases g z ∈ pts)
      case False
      with that have g z0 ∉ pts
        using X by metis
      thus ?thesis
        by (auto simp: isolated_points_of_def)
    next
      case True
      have eventually (λw. w ∈ X) (at z0)
        by (intro eventually_at_in_open') fact+
      hence eventually (λw. w ∈ g −‘ pts ∩ B) (at z0)
        by eventually_elim (use X True in fastforce)
      hence frequently (λw. w ∈ g −‘ pts ∩ B) (at z0)
        by (meson at_neq_bot eventually_frequently)
      thus z0 ∉ isolated_points_of (g −‘ pts ∩ B)
        unfolding isolated_points_of_def by (auto simp: frequently_def)
    qed
    moreover have eventually (λx. x ∈ X) (at z)
      by (intro eventually_at_in_open') fact+
    ultimately show ?thesis
      by (auto elim!: eventually_mono)
  qed
  thus ¬z islimpt isolated_points_of (g −‘ pts ∩ B)
    by (auto simp: islimpt_conv_frequently_at frequently_def)
next
  have f ∘ g analytic_on (⋃z∈B − isolated_points_of (g −‘ pts ∩ B). {z})
    unfolding analytic_on_UN
  proof
    fix z assume z: z ∈ B − isolated_points_of (g −‘ pts ∩ B)
    hence z ∈ B by blast
```

```
    have g': g analytic_on {z}
      by (rule holomorphic_on_imp_analytic_at[OF g]) (use assms z in auto)
    show f ∘ g analytic_on {z}
    proof (cases g z ∈ pts)
      case False
      show ?thesis
      proof (rule analytic_on_compose)
        show f analytic_on g ` {z} using False z assms
          by (auto intro!: meromorphic_on_imp_analytic_at[OF f])
      qed fact
    next
      case True
      show ?thesis
      proof (cases isolated_zero (λw. g w - g z) z)
        case False
        hence eventually (λw. g w - g z = 0) (nhds z)
          by (rule non_isolated_zero) (auto intro!: analytic_intros g')
        hence f ∘ g analytic_on {z} ⟷ (λ_. f (g z)) analytic_on {z}
          by (intro analytic_at_cong) (auto elim!: eventually_mono)
        thus ?thesis
          by simp
      next
        case True
        hence ev: eventually (λw. g w ≠ g z) (at z)
          by (auto simp: isolated_zero_def)

        have ¬g z islimpt pts
          using ‹g z ∈ pts› f by (auto simp: meromorphic_on_def)
        hence eventually (λw. w ∉ pts) (at (g z))
          by (auto simp: islimpt_conv_frequently_at frequently_def)
        moreover have g -z→ g z
          using analytic_at_imp_isCont[OF g'] isContD by blast
        with ev have filterlim g (at (g z)) (at z)
          by (auto simp: filterlim_at)
        ultimately have eventually (λw. g w ∉ pts) (at z)
          using eventually_compose_filterlim by blast
        hence z ∈ isolated_points_of (g -` pts ∩ B)
          using ‹g z ∈ pts› ‹z ∈ B›
          by (auto simp: isolated_points_of_def elim!: eventually_mono)
        with z show ?thesis by simp
      qed
    qed
  qed
  also have ... = B - isolated_points_of (g -` pts ∩ B)
    by blast
  finally show (λx. f (g x)) holomorphic_on B - isolated_points_of (g -` pts ∩
B)
    unfolding o_def using analytic_imp_holomorphic by blast
qed (auto simp: isolated_points_of_def ‹open B›)
```

lemma *meromorphic_on_compose'*:
assumes *f*: *f meromorphic_on A pts* **and** *g*: *g holomorphic_on B*
assumes *open B* **and** *g ‘ B ⊆ A* **and** *pts' = (isolated_points_of (g −‘ pts ∩ B))*
shows *(λx. f (g x)) meromorphic_on B pts'*
using *meromorphic_on_compose*[*OF assms*(*1−4*)] *assms*(*5*) **by** *simp*

lemma *meromorphic_on_inverse'*: *inverse meromorphic_on UNIV 0*
unfolding *meromorphic_on_def*
by (*auto intro*!: *holomorphic_intros singularity_intros not_essential_inverse isolated_singularity_at_inverse simp*: *islimpt_finite*)

lemma *meromorphic_on_inverse* [*meromorphic_intros*]:
assumes *mero*: *f meromorphic_on A pts*
shows *(λz. inverse (f z)) meromorphic_on A (pts ∪ {z∈A. isolated_zero f z})*
proof −
have *open A*
using *mero* **by** (*auto simp*: *meromorphic_on_def*)
have *open'*: *open (A − pts)*
by (*intro meromorphic_imp_open_diff*[*OF mero*])
have *holo*: *f holomorphic_on A − pts*
using *assms* **by** (*auto simp*: *meromorphic_on_def*)
have *ana*: *f analytic_on A − pts*
using *open' holo* **by** (*simp add*: *analytic_on_open*)

show *?thesis*
unfolding *meromorphic_on_def*
proof (*intro conjI ballI*)
fix *z* **assume** *z*: *z ∈ pts ∪ {z∈A. isolated_zero f z}*
have *isolated_singularity_at f z ∧ not_essential f z*
proof (*cases z ∈ pts*)
case *False*
have *f holomorphic_on A − pts − {z}*
by (*intro holomorphic_on_subset*[*OF holo*]) *auto*
hence *isolated_singularity_at f z*
by (*rule isolated_singularity_at_holomorphic*)
(*use z False* **in** ‹*auto intro*!: *meromorphic_imp_open_diff*[*OF mero*]›)
moreover have *not_essential f z*
using *z False*
by (*intro not_essential_holomorphic*[*OF holo*] *meromorphic_imp_open_diff*[*OF mero*]) *auto*
ultimately show *?thesis* **by** *blast*
qed (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)
thus *isolated_singularity_at (λz. inverse (f z)) z not_essential (λz. inverse (f z)) z*
by (*auto intro*!: *isolated_singularity_at_inverse not_essential_inverse*)
next
fix *z* **assume** *z ∈ A*
hence *¬ z islimpt {z∈A. isolated_zero f z}*

```
    by (rule not_islimpt_isolated_zeros[OF mero])
  thus ¬ z islimpt pts ∪ {z ∈ A. isolated_zero f z} using ‹z ∈ A›
    using mero by (auto simp: islimpt_Un meromorphic_on_def)
 next
  show pts ∪ {z ∈ A. isolated_zero f z} ⊆ A
    using mero by (auto simp: meromorphic_on_def)
 next
  have (λz. inverse (f z)) analytic_on (⋃w∈A − (pts ∪ {z ∈ A. isolated_zero f
z}) . {w})
    unfolding analytic_on_UN
  proof (intro ballI)
    fix w assume w: w ∈ A − (pts ∪ {z ∈ A. isolated_zero f z})
    show (λz. inverse (f z)) analytic_on {w}
    proof (cases f w = 0)
      case False
      thus ?thesis using w
        by (intro analytic_intros analytic_on_subset[OF ana]) auto
    next
      case True
      have eventually (λw. f w = 0) (nhds w)
        using True w by (intro non_isolated_zero analytic_on_subset[OF ana])
auto
      hence (λz. inverse (f z)) analytic_on {w} ⟷ (λ_. 0) analytic_on {w}
        using w by (intro analytic_at_cong refl) auto
      thus ?thesis
        by simp
    qed
  qed
  also have ... = A − (pts ∪ {z ∈ A. isolated_zero f z})
    by blast
  finally have (λz. inverse (f z)) analytic_on ... .
  moreover have open (A − (pts ∪ {z ∈ A. isolated_zero f z}))
    using closedin_isolated_zeros[OF mero] open' ‹open A›
   by (metis (no_types, lifting) Diff_Diff_Int Diff_Un closedin_closed open_Diff
open_Int)
    ultimately show (λz. inverse (f z)) holomorphic_on A − (pts ∪ {z ∈ A.
isolated_zero f z})
    by (subst (asm) analytic_on_open) auto
 qed (use assms in ‹auto simp: meromorphic_on_def islimpt_Un
                       intro!: holomorphic_intros singularity_intros›)
qed

