Complex Analysis

October 24, 2026

Contents

1 Contour integration **6**
- 1.1 Definition . . . 6
- 1.2 Reversing a path . . . 9
- 1.3 Joining two paths together . . . 10
- 1.4 Shifting the starting point of a (closed) path . . . 14
- 1.5 More about straight-line paths . . . 17
- 1.6 Relation to subpath construction . . . 18
- 1.7 Cauchy's theorem where there's a primitive . . . 22
- 1.8 Arithmetical combining theorems . . . 24
- 1.9 Operations on path integrals . . . 25
- 1.10 Arithmetic theorems for path integrability . . . 27
- 1.11 Reversing a path integral . . . 30
- 1.12 Reversing the order in a double path integral . . . 33
- 1.13 Partial circle path . . . 35
- 1.14 Special case of one complete circle . . . 45
- 1.15 Uniform convergence of path integral . . . 50

2 Complex Path Integrals and Cauchy's Integral Theorem **53**
- 2.1 The key quadrisection step . . . 55
- 2.2 Cauchy's theorem for triangles . . . 57
- 2.3 Version needing function holomorphic in interior only . . . 62
- 2.4 Version allowing finite number of exceptional points . . . 69
- 2.5 Cauchy's theorem for an open starlike set . . . 71
- 2.6 Cauchy's theorem for a convex set . . . 74
- 2.7 Generalize integrability to local primitives . . . 77
- 2.8 Homotopy forms of Cauchy's theorem . . . 86

3 Winding numbers **91**
- 3.1 Definition . . . 91
  - 3.1.1 Useful sufficient conditions for the winding number to be positive . . . 99

3.2 The winding number is an integer . . . 101
3.3 Continuity of winding number and invariance on connected sets 107
3.4 The winding number is constant on a connected region . . . 111
3.5 Winding number is zero "outside" a curve . . . 112
3.6 More winding number properties . . . 119
3.7 Winding number for a triangle . . . 124
3.8 Winding numbers for simple closed paths . . . 127
3.9 Winding number for rectangular paths . . . 143

4 Cauchy's Integral Formula 146
4.1 Proof . . . 146
4.2 General stepping result for derivative formulas . . . 148
4.3 Existence of all higher derivatives . . . 154
4.4 Morera's theorem . . . 158
4.5 Combining theorems for higher derivatives including Leibniz rule . . . 160
4.6 A holomorphic function is analytic, i.e. has local power series 169
4.7 The Liouville theorem and the Fundamental Theorem of Algebra . . . 172
4.8 Weierstrass convergence theorem . . . 174
4.9 Some more simple/convenient versions for applications . . . 181
4.10 More on winding numbers . . . 182
4.11 On analytic functions defined by a series . . . 184
4.12 Equality between holomorphic functions, on open ball then connected set . . . 189
4.13 Some basic lemmas about poles/singularities . . . 190
4.14 General, homology form of Cauchy's theorem . . . 194
4.15 Cauchy's inequality and more versions of Liouville . . . 207
4.16 The uniform Weierstrass convergence theorem . . . 212
4.17 Complex functions and power series . . . 219

5 Conformal Mappings and Consequences of Cauchy's Integral Theorem 225
5.1 Analytic continuation . . . 226
5.2 Open mapping theorem . . . 229
5.3 Maximum modulus principle . . . 233
5.4 Factoring out a zero according to its order . . . 235
5.5 Entire proper functions are precisely the non-trivial polynomials 244
5.6 Relating invertibility and nonvanishing of derivative . . . 246
5.6.1 Hence a nice clean inverse function theorem . . . 249
5.6.2 Holomorphism of covering maps and lifts. . . . 250
5.7 The Schwarz Lemma . . . 252
5.8 The Schwarz reflection principle . . . 255
5.9 Bloch's theorem . . . 261

6 The Great Picard Theorem and its Applications 270
6.1 Schottky's theorem . . . 270
6.2 The Little Picard Theorem . . . 279
6.3 The Arzelà–Ascoli theorem . . . 284
6.3.1 Montel's theorem . . . 287
6.4 Some simple but useful cases of Hurwitz's theorem . . . 292
6.5 The Great Picard theorem . . . 297

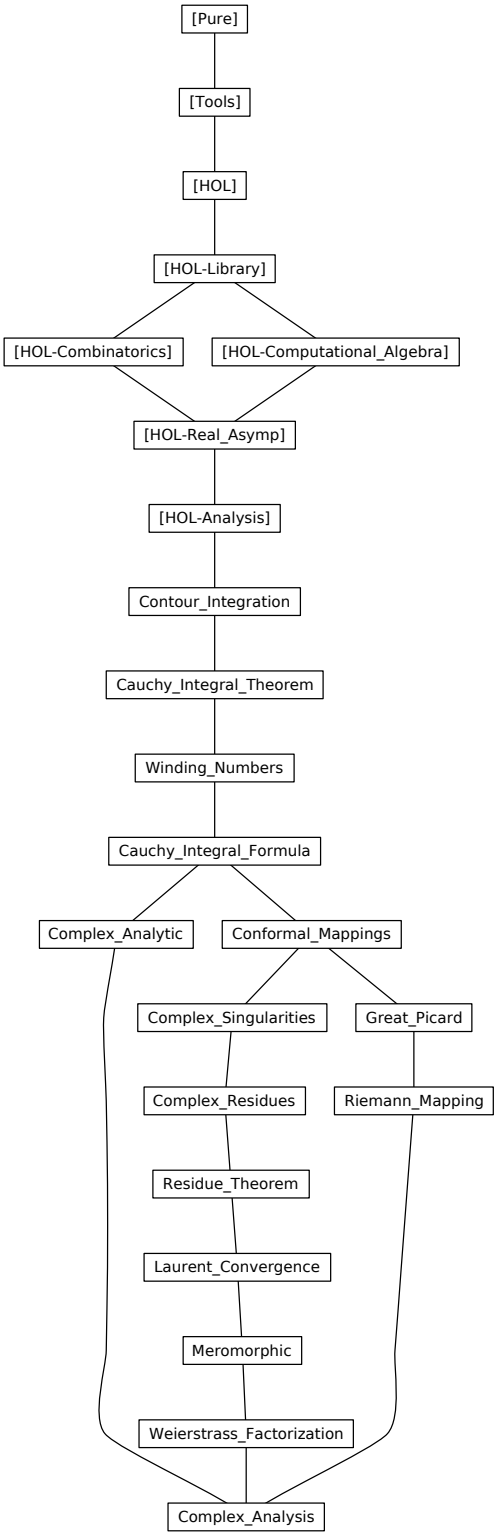
7 Moebius functions, Equivalents of Simply Connected Sets, Riemann Mapping Theorem 311
7.1 Moebius functions are biholomorphisms of the unit disc . . . 311
7.2 A big chain of equivalents of simple connectedness for an open set . . . 313
7.3 A further chain of equivalences about components of the complement of a simply connected set . . . 329
7.4 Further equivalences based on continuous logs and sqrts . . . 337
7.5 More Borsukian results . . . 340
7.6 Finally, the Riemann Mapping Theorem . . . 341
7.7 Applications to Winding Numbers . . . 342
7.8 The winding number defines a continuous logarithm for the path itself . . . 342
7.9 Winding number equality is the same as path/loop homotopy in C - 0 . . . 345
7.10 Non-essential singular points . . . 348
7.11 Isolated singularities . . . 353
7.12 The order of non-essential singularities (i.e. removable singularities or poles) . . . 372
7.13 Isolated points . . . 403
7.14 Isolated zeros . . . 404
7.15 Definition of residues . . . 406
7.16 Poles and residues of some well-known functions . . . 418

8 The Residue Theorem, the Argument Principle and Rouché's Theorem 419
8.1 Cauchy's residue theorem . . . 420
8.2 The argument principle . . . 430
8.3 Coefficient asymptotics for generating functions . . . 435
8.4 Rouche's theorem . . . 438
8.5 More Laurent expansions . . . 495
8.6 Formal convergence versus analytic convergence . . . 496
8.7 Remove singular points . . . 497
8.8 Meromorphicity . . . 504
8.9 Nice meromorphicity . . . 512
8.10 Closure properties and proofs for individual functions . . . 516

8.11 Meromorphic functions and zero order 525
8.12 More on poles and zeros 530
8.13 A relation for the zero order of a function 539

9 The Weierstraß Factorisation Theorem 548
9.1 The elementary factors 548
9.2 Infinite products of elementary factors 552
9.3 Writing a quotient as an exponential 560
9.4 Constructing the sequence of zeros 563
9.5 The factorisation theorem for holomorphic functions 569
9.6 The factorisation theorem for meromorphic functions 576

10 Complex Analyticity and the Complexification of Real-Analytic Functions 584
10.1 A CR-linear self-map of the plane is multiplication by a scalar 584
10.2 Several-variable Cauchy–Riemann criterion 585
10.3 Real analyticity via holomorphic extension 587


[Pure]
[Tools]
[HOL]
[HOL-Library]
[HOL-Combinatorics]
[HOL-Computational_Algebra]
[HOL-Real_Asymp]
[HOL-Analysis]
Contour_Integration
Cauchy_Integral_Theorem
Winding_Numbers
Cauchy_Integral_Formula
Complex_Analytic
Conformal_Mappings
Complex_Singularities
Great_Picard
Complex_Residues
Riemann_Mapping
Residue_Theorem
Laurent_Convergence
Meromorphic
Weierstrass_Factorization
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
unfolding *contour_integral_def has_contour_integral_def contour_integrable_on_def*
using *has_integral_unique* **by** *blast*

lemma *has_contour_integral_cong*:
assumes $\bigwedge z.\ z \in path_image\ g \Longrightarrow f\ z = f'\ z\ g = g'\ c = c'$
shows $(f\ has_contour_integral\ c)\ g \longleftrightarrow (f'\ has_contour_integral\ c')\ g'$
unfolding *has_contour_integral_def assms*(*2*,*3*)
by (*intro has_integral_cong*) (*auto simp*: *assms path_image_def intro*!: *assms*(*1*))

lemma *has_contour_integral_eqpath*:
$[\![(f\ has_contour_integral\ y)\ p;\ f\ contour_integrable_on\ \gamma;$
contour_integral p f = contour_integral γ f$]\!]$
$\Longrightarrow (f\ has_contour_integral\ y)\ \gamma$
using *contour_integrable_on_def contour_integral_unique* **by** *auto*

lemma *has_contour_integral_integral*:
f contour_integrable_on i $\Longrightarrow$ (*f has_contour_integral* (*contour_integral i f*))
i
by (*metis contour_integral_unique contour_integrable_on_def*)

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

lemma *has_contour_integral_translate*:
(*f has_contour_integral I*) ((+) *z* ∘ *g*) ⟷ ((λ*x*. *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*) ⟷ (λ*x*. *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* (λ*x*. *f* (*x* + *z*))
by (*simp add*: *contour_integral_def contour_integrable_translate has_contour_integral_translate*)

lemma *has_contour_integral_integrable*: (*f has_contour_integral i*) *g* $\Longrightarrow$ *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*:
$(\lambda x.\ f\ (g\ x) * vector_derivative\ p\ (at\ x\ within\ \{a..b\}))\ integrable_on\ \{a..b\} \longleftrightarrow$
$(\lambda x.\ f\ (g\ x) * vector_derivative\ p\ (at\ x))\ integrable_on\ \{a..b\}$
by (*simp add*: *integrable_on_def has_integral_localized_vector_derivative*)

lemma *has_contour_integral*:
$(f\ has_contour_integral\ i)\ g \longleftrightarrow$
$((\lambda x.\ f\ (g\ x) * vector_derivative\ g\ (at\ x))\ has_integral\ i)\ \{0..1\}$
by (*simp add*: *has_integral_localized_vector_derivative has_contour_integral_def*)

lemma *contour_integrable_on*:
$f\ contour_integrable_on\ g \longleftrightarrow$
$(\lambda t.\ f(g\ t) * vector_derivative\ g\ (at\ t))\ integrable_on\ \{0..1\}$
by (*simp add*: *has_contour_integral integrable_on_def contour_integrable_on_def*)

lemma *has_contour_integral_mirror_iff*:
assumes *valid_path g*
shows $(f\ has_contour_integral\ I)\ (-g) \longleftrightarrow ((\lambda x.\ -f\ (-\ x))\ has_contour_integral\ I)\ g$
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: $\bigwedge x.\ x \in \{0..1\} - S \Longrightarrow g\ differentiable\ at\ x\ within\ \{0..1\}$
unfolding *piecewise_differentiable_on_def* **by** *blast*
have S': *g differentiable at x* **if** $x \in \{0..1\} - (\{0,\ 1\} \cup S)$ **for** x
proof −
from *that* **have** $x \in interior\ \{0..1\}$ **by** *auto*
with S[*of x*] *that* **show** *?thesis* **by** (*auto simp*: *at_within_interior*[*of* _ $\{0..1\}$])
qed
have $(f\ has_contour_integral\ I)\ (-g) \longleftrightarrow$
$((\lambda x.\ f\ (-\ g\ x) * vector_derivative\ (-g)\ (at\ x))\ has_integral\ I)\ \{0..1\}$
by (*simp add*: *has_contour_integral*)
also have ... $\longleftrightarrow ((\lambda x.\ -f\ (-\ g\ x) * vector_derivative\ g\ (at\ x))\ has_integral\ I)\ \{0..1\}$
by (*intro has_integral_spike_finite_eq*[*of* $S \cup \{0,\ 1\}$])
(*insert* ‹*finite* S› S', *auto simp*: *o_def fun_Compl_def*)
also have ... $\longleftrightarrow ((\lambda x.\ -f\ (-x))\ has_contour_integral\ I)\ g$
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* $(-g) \longleftrightarrow (\lambda x.\ -f\ (-\ x))$ *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* $(\lambda x.\ -f\ (-\ x))$
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* $(\{0..1\} - S)$ $x \notin (-)\ 1\ `\ S\ 0 \le x\ x \le 1$
have *vector_derivative* $(\lambda x.\ g\ (1 - x))$ (*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 $(g \circ (\lambda x.\ 1 - x)$ *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'*: $((\lambda x.\ g\ (1 - x))$ *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 $((\lambda x.\ -\ (f\ (g\ (1 - x)) * vector_derivative\ g\ (at\ (1 - x)\ within\ \{0..1\})))$ *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 = (λx. 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: ((λx. f (g1 x) ∗ vector_derivative g1 (at x)) has_integral i1) {0..1}
   and 2: ((λx. f (g2 x) ∗ vector_derivative g2 (at x)) has_integral i2) {0..1}
    using assms
    by (auto simp: has_contour_integral)
```

```
  have i1: ((λx. (2*f (g1 (2*x))) * vector_derivative g1 (at (2*x))) has_integral
i1) {0..1/2}
    and i2: ((λx. (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 (λx. 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 (λx. 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 *: (λx. (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 (λx. 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 §: ((λx. 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 ((λx. 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 (λx. 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 *: (λx. (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 (λx. if x*2 ≤ 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 ∉ s2 for z
 proof (rule vector_derivative_at [OF has_vector_derivative_transform_within])
    show 0 < |z/2|
      using that by auto
    have §: ((λx. 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 ((λx. 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} ∪ 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*⟧
⟹ *f contour_integrable_on* (*g1 +++ g2*) ⟷ *f contour_integrable_on g1* ∧ *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*⟧
⟹ *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*))
by (*smt* (*verit*, *ccfv_threshold*) *Henstock_Kurzweil_Integration.integral_combine a add.commute atLeastAtMost_iff has_integral_iff*)
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 ((λx. 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 ∉ (λx. x - a + 1) ` S for x
  unfolding shiftpath_def
proof (rule vector_derivative_at [OF has_vector_derivative_transform_within])
  have ((λx. 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: va1) (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 integrable_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*} (λ*x*. *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 §: (λ*t*. *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 ∗: ((λ*x*. *f* (*g* ((*v* − *u*) ∗ *x* + *u*)) ∗ *vector_derivative g* (*at* ((*v* − *u*) ∗ *x* + *u*))) *has_integral* (*1* / (*v* − *u*)) ∗ *integral* {*u..v*} (λ*t*. *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* (λ*x*. *g* ((*v*−*u*) ∗ *x* + *u*)) (*at x*) = (*v*−*u*) ∗$_R$ *vector_derivative g* (*at* ((*v*−*u*) ∗ *x* + *u*))
if *x* ∈ {*0..1*} *x* ∉ (λ*t*. (*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)
  by (smt (verit, ccfv_threshold) assms contour_integrable_on_def contour_integrable_reversepath_eq
      has_contour_integral_subpath reversepath_subpath valid_path_subpath)

lemma has_integral_contour_integral_subpath:
  assumes f contour_integrable_on g valid_path g u ∈ {0..1} v ∈ {0..1} u ≤ v
    shows ((λx. f(g x) ∗ vector_derivative g (at x))
           has_integral  contour_integral (subpath u v g) f) {u..v}
          (is (?fg has_integral _)_)
proof −
  have (?fg has_integral integral {u..v} ?fg) {u..v}
    using assms contour_integrable_on integrable_on_subinterval by fastforce
  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} (λx. 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
  by (smt (verit) Henstock_Kurzweil_Integration.integral_combine assms
      has_integral_contour_integral_subpath has_integral_iff)