lemma meromorphic_on_inverse'' [meromorphic_intros]:
 assumes f meromorphic_on A pts {z∈A. f z = 0} ⊆ pts
 shows   (λz. inverse (f z)) meromorphic_on A pts
proof −
 have (λz. inverse (f z)) meromorphic_on A (pts ∪ {z ∈ A. isolated_zero f z})
   by (intro meromorphic_on_inverse assms)
 also have (pts ∪ {z ∈ A. isolated_zero f z}) = pts
```

using *assms*(*2*) **by** (*auto simp*: *isolated_zero_def*)
finally show *?thesis* **.**
qed

lemma *meromorphic_on_divide* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts* **and** *g meromorphic_on A pts*
shows (λ*z*. *f z* / *g z*) *meromorphic_on A* (*pts* ∪ {*z*∈*A*. *isolated_zero g z*})
proof −
have *mero1*: (λ*z*. *inverse* (*g z*)) *meromorphic_on A* (*pts* ∪ {*z*∈*A*. *isolated_zero g z*})
by (*intro meromorphic_intros assms*)
have *sparse*: ∀ *x*∈*A*. ¬ *x islimpt pts* ∪ {*z*∈*A*. *isolated_zero g z*} **and** *pts* ⊆ *A*
using *mero1* **by** (*auto simp*: *meromorphic_on_def*)
have *mero2*: *f meromorphic_on A* (*pts* ∪ {*z*∈*A*. *isolated_zero g z*})
by (*rule meromorphic_on_superset_pts*[*OF assms*(*1*)]) (*use sparse* ‹*pts* ⊆ *A*› **in** *auto*)
have (λ*z*. *f z* ∗ *inverse* (*g z*)) *meromorphic_on A* (*pts* ∪ {*z*∈*A*. *isolated_zero g z*})
by (*intro meromorphic_on_mult mero1 mero2*)
thus *?thesis*
by (*simp add*: *field_simps*)
qed

lemma *meromorphic_on_divide′* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts g meromorphic_on A pts* {*z*∈*A*. *g z* = *0*} ⊆ *pts*
shows (λ*z*. *f z* / *g z*) *meromorphic_on A pts*
proof −
have (λ*z*. *f z* ∗ *inverse* (*g z*)) *meromorphic_on A pts*
by (*intro meromorphic_intros assms*)
thus *?thesis*
by (*simp add*: *field_simps*)
qed

lemma *meromorphic_on_cmult_left* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*x*. *c* ∗ *f x*) *meromorphic_on A pts*
using *assms* **by** (*intro meromorphic_intros*) (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_cmult_right* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*x*. *f x* ∗ *c*) *meromorphic_on A pts*
using *assms* **by** (*intro meromorphic_intros*) (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_scaleR* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*x*. *c* ∗*R* *f x*) *meromorphic_on A pts*
using *assms* **unfolding** *scaleR_conv_of_real*
by (*intro meromorphic_intros*) (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_sum* [*meromorphic_intros*]:
assumes $\bigwedge y.\ y \in I \Longrightarrow f\ y$ *meromorphic_on* A *pts*
assumes $I \neq \{\} \vee$ *open* $A \wedge$ *pts* $\subseteq A \wedge (\forall x{\in}A.\ \neg x$ *islimpt pts*$)$
shows $(\lambda x.\ \sum y{\in}I.\ f\ y\ x)$ *meromorphic_on* A *pts*
proof −
have $*$: *open* $A \wedge$ *pts* $\subseteq A \wedge (\forall x{\in}A.\ \neg x$ *islimpt pts*$)$
using *assms*(*2*)
proof
assume $I \neq \{\}$
then obtain x **where** $x \in I$
by *blast*
from *assms*(*1*)[*OF this*] **show** *?thesis*
by (*auto simp*: *meromorphic_on_def*)
qed *auto*
show *?thesis*
using *assms*(*1*)
by (*induction I rule*: *infinite_finite_induct*) (*use* $*$ **in** ‹*auto intro*!: *meromorphic_intros*›)
qed

lemma *meromorphic_on_prod* [*meromorphic_intros*]:
assumes $\bigwedge y.\ y \in I \Longrightarrow f\ y$ *meromorphic_on* A *pts*
assumes $I \neq \{\} \vee$ *open* $A \wedge$ *pts* $\subseteq A \wedge (\forall x{\in}A.\ \neg x$ *islimpt pts*$)$
shows $(\lambda x.\ \prod y{\in}I.\ f\ y\ x)$ *meromorphic_on* A *pts*
proof −
have $*$: *open* $A \wedge$ *pts* $\subseteq A \wedge (\forall x{\in}A.\ \neg x$ *islimpt pts*$)$
using *assms*(*2*)
proof
assume $I \neq \{\}$
then obtain x **where** $x \in I$
by *blast*
from *assms*(*1*)[*OF this*] **show** *?thesis*
by (*auto simp*: *meromorphic_on_def*)
qed *auto*
show *?thesis*
using *assms*(*1*)
by (*induction I rule*: *infinite_finite_induct*) (*use* $*$ **in** ‹*auto intro*!: *meromorphic_intros*›)
qed

lemma *meromorphic_on_power* [*meromorphic_intros*]:
assumes f *meromorphic_on* A *pts*
shows $(\lambda x.\ f\ x\ \hat{}\ n)$ *meromorphic_on* A *pts*
proof −
have $(\lambda x.\ \prod i{\in}\{..{<}n\}.\ f\ x)$ *meromorphic_on* A *pts*
by (*intro meromorphic_intros assms*(*1*)) (*use assms* **in** ‹*auto simp*: *meromorphic_on_def*›)
thus *?thesis*

by *simp*
qed

lemma *meromorphic_on_power_int* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts*
shows (λ*z*. *f z powi n*) *meromorphic_on A* (*pts* ∪ {*z* ∈ *A*. *isolated_zero f z*})
proof −
have *inv*: (λ*x*. *inverse* (*f x*)) *meromorphic_on A* (*pts* ∪ {*z* ∈ *A*. *isolated_zero f z*})
by (*intro meromorphic_intros assms*)
have ∗: *f meromorphic_on A* (*pts* ∪ {*z* ∈ *A*. *isolated_zero f z*})
by (*intro meromorphic_on_superset_pts* [*OF assms*(*1*)])
(*use inv* **in** ‹*auto simp*: *meromorphic_on_def*›)
show *?thesis*
proof (*cases n* ≥ *0*)
case *True*
have (λ*x*. *f x* ^ *nat n*) *meromorphic_on A* (*pts* ∪ {*z* ∈ *A*. *isolated_zero f z*})
by (*intro meromorphic_intros* ∗)
thus *?thesis*
using *True* **by** (*simp add*: *power_int_def*)
next
case *False*
have (λ*x*. *inverse* (*f x*) ^ *nat* (−*n*)) *meromorphic_on A* (*pts* ∪ {*z* ∈ *A*. *isolated_zero f z*})
by (*intro meromorphic_intros assms*)
thus *?thesis*
using *False* **by** (*simp add*: *power_int_def*)
qed
qed

lemma *meromorphic_on_power_int′* [*meromorphic_intros*]:
assumes *f meromorphic_on A pts n* ≥ *0* ∨ (∀ *z*∈*A*. *isolated_zero f z* ⟶ *z* ∈ *pts*)
shows (λ*z*. *f z powi n*) *meromorphic_on A pts*
proof (*cases n* ≥ *0*)
case *True*
have (λ*z*. *f z* ^ *nat n*) *meromorphic_on A pts*
by (*intro meromorphic_intros assms*)
thus *?thesis*
using *True* **by** (*simp add*: *power_int_def*)
next
case *False*
have (λ*z*. *f z powi n*) *meromorphic_on A* (*pts* ∪ {*z*∈*A*. *isolated_zero f z*})
by (*rule meromorphic_on_power_int*) *fact*
also from *assms*(*2*) *False* **have** *pts* ∪ {*z*∈*A*. *isolated_zero f z*} = *pts*
by *auto*
finally show *?thesis* **.**
qed

lemma *has_laurent_expansion_on_imp_meromorphic_on*:
assumes *open A*
assumes *laurent*: $\bigwedge z.\ z \in A \Longrightarrow \exists F.\ (\lambda w.\ f\ (z + w))$ *has_laurent_expansion F*
shows f *meromorphic_on A* $\{z{\in}A.\ \neg f$ *analytic_on* $\{z\}\}$
unfolding *meromorphic_on_def*
proof (*intro conjI ballI*)
fix z **assume** $z \in \{z{\in}A.\ \neg f$ *analytic_on* $\{z\}\}$
then obtain F **where** F: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion F*
using *laurent*[*of z*] **by** *blast*
from F **show** *not_essential f z isolated_singularity_at f z*
using *has_laurent_expansion_not_essential has_laurent_expansion_isolated*
by *blast+*
next
fix z **assume** z: $z \in A$
obtain F **where** F: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion F*
using *laurent*[*of z*] ‹$z \in A$› **by** *blast*
from F **have** *isolated_singularity_at f z*
using *has_laurent_expansion_isolated z* **by** *blast*
then obtain r **where** r: $r > 0$ f *analytic_on ball z r* $-$ $\{z\}$
unfolding *isolated_singularity_at_def* **by** *blast*
have f *analytic_on* $\{w\}$ **if** $w \in$ *ball z r* $-$ $\{z\}$ **for** w
by (*rule analytic_on_subset*[*OF r*(*2*)]) (*use that* **in** *auto*)
hence *eventually* $(\lambda w.\ f$ *analytic_on* $\{w\})$ (*at z*)
using *eventually_at_in_open*[*of ball z r z*] ‹$r > 0$› **by** (*auto elim*!: *eventually_mono*)
hence $\neg z$ *islimpt* $\{w.\ \neg f$ *analytic_on* $\{w\}\}$
by (*auto simp*: *islimpt_conv_frequently_at frequently_def*)
thus $\neg z$ *islimpt* $\{w{\in}A.\ \neg f$ *analytic_on* $\{w\}\}$
using *islimpt_subset*[*of z* $\{w{\in}A.\ \neg f$ *analytic_on* $\{w\}\}$ $\{w.\ \neg f$ *analytic_on* $\{w\}\}$] **by** *blast*
next
have f *analytic_on A* $-$ $\{w{\in}A.\ \neg f$ *analytic_on* $\{w\}\}$
by (*subst analytic_on_analytic_at*) *auto*
thus f *holomorphic_on A* $-$ $\{w{\in}A.\ \neg f$ *analytic_on* $\{w\}\}$
by (*meson analytic_imp_holomorphic*)
qed (*use assms* **in** *auto*)

lemma *meromorphic_on_imp_has_laurent_expansion*:
assumes f *meromorphic_on A pts* $z \in A$
shows $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion laurent_expansion f z*
proof (*cases* $z \in$ *pts*)
case *True*
thus *?thesis*
using *assms* **by** (*intro not_essential_has_laurent_expansion*) (*auto simp*: *meromorphic_on_def*)
next
case *False*
have f *holomorphic_on* $(A - pts)$
using *assms* **by** (*auto simp*: *meromorphic_on_def*)

```
  moreover have z ∈ A − pts open (A − pts)
   using assms(2) False by (auto intro!: meromorphic_imp_open_diff[OF assms(1)])
  ultimately have f analytic_on {z}
    unfolding analytic_at by blast
  thus ?thesis
    using isolated_singularity_at_analytic not_essential_analytic
        not_essential_has_laurent_expansion by blast
qed

lemma
  assumes isolated_singularity_at f z f −z→ c
  shows   eventually_remove_sings_eq_nhds':
          eventually (λw. remove_sings f w = (if w = z then c else f w)) (nhds z)
    and   remove_sings_analytic_at_singularity: remove_sings f analytic_on {z}
proof −
  have eventually (λw. w ≠ z) (at z)
    by (auto simp: eventually_at_filter)
  hence eventually (λw. remove_sings f w = (if w = z then c else f w)) (at z)
    using eventually_remove_sings_eq_at[OF assms(1)]
    by eventually_elim auto
  moreover have remove_sings f z = c
    using assms by auto
  ultimately show ev: eventually (λw. remove_sings f w = (if w = z then c else
f w)) (nhds z)
    by (simp add: eventually_at_filter)

  have (λw. if w = z then c else f w) analytic_on {z}
    by (intro removable_singularity' assms)
  also have ?this ⟷ remove_sings f analytic_on {z}
    using ev by (intro analytic_at_cong) (auto simp: eq_commute)
  finally show ... .
qed

lemma remove_sings_meromorphic_on:
  assumes f meromorphic_on A pts ⋀z. z ∈ pts − pts' ⟹ ¬is_pole f z pts' ⊆
pts
  shows   remove_sings f meromorphic_on A pts'
  unfolding meromorphic_on_def
proof safe
  have remove_sings f analytic_on {z} if z ∈ A − pts' for z
  proof (cases z ∈ pts)
    case False
    hence ∗: f analytic_on {z}
      using assms meromorphic_imp_open_diff[OF assms(1)] that
      by (force simp: meromorphic_on_def analytic_at)
    have remove_sings f analytic_on {z} ⟷ f analytic_on {z}
      by (intro analytic_at_cong eventually_remove_sings_eq_nhds ∗ refl)
    thus ?thesis using ∗ by simp
  next
```

```
    case True
    have isol: isolated_singularity_at f z
      using True using assms by (auto simp: meromorphic_on_def)
    from assms(1) have not_essential f z
      using True by (auto simp: meromorphic_on_def)
    with assms(2) True that obtain c where f -z→ c
      by (auto simp: not_essential_def)
    thus remove_sings f analytic_on {z}
      by (intro remove_sings_analytic_at_singularity isol)
  qed
  hence remove_sings f analytic_on A - pts'
    by (subst analytic_on_analytic_at) auto
  thus remove_sings f holomorphic_on A - pts'
     using meromorphic_imp_open_diff'[OF assms(1,3)] by (subst (asm) ana-
lytic_on_open)
qed (use assms islimpt_subset[OF _ assms(3)] in ‹auto simp: meromorphic_on_def›)

lemma remove_sings_holomorphic_on:
  assumes f meromorphic_on A pts ⋀z. z ∈ pts ⟹ ¬is_pole f z
  shows   remove_sings f holomorphic_on A
  using remove_sings_meromorphic_on[OF assms(1), of {}] assms(2)
  by (auto simp: meromorphic_on_no_singularities)

lemma meromorphic_on_Ex_iff:
  (∃ pts. f meromorphic_on A pts) ⟷
     open A ∧ (∀ z∈A. ∃ F. (λw. f (z + w)) has_laurent_expansion F)
proof safe
  fix pts assume ∗: f meromorphic_on A pts
  from ∗ show open A
    by (auto simp: meromorphic_on_def)
  show ∃ F. (λw. f (z + w)) has_laurent_expansion F if z ∈ A for z
    using that ∗
   by (intro exI[of _ laurent_expansion f z] meromorphic_on_imp_has_laurent_expansion)
qed (blast intro!: has_laurent_expansion_on_imp_meromorphic_on)

lemma is_pole_inverse_holomorphic_pts:
  fixes pts::complex set and f::complex ⇒ complex
  defines g ≡ λx. (if x∈pts then 0 else inverse (f x))
  assumes mer: f meromorphic_on D pts
    and non_z: ⋀z. z ∈ D - pts ⟹ f z ≠ 0
    and all_poles:∀ x. is_pole f x ⟷ x∈pts
  shows g holomorphic_on D
proof -
  have open D and f_holo: f holomorphic_on (D-pts)
    using mer by (auto simp: meromorphic_on_def)
  have ∃ r. r>0 ∧ f analytic_on ball z r - {z}
          ∧ (∀ x ∈ ball z r - {z}. f x≠0) if z∈pts for z
  proof -
    have isolated_singularity_at f z is_pole f z
```

```
    using mer meromorphic_on_def that all_poles by blast+
  then obtain r1 where r1>0 and fan: f analytic_on ball z r1 − {z}
    by (meson isolated_singularity_at_def)
  obtain r2 where r2>0 ∀ x ∈ ball z r2 − {z}. f x≠0
    using non_zero_neighbour_pole[OF ‹is_pole f z›]
    unfolding eventually_at by (metis Diff_iff UNIV_I dist_commute insertI1
mem_ball)
  define r where r = min r1 r2
  have r>0 by (simp add: ‹0 < r2› ‹r1>0› r_def)
  moreover have f analytic_on ball z r − {z}
    using r_def by (force intro: analytic_on_subset [OF fan])
  moreover have ∀ x ∈ ball z r − {z}. f x≠0
    by (simp add: ‹∀ x∈ball z r2 − {z}. f x ≠ 0› r_def)
  ultimately show ?thesis by auto
 qed
 then obtain get_r where r_pos:get_r z>0
    and r_ana:f analytic_on ball z (get_r z) − {z}
    and r_nz:∀ x ∈ ball z (get_r z) − {z}. f x≠0
  if z∈pts for z
  by metis
 define p_balls where p_balls ≡ ⋃z∈pts. ball z (get_r z)
 have g_ball:g holomorphic_on ball z (get_r z) if z∈pts for z
 proof −
  have (λx. if x = z then 0 else inverse (f x)) holomorphic_on ball z (get_r z)
  proof (rule is_pole_inverse_holomorphic)
    show f holomorphic_on ball z (get_r z) − {z}
      using analytic_imp_holomorphic r_ana that by blast
    show is_pole f z
      using mer meromorphic_on_def that all_poles by force
    show ∀ x∈ball z (get_r z) − {z}. f x ≠ 0
      using r_nz that by metis
  qed auto
  then show ?thesis unfolding g_def
    by (smt (verit, ccfv_SIG) Diff_iff Elementary_Metric_Spaces.open_ball
        all_poles analytic_imp_holomorphic empty_iff
        holomorphic_transform insert_iff not_is_pole_holomorphic
        open_delete r_ana that)
 qed
 then have g holomorphic_on p_balls
 proof −
  have g analytic_on p_balls
    unfolding p_balls_def analytic_on_UN
    using g_ball by (simp add: analytic_on_open)
  moreover have open p_balls using p_balls_def by blast
  ultimately show ?thesis
    by (simp add: analytic_imp_holomorphic)
 qed
 moreover have g holomorphic_on D−pts
 proof −
```

```
    have (λz. inverse (f z)) holomorphic_on D − pts
      using f_holo holomorphic_on_inverse non_z by blast
    then show ?thesis
      by (metis DiffD2 g_def holomorphic_transform)
  qed
  moreover have open p_balls
    using p_balls_def by blast
  ultimately have g holomorphic_on (p_balls ∪ (D−pts))
    by (simp add: holomorphic_on_Un meromorphic_imp_open_diff[OF mer])
  moreover have D ⊆ p_balls ∪ (D−pts)
    unfolding p_balls_def using ‹⋀z. z ∈ pts ⟹ 0 < get_r z› by force
  ultimately show g holomorphic_on D by (meson holomorphic_on_subset)
qed

lemma meromorphic_imp_analytic_on:
  assumes f meromorphic_on D pts
  shows f analytic_on (D − pts)
 by (metis assms analytic_on_open meromorphic_imp_open_diff meromorphic_on_def)

lemma meromorphic_imp_constant_on:
  assumes merf: f meromorphic_on D pts
      and f constant_on (D − pts)
      and ∀ x∈pts. is_pole f x
    shows f constant_on D
proof −
  obtain c where c:⋀z. z ∈ D−pts ⟹ f z = c
    by (meson assms constant_on_def)

  have f z = c if z ∈ D for z
  proof (cases is_pole f z)
    case True
     then obtain r0 where r0 > 0 and r0: f analytic_on ball z r0 − {z} and
pol: is_pole f z
      using merf unfolding meromorphic_on_def isolated_singularity_at_def
      by (metis ‹z ∈ D› insert_Diff insert_Diff_if insert_iff merf
          meromorphic_imp_open_diff not_is_pole_holomorphic)
    have open D
      using merf meromorphic_on_def by auto
    then obtain r where r > 0 ball z r ⊆ D r ≤ r0
      by (smt (verit, best) ‹0 < r0› ‹z ∈ D› openE order_subst2 subset_ball)
    have r: f analytic_on ball z r − {z}
     by (meson Diff_mono ‹r ≤ r0› analytic_on_subset order_refl r0 subset_ball)
    have ball z r − {z} ⊆ −pts
      using merf r unfolding meromorphic_on_def
      by (meson ComplI Elementary_Metric_Spaces.open_ball
       analytic_imp_holomorphic assms(3) not_is_pole_holomorphic open_delete
subsetI)
    with ‹ball z r ⊆ D› have ball z r − {z} ⊆ D−pts
      by fastforce
```

```
    with c have c': ⋀u. u ∈ ball z r − {z} ⟹ f u = c
      by blast
    have False if ∀F x in at z. cmod c + 1 ≤ cmod (f x)
    proof −
      have ∀F x in at z within ball z r − {z}. cmod c + 1 ≤ cmod (f x)
      by (smt (verit, best) Diff_UNIV Diff_eq_empty_iff eventually_at_topological
insert_subset that)
      with ‹r > 0› show ?thesis
      apply (simp add: c' eventually_at_filter topological_space_class.eventually_nhds
open_dist)
        by (metis dist_commute min_less_iff_conj perfect_choose_dist)
    qed
    with pol show ?thesis
      by (auto simp: is_pole_def filterlim_at_infinity_conv_norm_at_top filter-
lim_at_top)
  next
    case False
    then show ?thesis by (meson DiffI assms(3) c that)
  qed
  then show ?thesis
    by (simp add: constant_on_def)
qed


lemma meromorphic_isolated:
  assumes merf: f meromorphic_on D pts and p∈pts
  obtains r where r>0 ball p r ⊆ D ball p r ∩ pts = {p}
proof −
  have ∀z∈D. ∃e>0. finite (pts ∩ ball z e)
    using merf unfolding meromorphic_on_def islimpt_eq_infinite_ball
    by auto
  then obtain r0 where r0:r0>0 finite (pts ∩ ball p r0)
    by (metis assms(2) in_mono merf meromorphic_on_def)
  moreover define pts' where pts' = pts ∩ ball p r0 − {p}
  ultimately have finite pts'
    by simp

  define r1 where r1=(if pts'={} then r0 else
                      min (Min {dist p' p |p'. p'∈pts'}/2) r0)
  have r1>0 ∧ pts ∩ ball p r1 − {p} = {}
  proof (cases pts'={})
    case True
    then show ?thesis
      using pts'_def r0(1) r1_def by presburger
  next
    case False
    define S where S={dist p' p |p'. p'∈pts'}