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*)
with *assms* **show** $f'\ (g'\ x) * \textit{vector_derivative}\ g'\ (\textit{at}\ x) = f\ (g\ x) * \textit{vector_derivative}\ g\ (\textit{at}\ x)$
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*) ⟷
$((\lambda x.\ f\ (\textit{of_real}\ x))$ *has_integral I*) {*Re a..Re b*}
proof −
have [*simp*]: *of_real* (*Re a*) = *a of_real* (*Re b*) = *b* **and** *a* ≠ *b*
using *assms* **by** (*simp_all add*: *complex_eq_iff*)
have $((\lambda x.\ f\ (\textit{of_real}\ x))$ *has_integral I*) (*cbox* (*Re a*) (*Re b*)) ⟷
$((\lambda x.\ f\ (a + b * \textit{of_real}\ x - a * \textit{of_real}\ x))$ *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 $(\lambda x.\ f\ (a + b * \textit{of_real}\ x - a * \textit{of_real}\ x)) =$
$(\lambda x.\ (f\ (a + b * \textit{of_real}\ x - a * \textit{of_real}\ x) * (b - a))\ /_R$ (*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*} ⟷
$((\lambda x.\ f\ (\textit{linepath}\ a\ b\ x) * (b - a))$ *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*
using *has_contour_integral_def has_contour_integral_linepath* **by** *presburger*
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*) ⟷
$(\lambda x.\ f\ (\textit{of_real}\ x))$ *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*
by (*metis assms has_contour_integral_integral has_contour_integral_linepath_Reals_iff integral_unique*)
next
case *False*
thus *?thesis*
by (*simp add*: *assms contour_integrable_linepath_Reals_iff 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 DERIV_continuous_on continuous_on_subset image_subsetI*)
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)
  } then show ?thesis
    using assms cfg ‹finite K›
   by (force simp: at_within_Icc_at intro: fundamental_theorem_of_calculus_interior_strong
[OF countable_finite])
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)


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*)) ∗ *measure lborel* (*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_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 $\Longrightarrow$ ($\lambda x.\ -(f\ x)$) *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*⟧ $\Longrightarrow$ ($\lambda x.$ *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*⟧ $\Longrightarrow$ ($\lambda x.$ *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 $\Longrightarrow$ ($\lambda x.\ c * f\ x$) *contour_integrable_on g*
using *has_contour_integral_lmul contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_rmul*:
f contour_integrable_on g $\Longrightarrow$ ($\lambda x.\ f\ x * c$) *contour_integrable_on g*
using *has_contour_integral_rmul contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_div*:
f contour_integrable_on g $\Longrightarrow$ ($\lambda x.\ f\ x\ /\ c$) *contour_integrable_on g*
using *has_contour_integral_div contour_integrable_on_def*
by *fastforce*

lemma *contour_integrable_sum*:
⟦*finite s*; $\bigwedge a.\ a \in s \Longrightarrow$ (*f a*) *contour_integrable_on p*⟧
$\Longrightarrow$ ($\lambda x.$ *sum* ($\lambda a.$ *f a x*) *s*) *contour_integrable_on p*
unfolding *contour_integrable_on_def* **by** (*metis has_contour_integral_sum*)

lemma *contour_integrable_neg_iff*:
($\lambda x.\ -f\ x$) *contour_integrable_on g* $\longleftrightarrow$ *f contour_integrable_on g*
using *contour_integrable_neg*[*of f g*] *contour_integrable_neg*[*of* $\lambda x.\ -f\ x\ g$] **by** *auto*

lemma *contour_integrable_lmul_iff*:
c $\neq$ *0* $\Longrightarrow$ ($\lambda x.\ c * f\ x$) *contour_integrable_on g* $\longleftrightarrow$ *f contour_integrable_on g*
using *contour_integrable_lmul*[*of f g c*] *contour_integrable_lmul*[*of* $\lambda x.\ c * f\ x$

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*)

lemma *contour_integral_rmul*: *contour_integral g* (λ*x*. *f x* ∗ *c*) = *contour_integral g f* ∗ *c*
proof (*cases c* = *0*)
case [*simp*]: *False*
show *?thesis*
proof (*cases f contour_integrable_on g*)
case *True*
thus *?thesis*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_rmul*)
next
case *False*
thus *?thesis*
using *contour_integrable_rmul_iff not_integrable_contour_integral* **by** *force*
qed
qed *auto*

lemma *contour_integral_lmul*: *contour_integral g* (λ*x*. *c* ∗ *f x*) = *c* ∗ *contour_integral g f*
by (*subst* (*1 2*) *mult.commute*) (*rule contour_integral_rmul*)

lemma *contour_integral_divide*: *contour_integral g* (λ*x*. *f x* / *c*) = *contour_integral g f* / *c*
using *contour_integral_rmul*[*of g f inverse c*] **by** (*simp add*: *field_simps*)

lemma *uniform_limit_contour_integral_linepath*:
assumes *u*: *uniform_limit* (*path_image* (*linepath a b*)) *f g F*
assumes *c*: ⋀*n*. *continuous_on* (*path_image* (*linepath a b*)) (*f n*)
assumes [*simp*]: *F* ≠ *bot*
obtains *I J* **where**
⋀*n*. (*f n has_contour_integral I n*) (*linepath a b*)
(*g has_contour_integral J*) (*linepath a b*)
(*I* ⟶ *J*) *F*
proof (*rule uniform_limit_integral*)

```
  note [continuous_intros] = continuous_on_compose2[OF c]

  show uniform_limit {0..1} (λx t. f x (linepath a b t) * (b - a))
        (λt. g (linepath a b t) * (b - a)) F
  proof (rule uniform_limit_intros)
    show uniform_limit {0..1} (λx t. f x (linepath a b t))
          (λt. g (linepath a b t)) F
      using u unfolding path_image_def by (rule uniform_limit_compose') auto
  qed

  show continuous_on {0..1} (λt. f n (linepath a b t) * (b - a)) for n
    by (intro continuous_intros; unfold path_image_def) auto

  fix I J
  assume I: ⋀n. ((λt. f n (linepath a b t) * (b - a)) has_integral I n) {0..1}
     and J: ((λt. g (linepath a b t) * (b - a)) has_integral J) {0..1}
     and lim: (I ⟶ J) F
  show ?thesis
   by (rule that[of I J]) (use I J lim in ‹auto simp: has_contour_integral›)
qed auto


lemma contour_integral_sums_linepath:
  assumes u: uniform_limit (closed_segment a b) (λN w. ∑n<N. f n w) g se-
quentially
  assumes c: ⋀n. continuous_on (closed_segment a b) (f n)
  obtains J where
    (g has_contour_integral J) (linepath a b)
    (λn. contour_integral (linepath a b) (f n)) sums J
proof (rule uniform_limit_contour_integral_linepath)
  show uniform_limit (path_image (linepath a b)) (λN w. ∑n<N. f n w) g se-
quentially
    using u by simp
next
  show continuous_on (path_image (linepath a b)) (λw. ∑n<N. f n w) for N
    by (intro continuous_intros continuous_on_subset[OF c]) simp_all
next
  fix I J
  assume 1: ⋀N. ((λw. ∑n<N. f n w) has_contour_integral I N) (linepath a b)
  assume 2: (g has_contour_integral J) (linepath a b) and 3: (I ⟶ J) sequen-
tially
  have 4: I = (λN. (∑n<N. contour_integral (linepath a b) (f n)))
  proof
    fix N :: nat
    have f n contour_integrable_on (linepath a b) for n
      by (intro contour_integrable_continuous_linepath assms)
    hence ((λw. ∑n<N. f n w) has_contour_integral
           (∑n<N. contour_integral (linepath a b) (f n))) (linepath a b)
     using c by (intro has_contour_integral_sum) (simp_all add: has_contour_integral_integral)
```

with *1*[*of N*] **show** *I N* = (∑ *n*<*N*. *contour_integral* (*linepath a b*) (*f n*))
using *contour_integral_unique* **by** *metis*
qed
have *5*: (λ*n*. *contour_integral* (*linepath a b*) (*f n*)) *sums J*
using *1 2 3 4* **unfolding** *sums_def* **by** *blast*
from *that*[*OF 2 5*] **show** *?thesis* **.**
qed *auto*

lemma *contour_integral_linepath_same_Re*:
assumes *Re z* = *c Re z′* = *c Im z* = *a Im z′* = *b a* < *b*
shows *contour_integral* (*linepath z z′*) *f* =
i ∗ *integral* {*a*..*b*} (λ*x*. *f* (*Complex c x*))
proof −
have *zz′*: *z* = *Complex c a z′* = *Complex c b*
using *assms* **by** (*auto simp*: *complex_eq_iff*)
have *contour_integral* (*linepath z z′*) *f* =
(*z′* − *z*) ∗ *integral* {*0*..*1*} (λ*x*. *f* (*linepath z z′ x*))
by (*simp add*: *contour_integral_integral*)
also have *z′* − *z* = i ∗ *of_real* (*b* − *a*)
by (*simp add*: *zz′ Complex_eq algebra_simps*)
also have *integral* {*0*..*1*} (λ*x*. *f* (*linepath z z′ x*)) =
integral {*0*..*1*} (λ*x*. *f* (*Complex c* (*linepath a b x*)))
by (*simp add*: *linepath_def Complex_eq scaleR_conv_of_real algebra_simps zz′*)
also have ... = *integral* {*0*..(*b* − *a*) / (*b* − *a*)} (λ*x*. *f* (*Complex c* (*a* + (*b* − *a*) ∗ *x*)))
using ‹*a* < *b*› **by** (*simp add*: *algebra_simps linepath_def*)
also have {*0*..(*b* − *a*) / (*b* − *a*)} = (λ*x*. *x* / (*b* − *a*)) ‘ {*0*..*b* − *a*}
using ‹*a* < *b*› **by** *simp*
also have *integral* ... (λ*x*. *f* (*Complex c* (*a* + (*b* − *a*) ∗ *x*))) =
integral {*a*−*a*..*b*−*a*} (λ*x*. *f* (*Complex c* (*x* + *a*))) / *of_real* (*b* − *a*)
using ‹*a* < *b*› **by** (*subst integral_stretch_real*) (*auto simp*: *scaleR_conv_of_real add_ac*)
also have ... = *integral* {*a*..*b*} (λ*x*. *f* (*Complex c x*)) / *of_real* (*b* − *a*)
by (*subst integral_shift_real_ivl*) (*rule refl*)
finally show *?thesis*
using ‹*a* < *b*› **by** *simp*
qed

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
    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
      apply (simp add: c' scaleR_conv_of_real 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
  }
  then show ?thesis
```

using *f k* **unfolding** *has_contour_integral_linepath*
by (*simp add*: *linepath_def has_integral_combine* [*OF _ _ fi*])
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*])
then have (*f has_contour_integral*
contour_integral (*linepath a c*) *f* + *contour_integral* (*linepath c b*) *f*) (*linepath a b*)
by (*meson c contour_integrable_continuous_linepath*
has_contour_integral_integral has_contour_integral_split k)
then show *?thesis*
by (*metis contour_integral_unique*)
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*} (λ*t*. *vector_derivative h* (*at t*))
shows *contour_integral g* (λ*w*. *contour_integral h* (*f w*)) =
contour_integral h (λ*z*. *contour_integral g* (λ*w*. *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*. (λ*t*. *f* (*g x*) (*h t*)) = (λ(*y1*,*y2*). *f y1 y2*) ∘ (λ*t*. (*g x*, *h t*))
by (*rule ext*) *simp*
have *fgh2*: ⋀*x*. (λ*t*. *f* (*g t*) (*h x*)) = (λ(*y1*,*y2*). *f y1 y2*) ∘ (λ*t*. (*g t*, *h x*))
by (*rule ext*) *simp*
have *fcon_im1*: ⋀*x*. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *continuous_on* ((λ*t*. (*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* ((λ*t*. (*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*)) ((λ*x*. *vector_derivative g* (*at x*)) ∘ *fst*)
continuous_on (*cbox* (*0*, *0*) (*1*::*real*, *1*)) ((λ*x*. *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*)) (λ*z*. *vector_derivative g* (*at* (*fst z*)))
and *hvcon′*: *continuous_on* (*cbox* (*0*, *0*) (*1*::*real*, *1*)) (λ*x*. *vector_derivative h* (*at* (*snd x*)))
by *auto*
have *continuous_on* ((λ*x*. (*g* (*fst x*), *h* (*snd x*))) ‘ *cbox* (*0*,*0*) (*1*,*1*)) (λ(*y1*, *y2*). *f y1 y2*)
by (*auto simp*: *path_image_def intro*: *continuous_on_subset* [*OF fcon*])
then have *continuous_on* (*cbox* (*0*, *0*) (*1*, *1*)) ((λ(*y1*, *y2*). *f y1 y2*) ∘ (λ*w*. ((*g* ∘ *fst*) *w*, (*h* ∘ *snd*) *w*)))

```
    by (intro gcon hcon continuous_intros | simp)+
  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)+
  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)+
    by (simp add: mult.commute mult.left_commute)
  also have ... = contour_integral h (λz. contour_integral g (λw. 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: ((λz. 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 ((λx. − (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 ((λx. 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 $f\ (-\ \gamma\ x) * vector_derivative\ (uminus \circ \gamma)\ (at\ x\ within\ \{0..1\}) =$
$-\ (f\ (-\ \gamma\ x) * vector_derivative\ \gamma\ (at\ x\ within\ \{0..1\}))$
if $x \in \{0..1\} - S$ **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 *not_on_circlepathI*:

assumes *cmod* $(z-z0) \neq |r|$
shows $z \notin$ *path_image* (*part_circlepath z0 r st tt*)
using *assms*
by (*auto simp add*: *path_image_def image_def part_circlepath_def norm_mult*)

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

lemma *pathfinish_part_circlepath* [*simp*]:
pathfinish(*part_circlepath z r s t*) = $z + r*exp(\mathrm{i}*t)$
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*
($\mathrm{i} * r * (of_real\ t - of_real\ s) * exp(\mathrm{i} * linepath\ s\ t\ x)$))
(*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*) =
$\mathrm{i} * r * (of_real\ t - of_real\ s) * exp(\mathrm{i} * linepath\ s\ t\ x)$
using *has_vector_derivative_part_circlepath vector_derivative_at* **by** *blast*

lemma *vector_derivative_part_circlepath01*:
$[\![0 \le x; x \le 1]\!]$
$\Longrightarrow$ *vector_derivative* (*part_circlepath z r s t*) (*at x within* $\{0..1\}$) =
$\mathrm{i} * r * (of_real\ t - of_real\ s) * exp(\mathrm{i} * linepath\ s\ t\ x)$
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 $s \leq t$
shows *path_image* (*part_circlepath z r s t*) = {$z + r * exp(\mathrm{i} * of_real\ x) \mid x.$ $s \leq x \wedge x \leq t$}
proof −
{ fix *z*::*real*
assume $0 \leq z\ z \leq 1$
with ‹$s \leq t$› **have** $\exists x.\ (exp\ (\mathrm{i} * linepath\ s\ t\ z) = exp\ (\mathrm{i} * of_real\ x)) \wedge s \leq x \wedge x \leq t$
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 $s \leq z\ z \leq t$
then have $z + of_real\ r * exp\ (\mathrm{i} * of_real\ z) \in (\lambda x.\ z + of_real\ r * exp\ (\mathrm{i} * linepath\ s\ t\ x))$ ‘ {*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*) = $(\lambda x.\ z + r * cis\ x)$ ‘ *closed_segment s t*
by (*metis* (*no_types*, *lifting*) *ext cis_conv_exp image_image linepath_image_01 part_circlepath_def path_image_def*)

lemma *path_image_part_circlepath_subset*:
$[\![s \leq t;\ 0 \leq r]\!] \Longrightarrow$ *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 $w \in$ *path_image*(*part_circlepath z r s t*) $s \leq t\ 0 \leq r$
shows *norm*$(w - z) = r$
by (*smt* (*verit*) *assms dist_norm mem_Collect_eq norm_minus_commute path_image_part_circlepath_sub sphere_def subsetD*)

lemma *path_image_part_circlepath_subset′*:
assumes $r \geq 0$
shows *path_image* (*part_circlepath z r s t*) ⊆ *sphere z r*
by (*smt* (*verit*) *assms path_image_part_circlepath_subset reversepath_part_circlepath*

reversepath_simps(*2*))

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_integrable_on_compose_cnj_iff*:
assumes *valid_path* γ
shows *f contour_integrable_on* (*cnj* ∘ γ) ⟷ (*cnj* ∘ *f* ∘ *cnj*) *contour_integrable_on* γ
proof −
from *assms* **obtain** *S* **where** *S*: *finite S* γ *C1_differentiable_on* {*0..1*} − *S*
unfolding *valid_path_def piecewise_C1_differentiable_on_def* **by** *blast*
have *f contour_integrable_on* (*cnj* ∘ γ) ⟷
((λ*t*. *cnj* (*cnj* (*f* (*cnj* (γ *t*))) ∗ *vector_derivative* γ (*at t*))) *integrable_on* {*0..1*})
unfolding *contour_integrable_on o_def*
proof (*intro integrable_spike_finite_eq* [*OF S*(*1*)])
fix *t* :: *real* **assume** *t* ∈ {*0..1*} − *S*
hence γ *differentiable at t*
using *S*(*2*) **by** (*meson C1_differentiable_on_eq*)
hence *vector_derivative* (λ*x*. *cnj* (γ *x*)) (*at t*) = *cnj* (*vector_derivative* γ (*at t*))
by (*rule vector_derivative_cnj*)
thus *f* (*cnj* (γ *t*)) ∗ *vector_derivative* (λ*x*. *cnj* (γ *x*)) (*at t*) =
cnj (*cnj* (*f* (*cnj* (γ *t*))) ∗ *vector_derivative* γ (*at t*))
by *simp*
qed
also have ... ⟷ ((λ*t*. *cnj* (*f* (*cnj* (γ *t*))) ∗ *vector_derivative* γ (*at t*)) *integrable_on* {*0..1*})
by (*rule integrable_on_cnj_iff*)
also have ... ⟷ (*cnj* ∘ *f* ∘ *cnj*) *contour_integrable_on* γ
by (*simp add*: *contour_integrable_on o_def*)
finally show *?thesis* **.**
qed

lemma *contour_integral_cnj*:
assumes *valid_path* γ
shows *contour_integral* (*cnj* ∘ γ) *f* = *cnj* (*contour_integral* γ (*cnj* ∘ *f* ∘ *cnj*))
proof −
from *assms* **obtain** *S* **where** *S*: *finite S* γ *C1_differentiable_on* {*0..1*} − *S*
unfolding *valid_path_def piecewise_C1_differentiable_on_def* **by** *blast*
have *contour_integral* (*cnj* ∘ γ) *f* =
integral {*0..1*} (λ*t*. *cnj* (*cnj* (*f* (*cnj* (γ *t*))) ∗ *vector_derivative* γ (*at t*)))
unfolding *contour_integral_integral*
proof (*intro integral_spike*)
fix *t* **assume** *t* ∈ {*0..1*} − *S*
hence γ *differentiable at t*
using *S*(*2*) **by** (*meson C1_differentiable_on_eq*)

hence *vector_derivative* ($\lambda x.$ *cnj* (*γ x*)) (*at t*) = *cnj* (*vector_derivative γ* (*at t*))
by (*rule vector_derivative_cnj*)
thus *cnj* (*cnj* (*f* (*cnj* (*γ t*))) ∗ *vector_derivative γ* (*at t*)) =
f ((*cnj* ∘ *γ*) *t*) ∗ *vector_derivative* (*cnj* ∘ *γ*) (*at t*)
by (*simp add*: *o_def*)
qed (*use S*(*1*) **in** *auto*)
also have ... = *cnj* (*integral* {*0..1*} ($\lambda t.$ *cnj* (*f* (*cnj* (*γ t*))) ∗ *vector_derivative γ* (*at t*)))
by (*subst integral_cnj* [*symmetric*]) *auto*
also have ... = *cnj* (*contour_integral γ* (*cnj* ∘ *f* ∘ *cnj*))
by (*simp add*: *contour_integral_integral*)
finally show *?thesis* **.**
qed

lemma *contour_integral_negatepath*:
assumes *valid_path γ*
shows *contour_integral* (*uminus* ∘ *γ*) *f* = −(*contour_integral γ* ($\lambda x.$ *f* (−*x*))) (**is** *?lhs* = *?rhs*)
proof (*cases f contour_integrable_on* (*uminus* ∘ *γ*))
case *True*
hence ∗: (*f has_contour_integral ?lhs*) (*uminus* ∘ *γ*)
using *has_contour_integral_integral* **by** *blast*
have (($\lambda z.$ *f* (−*z*)) *has_contour_integral* − *contour_integral* (*uminus* ∘ *γ*) *f*) (*uminus* ∘ (*uminus* ∘ *γ*))
by (*rule has_contour_integral_negatepath*) (*use* ∗ *assms* **in** *auto*)
hence (($\lambda x.$ *f* (−*x*)) *has_contour_integral* −*?lhs*) *γ*
by (*simp add*: *o_def*)
thus *?thesis*
by (*simp add*: *contour_integral_unique*)
next
case *False*
hence ¬($\lambda z.$ *f* (− *z*)) *contour_integrable_on γ*
using *contour_integrable_negatepath*[*of γ f*] *assms* **by** *auto*
with *False* **show** *?thesis*
by (*simp add*: *not_integrable_contour_integral*)
qed

lemma *contour_integral_bound_part_circlepath*:
assumes *f contour_integrable_on part_circlepath c r a b*
assumes *B* ≥ *0 r* ≥ *0* ⋀*x*. *x* ∈ *path_image* (*part_circlepath c r a b*) ⟹ *norm* (*f x*) ≤ *B*
shows *norm* (*contour_integral* (*part_circlepath c r a b*) *f*) ≤ *B* ∗ *r* ∗ |*b* − *a*|
proof −
let *?I* = *integral* {*0..1*} ($\lambda x.$ *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*} (λx::*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) ⟷
        ((λt. 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) ⟷
        ((λx. 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 ... ⟷ ((λx. 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 ... ⟷ ((λx. 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 ... ⟷ ((λz. 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 ... ⟷ ((λt. 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) ⟷
        (λt. 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} (λt. f (c + r * cis t) * r * i * cis t)
proof (cases f contour_integrable_on part_circlepath c r a b)
  case True
  hence (λt. 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 ¬(λt. 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)
    have abs ` {x. 2 * |real_of_int x| * pi ≤ b + cmod (Ln w)}
    ⊆ {..⌊(b + cmod (Ln w)) / (2 * pi)⌋}
      by (auto simp: field_split_simps le_floor_iff)
    then show ∃k. abs ` {x. 2 * |of_int x| * pi ≤ b + cmod (Ln w)} ⊆ {..k}
      by blast
  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 (of_real t − 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(ii * of_real s)
      have fin: finite (of_real −` {z. cmod z ≤ 1 ∧ exp (ii * 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
     using has_integral_bound [where 'a=real, simplified, OF _ **]
    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*: $\bigwedge x\ y$::*real*. $0 \le x \wedge x \le 1 \wedge 0 \le y \wedge y \le 1$
$\Longrightarrow (x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0 \longleftrightarrow |x - y| \in \{0,1\})$
by *auto*
have **: $\bigwedge x\ y$. $(\exists n.\ (complex_of_real\ x - of_real\ y) * (of_real\ t - of_real\ s) = 2 * (of_int\ n * of_real\ pi)) \longleftrightarrow$
$(\exists n.\ |x - y| * (t - s) = 2 * (of_int\ n * pi))$
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*: $|s - t| \le 2 * pi$
if $\bigwedge x.\ 0 \le x \wedge x \le 1 \Longrightarrow (\exists n.\ x * (t - s) = 2 * (real_of_int\ n * pi)) \longrightarrow x = 0 \vee x = 1$
proof (*rule ccontr*)
assume $\neg\ |s - t| \le 2 * pi$
then have *: $\bigwedge n.\ t - s \neq of_int\ n * |s - t|$
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** $x \neq 0$ $x * (t - s) = 2 * (real_of_int\ n * pi)$ **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*: $\bigwedge P.\ (\forall x \in \{0..1\}.\ \forall y \in \{0..1\}.\ P\ |x - y|) \longleftrightarrow (\forall x$::*real*. $0 \le x \wedge x \le 1 \longrightarrow P\ x)$
by *force*
have $\bigwedge x\ n.\ [\![s \neq t;\ |s - t| \le 2 * pi;\ 0 \le x;\ x < 1;$
$x * (t - s) = 2 * (real_of_int\ n * pi)]\!]$
$\Longrightarrow x = 0$
by (*rule ccontr*) (*auto simp*: *2 field_split_simps abs_mult dest*: *of_int_leD*)
then
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*)
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* $*$ *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 $s*y + t*x = s*x + (t*y + \text{of_int } n * (pi * 2))$
by (*force simp*: *algebra_simps linepath_def dest*: *arg_cong* [**where** *f*=*Re*])
with ‹$x \neq y$› **have** *st*: $s-t = (\text{of_int } n * (pi * 2)) / (y-x)$
by (*force simp*: *field_simps*)
have $|\text{real_of_int } n| < |y - x|$
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 * ii * r * exp (2 * of_real pi * ii * 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 * ii * r * exp(2 * of_real pi * ii * 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 * ii * r * exp(2 * of_real pi * ii * 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 ∈ (λu. 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 *circlepath_inj_on*:
 assumes *r*>*0*
 shows *inj_on* (*circlepath z r*) {*0*..<*1*}
proof (*rule inj_onI*)
 fix *x y*
 assume *x*: *x* ∈ {*0*..<*1*} **and** *y*: *y* ∈ {*0*..<*1*} **and** *eq*: *circlepath z r x* = *circlepath z r y*
 define *c* **where** *c* ≡ *2* ∗ *pi* ∗ i
 have *c*≠*0* **unfolding** *c_def* **by** *auto*
 from *eq* **have** *exp* (*c* ∗ *x*) = *exp* (*c* ∗ *y*)
  unfolding *circlepath c_def* **using** ‹*r*>*0*› **by** *auto*
 then obtain *n* **where** *c* ∗ *x* = *c* ∗ (*y* + *of_int n*)
  by (*auto simp add*: *exp_eq c_def algebra_simps*)
 then have *x*=*y*+*n* **using** ‹*c*≠*0*›
  by (*meson mult_cancel_left of_real_eq_iff*)
 then show *x*=*y* **using** *x y* **by** *auto*
qed

lemma *contour_integral_circlepath*:
 assumes *r* > *0*
 shows *contour_integral* (*circlepath z r*) (λ*w*. *1* / (*w* − *z*)) = *2* ∗ *of_real pi* ∗ i
proof (*rule contour_integral_unique*)
 show ((λ*w*. *1* / (*w* − *z*)) *has_contour_integral 2* ∗ *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

lemma *uniform_limit_contour_integral_circlepath*:
 assumes *u*: *uniform_limit* (*path_image* (*circlepath z r*)) *f g F*
 assumes *c*: ⋀*n*. *continuous_on* (*path_image* (*circlepath z r*)) (*f n*)
 assumes [*simp*]: *F* ≠ *bot*

```
  obtains I J where
    ⋀n. (f n has_contour_integral I n) (circlepath z r)
    (g has_contour_integral J) (circlepath z r)
    (I ⟶ J) F
proof (rule uniform_limit_integral)
  note [continuous_intros] = continuous_on_compose2[OF c]

  show uniform_limit {0..2*pi} (λx t. f x (z + of_real r * cis t) * (of_real r * i
* cis t))
         (λt. g (z + of_real r * cis t) * (of_real r * i * cis t)) F
  proof (rule uniform_lim_mult)
    show uniform_limit {0..2*pi} (λx t. f x (z + of_real r * cis t))
          (λt. g (z + of_real r * cis t)) F
     using u by (rule uniform_limit_compose') (auto simp: dist_norm norm_mult)
  next
     show uniform_limit {0..2*pi} (λa b. complex_of_real r * i * cis b) (λa.
complex_of_real r * i * cis a) F
      by (intro uniform_limit_intros)
  next
    have continuous_on (path_image (circlepath z r)) g
    by (rule uniform_limit_theorem[OF _ u]) (use c in ‹auto intro!: always_eventually›)
    note [continuous_intros] = continuous_on_compose2[OF this]
    have compact ((λa. g (z + complex_of_real r * cis a)) ` {0..2*pi})
    by (intro compact_continuous_image continuous_intros) (auto simp: dist_norm
norm_mult)
    thus bounded ((λa. g (z + complex_of_real r * cis a)) ` {0..2*pi})
      by (rule compact_imp_bounded)
  next
    show bounded ((λa. complex_of_real r * i * cis a) ` {0..2*pi})
    by (intro compact_imp_bounded compact_continuous_image continuous_intros)
auto
  qed

  show continuous_on {0..2*pi} (λt. f n (z + complex_of_real r * cis t) *
        (complex_of_real r * i * cis t)) for n
    by (intro continuous_intros) (auto simp: dist_norm norm_mult)

  fix I J
   assume I: ⋀n. ((λt. f n (z + of_real r * cis t) * (of_real r * i * cis t))
has_integral I n) {0..2*pi}
     and J: ((λt. g (z + of_real r * cis t) * (of_real r * i * cis t)) has_integral J)
{0..2*pi}
     and lim: (I ⟶ J) F
  show ?thesis
    by (rule that[of I J])
      (use I J lim in ‹simp_all add: circlepath_def has_contour_integral_part_circlepath_iff
mult_ac›)
qed auto
```

lemma *contour_integral_sums_circlepath*:
assumes *u*: *uniform_limit* (*path_image* (*circlepath z r*)) (λ*N w*. ∑ *n*<*N*. *f n w*) *g sequentially*
assumes *c*: ⋀*n*. *continuous_on* (*path_image* (*circlepath z r*)) (*f n*)
obtains *J* **where**
(*g has_contour_integral J*) (*circlepath z r*)
(λ*n*. *contour_integral* (*circlepath z r*) (*f n*)) *sums J*
using *u*
proof (*rule uniform_limit_contour_integral_circlepath*)
show *continuous_on* (*path_image* (*circlepath z r*)) (λ*w*. ∑ *n*<*N*. *f n w*) **for** *N*
by (*intro continuous_intros c*)
next
fix *I J*
assume *1*: ⋀*N*. ((λ*w*. ∑ *n*<*N*. *f n w*) *has_contour_integral I N*) (*circlepath z r*)
assume *2*: (*g has_contour_integral J*) (*circlepath z r*) **and** *3*: (*I* ⟶ *J*) *sequentially*
have *4*: *I* = (λ*N*. (∑ *n*<*N*. *contour_integral* (*circlepath z r*) (*f n*)))
proof
fix *N* :: *nat*
have *f n contour_integrable_on* (*circlepath z r*) **for** *n*
by (*intro contour_integrable_continuous_circlepath assms*)
hence ((λ*w*. ∑ *n*<*N*. *f n w*) *has_contour_integral* (∑ *n*<*N*. *contour_integral* (*circlepath z r*) (*f n*))) (*circlepath z r*)
using *c* **by** (*intro has_contour_integral_sum*) (*simp_all add*: *has_contour_integral_integral*)
with *1*[*of N*] **show** *I N* = (∑ *n*<*N*. *contour_integral* (*circlepath z r*) (*f n*))
using *contour_integral_unique* **by** *metis*
qed
have *5*: (λ*n*. *contour_integral* (*circlepath z r*) (*f n*)) *sums J*
using *1 2 3 4* **unfolding** *sums_def* **by** *blast*
from *that*[*OF 2 5*] **show** *?thesis* **.**
qed *auto*