    have nempty:S ≠ {}
```

```
    using False S_def by blast
  have finite:finite S
    using ‹finite pts'› S_def by simp

  have r1>0
  proof −
    have r1=min (Min S/2) r0
      using False unfolding S_def r1_def by auto
    moreover have Min S∈S
      using ‹S≠{}› ‹finite S›  Min_in by auto
    then have Min S>0 unfolding S_def
      using pts'_def by force
    ultimately show ?thesis using ‹r0>0› by auto
  qed
  moreover have pts ∩ ball p r1 − {p} = {}
  proof (rule ccontr)
    assume pts ∩ ball p r1 − {p} ≠ {}
    then obtain p' where p'∈pts ∩ ball p r1 − {p} by blast
    moreover have r1≤r0 using r1_def by auto
    ultimately have p'∈pts' unfolding pts'_def
      by auto
    then have dist p' p≥Min S
      using S_def eq_Min_iff local.finite by blast
    moreover have dist p' p < Min S
      using ‹p'∈pts ∩ ball p r1 − {p}› False unfolding r1_def
      apply (fold S_def)
      by (smt (verit, ccfv_threshold) DiffD1 Int_iff dist_commute
          dist_triangle_half_l mem_ball)
    ultimately show False by auto
  qed
  ultimately show ?thesis by auto
qed
then have r1>0 and r1_pts:pts ∩ ball p r1 − {p} = {} by auto

obtain r2 where r2>0 ball p r2 ⊆ D
  by (metis assms(2) merf meromorphic_on_def openE subset_eq)
define r where r=min r1 r2
have r > 0 unfolding r_def
  by (simp add: ‹0 < r1› ‹0 < r2›)
moreover have ball p r ⊆ D
  using ‹ball p r2 ⊆ D› r_def by auto
moreover have ball p r ∩ pts = {p}
  using assms(2) ‹r>0› r1_pts
  unfolding r_def by auto
ultimately show ?thesis using that by auto
qed

lemma meromorphic_pts_closure:
  assumes merf: f meromorphic_on D pts
```

```
  shows pts ⊆ closure (D − pts)
proof −
  have p islimpt (D − pts) if p∈pts for p
  proof −
    obtain r where r>0 ball p r ⊆ D ball p r ∩ pts = {p}
      using meromorphic_isolated[OF merf ‹p∈pts›] by auto
    from ‹r>0›
    have p islimpt ball p r − {p}
      by (meson open_ball ball_subset_cball in_mono islimpt_ball
          islimpt_punctured le_less open_contains_ball_eq)
    moreover have  ball p r − {p} ⊆ D − pts
      using ‹ball p r ∩ pts = {p}› ‹ball p r ⊆ D› by fastforce
    ultimately show ?thesis
      using islimpt_subset by auto
  qed
  then show ?thesis by (simp add: islimpt_in_closure subset_eq)
qed

lemma nconst_imp_nzero_neighbour:
  assumes merf: f meromorphic_on D pts
    and f_nconst:¬(∀ w∈D−pts. f w=0)
    and z∈D and connected D
  shows (∀F w in at z. f w ≠ 0 ∧ w ∈ D − pts)
proof −
  obtain β where β:β ∈ D − pts f β≠0
    using f_nconst by auto

  have ?thesis if z∉pts
  proof −
    have ∀F w in at z. f w ≠ 0 ∧ w ∈ D − pts
      apply (rule non_zero_neighbour_alt[of f D−pts z  β])
      subgoal using merf meromorphic_on_def by blast
      subgoal using merf meromorphic_imp_open_diff by auto
      subgoal using assms(4) merf meromorphic_imp_connected_diff by blast
      subgoal by (simp add: assms(3) that)
      using β by auto
    then show ?thesis by (auto elim:eventually_mono)
  qed
  moreover have ?thesis if z∈pts ¬ f −z→ 0
  proof −
    have ∀F w in at z. w ∈ D − pts
      using merf[unfolded meromorphic_on_def islimpt_iff_eventually] ‹z∈D›
      using eventually_at_in_open′ eventually_elim2 by fastforce
    moreover have ∀F w in at z. f w ≠ 0
    proof (cases  is_pole f z)
      case True
      then show ?thesis using non_zero_neighbour_pole by auto
    next
      case False
```

```
    moreover have not_essential f z
      using merf meromorphic_on_def that(1) by fastforce
    ultimately obtain c where c≠0 f ─z→ c
      by (metis ‹¬ f ─z→ 0› not_essential_def)
    then show ?thesis
      using tendsto_imp_eventually_ne by auto
  qed
  ultimately show ?thesis by eventually_elim auto
qed
moreover have ?thesis if z∈pts f ─z→ 0
proof ─
  define ff where ff=(λx. if x=z then 0 else f x)
  define A where A=D − (pts − {z})

  have f holomorphic_on A − {z}
    by (metis A_def Diff_insert analytic_imp_holomorphic
        insert_Diff merf meromorphic_imp_analytic_on that(1))
  moreover have open A
    using A_def merf meromorphic_imp_open_diff' by force
  ultimately have ff holomorphic_on A
    using ‹f ─z→ 0› unfolding ff_def
    by (rule removable_singularity)
  moreover have connected A
  proof ─
    have connected (D − pts)
      using assms(4) merf meromorphic_imp_connected_diff by auto
    moreover have D − pts ⊆ A
      unfolding A_def by auto
    moreover have A ⊆ closure (D − pts) unfolding A_def
      by (smt (verit, ccfv_SIG) Diff_empty Diff_insert
          closure_subset insert_Diff_single insert_absorb
          insert_subset merf meromorphic_pts_closure that(1))
    ultimately show ?thesis using connected_intermediate_closure
      by auto
  qed
  moreover have z ∈ A using A_def assms(3) by blast
  moreover have ff z = 0 unfolding ff_def by auto
  moreover have β ∈ A  using A_def β(1) by blast
  moreover have ff β ≠ 0 using β(1) β(2) ff_def that(1) by auto
  ultimately obtain r where 0 < r
      ball z r ⊆ A ⋀x. x ∈ ball z r − {z} ⟹ ff x ≠ 0
    using ‹open A› isolated_zeros[of ff A z β] by auto
  then show ?thesis unfolding eventually_at ff_def
    by (intro exI[of _ r]) (auto simp: A_def dist_commute ball_def)
qed
ultimately show ?thesis by auto
qed

lemma nconst_imp_nzero_neighbour':
```

assumes *merf*: *f meromorphic_on D pts*
and *f_nconst*:¬(∀ *w*∈*D*−*pts*. *f w*=*0*)
and *z*∈*D* **and** *connected D*
shows ∀ *F w in at z*. *f w* ≠ *0*
using *nconst_imp_nzero_neighbour*[*OF assms*]
by (*auto elim*:*eventually_mono*)

lemma *meromorphic_compact_finite_zeros*:
assumes *merf*:*f meromorphic_on D pts*
and *compact S S* ⊆ *D connected D*
and *f_nconst*:¬(∀ *w*∈*D*−*pts*. *f w*=*0*)
shows *finite* ({*x*∈*S*. *f x*=*0*})
proof −
have *finite* ({*x*∈*S*. *f x*=*0* ∧ *x* ∉ *pts*})
proof (*rule ccontr*)
assume *infinite* {*x* ∈ *S*. *f x* = *0* ∧ *x* ∉ *pts*}
then obtain *z* **where** *z*∈*S* **and** *z_lim*:*z islimpt* {*x* ∈ *S*. *f x* = *0*
∧ *x* ∉ *pts*}
using ‹*compact S*› **unfolding** *compact_eq_Bolzano_Weierstrass*
by *auto*

from *z_lim*
have ∃ *F x in at z*. *f x* = *0* ∧ *x* ∈ *S* ∧ *x* ∉ *pts*
unfolding *islimpt_iff_eventually not_eventually* **by** *simp*
moreover have ∀ *F w in at z*. *f w* ≠ *0* ∧ *w* ∈ *D* − *pts*
using *nconst_imp_nzero_neighbour*[*OF merf f_nconst* _ ‹*connected D*›]
‹*z*∈*S*› ‹*S* ⊆ *D*›
by *auto*
ultimately have ∃ *F x in at z*. *False*
by (*simp add*: *eventually_mono frequently_def*)
then show *False* **by** *auto*
qed
moreover have *finite* (*S* ∩ *pts*)
using *meromorphic_compact_finite_pts*[*OF merf* ‹*compact S*› ‹*S* ⊆ *D*›] **.**
ultimately have *finite* ({*x*∈*S*. *f x*=*0* ∧ *x* ∉ *pts*} ∪ (*S* ∩ *pts*))
unfolding *finite_Un* **by** *auto*
then show *?thesis* **by** (*elim rev_finite_subset*) *auto*
qed

lemma *meromorphic_onI* [*intro?*]:
assumes *open A pts* ⊆ *A*
assumes *f holomorphic_on A* − *pts* ⋀*z*. *z* ∈ *A* ⟹ ¬*z islimpt pts*
assumes ⋀*z*. *z* ∈ *pts* ⟹ *isolated_singularity_at f z*
assumes ⋀*z*. *z* ∈ *pts* ⟹ *not_essential f z*
shows *f meromorphic_on A pts*
using *assms* **unfolding** *meromorphic_on_def* **by** *blast*

lemma *Polygamma_plus_of_nat*:
assumes ∀ *k*<*m*. *z* ≠ −*of_nat k*