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 ((λ*n*. *contour_integral* γ (*f n*)) ⟶ *contour_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 §: ∀\<^sub>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
    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 [OF ev_fint §]]
      by (fastforce simp: contour_integrable_on path_image_def)
    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
      proof −
      have cmod (l (γ x) ∗ vector_derivative γ (at x) − f a (γ x) ∗ vector_derivative
γ (at x)) ≤ B ∗ e / (|B| + 1)
        using noleB [OF that] fga [OF that] ‹0 ≤ B› ‹0 < e›
        by (fastforce simp: mult_ac dest: mult_mono [OF less_imp_le] simp add:
norm_mult left_diff_distrib [symmetric] norm_minus_commute divide_simps)
       also have ... ≤ e
         using ‹0 ≤ B› ‹0 < e› by (simp add: field_split_simps)
       finally show ?thesis .
      qed
    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': ∀\<^sub>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)

lemma has_contour_integral_linepath_same_Re_iff:
 assumes Re z = c Re z' = c Im z = a Im z' = b a < b
 shows   (f has_contour_integral I) (linepath z z') ⟷
         ((λx. f (Complex c x)) has_integral (-i * I)) {a..b}
proof -
 have (f has_contour_integral I) (linepath z z') ⟷
       ((λx. f (linepath z z' x) * (z' - z)) has_integral I) {0..1}
   by (subst has_contour_integral_linepath) simp_all
  also have ... ⟷ ((λx. f (c + (a + (b - a) * x) *R i) * (i * (b - a)))
has_integral I) {0..1}
   using assms
```

by (*intro has_integral_cong arg_cong2*[*of* _ _ _ _ (∗)] *arg_cong*[*of* _ _ *f*])
(*auto simp*: *linepath_def complex_eq_iff algebra_simps*)
also have {*0..1*} = (λ*x*. *x* / (*b* − *a*)) ‘ {*0..b*−*a*}
using *assms* **by** *simp*
also have ((λ*x*. *f* (*c* + (*a* + (*b*−*a*) ∗ *x*) ∗$_R$ i) ∗ (i ∗ (*b*−*a*))) *has_integral I*) ...
⟷
((λ*x*. *f* (*c* + (*a* + *x*) ∗$_R$ i) ∗ (i ∗ (*b*−*a*))) *has_integral* ((*b*−*a*) ∗$_R$ *I*))
{*0..b*−*a*}
by (*subst has_integral_stretch_real_iff*) (*use assms* **in** *simp_all*)
also have ... ⟷ ((λ*x*. *of_real* (*b*−*a*) ∗ i ∗ (*f* (*c* + *x* ∗$_R$ i))) *has_integral* (*b*−*a*)
∗$_R$ *I*) {*a..b*}
by (*subst has_integral_shift_real_ivl_iff*[**where** *c* = −*a*])
(*simp_all add*: *scaleR_conv_of_real mult_ac*)
also have ... ⟷ ((λ*x*. *f* (*c* + *x* ∗$_R$ i)) *has_integral* (−i ∗ *I*)) {*a..b*}
by (*subst has_integral_mult_right_iff*) (*use assms* **in** ‹*auto simp*: *scaleR_conv_of_real*›)
finally show *?thesis*
by (*simp add*: *scaleR_conv_of_real Complex_eq mult.commute*)
qed

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*: $(\lambda x.\ \mathit{indicat_real}\ (-\ \mathbb{R}_{\leq 0})\ x *_R \mathit{Ln}\ x) \in \mathit{borel_measurable}\ \mathit{borel}$
by (*intro borel_measurable_continuous_on_indicator continuous_intros*) *auto*
have $(\lambda x.\ \mathit{if}\ x \in \mathbb{R}_{\leq 0}\ \mathit{then}\ \mathit{ln}\ (-\mathit{Re}\ x) + \mathrm{i} * \mathit{pi}\ \mathit{else}\ \mathit{indicator}\ (-\mathbb{R}_{\leq 0})\ x *_R \mathit{Ln}\ x) \in \mathit{borel} \to_M \mathit{borel}$
(**is** *?f* ∈ _) **by** (*rule measurable_If_set*[*OF* _ *cont*]) *auto*
hence $(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \mathit{Ln}\ 0\ \mathit{else}\ ?f\ x) \in \mathit{borel} \to_M \mathit{borel}$ **by** *measurable*
also have $(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ \mathit{Ln}\ 0\ \mathit{else}\ ?f\ x) = \mathit{Ln}$
by (*auto simp*: *fun_eq_iff* ∗∗ *nonpos_Reals_def*)
finally show *?thesis* **.**
qed

lemma *powr_complex_measurable* [*measurable*]:
assumes [*measurable*]: $f \in \mathit{measurable}\ M\ \mathit{borel}\ g \in \mathit{measurable}\ M\ \mathit{borel}$
shows $(\lambda x.\ f\ x\ \mathit{powr}\ g\ x :: \mathit{complex}) \in \mathit{measurable}\ M\ \mathit{borel}$
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 $\mathit{midpoint}\ a\ b - a = (1/2) *_R (b - a)$
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 $\mathit{midpoint}\ a\ b - a = (1/2) *_R (b - a)$
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 $\bigwedge x.\ x \in \mathit{UNIV} \Longrightarrow ((\lambda x.\ m\ /\ 2 * x^2 + d * x)\ \mathit{has_field_derivative}\ m * x + d)\ (\mathit{at}\ x)$
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': ⋀y. ⟦cmod (y − x) ≤ d; y ∈ S⟧ ⟹ cmod (f y − f x − f' ∗ (y − x))
≤ e ∗ cmod (y − x)
      and le: cmod (a − b) ≤ d cmod (b − c) ≤ d cmod (c − a) ≤ d
      and xc: x ∈ convex hull {a, b, c}
      and S: convex hull {a, b, c} ⊆ 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) (λy. f y − f x − f' ∗ (y−x)) +
          contour_integral (linepath b c) (λy. f y − f x − f' ∗ (y−x)) +
          contour_integral (linepath c a) (λy. 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 ∈ convex hull {a,b,c}
      have cmod (f y − f x − f' ∗ (y − x)) ≤ e*norm(y − x)
      proof (rule f')
        show cmod (y − x) ≤ d
          by (metis triangle_points_closer [OF xc yc] le norm_minus_commute
order_trans)
      qed (use S yc in blast)
```

```
    also have ... ≤ 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)) ≤ 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 con-
tour_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*: $\bigwedge x\ y\ z\ n.\ At\ x\ y\ z\ n \Longrightarrow \exists x'\ y'\ z'.\ At\ x'\ y'\ z'\ (Suc\ n) \wedge Follows\ x'\ y'\ z'\ x\ y\ z$
obtains *f g h* **where**
$f\ 0 = x0\ g\ 0 = y0\ h\ 0 = z0$
$\bigwedge n.\ At\ (f\ n)\ (g\ n)\ (h\ n)\ n$
$\bigwedge n.\ Follows\ (f(Suc\ n))\ (g(Suc\ n))\ (h(Suc\ n))\ (f\ n)\ (g\ n)\ (h\ n)$
proof −
interpret *three*: *Chain* (*x0*,*y0*,*z0*) $\lambda(x,y,z).\ At\ x\ y\ z\ \lambda(x',y',z').\ \lambda(x,y,z).\ Follows\ x'\ y'\ z'\ x\ y\ z$
proof qed (*use At0 AtSuc* **in** *auto*)
show *?thesis*
proof
show $\bigwedge n.\ Follows\ (fst\ (three.f\ (Suc\ n)))\ (fst\ (snd\ (three.f\ (Suc\ n))))$
$(snd\ (snd\ (three.f\ (Suc\ n))))\ (fst\ (three.f\ n))$
$(fst\ (snd\ (three.f\ n)))\ (snd\ (snd\ (three.f\ n)))$
$\bigwedge n.\ At\ (fst\ (three.f\ n))\ (fst\ (snd\ (three.f\ n)))\ (snd\ (snd\ (three.f\ n)))\ n$
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* = $\lambda x\ y.$ *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*: $y \neq 0$
define *K* **where** $K = 1 + max\ (dist\ a\ b)\ (max\ (dist\ b\ c)\ (dist\ c\ a))$
define *e* **where** $e = norm\ y\ /\ K\hat{}2$
have *K1*: $K \geq 1$ **by** (*simp add*: *K_def max.coboundedI1*)
then have *K*: $K > 0$ **by** *linarith*
have [*iff*]: $dist\ a\ b \leq K\ dist\ b\ c \leq K\ dist\ c\ a \leq K$
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*} ∧
$dist\ x\ y \leq K/2\hat{}n \wedge dist\ y\ z \leq K/2\hat{}n \wedge dist\ z\ x \leq K/2\hat{}n \wedge$
$norm(?pathint\ x\ y + ?pathint\ y\ z + ?pathint\ z\ x) \geq e*(K/2\hat{}n)\hat{}2$

```
    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 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 ≤ 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 ≤ 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* ≡ λ*x 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)) * measure
lborel (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 mi-
nus_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* ≠ *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* ∈ *S closed_segment b c* ⊆ *S* **and** *subs*: ⋀*x*. *x* ∈ *S* ⟹ *closed_segment a x* ⊆ *S*
shows *convex hull* {*a*,*b*,*c*} ⊆ *S*
proof −
have *convex hull* {*b*, *c*} ⊆ *S*
using *assms*(*2*) *segment_convex_hull* **by** *auto*
then have ⋀*u v d*. ⟦*0* ≤ *u*; *0* ≤ *v*; *u* + *v* = *1*; *d* ∈ *convex hull* {*b*, *c*}⟧ ⟹ *u* *$_R$ *a* + *v* *$_R$ *d* ∈ *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 ((λ*x*. *contour_integral*(*linepath a x*) *f*) *has_field_derivative f x*) (*at x*)
proof −
let *?pathint* = λ*x 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 $(\lambda y.\ (?pathint\ x\ y - f\ x * (y - x))\ /_R\ cmod\ (y - x)) \longrightarrow x\rightarrow\ 0$
by (*simp add*: *Lim_at dist_norm inverse_eq_divide*)
then have $(\lambda y.\ (1\ /\ cmod\ (y - x)) *_R\ (?pathint\ a\ y - (?pathint\ a\ x + f\ x * (y - x)))) \longrightarrow x\rightarrow\ 0$
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

Existence of a primitive

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*: $\bigwedge x.\ x \in S - k \Longrightarrow f\ field_differentiable\ at\ x$
shows $\exists g.\ \forall x \in S.\ (g\ has_field_derivative\ f\ x)\ (at\ x)$
proof −
obtain *a* **where** *a*: $a{\in}S$ **and** *a_cs*: $\bigwedge x.\ x{\in}S \Longrightarrow closed_segment\ a\ x \subseteq S$
using *S* **by** (*auto simp*: *starlike_def*)
{ fix *x b c*
assume $x \in S$ *closed_segment b c* $\subseteq S$
then have *abcs*: *convex hull* {*a*, *b*, *c*} $\subseteq S$
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*⟧
⟹ (*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* ⊆ *S*; *pathfinish g* = *pathstart g*⟧
⟹ (*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*: ⋀*b c*. ⟦*b* ∈ *S*; *c* ∈ *S*⟧
⟹ *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* = 0
shows ((λ*x*. *contour_integral*(*linepath a x*) *f*) *has_field_derivative f x*) (*at x within S*)
proof −
let *?pathint* = λ*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 ((λ*w*. *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
          ⋀a b c. ⟦a ∈ S; b ∈ S; c ∈ S⟧ ⟹ (f has_contour_integral 0) (linepath a
b +++ linepath b c +++ linepath c a)
  obtains g where ⋀x. x ∈ S ⟹ (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: ⋀x. x ∈ interior S − K ⟹ f field_differentiable at x
  obtains g where ⋀x. x ∈ S ⟹ (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** ⋀*x*. *x* ∈ *S* ⟹ (*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*;
⋀*x*. *x* ∈ *interior S* − *K* ⟹ *f field_differentiable at x*;
valid_path g; *path_image g* ⊆ *S*; *pathfinish g* = *pathstart g*⟧
⟹ (*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 $f :: complex \Rightarrow complex$
assumes *gpd*: g *piecewise_differentiable_on* $\{a..b\}$
and *dh*: $\bigwedge x.\ x \in S \Longrightarrow (f$ *has_field_derivative* $f'\ x)$ (*at* x *within* $S)$
and *gs*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g\ x \in S$
shows $(\lambda x.\ f'\ (g\ x) * vector_derivative\ g\ (at\ x\ within\ \{a..b\}))$ *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 $f :: complex \Rightarrow complex$
assumes *gpd*: g *piecewise_differentiable_on* $\{a..b\}$
and *dh*: $\bigwedge x.\ x \in S$
$\Longrightarrow \exists d\ h.\ 0 < d \wedge$
$(\forall y.\ norm(y - x) < d \longrightarrow (h$ *has_field_derivative* $f\ y)$ (*at* y *within* $S))$
and *gs*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g\ x \in S$
shows $(\lambda x.\ f(g\ x) * vector_derivative\ g\ (at\ x))$ *integrable_on* $\{a..b\}$
proof −
{ fix x
assume x: $a \le x\ x \le b$
obtain $d\ h$ **where** d: $0 < d$
and h: $(\bigwedge y.\ norm(y - g\ x) < d \Longrightarrow (h$ *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*: $\bigwedge x'.\ x' \in \{a..b\} \Longrightarrow |x' - x| < e \Longrightarrow cmod\ (g\ x' - g\ x) < d$
using x d **by** (*fastforce simp*: *dist_norm continuous_on_iff*)
have $\exists e>0.\ \forall u\ v.\ u \le x \wedge x \le v \wedge \{u..v\} \subseteq ball\ x\ e \wedge (u \le v \longrightarrow a \le u \wedge v \le b) \longrightarrow$
$(\lambda x.\ f\ (g\ x) * vector_derivative\ g\ (at\ x))$ *integrable_on* $\{u..v\}$
proof −
have $(\lambda x.\ f\ (g\ x) * vector_derivative\ g\ (at\ x\ within\ \{u..v\}))$ *integrable_on* $\{u..v\}$
if $u \le x\ x \le v$ **and** *ball*: $\{u..v\} \subseteq ball\ x\ e$ **and** *auvb*: $u \le v \Longrightarrow a \le u \wedge v \le b$
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 $\bigwedge x.\ x \in g\ `\ \{u..v\} \Longrightarrow (h$ *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 $\exists h.\ (\forall y.\ cmod\ (y - z) < d \longrightarrow (h\ has_field_derivative\ f\ y)\ (at\ y\ within\ S))$
by (*force simp*: *dist_norm norm_minus_commute*)
then have $\exists d\ h.\ 0 < d \wedge (\forall y.\ cmod\ (y - z) < d \longrightarrow (h\ has_field_derivative\ f\ y)\ (at\ y\ within\ S))$
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* $\subseteq$ *S*
shows *f contour_integrable_on g*
proof −
have $\bigwedge x.\ x \in S \Longrightarrow f\ field_differentiable\ at\ x$
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 *analytic_imp_contour_integrable*:
assumes *f analytic_on path_image p valid_path p*
shows *f contour_integrable_on p*
by (*meson analytic_on_holomorphic assms contour_integrable_holomorphic_simple*)

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* ⇒ (*real* ⇒ *'a*) ⇒ (*real* ⇒ *'a*::*topological_space*) ⇒ *bool*
  where *linked_paths atends g h* ==
      (*if atends then pathstart h* = *pathstart g* ∧ *pathfinish h* = *pathfinish g*
            else pathfinish g = *pathstart g* ∧ *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* ⊆ *S*
  shows ∃ *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*) ∧
              linked_paths atends g h
              ⟶ *path_image g* ⊆ *S* ∧ *path_image h* ⊆ *S* ∧
             (∀ *f*. *f holomorphic_on S* ⟶ *contour_integral h f* = *contour_integral g f*))
proof −
  have ∀ *z*. ∃ *e*. *z* ∈ *path_image p* ⟶ *0* < *e* ∧ *ball z e* ⊆ *S*
    using *open_contains_ball os p*(*2*) **by** *blast*
  then obtain *ee* **where** *ee*: ⋀*z*. *z* ∈ *path_image p* ⟹ *0* < *ee z* ∧ *ball z* (*ee z*) ⊆ *S*
    by *metis*
  define *cover* **where** *cover* = (λ*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* = ((λ*z*. *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 holomorphic_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* − *contour_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 = - contour_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* (λ*x*. *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* ∘ (λ*u*. (*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*: *order_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 = (λt. ball t (ee t / 3)) ‘ {0..1}
 obtain C' where C': C' ⊆ C finite C' and C'01: {0..1} ⊆ ⋃ C'
 proof (rule compactE [OF compact_interval])
   show {0..1} ⊆ ⋃ C
     using ee [THEN conjunct1] by (auto simp: C_def dist_norm)
 qed (use C_def in auto)
 define kk where kk = {t ∈ {0..1}. ball t (ee t / 3) ∈ C'}
 have kk01: kk ⊆ {0..1} by (auto simp: kk_def)
 define e where e = Min (ee ‘ kk)
 have C'_eq: C' = (λt. ball t (ee t / 3)) ‘ kk
   using C' by (auto simp: kk_def C_def)
 have ee_pos[simp]: ⋀t. t ∈ {0..1} ⟹ 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 ≠ {} using ‹{0..1} ⊆ ⋃ 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: ⋀i. i ∈ kk ⟹ e ≤ ee i
    using ‹finite kk› by (simp add: e_def Min_le_iff [of ee ‘ kk])
  have plus: ∃t ∈ kk. x ∈ ball t (ee t / 3) if x ∈ {0..1} for x
    using C' subsetD [OF C'01 that] unfolding C'_eq by blast
  have [OF order_refl]:
      ∃d. 0 < d ∧ (∀j. valid_path j ∧ (∀u ∈ {0..1}. norm(j u − k (n/N, u)) <
d) ∧ linked_paths atends g j
                    ⟶ contour_integral j f = contour_integral g f)
      if n ≤ 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 ∈ {0..1} (Suc n)/N ∈ {0..1}  by auto
    then obtain t where t: t ∈ kk n/N ∈ ball t (ee t / 3)
      using plus [of n/N] by blast
    then have nN_less: |n/N − t| < ee t
      by (simp add: dist_norm del: less_divide_eq_numeral1)
    have n'N_less: |real (Suc n) / real N − t| < 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 ∈ {0..1} using ‹kk ⊆ {0..1}› ‹t ∈ kk› by blast
    obtain d1 where d1 > 0 and d1:
       ⋀g1 g2. ⟦valid_path g1; 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* ∧
(∀ *t* ∈ {*0..1*}. *norm*(*γ t* − *p t*) < *e*) ∧
contour_integral p (λ*w*. *1*/(*w* − *z*)) = *2* ∗ *pi* ∗ i ∗ *n*

definition *winding_number*:: [*real* ⇒ *complex*, *complex*] ⇒ *complex* **where**
winding_number γ z ≡ *SOME n*. ∀ *e* > *0*. ∃ *p*. *winding_number_prop γ z e p n*

lemma *winding_number*:
assumes *path γ z* ∉ *path_image γ 0* < *e*
shows ∃ *p*. *winding_number_prop γ z e p* (*winding_number γ z*)
proof −
have *path_image γ* ⊆ *UNIV* − {*z*}
using *assms* **by** *blast*
then obtain *d*
where *d*: *d*>*0*
and *pi_eq*: ⋀*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* ⟶
path_image h1 ⊆ *UNIV* − {*z*} ∧ *path_image h2* ⊆ *UNIV* − {*z*} ∧

```
                (∀f. 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)
  then obtain h where h: polynomial_function h ∧ pathstart h = pathstart γ ∧
pathfinish h = pathfinish γ ∧
                      (∀t ∈ {0..1}. norm(h t − γ 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*i) * contour_integral h (λw. 1/(w − z))
  have ∃n. ∀e > 0. ∃p. winding_number_prop γ 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 ∧
            pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧ (∀t∈{0..1}.
cmod (p t − γ 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 (λw. 1 / (w − z)) holomorphic_on UNIV − {z}
        by (auto simp: intro!: holomorphic_intros)
      then have winding_number_prop γ 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 ∃p. winding_number_prop γ 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_prop_reversepath:
  assumes winding_number_prop γ z e p n
  shows   winding_number_prop (reversepath γ) z e (reversepath p) (-n)
proof -
  have p: valid_path p z ∉ path_image p pathstart p = pathstart γ
        pathfinish p = pathfinish γ ⋀t. t ∈ {0..1} ⟹ norm (γ t - p t) < e
        contour_integral p (λw. 1 / (w - z)) = 2 * complex_of_real pi * i * n
    using assms by (auto simp: winding_number_prop_def)
  show ?thesis
    unfolding winding_number_prop_def
  proof (intro conjI strip)
    show norm (reversepath γ t - reversepath p t) < e if t ∈ {0..1} for t
      unfolding reversepath_def using p(5)[of 1 - t] that by auto
    show contour_integral (reversepath p) (λw. 1 / (w - z)) =
           complex_of_real (2 * pi) * i * - n
      using p by (subst contour_integral_reversepath) auto
  qed (use p in auto)
qed

lemma winding_number_prop_reversepath_iff:
  winding_number_prop (reversepath γ) z e p n ⟷ winding_number_prop γ z e
(reversepath p) (-n)
  using winding_number_prop_reversepath[of reversepath γ z e p n]
        winding_number_prop_reversepath[of γ z e reversepath p -n] by auto


lemma winding_number_unique_loop:
  assumes γ: path γ z ∉ path_image γ
      and loop: pathfinish γ = pathstart γ
      and pi:
```

$\bigwedge e.\ e{>}0 \Longrightarrow \exists p.\ valid_path\ p \wedge z \notin path_image\ p \wedge$
$pathfinish\ p = pathstart\ p \wedge$
$(\forall t \in \{0..1\}.\ norm\ (\gamma\ t - p\ t) < e) \wedge$
$contour_integral\ p\ (\lambda w.\ 1/(w - z)) = 2 * pi * \mathrm{i} * n$
shows *winding_number* $\gamma\ z = n$
proof −
have *path_image* $\gamma \subseteq UNIV - \{z\}$
using *assms* **by** *blast*
then obtain *e*
where *e*: $e{>}0$
and *pi_eq*: $\bigwedge h1\ h2\ f.\ [\![valid_path\ h1;\ valid_path\ h2;$
$(\forall t{\in}\{0..1\}.\ cmod\ (h1\ t - \gamma\ t) < e \wedge cmod\ (h2\ t - \gamma\ t) < e);$
$pathfinish\ h1 = pathstart\ h1;\ pathfinish\ h2 = pathstart\ h2;\ f$
holomorphic_on UNIV $- \{z\}]\!] \Longrightarrow$
$contour_integral\ h2\ f = contour_integral\ h1\ f$
using *contour_integral_nearby_loops* [*of UNIV* $- \{z\}\ \gamma$] *assms* **by** (*auto simp*: *open_delete*)
obtain *p* **where** *p*:
$valid_path\ p \wedge z \notin path_image\ p \wedge pathfinish\ p = pathstart\ p \wedge$
$(\forall t \in \{0..1\}.\ norm\ (\gamma\ t - p\ t) < e) \wedge$
$contour_integral\ p\ (\lambda w.\ 1/(w - z)) = 2 * pi * \mathrm{i} * n$
using *pi* [*OF e*] **by** *blast*
obtain *q* **where** *q*: *winding_number_prop* $\gamma\ z\ e\ q$ (*winding_number* $\gamma\ z$)
using *winding_number* [*OF* $\gamma\ e$] **by** *blast*
have $2 * complex_of_real\ pi * \mathrm{i} * n = contour_integral\ p\ (\lambda w.\ 1\ /\ (w - z))$
using *p* **by** *auto*
also have $\ldots = contour_integral\ q\ (\lambda w.\ 1\ /\ (w - z))$
proof (*rule pi_eq*)
show $(\lambda w.\ 1\ /\ (w - z))$ *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 $\ldots = 2 * complex_of_real\ pi * \mathrm{i} * winding_number\ \gamma\ z$
using *q* **by** (*auto simp*: *winding_number_prop_def*)
finally have $2 * complex_of_real\ pi * \mathrm{i} * n = 2 * complex_of_real\ pi * \mathrm{i} *$
winding_number $\gamma\ z$ **.**
then show *?thesis*
by *simp*
qed

proposition *winding_number_valid_path*:
assumes *valid_path* γ *z* $\notin$ *path_image* γ
shows *winding_number* $\gamma\ z = 1/(2{*}pi{*}\mathrm{i}) * contour_integral\ \gamma\ (\lambda w.\ 1/(w - z))$
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* $\notin$ *path_image* γ
shows $((\lambda w.\ 1/(w - z))\ has_contour_integral\ (2{*}pi{*}\mathrm{i}{*}winding_number\ \gamma\ z))$
γ

by (*simp add*: *winding_number_valid_path has_contour_integral_integral contour_integrable_inversediff assms*)

lemma *winding_number_trivial* [*simp*]: $z \neq a \Longrightarrow$ *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 *γ1*: *path γ1 z* ∉ *path_image γ1*
and *γ2*: *path γ2 z* ∉ *path_image γ2*
and *pathfinish γ1* = *pathstart γ2*
shows *winding_number*(*γ1 +++ γ2*) *z* = *winding_number γ1 z* + *winding_number γ2 z*
proof (*rule winding_number_unique*)
show ∃ *p*. *winding_number_prop* (*γ1 +++ γ2*) *z e p*
(*winding_number γ1 z* + *winding_number γ2 z*) **if** *e* > *0* **for** *e*
proof −
obtain *p1* **where** *winding_number_prop γ1 z e p1* (*winding_number γ1 z*)
using ‹*0* < *e*› *γ1 winding_number* **by** *blast*
moreover
obtain *p2* **where** *winding_number_prop γ2 z e p2* (*winding_number γ2 z*)
using ‹*0* < *e*› *γ2 winding_number* **by** *blast*
ultimately
have *winding_number_prop* (*γ1+++γ2*) *z e* (*p1+++p2*) (*winding_number γ1 z* + *winding_number γ2 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 γ z* ∉ *path_image γ*
shows *winding_number*(*reversepath γ*) *z* = − (*winding_number γ z*)
proof (*rule winding_number_unique*)
show ∃ *p*. *winding_number_prop* (*reversepath γ*) *z e p* (− *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 have *winding_number_prop* (*reversepath γ*) *z e* (*reversepath p*) (− *wind-*

```
ing_number γ 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*≠*z* **and** *g*: ⋀*t*. ⟦*0*≤*t*; *t*≤*1*⟧ ⟹ *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*≠*z*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *winding_number_offset*: *winding_number p z* = *winding_number* (λ*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* (λ*w*. *p w* − *z*) *0 e r n*
by (*rule_tac x*=λ*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* (λ*w*. *p w* − *z*) *0 e g n*
then have *winding_number_prop p z e* (λ*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_offset_NO_MATCH*:
NO_MATCH 0 z ⟹ *winding_number p z* = *winding_number* (λ*w*. *p w* − *z*) *0*
using *winding_number_offset* **by** *metis*

lemma *winding_number_negatepath*:

```
  assumes γ: valid_path γ and 0: 0 ∉ 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 ((λz. 1/z) has_contour_integral contour_integral γ ((/) 1)) γ
    by (rule has_contour_integral_integral)
  then have ((λz. 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

lemma winding_number_cnj:
  assumes path γ z ∉ path_image γ
  shows   winding_number (cnj ∘ γ) (cnj z) = −cnj (winding_number γ z)
proof (rule winding_number_unique)
  show ∃p. winding_number_prop (cnj ∘ γ) (cnj z) e p (−cnj (winding_number
γ z))
    if e > 0 for e
  proof −
    from winding_number[OF assms(1,2) ‹e > 0›]
    obtain p where winding_number_prop γ z e p (winding_number γ z)
      by blast
    then have p: valid_path p z ∉ path_image p
       pathstart p = pathstart γ
       pathfinish p = pathfinish γ
       ⋀t. t ∈ {0..1} ⟹ cmod (γ t − p t) < e and
       p_cont:contour_integral p (λw. 1 / (w − z)) =
         complex_of_real (2 ∗ pi) ∗ i ∗ winding_number γ z
      unfolding winding_number_prop_def by auto

    have valid_path (cnj ∘ p)
      using p(1) by (subst valid_path_cnj) auto
    moreover have cnj z ∉ path_image (cnj ∘ p)
      using p(2) by (auto simp: path_image_def)
    moreover have pathstart (cnj ∘ p) = pathstart (cnj ∘ γ)
      using p(3) by (simp add: pathstart_compose)
    moreover have pathfinish (cnj ∘ p) = pathfinish (cnj ∘ γ)
      using p(4) by (simp add: pathfinish_compose)
    moreover have cmod ((cnj ∘ γ) t − (cnj ∘ p) t) < e
      if t: t ∈ {0..1} for t
    proof −
      have (cnj ∘ γ) t − (cnj ∘ p) t = cnj (γ t − p t)
```

```
      by simp
    also have norm ... = norm (γ t - p t)
      by (subst complex_mod_cnj) auto
    also have ... < e
      using p(5)[OF t] by simp
    finally show ?thesis .
  qed
  moreover have contour_integral (cnj ∘ p) (λw. 1 / (w - cnj z)) =
      cnj (complex_of_real (2 * pi) * i * winding_number γ z) (is ?L=?R)
  proof -
    have ?L = contour_integral (cnj ∘ p) (cnj ∘ (λw. 1 / (cnj w - z)))
      by (simp add: o_def)
    also have ... = cnj (contour_integral p (λx. 1 / (x - z)))
      using p(1) by (subst contour_integral_cnj) (auto simp: o_def)
    also have ... = ?R
      using p_cont by simp
    finally show ?thesis .
  qed
  ultimately show ?thesis
    by (intro exI[of _ cnj ∘ p]) (auto simp: winding_number_prop_def)
 qed
 show path (cnj ∘ γ)
   by (intro path_continuous_image continuous_intros) (use assms in auto)
 show cnj z ∉ path_image (cnj ∘ γ)
   using ‹z ∉ path_image γ› unfolding path_image_def by auto
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 (λw. 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 = λx. 1 / (γ x − z) ∗ vector_derivative γ (at x)
  let ?int = λz. contour_integral γ (λw. 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 ((λa. 1 / (a − z)) has_contour_integral contour_integral γ (λw. 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 ((λx. 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 $z \notin$ *path_image* γ
and *e*: $0 < e$
and *ge*: $\bigwedge x.\ [\![0 < x; x < 1]\!] \Longrightarrow e \leq Im\ (vector_derivative\ \gamma\ (at\ x) * cnj(\gamma\ x - z))$
shows $0 < Re\ (winding_number\ \gamma\ z)$
proof −
have *bm*: *bounded* $((\lambda w.\ w - z)\ `\ (path_image\ \gamma))$
using *bounded_translation* [*of* _ $-z$] γ **by** (*simp add*: *bounded_valid_path_image*)
then obtain B **where** B: $B > 0$ **and** *Bno*: $\bigwedge x.\ x \in (\lambda w.\ w - z)\ `\ (path_image\ \gamma) \Longrightarrow norm\ x \leq B$
using *bounded_pos* [*THEN iffD1*, *OF bm*] **by** *blast*
{ fix *x*::*real* **assume** *x*: $0 < x\ x < 1$
then have *B2*: *cmod* $(\gamma\ x - z)\hat{}2 \leq B\hat{}2$ **using** *Bno* [*of* $\gamma\ x - z$]
by (*simp add*: *path_image_def power2_eq_square mult_mono'*)
with x **have** $\gamma\ x \neq z$ **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 (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*)) *integrable_on* {*a*..*b*}
(**is** *?thesis1*)
exp (− (*integral* {*a*..*b*} (λ*x*. *vector_derivative* γ (*at x*) / (γ *x* − *z*)))) ∗ (γ *b* − *z*) = γ *a* − *z*
(**is** *?thesis2*)
proof −
let *?D*γ = λ*x*. *vector_derivative* γ (*at x*)
have [*simp*]: ⋀*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 ○: *open* ({*a*<..<*b*} − *k*)
using ‹*finite k*› **by** (*simp add*: *finite_imp_closed open_Diff*)
moreover have {*a*<..<*b*} − *k* ⊆ {*a*..*b*} − *k*
by *force*
ultimately have *g_diff_at*: ⋀*x*. ⟦*x* ∉ *k*; *x* ∈ {*a*<..<*b*}⟧ ⟹ γ *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*))) (λ*w*. *1* / (*w* − *z*))
by (*auto simp*: *dist_norm intro*!: *continuous_intros*)
moreover have ⋀*x*. *cmod* (*w* − *x*) < *cmod* (*w* − *z*) ⟹ ∃*f'*. ((λ*w*. *1* / (*w* − *z*)) *has_field_derivative f'*) (*at x*)
by (*auto simp*: *intro*!: *derivative_eq_intros*)
ultimately have ∃*h*. ∀*y*. *norm*(*y* − *w*) < *norm*(*w* − *z*) ⟶ (*h has_field_derivative* *1*/(*y* − *z*)) (*at y*)
using *holomorphic_convex_primitive* [*of ball w* (*norm*(*w* − *z*)) {} λ*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*: ⋀*w y*. $w \neq z \Longrightarrow$ *norm*(*y* − *w*) < *norm*(*w* − *z*) ⟹ (*h w has_field_derivative* *1*/(*y* − *z*)) (*at y*)
by *meson*
have *exy*: ∃*y*. ((λ*x*. *inverse* (γ *x* − *z*) ∗ *?D*γ *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*: $(\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))\ integrable_on\ \{a..b\}$
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* $\circ$] **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 (λx. x *R d)) (at x)
        by (auto simp add: differentiable_iff_scaleR)
       show ((λc. exp (− integral {a..c} (λx. ?Dγ x / (γ x − z))) * (γ c − z))
has_derivative (λh. 0))
        (at x within {a..b})
      proof (rule exp_fg [unfolded has_vector_derivative_def, simplified])
        show (γ has_derivative (λc. c *R d)) (at x within {a..b})
          using d by (blast intro: has_derivative_at_withinI)
       have ((λx. integral {a..x} (λx. ?Dγ x / (γ x − z))) has_vector_derivative
d / (γ 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}) (λx. vector_derivative γ (at x) /
(γ 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 countable_finite 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* (λ*w*. *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* (λ*w*. *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* (λ*w*. *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* ∉ *path_image* γ; *w* ≠ *z*;
    ⋀*a*::*real*. *0* < *a* ⟹ *z* + *of_real a* ∗ (*w* − *z*) ∉ *path_image* γ⟧
   ⟹ *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* ∉ *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* ∉ *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:
    ⋀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⟧
            ⟹
             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 order_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 ... ≤ 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)) ≤ e
    proof (cases x=z ∨ 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 = (λx. 1 / (x - w) - 1 / (x - z))
  have cmod (contour_integral p ?f) ≤ 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 ∀u∈- cball z (3 / 4 * pe). cmod (?f u) ≤ 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* ∉ *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 ∃ *d>0*. ∀ *x'*. *dist x' z* < *d* ⟶ *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** δ = *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*) (λ*w*. *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* (λ*w*. *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 ∃ *e*>*0*. ∀ *y*. *dist y z* < *e* ⟶ *y* ∉ *path_image γ* ∧ *winding_number γ y* = *winding_number γ 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* ∈ *outside* (*path_image γ*)
shows *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*
obtain *w*::*complex* **where** *w*: *w* ∉ *ball 0* (*B* + *1*)
by (*metis abs_of_nonneg le_less less_irrefl mem_ball_0 norm_of_real*)
have − *ball 0* (*B* + *1*) ⊆ *outside* (*path_image γ*)
using *B subset_ball* **by** (*intro outside_subset_convex*) *auto*
then have *wout*: *w* ∈ *outside* (*path_image γ*)
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 γ z* = *winding_number γ w*
by (*metis* (*no_types*, *opaque_lifting*) *constant_on_def z*)
also have ... = *0*
proof −
have *wnot*: *w* ∉ *path_image γ* **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: (⋀t. ⟦0 ≤ t; t ≤ 1⟧ ⟹ cmod (p t - γ t) < 1)
                and pge: (⋀t. ⟦0 ≤ t; t ≤ 1⟧ ⟹ cmod (p t - γ 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 ∃p. valid_path p ∧ w ∉ path_image p ∧
                  pathstart p = pathstart γ ∧ pathfinish p = pathfinish γ ∧
                  (∀t∈{0..1}. cmod (γ t - p t) < e) ∧ contour_integral p (λwa. 1
/ (wa - w)) = 0
    proof (intro exI conjI)
      have ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ 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 ∉ 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 $\exists B.\ \forall z.\ B \leq$ *norm* $z \longrightarrow$ *winding_number* γ 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*
  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 $z \notin$ *cball 0 B*
      using *that* **by** *auto*
    show *path_image* $\gamma \subseteq$ *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** $\bigwedge x.$ *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* $\gamma \subseteq S$⟧
    ⟹ $\exists z.\ z \in S \wedge$ *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*: $S \cap$ *path_image* γ = {}
      and z: $z \in S$ **and** *wn_nz*: *winding_number* γ z ≠ *0* **and** *w*: $w \in S \cup$ *inside S*
    shows *winding_number* γ w = *winding_number* γ z
proof −
  have *ssb*: $S \subseteq$ *inside*(*path_image* γ)
  proof
    fix x :: *complex*
    assume $x \in S$
    hence $x \notin$ *path_image* γ
      by (*meson disjoint_iff_not_equal S_disj*)
    thus $x \in$ *inside* (*path_image* γ)