```
  shows   Polygamma n (z + of_nat m) =
            Polygamma n z + (-1) ^ n * fact n * (∑k<m. 1 / (z + of_nat k) ^
Suc n)
  using assms
proof (induction m)
  case (Suc m)
  have Polygamma n (z + of_nat (Suc m)) = Polygamma n (z + of_nat m + 1)
    by (simp add: add_ac)
  also have ... = Polygamma n (z + of_nat m) + (-1) ^ n * fact n * (1 / ((z
+ of_nat m) ^ Suc n))
    using Suc.prems by (subst Polygamma_plus1) (auto simp: add_eq_0_iff2)
  also have Polygamma n (z + of_nat m) =
              Polygamma n z + (-1) ^ n * (∑k<m. 1 / (z + of_nat k) ^ Suc n)
* fact n
    using Suc.prems by (subst Suc.IH) auto
  finally show ?case
    by (simp add: algebra_simps)
qed auto

lemma tendsto_Gamma [tendsto_intros]:
  assumes (f ⟶ c) F c ∉ ℤ≤0
  shows   ((λz. Gamma (f z)) ⟶ Gamma c) F
  by (intro isCont_tendsto_compose[OF _ assms(1)] continuous_intros assms)

lemma tendsto_Polygamma [tendsto_intros]:
  fixes f :: _ ⇒ 'a :: {real_normed_field,euclidean_space}
  assumes (f ⟶ c) F c ∉ ℤ≤0
  shows   ((λz. Polygamma n (f z)) ⟶ Polygamma n c) F
  by (intro isCont_tendsto_compose[OF _ assms(1)] continuous_intros assms)

lemma analytic_on_Gamma' [analytic_intros]:
  assumes f analytic_on A ∀x∈A. f x ∉ ℤ≤0
  shows   (λz. Gamma (f z)) analytic_on A
 using analytic_on_compose_gen[OF assms(1) analytic_Gamma[of f ` A]] assms(2)
  by (auto simp: o_def)

lemma analytic_on_Polygamma' [analytic_intros]:
  assumes f analytic_on A ∀x∈A. f x ∉ ℤ≤0
  shows   (λz. Polygamma n (f z)) analytic_on A
  using analytic_on_compose_gen[OF assms(1) analytic_on_Polygamma[of f `
A n]] assms(2)
  by (auto simp: o_def)

lemma
  shows is_pole_Polygamma: is_pole (Polygamma n) (-of_nat m :: complex)
  and   zorder_Polygamma: zorder (Polygamma n) (-of_nat m) = -int (Suc n)
  and   residue_Polygamma: residue (Polygamma n) (-of_nat m) = (if n = 0
then -1 else 0)
proof -
```

```
  define g1 :: complex ⇒ complex where
    g1 = (λz. Polygamma n (z + of_nat (Suc m)) +
           (−1) ^ Suc n * fact n * (∑k<m. 1 / (z + of_nat k) ^ Suc n))
  define g :: complex ⇒ complex where
    g = (λz. g1 z + (−1) ^ Suc n * fact n / (z + of_nat m) ^ Suc n)
  define F where F = fps_to_fls (fps_expansion g1 (−of_nat m)) + fls_const
((−1) ^ Suc n * fact n) / fls_X ^ Suc n
   have F_altdef: F = fps_to_fls (fps_expansion g1 (−of_nat m)) + fls_shift
(n+1) (fls_const ((−1) ^ Suc n * fact n))
    by (simp add: F_def del: power_Suc)

  have ¬(−of_nat m) islimpt (ℤ≤0 :: complex set)
    by (intro discrete_imp_not_islimpt[where e = 1])
       (auto elim!: nonpos_Ints_cases simp: dist_of_int)
  hence eventually (λz::complex. z ∉ ℤ≤0) (at (−of_nat m))
    by (auto simp: islimpt_conv_frequently_at frequently_def)
  hence ev: eventually (λz. Polygamma n z = g z) (at (−of_nat m))
  proof eventually_elim
    case (elim z)
    hence ∗: ∀k<Suc m. z ≠ − of_nat k
      by auto
    thus ?case
      using Polygamma_plus_of_nat[of Suc m z n, OF ∗]
      by (auto simp: g_def g1_def algebra_simps)
  qed

  have (λw. g (−of_nat m + w)) has_laurent_expansion F
    unfolding g_def F_def
   by (intro laurent_expansion_intros has_laurent_expansion_fps analytic_at_imp_has_fps_expansion)
       (auto simp: g1_def intro!: laurent_expansion_intros analytic_intros)
 also have ?this ⟷ (λw. Polygamma n (−of_nat m + w)) has_laurent_expansion
F
    using ev by (intro has_laurent_expansion_cong refl)
                (simp_all add: eq_commute at_to_0' eventually_filtermap)
  finally have ∗: (λw. Polygamma n (−of_nat m + w)) has_laurent_expansion
F .

  have subdegree: fls_subdegree F = −int (Suc n) unfolding F_def
     by (subst fls_subdegree_add_eq2) (simp_all add: fls_subdegree_fls_to_fps
fls_divide_subdegree)
  have [simp]: F ≠ 0
    using subdegree by auto

  show is_pole (Polygamma n) (−of_nat m :: complex)
    using ∗ by (rule has_laurent_expansion_imp_is_pole) (auto simp: subdegree)
  show zorder (Polygamma n) (−of_nat m :: complex) = −int (Suc n)
    by (subst has_laurent_expansion_zorder[OF ∗]) (auto simp: subdegree)
  show residue (Polygamma n) (−of_nat m :: complex) = (if n = 0 then −1 else
0)
```

by (*subst has_laurent_expansion_residue*[*OF* ∗]) (*auto simp*: *F_altdef*)
qed

lemma *Gamma_meromorphic_on* [*meromorphic_intros*]: *Gamma meromorphic_on UNIV* $\mathbb{Z}_{\leq 0}$
proof
show ¬*z islimpt* $\mathbb{Z}_{\leq 0}$ **for** *z* :: *complex*
by (*intro discrete_imp_not_islimpt*[*of 1*]) (*auto elim*!: *nonpos_Ints_cases simp*: *dist_of_int*)
next
fix *z* :: *complex* **assume** *z*: $z \in \mathbb{Z}_{\leq 0}$
then obtain *n* **where** *n*: $z = -of\_nat\ n$
by (*elim nonpos_Ints_cases*′)
show *not_essential Gamma z*
by (*auto simp*: *n intro*!: *is_pole_imp_not_essential is_pole_Gamma*)
have ∗: *open* $(-(\mathbb{Z}_{\leq 0} - \{z\}))$
by (*intro open_Compl discrete_imp_closed*[*of 1*]) (*auto elim*!: *nonpos_Ints_cases simp*: *dist_of_int*)
have *Gamma holomorphic_on* $-(\mathbb{Z}_{\leq 0} - \{z\}) - \{z\}$
by (*intro holomorphic_intros*) *auto*
thus *isolated_singularity_at Gamma z*
by (*rule isolated_singularity_at_holomorphic*) (*use z* ∗ **in** *auto*)
qed (*auto intro*!: *holomorphic_intros*)

lemma *Polygamma_meromorphic_on* [*meromorphic_intros*]: *Polygamma n meromorphic_on UNIV* $\mathbb{Z}_{\leq 0}$
proof
show ¬*z islimpt* $\mathbb{Z}_{\leq 0}$ **for** *z* :: *complex*
by (*intro discrete_imp_not_islimpt*[*of 1*]) (*auto elim*!: *nonpos_Ints_cases simp*: *dist_of_int*)
next
fix *z* :: *complex* **assume** *z*: $z \in \mathbb{Z}_{\leq 0}$
then obtain *m* **where** *n*: $z = -of\_nat\ m$
by (*elim nonpos_Ints_cases*′)
show *not_essential* (*Polygamma n*) *z*
by (*auto simp*: *n intro*!: *is_pole_imp_not_essential is_pole_Polygamma*)
have ∗: *open* $(-(\mathbb{Z}_{\leq 0} - \{z\}))$
by (*intro open_Compl discrete_imp_closed*[*of 1*]) (*auto elim*!: *nonpos_Ints_cases simp*: *dist_of_int*)
have *Polygamma n holomorphic_on* $-(\mathbb{Z}_{\leq 0} - \{z\}) - \{z\}$
by (*intro holomorphic_intros*) *auto*
thus *isolated_singularity_at* (*Polygamma n*) *z*
by (*rule isolated_singularity_at_holomorphic*) (*use z* ∗ **in** *auto*)
qed (*auto intro*!: *holomorphic_intros*)

theorem *argument_principle*′:
fixes *f*::*complex* ⇒ *complex* **and** *poles s*:: *complex set*
— *pz* is the set of non-essential singularities and zeros

```
  defines pz ≡ {w∈s. f w = 0 ∨ w ∈ poles}
  assumes open s and
          connected s and
          f_holo:f holomorphic_on s−poles and
          h_holo:h holomorphic_on s and
          valid_path g and
          loop:pathfinish g = pathstart g and
          path_img:path_image g ⊆ s − pz and
          homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0 and
          finite:finite pz and
          poles:∀ p∈s∩poles. not_essential f p
  shows contour_integral g (λx. deriv f x * h x / f x) = 2 * pi * i *
          (∑ p∈pz. winding_number g p * h p * zorder f p)
proof −
  define ff where ff = remove_sings f

  have finite':finite (s ∩ poles)
    using finite unfolding pz_def by (auto elim:rev_finite_subset)

  have isolated:isolated_singularity_at f z if z∈s for z
  proof (rule isolated_singularity_at_holomorphic)
    show f holomorphic_on (s−(poles−{z})) − {z}
      by (metis Diff_empty Diff_insert Diff_insert0 Diff_subset
          f_holo holomorphic_on_subset insert_Diff)
    show open (s − (poles − {z}))
      by (metis Diff_Diff_Int Int_Diff assms(2) finite' finite_Diff
          finite_imp_closed inf.idem open_Diff)
    show z ∈ s − (poles − {z})
      using assms(4) that by auto
  qed

  have not_ess:not_essential f w if w∈s for w
    by (metis Diff_Diff_Int Diff_iff Int_Diff Int_absorb assms(2)
        f_holo finite' finite_imp_closed not_essential_holomorphic
        open_Diff poles that)

  have nzero:∀ F x in at w. f x ≠ 0 if w∈s for w
  proof (rule ccontr)
    assume ¬ (∀ F x in at w. f x ≠ 0)
    then have ∃ F x in at w. f x = 0
      unfolding not_eventually by simp
    moreover have ∀ F x in at w. x∈s
      by (simp add: assms(2) eventually_at_in_open' that)
    ultimately have ∃ F x in at w. x∈{w∈s. f w = 0}
      apply (elim frequently_rev_mp)
      by (auto elim:eventually_mono)
    from frequently_at_imp_islimpt[OF this]
    have w islimpt {w ∈ s. f w = 0} .
    then have infinite({w ∈ s. f w = 0} ∩ ball w 1)
```