by (*metis Compl_iff S_disj UnE* γ ‹$x \in S$› *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 $z \in S \cup$ *inside S*
using *z* **by** *blast*
show *connected* ($S \cup$ *inside S*)
by (*simp add*: *cls connected_with_inside cos*)
show ($S \cup$ *inside S*) $\cap$ *path_image* $\gamma = \{\}$
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*} (λx. *winding_number*(*subpath 0 x* γ) *z*)
proof (*rule continuous_on_eq*)
let *?f* = λx. *integral* {*0..x*} (λt. *vector_derivative* γ (*at t*) / (γ *t* − *z*))
show *continuous_on* {*0..1*} (λx. *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* γ) *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* γ) (λw. *1*/(*w* − *z*))
using *assms x*
by (*simp add*: *contour_integral_subcontour_integral* [*OF contour_integrable_inversediff*])
also have ... = *winding_number* (*subpath 0 x* γ) *z*
proof (*subst winding_number_valid_path*)
show *z* $\notin$ *path_image* (*subpath 0 x* γ)
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* γ *z*)
shows $\exists t \in$ {*0..1*}. *Re*(*winding_number*(*subpath 0 t* γ) *z*) = *w*
proof −

have *continuous_on* {*0..1*} (λ*x*. *winding_number* (*subpath 0 x* γ) *z*)
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* (*winding_number* (*subpath 0 0* γ) *z*) ≤ *w w* ≤ *Re* (*winding_number* (*subpath 0 1* γ) *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* ∉ *path_image* γ **and** *Re*(*winding_number* γ *z*) ≤ *w w* ≤ *0*
shows ∃ *t* ∈ {*0..1*}. *Re*(*winding_number*(*subpath 0 t* γ) *z*) = *w*
proof −
have *continuous_on* {*0..1*} (λ*x*. *winding_number* (*subpath 0 x* γ) *z*)
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* (*winding_number* (*subpath 0 0* γ) *z*) ≥ *w w* ≥ *Re* (*winding_number* (*subpath 0 1* γ) *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 γ z)| < 1/2
proof −
 have z ∉ path_image γ using assms by auto
 with assms have Re (winding_number γ z) < 0 ⟹ − Re (winding_number γ
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 ∉ path_image γ› winding_number_lt_half_lemma by fastforce
qed

lemma winding_number_le_half:
 assumes γ: valid_path γ and z: z ∉ path_image γ
    and anz: a ≠ 0 and azb: a · z ≤ b and pag: path_image γ ⊆ {w. a · w ≥ b}
   shows |Re (winding_number γ z)| ≤ 1/2
proof −
 { assume wnz_12: |Re (winding_number γ 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: ⋀x'. dist x' z < d ⟹ dist (winding_number
γ x') (winding_number γ z) < |Re(winding_number γ 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 · z ∗ 6 ≤ 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 · z' ≤ 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 γ z' − winding_number γ z) < |Re (winding_number
```

```
γ z)| − 1/2
      using d [of z'] anz ‹d>0› by (simp add: dist_norm z'_def)
    then have 1/2 < |Re (winding_number γ z)| − cmod (winding_number γ z'
− winding_number γ z)
      by simp
    then have 1/2 < |Re (winding_number γ z)| − |Re (winding_number γ z')
− Re (winding_number γ z)|
       using abs_Re_le_cmod [of winding_number γ z' − winding_number γ z]
by simp
    then have wnz_12': |Re (winding_number γ z')| > 1/2
      by linarith
    moreover have |Re (winding_number γ z')| < 1/2
    proof (rule winding_number_lt_half [OF γ *])
      show path_image γ ⊆ {w. b − d / 3 * cmod a < a · 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
```

lemma *winding_number_linepath_neg_lt*:
  assumes $0 < Im\ ((a - b) * cnj\ (a - z))$
  shows $Re(winding_number(linepath\ a\ b)\ z) < 0$
proof −
  have *z*: $z \notin path_image\ (linepath\ a\ b)$
    using *assms*
    by (*simp add*: *closed_segment_def*) (*force simp*: *algebra_simps*)
  have $Re(winding_number(linepath\ a\ b)\ z) =$
      $-Re(winding_number(reversepath\ (linepath\ a\ b))\ z)$
    by (*subst winding_number_reversepath*) (*use z* **in** *auto*)
  also have $\ldots = - Re(winding_number(linepath\ b\ a)\ z)$
    by *simp*
  finally have $Re\ (winding_number\ (linepath\ a\ b)\ z)$
    $= - Re\ (winding_number\ (linepath\ b\ a)\ z)$ **.**
  moreover have $0 < Re\ (winding_number\ (linepath\ b\ a)\ z)$
    using *winding_number_linepath_pos_lt*[*OF assms*] **.**
  ultimately show *?thesis* **by** *auto*
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*: $z \notin path_image\ g$ **and** *pah*: $z \notin path_image\ h$
    using *homotopic_paths_imp_subset* [*OF assms*] **by** *auto*
  ultimately obtain *d e* **where** $d > 0\ e > 0$
      and *d*: $\bigwedge p.\ [\![path\ p;\ pathstart\ p = pathstart\ g;\ pathfinish\ p = pathfinish\ g;$
$\forall\, t{\in}\{0..1\}.\ norm\ (p\ t - g\ t) < d]\!]$
        $\Longrightarrow homotopic_paths\ (-\{z\})\ g\ p$
      and *e*: $\bigwedge q.\ [\![path\ q;\ pathstart\ q = pathstart\ h;\ pathfinish\ q = pathfinish\ h;$
$\forall\, t{\in}\{0..1\}.\ norm\ (q\ t - h\ t) < e]\!]$
        $\Longrightarrow homotopic_paths\ (-\{z\})\ h\ q$
    using *homotopic_nearby_paths* [*of g* $-\{z\}$] *homotopic_nearby_paths* [*of h*
$-\{z\}$] **by** *force*
  obtain *p* **where** *p*:
      valid_path p $z \notin path_image\ p$
      pathstart p = pathstart g pathfinish p = pathfinish g
      and *gp_less*:$\forall\, t{\in}\{0..1\}.\ cmod\ (g\ t - p\ t) < d$
      and *pap*: $contour_integral\ p\ (\lambda w.\ 1\ /\ (w - z)) = complex_of_real\ (2 * pi)$
$*\ \mathrm{i} *\ winding_number\ g\ z$
   using *winding_number* [*OF* ‹*path g*› *pag* ‹*0 < d*›] **unfolding** *winding_number_prop_def*
by *blast*
  obtain *q* **where** *q*:
      valid_path q $z \notin path_image\ q$

```
      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*: $\forall t{\in}\{0..1\}$. *cmod* (*h t − q t*) < *e*
and *paq*: *contour_integral q* (λ*w*. *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* (λ*w*. *1/(w − z)*) = *contour_integral q* (λ*w*. *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 $w \notin$ *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*: $0 < r \Longrightarrow$ *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 winding_number_circlepath':
  assumes r ≥ 0 dist w z ≠ r
  shows   winding_number (circlepath z r) w = indicator (ball z r) w
proof (cases dist w z < r)
  case True
  thus ?thesis
   using winding_number_circlepath[of w z r] by (auto simp: dist_norm norm_minus_commute)
next
  case False
  have winding_number (circlepath z r) w = 0
    by (rule winding_number_zero_outside[of _ cball z r])
       (use assms False in ‹auto simp: dist_commute›)
  thus ?thesis
    using False assms by (auto simp: dist_commute)
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 ∈ inside (path_image c1 ∪ 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 umi-
nus_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 ∩ path_image p = {}
obtains z where z ∈ 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 ∩ path_image (linepath (a − e) (a + e)) ⊆ {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 ∈ path_image ?pae
    by (simp add: assms path_image_join) (metis midpoint_in_closed_segment)
  then have a ∈ frontier(inside (path_image ?pae))
    using assms by (simp add: Jordan_inside_outside )
  with ‹0 < e› obtain c where c: c ∈ inside (path_image ?pae)
                and dac: dist a c < e
    by (auto simp: frontier_straddle)
  then have c ∉ path_image ?pae
    using inside_no_overlap by blast
  then have c ∉ path_image p c ∉ closed_segment (a − e) (a + e)
    by (simp_all add: assms path_image_join)
  with ‹0 < e› dac have c ∉ affine hull {a − e, a + e}
    by (simp add: segment_as_ball not_le)
  with ‹0 < e› have ∗: ¬ 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) ∈
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* ∈ *frontier* (*inside* (*path_image p*))
and *d_int*: ⋀ε. ⟦*0* ≤ ε; ε < *d*⟧ ⟹ (*a* − ε) ∈ *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* ∈ *frontier* (*inside* (*path_image p*))
and *e_int*: ⋀ε. ⟦*0* ≤ ε; ε < *e*⟧ ⟹ (*a* + ε) ∈ *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* ≤ *t0 t0* ≤ *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*: ⋀*z*. *z* ∈ *inside*(*path_image p*) ⟹ *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* ∈ *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* ≠ *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*} ⊆ *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* ⊆ *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* ∈ *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* ∈ *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 ‹$0 \le t$› ‹*simple_path q*› ‹$t \le 1$› ‹$t \neq 0$› ‹$t \neq 1$›
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 ‹$0 \le t$› ‹*simple_path q*› ‹$t \le 1$› ‹$t \neq 0$›
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* $(a - d)$ $(a + e) \subseteq \{a + e, a - d\}$
using *qt paq_Int_cs* ‹*simple_path q*› ‹$0 \le t$› ‹$t \le 1$›
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 ‹$0 \le t$› ‹$t \le 1$› **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*: $z \notin$ *closed_segment* $(a + e)$ $(a - d)$
using *zin q01* **by** (*simp add*: *path_image_join closed_segment_commute inside_def*)
have *z_notin_0t*: $z \notin$ *path_image* (*subpath 0 t q*)
by (*metis* (*no_types*, *opaque_lifting*) *IntI Un_upper1 subsetD empty_iff inside_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* ‹$0 \le t$› ‹*simple_path q*› ‹$t \le 1$› *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*: $z \in$ *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*: ‹$0 \le t$› ‹*simple_path q*› ‹$t \le 1$› ‹$t \neq 0$› ‹$t \neq 1$› *simple_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 wind-
ing_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 $z \in \mathbb{Z}$
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 $z \notin$ *path_image* γ
shows *winding_number* γ $z \in \{-1,0,1\}$
proof −
consider $z \in$ *inside* (*path_image* γ) | $z \in$ *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; $z \notin$ *path_image* γ; $0 <$ *Re*(*winding_number* γ z)⟧
⟹ *winding_number* γ z = *1*
using *simple_closed_path_winding_number_cases*
by *fastforce*

lemma *simple_closed_path_winding_number_neg*:
⟦*simple_path* γ; *pathfinish* γ = *pathstart* γ; $z \notin$ *path_image* γ; *Re* (*winding_number* γ z) $<$ *0*⟧
⟹ *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 $z \in$ *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 ∗ i)
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) (λw. 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 holomor-
phic_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} (λx. 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 ((λx. 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 ((λw. f w / (w−z)) has_contour_integral (2*pi * ii * winding_number
γ z * f z)) γ
proof −
  let ?fz = λw. (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›)
 note ∗∗ = has_contour_integral_add [OF has_contour_integral_lmul [OF has_contour_integral_winding_
[OF vpg znotin], of f z] ∗]
  show ?thesis
    apply (rule has_contour_integral_eq)
    using znotin ∗∗ 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 $((\lambda w.\ f\ w\ /\ (w-z))$ *has_contour_integral* $(2*pi * \mathrm{i} * winding_number\ \gamma\ z * f\ z))\ \gamma$
proof −
have $\bigwedge x.\ x \in interior\ S \Longrightarrow f$ *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*: *holomorphic_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 $((\lambda u.\ f\ u/(u-w))$ *has_contour_integral* $(2 * of_real\ pi * \mathrm{i} * f\ w))$
(*circlepath z r*)
proof −
have $r > 0$
using *assms le_less_trans norm_ge_zero* **by** *blast*
have $((\lambda u.\ f\ u\ /\ (u-w))$ *has_contour_integral* $(2 * pi) * \mathrm{i} * winding_number$ (*circlepath z r*) $w * f\ w)$
(*circlepath z r*)
proof (*rule Cauchy_integral_formula_weak* [**where** S = *cball z r* **and** $k = \{\}$])
show $\bigwedge x.\ x \in interior$ (*cball z r*) $- \{\} \Longrightarrow$
f field_differentiable at x
using *holf holomorphic_on_imp_differentiable_at* **by** *auto*
have $w \notin$ *sphere z r*
by *simp* (*metis dist_commute dist_norm not_le order_refl wz*)
then show *path_image* (*circlepath z r*) $\subseteq$ *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 $((\lambda u.\ f\ u/(u-w))$ *has_contour_integral* $(2 * of_real\ pi * \mathrm{i} * f\ w))$
(*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*:
fixes f' :: *complex* ⇒ *complex*
defines $h \equiv \lambda k\ w\ u.\ f'\ u\ /\ (u-w)\hat{}k$

```
  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 ⟹ (h k w 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 h (Suc k) w contour_integrable_on γ
       and (f has_field_derivative (k * contour_integral γ (h (Suc k) w))) (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: (λz. (h k x z − h k w z) / (x * k − w * k)) contour_integrable_on γ
    if x ≠ w cmod (x−w) < d for x::complex
  proof −
    have x ∈ S − path_image γ
      by (metis d dist_commute dist_norm mem_ball subsetD that(2))
    then show ?thesis
    using contour_integrable_diff contour_integrable_div contour_integrable_on_def
int w
      by meson
  qed
  then have 1: ∀F x in at w. (λz. (h k x z − h k w z) / (x−w) / of_nat k)
                       contour_integrable_on γ
    unfolding eventually_at
    by (force intro: exI [where x=d] simp add: ‹d > 0› dist_norm field_simps)
  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, 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 unfolding path_image_def
          by (smt (verit, best) dist_norm imageE insert_Diff mem_ball sub-
set_Diff_insert)
      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
      using ‹k ≠ 0› ‹x ≠ u› ‹u ≠ w› le_less_trans [OF _ e]
      by (simp add: field_simps flip: norm_divide)
  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) (λx z. (h k x z − h k w z) / (x−w) / k)
(h (Suc k) w) (at w)
  unfolding uniform_limit_iff dist_norm
 proof clarify
```

```
    fix e::real
    assume 0 < e
    have *: cmod ((h k x (γ u) − h k w (γ u)) / ((x−w) * k) − h (Suc k) w (γ
u)) < e
          if ec: cmod ((inverse (γ u − x) ^ k − inverse (γ u − w) ^ k) / ((x−w) *
k) −
                      inverse (γ u − w) * inverse (γ u − w) ^ k) < e / C
                and x: 0 ≤ u u ≤ 1
              for x u
    proof (cases (f' (γ u)) = 0)
      case True then show ?thesis by (simp add: ‹0 < e› h_def)
    next
      case False
      have cmod ((h k x (γ u) − h k w (γ u)) / ((x−w) * k) − h (Suc k) w (γ u))
=
            cmod (f' (γ u)) * ((inverse (γ u − x) ^ k − inverse (γ u − w) ^ k) /
((x−w) * k) −
                               inverse (γ u − w) * inverse (γ u − w) ^ k))
        by (simp add: h_def field_simps)
      also have ... = cmod (f' (γ u)) *
                      cmod ((inverse (γ u − x) ^ k − inverse (γ u − w) ^ k) / ((x−w)
* k) −
                               inverse (γ u − w) * inverse (γ u − w) ^ k)
        by (simp add: norm_mult)
      also have ... < cmod (f' (γ u)) * (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 u in at w.
           ∀x∈path_image γ.
            cmod ((h k u x − h k w x) / (u−w) / of_nat k − h (Suc k) w x) < e
      using twom [OF divide_pos_pos [OF ‹0 < e› ‹C > 0›]] *
      unfolding path_image_def h_def
      by (force elim: eventually_mono)
  qed
  show h (Suc k) w contour_integrable_on γ
    using contour_integral_uniform_limit [OF 1 2 leB γ] by (simp add: h_def)
  have *: (λu. contour_integral γ (λx. (h k u x − h k w x) / (u−w) / k))
          −w→ contour_integral γ (h (Suc k) w)
    by (rule contour_integral_uniform_limit [OF 1 2 leB γ]) auto
  have **: contour_integral γ (λx. (h k u x − h k w x) / ((u−w) * k)) =
             (f u − f w) / (u−w) / k
    if dist u w < d for u
  proof −
    have u ∈ S − path_image γ
      by (metis subsetD d dist_commute mem_ball that)
```

```
    then have (h k u has_contour_integral f u) γ (h k w has_contour_integral f
w) γ
      using w by (simp_all add: field_simps int)
    then show ?thesis
      using contour_integral_unique has_contour_integral_diff
        has_contour_integral_div by force
  qed
  show ?thes2
    unfolding has_field_derivative_iff
    by (simp add: ‹k ≠ 0› ** Lim_transform_within [OF tendsto_mult_left [OF
*] ‹0 < d›])
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*
unfolding *power2_eq_square*
using *int Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1*]
by *fastforce*
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*)
unfolding *power2_eq_square*
using *int Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1* **and** *g* = λ*x*. *2* ∗ *of_real pi* ∗ i ∗ *f x*]
by *fastforce*
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 holomorphic_deriv_compose:
  assumes g: g holomorphic_on B and f: f holomorphic_on A and f ‘ A ⊆ B
open B
 shows  (λx. deriv g (f x)) holomorphic_on A
 using holomorphic_on_compose_gen [OF f holomorphic_deriv[OF g]] assms
 by (auto simp: o_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* ⟹ (*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* ∈ *S*⟧
⟹ ((*deriv* ^^ *n*) *f has_field_derivative* (*deriv* ^^ (*Suc n*)) *f x*) (*at x*)
using *holomorphic_derivI holomorphic_higher_deriv* **by** *fastforce*

lemma *higher_deriv_cmult*:
assumes *f holomorphic_on A x* ∈ *A open A*
shows (*deriv* ^^ *j*) (λ*x*. *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*) (λ*x*. *c* ∗ *f x*)) *x* = *deriv* (λ*x*. *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 *higher_deriv_cmult′*:
assumes *f analytic_on* {*x*}
shows (*deriv* ^^ *j*) (λ*x*. *c* ∗ *f x*) *x* = *c* ∗ (*deriv* ^^ *j*) *f x*
using *assms higher_deriv_cmult*[*of f _ x j c*] *assms*
using *analytic_at_two* **by** *blast*

lemma *deriv_cmult′*:
assumes *f analytic_on* {*x*}
shows *deriv* (λ*x*. *c* ∗ *f x*) *x* = *c* ∗ *deriv f x*
using *higher_deriv_cmult′*[*OF assms*, *of 1 c*] **by** *simp*

lemma *analytic_derivI*:
assumes *f analytic_on* {*z*}
shows (*f has_field_derivative* (*deriv f z*)) (*at z within A*)
using *assms holomorphic_derivI*[*of f _ z*] *analytic_at* **by** *blast*

lemma *analytic_logderivI*:
assumes *f analytic_on* {*z*} *f z* ≠ *0*
shows (*f has_log_derivative* (*deriv f z* / *f z*)) (*at z*)
by (*intro has_field_derivative_imp_has_log_derivative′ analytic_derivI assms*)

lemma *deriv_compose_analytic*:
fixes *f g* :: *complex* ⇒ *complex*
assumes *f analytic_on* {*g z*} *g analytic_on* {*z*}
shows *deriv* (λ*x*. *f* (*g x*)) *z* = *deriv f* (*g z*) ∗ *deriv g z*
proof −
have ((*f* ∘ *g*) *has_field_derivative* (*deriv f* (*g z*) ∗ *deriv g z*)) (*at z*)
by (*intro DERIV_chain analytic_derivI assms*)
thus *?thesis*
by (*auto intro*!: *DERIV_imp_deriv simp add*: *o_def*)
qed

lemma *valid_path_compose_holomorphic*:
assumes *valid_path g f holomorphic_on S* **and** *open S path_image g* ⊆ *S*
shows *valid_path* (*f* ∘ *g*)
by (*meson assms holomorphic_deriv holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at*
holomorphic_on_subset subsetD valid_path_compose)

lemma *valid_path_compose_analytic*:
assumes *valid_path g* **and** *holo*:*f analytic_on S* **and** *path_image g* ⊆ *S*
shows *valid_path* (*f* ∘ *g*)
proof (*rule valid_path_compose*[*OF* ‹*valid_path g*›])
fix *x* **assume** *x* ∈ *path_image g*
then show *f field_differentiable at x*
using *analytic_on_imp_differentiable_at analytic_on_open assms holo* **by** *blast*
next
show *continuous_on* (*path_image g*) (*deriv f*)
by (*intro holomorphic_on_imp_continuous_on analytic_imp_holomorphic analytic_intros*
analytic_on_subset[*OF holo*] *assms*)
qed

lemma *analytic_on_deriv* [*analytic_intros*]:
assumes *f analytic_on g* ‘ *A*
assumes *g analytic_on A*
shows (λ*x*. *deriv f* (*g x*)) *analytic_on A*
proof −
have (*deriv f* ∘ *g*) *analytic_on A*
by (*rule analytic_on_compose_gen*[*OF assms*(*2*) *analytic_deriv*[*OF assms*(*1*)]]) *auto*
thus *?thesis*
by (*simp add*: *o_def*)
qed

lemma *contour_integral_comp_analyticW*:
assumes *f analytic_on s valid_path* γ *path_image* γ ⊆ *s*
shows *contour_integral* (*f* ∘ γ) *g* = *contour_integral* γ (λ*w*. *deriv f w* ∗ *g* (*f w*))

proof −
obtain *spikes* **where** *finite spikes* **and** γ_*diff*: γ *C1_differentiable_on* $\{0..1\}$ − *spikes*
using ‹*valid_path* γ› **unfolding** *valid_path_def piecewise_C1_differentiable_on_def*
by *auto*
show *contour_integral* $(f \circ \gamma)$ *g*
$=$ *contour_integral* γ $(\lambda w.$ *deriv f w* $*$ *g* $(f\ w))$
unfolding *contour_integral_integral*
proof (*rule integral_spike*[*OF negligible_finite*[*OF* ‹*finite spikes*›]])
fix *t*::*real* **assume** *t*:*t* $\in \{0..1\}$ − *spikes*
then have γ *differentiable at t*
using γ_*diff* **unfolding** *C1_differentiable_on_eq* **by** *auto*
moreover have *f field_differentiable at* $(\gamma\ t)$
proof −
have $\gamma\ t \in s$ **using** *t assms* **unfolding** *path_image_def* **by** *auto*
thus *?thesis*
using ‹*f analytic_on s*› *analytic_on_imp_differentiable_at* **by** *blast*
qed
ultimately show *deriv f* $(\gamma\ t) * g\ (f\ (\gamma\ t)) *$ *vector_derivative* γ $(at\ t) =$
$g\ ((f \circ \gamma)\ t) *$ *vector_derivative* $(f \circ \gamma)$ $(at\ t)$
by (*subst vector_derivative_chain_at_general*) (*simp_all add*:*field_simps*)
qed
qed

corollary *contour_integral_affine*:
assumes *valid_path* γ $c \neq 0$
shows *contour_integral* $((\lambda x.\ c * x + b) \circ \gamma)$ $f =$ *contour_integral* γ $(\lambda w.\ c * f\ (c * w + b))$
proof −
define *ff* **where** $ff \equiv \lambda x.\ c{*}x{+}b$
have *contour_integral* $(ff \circ \gamma)$ $f =$ *contour_integral* γ $(\lambda w.$ *deriv ff w* $*$ *f* $(ff\ w))$
proof (*rule contour_integral_comp_analyticW*)
show *ff analytic_on UNIV path_image* $\gamma \subseteq$ *UNIV valid_path* γ
unfolding *ff_def* **using** ‹*valid_path* γ›
by (*auto intro*: *analytic_intros*)
qed
also have $\ldots =$ *contour_integral* γ $(\lambda w.\ c * f\ (c * w + b))$
proof −
have *deriv ff* $x = c$ *ff* $x = c{*}x{+}b$ **for** *x*
unfolding *ff_def* **by** *auto*
then show *?thesis* **by** *auto*
qed
finally show *?thesis* **unfolding** *ff_def* **.**
qed

4.4 Morera's theorem

lemma *Morera_local_triangle_ball*:
assumes $\bigwedge z.\ z \in S$

```
        ⟹ ∃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)
  shows f analytic_on S
proof −
  { fix z  assume z ∈ S
    with assms obtain e a where
          0 < e and z: z ∈ ball a e and contf: continuous_on (ball a e) f
       and 0: ⋀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
      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*) (λ*w*. *f w* + *g w*) *x*
by (*auto simp add*: *Suc*)
then have ((*deriv* ^^ *n*) (λ*w*. *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* λ*w*. (*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*) (λ*w*. *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*) (λ*w*. *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 + 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*) (λ*w*. *f* (*u* ∗ *w*+*c*))) *z* = *deriv* (λ*z*. *u*^*n* ∗ (*deriv* ^^ *n*) *f* (*u*∗*z*+*c*)) *z*
proof (*rule complex_derivative_transform_within_open* [*OF* _ *holo3 S Suc.prems*])
show (*deriv* ^^ *n*) (λ*w*. *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* (λ*z*. (*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 (λ*w*. (*deriv* ^^ *n*) *f* (*u* ∗ *w*+*c*)) *analytic_on S*
using *holomorphic_on_compose*[*OF* _ *holo2*] ‹(λ*w*. *u* ∗ *w*+*c*) *holomorphic_on S*›
by (*simp add*: *S analytic_on_open o_def*)
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 *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*) (λ*w*. *f* (*u* ∗ *w*)) *z* = *u*^*n* ∗ (*deriv* ^^ *n*) *f* (*u* ∗ *z*)
using *higher_deriv_compose_linear'* [**where** *c*=*0*] *assms* **by** *simp*

lemma *higher_deriv_add_at*:
assumes *f analytic_on* {*z*} *g analytic_on* {*z*}
shows (*deriv* ^^ *n*) (λ*w*. *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*) (λ*w*. *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*) (λ*w*. −(*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*) (λ*w*. *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*

lemma *higher_deriv_sum_at*:
assumes ⋀*x*. *x* ∈ *X* ⟹ *f x analytic_on* {*z*}
shows (*deriv* ^^ *n*) (λ*z*. ∑ *x*∈*X*. *f x z*) *z* = (∑ *x*∈*X*. (*deriv* ^^ *n*) (*f x*) *z*)
using *assms*
by (*induction X rule*: *infinite_finite_induct*)
(*auto simp*: *higher_deriv_add_at analytic_on_sum*)

lemma *deriv_scale*:
assumes *f analytic_on* {*c* ∗ *z*}
shows *deriv* (*f* ∘ (λ*z*. *c* ∗ *z*)) *z* = *c* ∗ *deriv f* (*c* ∗ *z*)
by (*subst* (*2*) *mult.commute*, *intro DERIV_imp_deriv DERIV_chain analytic_derivI assms*)
(*auto intro*!: *derivative_eq_intros*)

lemma *higher_deriv_scale*:
assumes *f analytic_on* {*c* ∗ *z*}
shows (*deriv* ^^ *n*) (*f* ∘ (λ*z*. *c* ∗ *z*)) *z* = *c* ^ *n* ∗ (*deriv* ^^ *n*) *f* (*c* ∗ *z*)
using *assms*
proof (*induction n arbitrary*: *f*)
case (*Suc n f*)
obtain *R* **where** *R*: *R* > *0 f holomorphic_on ball* (*c* ∗ *z*) *R*
using *analytic_at_ball Suc.prems* **by** *blast*
define *r* **where** *r* = (*if c* = *0 then 1 else R* / *norm c*)
have *r*: *r* > *0*
using *R* **by** (*auto simp*: *r_def*)

have (*deriv* ^^ *Suc n*) (*f* ∘ (λ*z*. *c* ∗ *z*)) *z* = (*deriv* ^^ *n*) (*deriv* (*f* ∘ (∗) *c*)) *z*
by (*subst funpow_Suc_right*) *auto*
also have (*deriv* ^^ *n*) (*deriv* (*f* ∘ (∗) *c*)) *z* = (*deriv* ^^ *n*) (λ*z*. *c* ∗ *deriv f* (*c* ∗ *z*)) *z*
proof (*intro higher_deriv_cong_ev*)
have *eventually* (λ*w*. *w* ∈ *ball z r*) (*nhds z*)
by (*intro eventually_nhds_in_open*) (*use r* **in** *auto*)
thus ∀ *F* *w in nhds z*. *deriv* (*f* ∘ (∗) *c*) *w* = *c* ∗ *deriv f* (*c* ∗ *w*)
proof *eventually_elim*
case (*elim w*)
have *c* ∗ *w* ∈ *ball* (*c* ∗ *z*) *R*
using *elim* ‹*R* > *0*› **by** (*auto simp*: *dist_mult_left r_def field_simps split*: *if_splits*)
moreover have *f analytic_on ball* (*c* ∗ *z*) *R*
using *R* **by** (*simp add*: *analytic_on_open*)
ultimately have *f analytic_on* {*c* ∗ *w*}

```
      using analytic_on_analytic_at by blast
    thus ?case
      by (rule deriv_scale)
  qed
 qed auto
 also have ... = c * (deriv ^^ n) (deriv f ∘ (λz. c * z)) z
   by (subst higher_deriv_cmult') (auto intro!: analytic_intros Suc.prems simp:
o_def)
 also have ... = c ^ Suc n * (deriv ^^ Suc n) f (c * z)
   by (subst Suc.IH)
      (auto intro!: analytic_intros Suc.prems simp: funpow_Suc_right simp del:
funpow.simps)
 finally show ?case .
qed auto
```