```
    unfolding islimpt_eq_infinite_ball by auto
  then have infinite({w ∈ s. f w = 0})
    by auto
  then have infinite pz unfolding pz_def
    by (smt (verit) Collect_mono_iff rev_finite_subset)
  then show False using finite by auto
qed

obtain pts' where pts':pts' ⊆ s ∩ poles
  finite pts' ff holomorphic_on s − pts' ∀ x∈pts'. is_pole ff x
  apply (elim get_all_poles_from_remove_sings
      [of f,folded ff_def,rotated −1])
  subgoal using f_holo by fastforce
  using ‹open s› poles finite' by auto

have pts'_sub_pz:{w ∈ s. ff w = 0 ∨ w ∈ pts'} ⊆ pz
proof −
  have w∈poles if w∈s w∈pts' for w
    by (meson in_mono le_infE pts'(1) that(2))
  moreover have f w=0 if w∈s w∉poles ff w=0 for w
  proof −
    have ¬ is_pole f w
      by (metis DiffI Diff_Diff_Int Diff_subset assms(2) f_holo
          finite' finite_imp_closed inf.absorb_iff2
          not_is_pole_holomorphic open_Diff that(1) that(2))
    then have f −w→ 0
      using remove_sings_eq_0_iff[OF not_ess[OF ‹w∈s›]] ‹ff w=0›
      unfolding ff_def by auto
    moreover have f analytic_on {w}
      using that(1,2) finite' f_holo assms(2)
      by (metis Diff_Diff_Int Diff_empty Diff_iff Diff_subset
          double_diff finite_imp_closed
          holomorphic_on_imp_analytic_at open_Diff)
    ultimately show ?thesis
      using ff_def remove_sings_at_analytic that(3) by presburger
  qed
  ultimately show ?thesis unfolding pz_def by auto
qed


have contour_integral g (λx. deriv f x * h x / f x)
      = contour_integral g (λx. deriv ff x * h x / ff x)
proof (rule contour_integral_eq)
  fix x assume x ∈ path_image g
  have f analytic_on {x}
  proof (rule holomorphic_on_imp_analytic_at[of _ s−poles])
    from finite'
    show open (s − poles)
      using ‹open s›
```

```
      by (metis Diff_Compl Diff_Diff_Int Diff_eq finite_imp_closed
          open_Diff)
    show x ∈ s − poles
      using path_img ‹x ∈ path_image g› unfolding pz_def by auto
  qed (use f_holo in simp)
  then show deriv f x * h x / f x = deriv ff x * h x / ff x
    unfolding ff_def by auto
qed
also have ... = complex_of_real (2 * pi) * i *
                  (∑ p∈{w ∈ s. ff w = 0 ∨ w ∈ pts'}.
                    winding_number g p * h p * of_int (zorder ff p))
proof (rule argument_principle[OF ‹open s› ‹connected s›, of ff pts' h g])
  show path_image g ⊆ s − {w ∈ s. ff w = 0 ∨ w ∈ pts'}
    using path_img pts'_sub_pz by auto
  show finite {w ∈ s. ff w = 0 ∨ w ∈ pts'}
    using pts'_sub_pz finite
    using rev_finite_subset by blast
qed (use pts' assms in auto)
also have ... = 2 * pi * i *
      (∑ p∈pz. winding_number g p * h p * zorder f p)
proof −
  have (∑ p∈{w ∈ s. ff w = 0 ∨ w ∈ pts'}.
     winding_number g p * h p * of_int (zorder ff p)) =
    (∑ p∈pz. winding_number g p * h p * of_int (zorder f p))
  proof (rule sum.mono_neutral_cong_left)
    have zorder f w = 0
      if w∈s  f w = 0 ∨ w ∈ poles ff w ≠ 0  w ∉ pts'
      for w
    proof −
      define F where F=laurent_expansion f w
      have has_l:(λx. f (w + x)) has_laurent_expansion F
        unfolding F_def
        apply (rule not_essential_has_laurent_expansion)
        using isolated not_ess ‹w∈s› by auto
      from has_laurent_expansion_eventually_nonzero_iff[OF this]
      have F ≠0
        using nzero ‹w∈s› by auto
      from tendsto_0_subdegree_iff[OF has_l this]
      have f −w→ 0 = (0 < fls_subdegree F) .
      moreover have ¬ (is_pole f w ∨ f −w→ 0)
        using remove_sings_eq_0_iff[OF not_ess[OF ‹w∈s›]] ‹ff w ≠ 0›
        unfolding ff_def by auto
      moreover have is_pole f w = (fls_subdegree F < 0)
        using is_pole_fls_subdegree_iff[OF has_l] .
      ultimately have fls_subdegree F = 0 by auto
      then show ?thesis
        using has_laurent_expansion_zorder[OF has_l ‹F≠0›] by auto
    qed
    then show ∀ i∈pz − {w ∈ s. ff w = 0 ∨ w ∈ pts'}.
```

```
      winding_number g i * h i * of_int (zorder f i) = 0
      unfolding pz_def by auto
    show ⋀x. x ∈ {w ∈ s. fff w = 0 ∨ w ∈ pts'} ⟹
      winding_number g x * h x * of_int (zorder fff x) =
      winding_number g x * h x * of_int (zorder f x)
      using isolated zorder_remove_sings[of f,folded fff_def] by auto
  qed (use pts'_sub_pz finite in auto)
  then show ?thesis by auto
 qed
 finally show ?thesis .
qed

lemma meromorphic_on_imp_isolated_singularity:
 assumes f meromorphic_on D pts z ∈ D
 shows   isolated_singularity_at f z
 by (meson DiffI assms(1) assms(2) holomorphic_on_imp_analytic_at isolated_singularity_at_analytic

     meromorphic_imp_open_diff meromorphic_on_def)

lemma meromorphic_imp_not_is_pole:
 assumes f meromorphic_on D pts z ∈ D - pts
 shows   ¬is_pole f z
proof -
 from assms have f analytic_on {z}
  using meromorphic_on_imp_analytic_at by blast
 thus ?thesis
  using analytic_at not_is_pole_holomorphic by blast
qed

lemma meromorphic_all_poles_iff_empty [simp]: f meromorphic_on pts pts ⟷
pts = {}
 by (auto simp: meromorphic_on_def holomorphic_on_def open_imp_islimpt)

lemma meromorphic_imp_nonsingular_point_exists:
 assumes f meromorphic_on A pts A ≠ {}
 obtains x where x ∈ A - pts
proof -
 have A ≠ pts
  using assms by auto
 moreover have pts ⊆ A
  using assms by (auto simp: meromorphic_on_def)
 ultimately show ?thesis
  using that by blast
qed

lemma meromorphic_frequently_const_imp_const:
 assumes f meromorphic_on A pts connected A
 assumes frequently (λw. f w = c) (at z)
 assumes z ∈ A - pts
```

```
  assumes w ∈ A − pts
  shows   f w = c
proof −
  have f w − c = 0
  proof (rule analytic_continuation[where f = λz. f z − c])
    show (λz. f z − c) holomorphic_on (A − pts)
     by (intro holomorphic_intros meromorphic_imp_holomorphic[OF assms(1)])
    show [intro]: open (A − pts)
      using assms meromorphic_imp_open_diff by blast
    show connected (A − pts)
      using assms meromorphic_imp_connected_diff by blast
    show {z∈A−pts. f z = c} ⊆ A − pts
      by blast
    have eventually (λz. z ∈ A − pts) (at z)
      using assms by (intro eventually_at_in_open') auto
    hence frequently (λz. f z = c ∧ z ∈ A − pts) (at z)
      by (intro frequently_eventually_frequently assms)
    thus z islimpt {z∈A−pts. f z = c}
      by (simp add: islimpt_conv_frequently_at conj_commute)
  qed (use assms in auto)
  thus ?thesis
    by simp
qed

lemma meromorphic_imp_eventually_neq:
  assumes f meromorphic_on A pts connected A ¬f constant_on A − pts
  assumes z ∈ A − pts
  shows   eventually (λz. f z ≠ c) (at z)
proof (rule ccontr)
  assume ¬eventually (λz. f z ≠ c) (at z)
  hence ∗: frequently (λz. f z = c) (at z)
    by (auto simp: frequently_def)
  have ∀ w∈A−pts. f w = c
    using meromorphic_frequently_const_imp_const [OF assms(1,2) ∗ assms(4)]
by blast
  hence f constant_on A − pts
    by (auto simp: constant_on_def)
  thus False
    using assms(3) by contradiction
qed

lemma meromorphic_frequently_const_imp_const':
  assumes f meromorphic_on A pts connected A ∀ w∈pts. is_pole f w
  assumes frequently (λw. f w = c) (at z)
  assumes z ∈ A
  assumes w ∈ A
  shows   f w = c
proof −
  have ¬is_pole f z
```

```
    using frequently_const_imp_not_is_pole[OF assms(4)] .
  with assms have z: z ∈ A − pts
    by auto
  have ∗: f w = c if w ∈ A − pts for w
   using that meromorphic_frequently_const_imp_const [OF assms(1,2,4) z] by
auto
  have ¬is_pole f u if u ∈ A for u
  proof −
    have is_pole f u ⟷ is_pole (λ_. c) u
    proof (rule is_pole_cong)
      have eventually (λw. w ∈ A − (pts − {u}) − {u}) (at u)
     by (intro eventually_at_in_open meromorphic_imp_open_diff' [OF assms(1)])
(use that in auto)
      thus eventually (λw. f w = c) (at u)
        by eventually_elim (use ∗ in auto)
    qed auto
    thus ?thesis
      by auto
  qed
  moreover have pts ⊆ A
    using assms(1) by (simp add: meromorphic_on_def)
  ultimately have pts = {}
    using assms(3) by auto
  with ∗ and ‹w ∈ A› show ?thesis
    by blast
qed

lemma meromorphic_imp_eventually_neq':
  assumes f meromorphic_on A pts connected A ∀ w∈pts. is_pole f w ¬f con-
stant_on A
  assumes z ∈ A
  shows   eventually (λz. f z ≠ c) (at z)
proof (rule ccontr)
  assume ¬eventually (λz. f z ≠ c) (at z)
  hence ∗: frequently (λz. f z = c) (at z)
    by (auto simp: frequently_def)
  have ∀ w∈A. f w = c
   using meromorphic_frequently_const_imp_const' [OF assms(1,2,3) ∗ assms(5)]
by blast
  hence f constant_on A
    by (auto simp: constant_on_def)
  thus False
    using assms(4) by contradiction
qed

lemma zorder_eq_0_iff_meromorphic:
  assumes f meromorphic_on A pts ∀ z∈pts. is_pole f z z ∈ A
  assumes eventually (λx. f x ≠ 0) (at z)
  shows   zorder f z = 0 ⟷ ¬is_pole f z ∧ f z ≠ 0
```

proof (*cases* $z \in pts$)
case *True*
from *assms* **obtain** F **where** F: $(\lambda x.\ f\ (z + x))$ *has_laurent_expansion* F
by (*metis True meromorphic_on_def not_essential_has_laurent_expansion*)
from F **and** *assms*(4) **have** [*simp*]: $F \neq 0$
using *has_laurent_expansion_eventually_nonzero_iff* **by** *blast*
show *?thesis* **using** *True assms*(2)
using *is_pole_fls_subdegree_iff* [*OF F*] *has_laurent_expansion_zorder* [*OF F*]
by *auto*
next
case *False*
have *ana*: *f analytic_on* {*z*}
using *meromorphic_on_imp_analytic_at False assms* **by** *blast*
hence ¬*is_pole f z*
using *analytic_at not_is_pole_holomorphic* **by** *blast*
moreover have *frequently* $(\lambda w.\ f\ w \neq 0)$ (*at z*)
using *assms*(4) **by** (*intro eventually_frequently*) *auto*
ultimately show *?thesis* **using** *zorder_eq_0_iff*[*OF ana*] *False*
by *auto*
qed

lemma *zorder_pos_iff_meromorphic*:
assumes *f meromorphic_on A pts* ∀ *z*∈*pts*. *is_pole f z z* ∈ *A*
assumes *eventually* $(\lambda x.\ f\ x \neq 0)$ (*at z*)
shows *zorder f z* > *0* ⟷ ¬*is_pole f z* ∧ *f z* = *0*
proof (*cases* $z \in pts$)
case *True*
from *assms* **obtain** F **where** F: $(\lambda x.\ f\ (z + x))$ *has_laurent_expansion* F
by (*metis True meromorphic_on_def not_essential_has_laurent_expansion*)
from F **and** *assms*(4) **have** [*simp*]: $F \neq 0$
using *has_laurent_expansion_eventually_nonzero_iff* **by** *blast*
show *?thesis* **using** *True assms*(2)
using *is_pole_fls_subdegree_iff* [*OF F*] *has_laurent_expansion_zorder* [*OF F*]
by *auto*
next
case *False*
have *ana*: *f analytic_on* {*z*}
using *meromorphic_on_imp_analytic_at False assms* **by** *blast*
hence ¬*is_pole f z*
using *analytic_at not_is_pole_holomorphic* **by** *blast*
moreover have *frequently* $(\lambda w.\ f\ w \neq 0)$ (*at z*)
using *assms*(4) **by** (*intro eventually_frequently*) *auto*
ultimately show *?thesis* **using** *zorder_pos_iff'*[*OF ana*] *False*
by *auto*
qed

lemma *zorder_neg_iff_meromorphic*:

assumes *f meromorphic_on A pts* ∀ *z*∈*pts*. *is_pole f z z* ∈ *A*
assumes *eventually* (λ*x*. *f x* ≠ *0*) (*at z*)
shows *zorder f z* < *0* ⟷ *is_pole f z*
proof −
have *frequently* (λ*x*. *f x* ≠ *0*) (*at z*)
using *assms* **by** (*intro eventually_frequently*) *auto*
moreover from *assms* **have** *isolated_singularity_at f z not_essential f z*
using *meromorphic_on_imp_isolated_singularity meromorphic_on_imp_not_essential*
by *blast*+
ultimately show *?thesis*
using *isolated_pole_imp_neg_zorder neg_zorder_imp_is_pole* **by** *blast*
qed

lemma *meromorphic_on_imp_discrete*:
assumes *mero*:*f meromorphic_on S pts* **and** *connected S*
and *nconst*:¬ (∀ *w*∈*S* − *pts*. *f w* = *c*)
shows *discrete* {*x*∈*S*. *f x*=*c*}
proof −
define *g* **where** *g*=(λ*x*. *f x* − *c*)
have ∀ *F w in at z*. *g w* ≠ *0* **if** *z* ∈ *S* **for** *z*
proof (*rule nconst_imp_nzero_neighbour'*[*of g S pts z*])
show *g meromorphic_on S pts* **using** *mero* **unfolding** *g_def*
by (*auto intro*:*meromorphic_intros*)
show ¬ (∀ *w*∈*S* − *pts*. *g w* = *0*) **using** *nconst* **unfolding** *g_def* **by** *auto*
qed *fact*+
then show *?thesis*
unfolding *discrete_altdef g_def*
using *eventually_mono* **by** *fastforce*
qed

lemma *meromorphic_isolated_in*:
assumes *merf*: *f meromorphic_on D pts p*∈*pts*
shows *p isolated_in pts*
by (*meson assms isolated_in_islimpt_iff meromorphic_on_def subsetD*)

lemma *remove_sings_constant_on*:
assumes *merf*: *f meromorphic_on D pts* **and** *connected D*
and *const*:*f constant_on* (*D* − *pts*)
shows (*remove_sings f*) *constant_on D*
proof −
have *remove_sings_const*: *remove_sings f constant_on D* − *pts*
using *const*
by (*metis constant_onE merf meromorphic_on_imp_analytic_at remove_sings_at_analytic*)

have *?thesis* **if** *D* = {}
using *that* **unfolding** *constant_on_def* **by** *auto*
moreover have *?thesis* **if** *D*≠{} {*x*∈*pts*. *is_pole f x*} = {}
proof −
obtain ξ **where** ξ ∈ (*D* − *pts*) ξ *islimpt* (*D* − *pts*)

```
  proof −
    have open (D − pts)
      using meromorphic_imp_open_diff[OF merf] .
    moreover have (D − pts) ≠ {} using ‹D≠{}›
      by (metis Diff_empty closure_empty merf
          meromorphic_pts_closure subset_empty)
    ultimately show ?thesis using open_imp_islimpt that by auto
  qed
  moreover have remove_sings f holomorphic_on D
    using remove_sings_holomorphic_on[OF merf] that by auto
  moreover note remove_sings_const
  moreover have open D
    using assms(1) meromorphic_on_def by blast
  ultimately show ?thesis
    using Conformal_Mappings.analytic_continuation'
          [of remove_sings f D D−pts ξ] ‹connected D›
    by auto
qed
moreover have ?thesis if D≠{} {x∈pts. is_pole f x} ≠ {}
proof −
  define PP where PP={x∈D. is_pole f x}
  have remove_sings f meromorphic_on D PP
    using merf unfolding PP_def
    apply (elim remove_sings_meromorphic_on)
    subgoal using assms(1) meromorphic_on_def by force
    subgoal using meromorphic_pole_subset merf by auto
    done
  moreover have remove_sings f constant_on D − PP
  proof −
    obtain ξ where ξ ∈ f ‘ (D − pts)
      by (metis Diff_empty Diff_eq_empty_iff ‹D ≠ {}› assms(1)
          closure_empty ex_in_conv imageI meromorphic_pts_closure)
    have ξ:∀ x∈D − pts. f x = ξ
      by (metis ‹ξ ∈ f ‘ (D − pts)› assms(3) constant_on_def image_iff)

    have remove_sings f x = ξ if x∈D − PP for x
    proof (cases x∈pts)
      case True
      then havex isolated_in pts
        using meromorphic_isolated_in[OF merf] by auto
      then obtain T0 where T0:open T0 T0 ∩ pts = {x}
        unfolding isolated_in_def by auto
      obtain T1 where T1:open T1 x∈T1 T1 ⊆ D
        using merf unfolding meromorphic_on_def
        using True by blast
      define T2 where T2 = T1 ∩ T0
      have open T2 x∈T2 T2 − {x} ⊆ D − pts
        using T0 T1 unfolding T2_def by auto
      then have ∀ w∈T2. w≠x ⟶ f w =ξ
```

```
        using ξ by auto
      then have ∀F x in at x. f x = ξ
        unfolding eventually_at_topological
        using ‹open T2› ‹x∈T2› by auto
      then have f −x→ ξ
        using tendsto_eventually by auto
      then show ?thesis by blast
    next
      case False
      then show ?thesis
        using ‹∀x∈D − pts. f x = ξ› assms(1)
          meromorphic_on_imp_analytic_at that by auto
    qed

    then show ?thesis unfolding constant_on_def by auto
  qed

  moreover have is_pole (remove_sings f) x if x∈PP for x
  proof −
    have isolated_singularity_at f x
      by (metis (mono_tags, lifting) DiffI PP_def assms(1)
          isolated_singularity_at_analytic mem_Collect_eq
          meromorphic_on_def meromorphic_on_imp_analytic_at that)
    then show ?thesis using that unfolding PP_def by simp
  qed
  ultimately show ?thesis
    using meromorphic_imp_constant_on
        [of remove_sings f D PP]
    by auto
 qed
 ultimately show ?thesis by auto
qed

lemma meromorphic_eq_meromorphic_extend:
 assumes f meromorphic_on A pts1 g meromorphic_on A pts1 ¬z islimpt pts2
 assumes ⋀z. z ∈ A − pts2 ⟹ f z = g z pts1 ⊆ pts2 z ∈ A − pts1
 shows   f z = g z
proof −
 have g analytic_on {z}
  using assms by (intro meromorphic_on_imp_analytic_at[OF assms(2)]) auto
 hence g −z→ g z
  using analytic_at_imp_isCont isContD by blast
 also have ?this ⟷ f −z→ g z
 proof (intro filterlim_cong)
  have eventually (λw. w ∉ pts2) (at z)
    using assms by (auto simp: islimpt_conv_frequently_at frequently_def)
  moreover have eventually (λw. w ∈ A) (at z)
      using assms by (intro eventually_at_in_open′) (auto simp: meromor-
phic_on_def)
```

```
    ultimately show ∀F x in at z. g x = f x
      by eventually_elim (use assms in auto)
  qed auto
  finally have f −z→ g z .
  moreover have f analytic_on {z}
   using assms by (intro meromorphic_on_imp_analytic_at[OF assms(1)]) auto
  hence f −z→ f z
    using analytic_at_imp_isCont isContD by blast
  ultimately show ?thesis
    using tendsto_unique by force
qed

lemma meromorphic_constant_on_extend:
  assumes f constant_on A − pts1 f meromorphic_on A pts1 f meromorphic_on
A pts2 pts2 ⊆ pts1
  shows   f constant_on A − pts2
proof −
  from assms(1) obtain c where c: ⋀z. z ∈ A − pts1 ⟹ f z = c
    unfolding constant_on_def by auto
  have f z = c if z ∈ A − pts2 for z
    using assms(3)
  proof (rule meromorphic_eq_meromorphic_extend[where z = z])
    show (λa. c) meromorphic_on A pts2
        by (intro meromorphic_on_const) (use assms in ‹auto simp: meromor-
phic_on_def›)
    show ¬z islimpt pts1
      using that assms by (auto simp: meromorphic_on_def)
  qed (use assms c that in auto)
  thus ?thesis
    by (auto simp: constant_on_def)
qed

lemma meromorphic_remove_sings_constant_on_imp_constant_on:
  assumes f meromorphic_on A pts
  assumes remove_sings f constant_on A
  shows   f constant_on A − pts
proof −
  from assms(2) obtain c where c: ⋀z. z ∈ A ⟹ remove_sings f z = c
    by (auto simp: constant_on_def)
  have f z = c if z ∈ A − pts for z
    using meromorphic_on_imp_analytic_at[OF assms(1) that] c[of z] that
    by auto
  thus ?thesis
    by (auto simp: constant_on_def)
qed
```

definition *singularities_on* :: *complex set* ⇒ (*complex* ⇒ *complex*) ⇒ *complex set*
where
singularities_on A f =
{*z*∈*A*. *isolated_singularity_at f z* ∧ *not_essential f z* ∧ ¬*f analytic_on* {*z*}}

lemma *singularities_on_subset*: *singularities_on A f* ⊆ *A*
by (*auto simp*: *singularities_on_def*)

lemma *pole_in_singularities_on*:
assumes *f meromorphic_on A pts z* ∈ *A is_pole f z*
shows *z* ∈ *singularities_on A f*
unfolding *singularities_on_def not_essential_def* **using** *assms*
using *analytic_at_imp_no_pole meromorphic_on_imp_isolated_singularity* **by** *force*

lemma *meromorphic_on_subset_pts*:
assumes *f meromorphic_on A pts pts'* ⊆ *pts f analytic_on pts* − *pts'*
shows *f meromorphic_on A pts'*
proof
show *open A pts'* ⊆ *A*
using *assms* **by** (*auto simp*: *meromorphic_on_def*)
show *isolated_singularity_at f z not_essential f z* **if** *z* ∈ *pts'* **for** *z*
using *assms that* **by** (*auto simp*: *meromorphic_on_def*)
show ¬*z islimpt pts'* **if** *z* ∈ *A* **for** *z*
using *assms that islimpt_subset* **unfolding** *meromorphic_on_def* **by** *blast*
have *f analytic_on A* − *pts*
using *assms*(*1*) *meromorphic_imp_analytic* **by** *blast*
with *assms* **have** *f analytic_on* (*A* − *pts*) ∪ (*pts* − *pts'*)
by (*subst analytic_on_Un*) *auto*
also have (*A* − *pts*) ∪ (*pts* − *pts'*) = *A* − *pts'*
using *assms* **by** (*auto simp*: *meromorphic_on_def*)
finally show *f holomorphic_on A* − *pts'*
using *analytic_imp_holomorphic* **by** *blast*
qed

lemma *meromorphic_on_imp_superset_singularities_on*:
assumes *f meromorphic_on A pts*
shows *singularities_on A f* ⊆ *pts*
proof
fix *z* **assume** *z* ∈ *singularities_on A f*
hence *z* ∈ *A* ¬*f analytic_on* {*z*}
by (*auto simp*: *singularities_on_def*)
with *assms* **show** *z* ∈ *pts*
by (*meson DiffI meromorphic_on_imp_analytic_at*)
qed

lemma *meromorphic_on_singularities_on*:
assumes *f meromorphic_on A pts*

```
  shows   f meromorphic_on A (singularities_on A f)
  using assms meromorphic_on_imp_superset_singularities_on[OF assms]
proof (rule meromorphic_on_subset_pts)
  have f analytic_on {z} if z ∈ pts − singularities_on A f for z
    using that assms by (auto simp: singularities_on_def meromorphic_on_def)
  thus f analytic_on pts − singularities_on A f
    using analytic_on_analytic_at by blast
qed

theorem Residue_theorem_inside:
  assumes f: f meromorphic_on s pts
           simply_connected s
  assumes g: valid_path g
           pathfinish g = pathstart g
           path_image g ⊆ s − pts
  defines pts1 ≡ pts ∩ inside (path_image g)
  shows finite pts1
    and contour_integral g f = 2 * pi * i * (∑p∈pts1. winding_number g p *
residue f p)
proof −
  note [dest] = valid_path_imp_path
  have cl_g [intro]: closed (path_image g)
    using g by (auto intro!: closed_path_image)
  have open s
    using f(1) by (auto simp: meromorphic_on_def)
  define pts2 where pts2 = pts − pts1

  define A where A = path_image g ∪ inside (path_image g)
  have closed A
    unfolding A_def using g by (intro closed_path_image_Un_inside) auto
  moreover have bounded A
   unfolding A_def using g by (auto intro!: bounded_path_image bounded_inside)
  ultimately have 1: compact A
    using compact_eq_bounded_closed by blast
  have 2: open (s − pts2)
   using f by (auto intro!: meromorphic_imp_open_diff' [OF f(1)] simp: pts2_def)
  have 3: A ⊆ s − pts2
    unfolding A_def pts2_def pts1_def
   using f(2) g(3) 2 subset_simply_connected_imp_inside_subset[of s path_image
g] ‹open s›
    by auto

  obtain ε where ε: ε > 0 (⋃x∈A. ball x ε) ⊆ s − pts2
    using compact_subset_open_imp_ball_epsilon_subset[OF 1 2 3] by blast
  define B where B = (⋃x∈A. ball x ε)

  have finite (A ∩ pts)
    using 1 3 by (intro meromorphic_compact_finite_pts[OF f(1)]) auto
  also have A ∩ pts = pts1
```

```
    unfolding pts1_def using g by (auto simp: A_def)
  finally show fin: finite pts1 .

  show contour_integral g f = 2 * pi * i * (∑p∈pts1. winding_number g p *
residue f p)
  proof (rule Residue_theorem)
    show open B
      by (auto simp: B_def)
  next
    have connected A
      unfolding A_def using g
      by (intro connected_with_inside closed_path_image connected_path_image)
auto
    hence connected (A ∪ B)
      unfolding B_def using g ‹ε > 0› f(2)
      by (intro connected_Un_UN connected_path_image valid_path_imp_path)
        (auto simp: simply_connected_imp_connected)
    also have A ∪ B = B
      using ε(1) by (auto simp: B_def)
    finally show connected B .
  next
    have f holomorphic_on (s - pts)
      by (intro meromorphic_imp_holomorphic f)
    moreover have B - pts1 ⊆ s - pts
      using ε unfolding B_def by (auto simp: pts1_def pts2_def)
    ultimately show f holomorphic_on (B - pts1)
      by (rule holomorphic_on_subset)
  next
    have path_image g ⊆ A - pts1
      using g unfolding pts1_def by (auto simp: A_def)
    also have ... ⊆ B - pts1
      unfolding B_def using ε(1) by auto
    finally show path_image g ⊆ B - pts1 .
  next
    show ∀z. z ∉ B ⟶ winding_number g z = 0
    proof safe
      fix z assume z: z ∉ B
      hence z ∉ A
        using ε(1) by (auto simp: B_def)
      hence z ∈ outside (path_image g)
        unfolding A_def by (simp add: union_with_inside)
      thus winding_number g z = 0
        using g by (intro winding_number_zero_in_outside) auto
    qed
  qed (use g fin in auto)
qed

theorem Residue_theorem':
  assumes f: f meromorphic_on s pts
```

```
        simply_connected s
  assumes g: valid_path g
        pathfinish g = pathstart g
        path_image g ⊆ s − pts
  assumes pts': finite pts'
          pts' ⊆ s
          ⋀z. z ∈ pts − pts' ⟹ winding_number g z = 0
  shows contour_integral g f = 2 * pi * i * (∑ p∈pts'. winding_number g p *
residue f p)
proof −
  note [dest] = valid_path_imp_path
  define pts1 where pts1 = pts ∩ inside (path_image g)

  have contour_integral g f = 2 * pi * i * (∑ p∈pts1. winding_number g p *
residue f p)
    unfolding pts1_def by (intro Residue_theorem_inside[OF f g])
  also have (∑ p∈pts1. winding_number g p * residue f p) =
        (∑ p∈pts'. winding_number g p * residue f p)
  proof (intro sum.mono_neutral_cong refl)
    show finite pts1
      unfolding pts1_def by (intro Residue_theorem_inside[OF f g])
    show finite pts'
      by fact
  next
    fix z assume z: z ∈ pts' − pts1
    show winding_number g z * residue f z = 0
    proof (cases z ∈ pts)
      case True
      with z have z ∉ path_image g ∪ inside (path_image g)
        using g(3) by (auto simp: pts1_def)
      hence z ∈ outside (path_image g)
        by (simp add: union_with_inside)
      hence winding_number g z = 0
        using g by (intro winding_number_zero_in_outside) auto
      thus ?thesis
        by simp
    next
      case False
      with z pts' have z ∈ s − pts
        by auto
      with f(1) have f analytic_on {z}
        by (intro meromorphic_on_imp_analytic_at)
      hence residue f z = 0
        using analytic_at residue_holo by blast
      thus ?thesis
        by simp
    qed
  next
    fix z assume z: z ∈ pts1 − pts'
```

```
    hence winding_number g z = 0
      using pts' by (auto simp: pts1_def)
    thus winding_number g z * residue f z = 0
      by simp
  qed
  finally show ?thesis .
qed

end
theory Complex_Analysis
imports
  Residue_Theorem
  Meromorphic
begin

end
```

References

[1]