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−)])
 have continuous_on (S−K) f
  using holf holomorphic_on_imp_continuous_on by auto
 then show continuous_on S f
  by (metis Diff_iff K S at_within_open continuous_on_eq_continuous_at
      continuous_within f finite_imp_closed open_Diff)
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 * ii * 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_on_subset fcd field_differentiable_def open_interior
      has_field_derivative_at_within holomorphic_derivI holomorphic_onI inte-
rior_subset)
 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: ((λu. f u / (u−w) ^ 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) ((λu. f u / (u−w) ^ Suc (Suc
k))) = X
    by (rule contour_integral_unique)
  have ⋀n. ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) w) (at w)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  then have dnf_diff: ⋀n. (deriv ^^ n) f field_differentiable (at w)
    by (force simp: field_differentiable_def)
  have deriv (λw. complex_of_real (2 * pi) * i / (fact k) * (deriv ^^ k) f w) w =
        of_nat (Suc k) * contour_integral (circlepath z r) (λu. f u / (u−w) ^ Suc
(Suc k))
    by (force intro!: DERIV_imp_deriv Cauchy_next_derivative_circlepath [OF f
Suc.IH _ Suc.prems])
  also have ... = of_nat (Suc k) * X
    by (simp only: con)
  finally have deriv (λw. ((2 * pi) * i / (fact k)) * (deriv ^^ k) f w) w = of_nat
(Suc k) * X .
  then have ((2 * pi) * i / (fact k)) * deriv (λw. (deriv ^^ k) f w) w = of_nat
(Suc k) * X
    by (metis deriv_cmult dnf_diff)
  then have deriv (λw. (deriv ^^ k) f w) w = of_nat (Suc k) * X / ((2 * pi) * 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* ∈ *ball z r*
shows (λ*u*. *f u* / (*u*−*w*)^(*Suc k*)) *contour_integrable_on* (*circlepath z r*)
(**is** *?thes1*)
and (*deriv* ^^ *k*) *f w* = (*fact k*) / (2 ∗ *pi* ∗ i) ∗ *contour_integral*(*circlepath z r*) (λ*u*. *f u*/(*u*−*w*)^(*Suc k*))
(**is** *?thes2*)
proof −
have ∗: ((λ*u*. *f u* / (*u*−*w*) ^ *Suc k*) *has_contour_integral* (2 ∗ *pi*) ∗ i / (*fact k*) ∗ (*deriv* ^^ *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*
by (*metis w dist_norm mem_ball norm_ge_zero not_less_iff_gr_or_eq*

order_less_le_trans)
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*: $\bigwedge k.\ (\lambda u.\ f\ u\ /\ (u-z)\ \hat{}\ Suc\ k)$ *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*: $\bigwedge u.\ u \in cball\ z\ r \Longrightarrow norm(f\ u) \le B$
by (*metis* (*no_types*) *bounded_pos compact_cball compact_continuous_image compact_imp_bounded contf image_eqI*)
obtain *k* **where** *k*: $0 < k\ k \le r$ **and** *wz_eq*: $norm(w-z) = r - k$
and *kle*: $\bigwedge u.\ norm(u-z) = r \Longrightarrow k \le norm(u-w)$
proof
show $\bigwedge u.\ cmod\ (u-z) = r \Longrightarrow r - dist\ z\ w \le cmod\ (u-w)$
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*) $(\lambda n\ x.\ (\sum k<n.\ (w-z)\ \hat{}\ k * (f\ x\ /\ (x-z)\ \hat{}\ Suc\ k)))\ (\lambda x.\ f\ x\ /\ (x-w))$ *sequentially*
unfolding *uniform_limit_iff dist_norm*
proof *clarify*
fix *e*::*real*
assume $0 < e$
have *rr*: $0 \le (r-k) / r\ (r-k) / r < 1$ **using** k **by** *auto*
obtain *n* **where** *n*: $((r-k) / r)\ \hat{}\ n < e / B * k$
using *arch_pow_inv* [*of e/B*k* (r-k)/r*] ‹*0 < e*› ‹*0 < B*› *k* **by** *force*
have *norm* $((\sum k<N.\ (w-z)\ \hat{}\ k * f\ u\ /\ (u-z)\ \hat{}\ Suc\ k) - f\ u\ /\ (u-w)) < e$
if $n \le N$ **and** *r*: $r = dist\ z\ u$ **for** *N u*
proof −
have *N*: $((r-k) / r)\ \hat{}\ N < e / B * k$
using *le_less_trans* [*OF power_decreasing n*]
using ‹$n \le N$› *k* **by** *auto*
have *u* [*simp*]: $(u \ne z) \wedge (u \ne w)$
using ‹*0 < r*› *r w* **by** *auto*
have *wzu_not1*: $(w-z) / (u-z) \ne 1$
by (*metis* (*no_types*) *dist_norm divide_eq_1_iff less_irrefl mem_ball norm_minus_commute r w*)
have *norm* $((\sum k<N.\ (w-z)\ \hat{}\ k * f\ u\ /\ (u-z)\ \hat{}\ Suc\ k) * (u-w) - f\ u)$
$= norm\ ((\sum k<N.\ (((w-z)\ /\ (u-z))\ \hat{}\ k)) * f\ u * (u-w) / (u-z) - f\ u)$
unfolding *sum_distrib_right sum_divide_distrib power_divide* **by** (*simp add*: *algebra_simps*)
also have $\ldots = norm\ ((((w-z)\ /\ (u-z))\ \hat{}\ N - 1) * (u-w) / (((w-z)\ /\ (u-z) - 1) * (u-z)) - 1) * norm\ (f\ u)$
using ‹*0 < B*›
apply (*simp add*: *geometric_sum* [*OF wzu_not1*] *flip*: *norm_mult*)
apply (*simp add*: *field_simps*)

```
      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: ⋀n.
          (∑k<n. contour_integral (circlepath z r) (λu. f u / (u−z) ^ Suc k) *
(w−z) ^ k) =
          contour_integral (circlepath z r) (λu. ∑k<n. (w−z) ^ k * (f u / (u−z)
^ Suc k))
   apply (subst contour_integral_sum)
    apply (simp_all only: § finite_lessThan contour_integral_lmul cint alge-
bra_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)) contour_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 eq*
using ‹*0 < r*› *contour_integral_uniform_limit_circlepath* [*OF eventuallyI ul*]

by *fastforce*
then have (λ*k*. *contour_integral* (*circlepath z r*) (λ*u*. *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 (λ*k*. *contour_integral* (*circlepath z r*) (λ*u*. *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 (λ*n*. (*w*−*z*) ^ *n* ∗ *contour_integral* (*circlepath z r*) (λ*u*. *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 λz. 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 (λx. norm
(f x) ≥ B) at_infinity
    obtains z where f z = 0
proof −
  { assume f: ⋀z. f z ≠ 0
    have 1: (λx. 1 / f x) holomorphic_on UNIV
      by (simp add: holomorphic_on_divide assms f)
    have 2: ((λx. 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* (λ*n*. *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*)) (λ*x*. *1* / (*2* ∗ *complex_of_real pi* ∗ i) ∗ *g x*)
by (*intro continuous_intros continuous_on_subset* [*OF contg*])
have *2*: ((λ*u*. *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*. (λ*u*. *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*) (λ*n x*. *f n x* / (*x*−*w*)) (λ*x*. *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 ⋀e. 0 < e ⟹ ∀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 ((λn. 2 * of_real pi * i * f n w) ⟶ 2 * of_real pi * i * g w) F
     by (rule tendsto_mult_left [OF tendstoI])
    then have ((λu. g u / (u−w)) has_contour_integral 2 * of_real pi * 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 ((λu. g u / (2 * of_real pi * i * (u−w))) has_contour_integral g
w) (circlepath z r)
     using has_contour_integral_div [where c = 2 * of_real pi * 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

lemma higher_deriv_complex_uniform_limit:
  assumes ulim: uniform_limit A f g F
      and f_holo: eventually (λn. f n holomorphic_on A) F
      and F: F ≠ bot
      and A: open A z ∈ A
    shows ((λn. (deriv ^^ m) (f n) z) ⟶ (deriv ^^ m) g z) F
proof −
  obtain r where r: r > 0 cball z r ⊆ A
    using A by (meson open_contains_cball)
  have r': ball z r ⊆ A
```

```
    using r by auto
  define h where h = (λn z. f n z − g z)
  define c where c = of_real (2*pi) * i / fact m
  have [simp]: c ≠ 0
    by (simp add: c_def)
  have g holomorphic_on ball z r ∧ continuous_on (cball z r) g
  proof (rule holomorphic_uniform_limit)
    show uniform_limit (cball z r) f g F
      by (rule uniform_limit_on_subset[OF ulim r(2)])
    show ∀F n in F. continuous_on (cball z r) (f n) ∧ f n holomorphic_on ball z
r using f_holo
      by eventually_elim
        (use holomorphic_on_subset[OF _ r(2)] holomorphic_on_subset[OF _ r']

          in ‹auto intro!: holomorphic_on_imp_continuous_on›)
  qed (use F in auto)
  hence g_holo: g holomorphic_on ball z r and g_cont: continuous_on (cball z
r) g
    by blast+

  have ulim': uniform_limit (sphere z r) (λn x. h n x / (x − z) ^ (Suc m)) (λ_.
0) F
  proof −
    have uniform_limit (sphere z r) (λn x. f n x / (x − z) ^ Suc m) (λx. g x / (x
− z) ^ Suc m) F
    proof (intro uniform_lim_divide uniform_limit_intros uniform_limit_on_subset[OF
ulim])
      have compact (g ‘ sphere z r)
        by (intro compact_continuous_image continuous_on_subset[OF g_cont])
auto
      thus bounded (g ‘ sphere z r)
        by (rule compact_imp_bounded)
      show r ^ Suc m ≤ norm ((x − z) ^ Suc m) if x ∈ sphere z r for x unfolding
norm_power
        by (intro power_mono) (use that r(1) in ‹auto simp: dist_norm norm_minus_commute›)
    qed (use r in auto)
    hence uniform_limit (sphere z r) (λn x. f n x / (x − z) ^ Suc m − g x / (x
− z) ^ Suc m)
            (λx. g x / (x − z) ^ Suc m − g x / (x − z) ^ Suc m) F
      by (intro uniform_limit_intros)
    thus ?thesis
      by (simp add: h_def diff_divide_distrib)
  qed

  have has_integral: eventually (λn. ((λu. h n u / (u − z) ^ Suc m) has_contour_integral

                        c * (deriv ^^ m) (h n) z) (circlepath z r)) F
    using f_holo
  proof eventually_elim
```

```
    case (elim n)
    show ?case
      unfolding c_def
    proof (rule Cauchy_has_contour_integral_higher_derivative_circlepath)
      show continuous_on (cball z r) (h n) unfolding h_def
        by (intro continuous_intros g_cont holomorphic_on_imp_continuous_on
              holomorphic_on_subset[OF elim] r)
      show h n holomorphic_on ball z r
      unfolding h_def by (intro holomorphic_intros g_holo holomorphic_on_subset[OF
elim] r')
    qed (use r(1) in auto)
  qed

  have ((λn. contour_integral (circlepath z r) (λu. h n u / (u − z) ^ Suc m))
⟶
        contour_integral (circlepath z r) (λu. 0 / (u − z) ^ Suc m)) F
  proof (rule contour_integral_uniform_limit_circlepath)
    show ∀F n in F. (λu. h n u / (u − z) ^ Suc m) contour_integrable_on
circlepath z r
    using has_integral by eventually_elim (blast intro: has_contour_integral_integrable)
  qed (use r(1) ‹F ≠ bot› ulim' in simp_all)
  hence ((λn. contour_integral (circlepath z r) (λu. h n u / (u − z) ^ Suc m))
⟶ 0) F
    by simp
  also have ?this ⟷ ((λn. c * (deriv ^^ m) (h n) z) ⟶ 0) F
  proof (rule tendsto_cong)
    show ∀F x in F. contour_integral (circlepath z r) (λu. h x u / (u − z) ^ Suc
m) =
                    c * (deriv ^^ m) (h x) z
      using has_integral by eventually_elim (simp add: contour_integral_unique)
  qed
  finally have ((λn. (deriv ^^ m) g z + c * (deriv ^^ m) (h n) z / c) ⟶
                (deriv ^^ m) g z + 0 / c) F
    by (intro tendsto_intros) auto
  also have ?this ⟷ ((λn. (deriv ^^ m) (f n) z) ⟶ (deriv ^^ m) g z) F
  proof (intro filterlim_cong)
    show ∀F n in F. (deriv ^^ m) g z + c * (deriv ^^ m) (h n) z / c = (deriv ^^
m) (f n) z
      using f_holo
    proof eventually_elim
      case (elim n)
        have (deriv ^^ m) (h n) z = (deriv ^^ m) (f n) z − (deriv ^^ m) g z
unfolding h_def
        by (rule higher_deriv_diff holomorphic_on_subset[OF elim r'] g_holo A)+
(use r(1) in auto)
      thus ?case
        by simp
    qed
  qed auto
```

finally show *?thesis* **.**
qed

lemma *deriv_complex_uniform_limit*:
assumes *ulim*: *uniform_limit A f g F*
and *f_holo*: *eventually* (*λn. f n holomorphic_on A*) *F*
and *F*: *F ≠ bot*
and *A*: *open A z ∈ A*
shows ((*λn. deriv* (*f n*) *z*) *⟶ deriv g z*) *F*
using *higher_deriv_complex_uniform_limit*[*OF assms*, *of 1*] **by** *simp*

lemma *logderiv_prodinf_complex_uniform_limit*:
fixes *f* :: *nat ⇒ complex ⇒ complex*
assumes *lim*: *uniform_limit A* (*λn x.* ∏ *k*<*n. f k x*) *P sequentially*
assumes *holo*: ⋀*k. f k holomorphic_on A*
assumes *nz*: *P z ≠ 0*
assumes *A*: *open A z ∈ A*
shows (*λk. deriv* (*f k*) *z* / *f k z*) *sums* (*deriv P z* / *P z*)
proof −
define *f′* **where** *f′* = (*λk. deriv* (*f k*))
note [*derivative_intros*] = *has_field_derivative_prod′*
have [*derivative_intros*]:
(*f k has_field_derivative f′ k z*) (*at z within B*) **if** *z ∈ A* **for** *B z k*
using *that holomorphic_derivI*[*OF holo*[*of k*], *of z B*] *A* **unfolding** *f′_def* **by** *auto*
have *lim′*: (*λn.* ∏ *k*<*n. f k z*) *⟶ P z*
using *lim* **by** (*rule tendsto_uniform_limitI*) *fact*+

have *nz′*: *f k z ≠ 0* **for** *k*
proof
assume *f k z = 0*
have *eventually* (*λn.* (∏ *k*<*n. f k z*) = *0*) *sequentially*
using *eventually_gt_at_top*[*of k*] **by** *eventually_elim* (*use ‹f k z = 0› in auto*)
hence (*λn.* (∏ *k*<*n. f k z*)) *⟶ 0*
by (*rule tendsto_eventually*)
with *lim′* **have** *P z = 0*
using *tendsto_unique sequentially_bot* **by** *blast*
with *nz* **show** *False*
by *simp*
qed

from *lim* **have** (*λn. deriv* (*λx.* ∏ *k*<*n. f k x*) *z*) *⟶ deriv P z*
by (*rule deriv_complex_uniform_limit*)
(*use A* **in** ‹*auto intro*!: *always_eventually holomorphic_intros holo*›)
also have (*λn. deriv* (*λx.* ∏ *k*<*n. f k x*) *z*) = (*λn.* (∏ *k*<*n. f k z*) ∗ (∑ *k*<*n. f′ k z* / *f k z*))
using ‹*z ∈ A*› **by** (*auto intro*!: *ext DERIV_imp_deriv derivative_eq_intros simp*: *nz′*)

```
  finally have (λn. (∏k<n. f k z) * (∑k<n. f' k z / f k z)) ⟶ deriv P z .
  hence (λn. (∏k<n. f k z) * (∑k<n. f' k z / f k z) / (∏k<n. f k z)) ⟶
deriv P z / P z
    by (intro tendsto_intros) (use nz lim' in auto)
  also have (λn. (∏k<n. f k z) * (∑k<n. f' k z / f k z) / (∏k<n. f k z)) =
            (λn. (∑k<n. f' k z / f k z))
    by (simp add: nz')
  finally show (λk. f' k z / f k z) sums (deriv P z / P z)
    unfolding sums_def .
qed
```

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

```
proposition has_complex_derivative_uniform_limit:
  fixes z::complex
  assumes cont: eventually (λn. continuous_on (cball z r) (f n) ∧
                           (∀w∈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: ¬ trivial_limit F and 0 < r
  obtains g' where
      continuous_on (cball z r) g
      ⋀w. w ∈ ball z r ⟹ (g has_field_derivative (g' w)) (at w) ∧ ((λn. f' n w)
⟶ 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': ⋀x. x ∈ ball z r ⟹ (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: ⋀x. x ∈ ball z r ⟹ deriv g x = g' x
    by (simp add: DERIV_imp_deriv)
  have tends_f'n_g': ((λn. f' n w) ⟶ g' w) F if w: w ∈ ball z r for w
  proof −
    have eq_f': ?conint (λx. f n x / (x−w)²) − ?conint (λx. 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** $d = (r - norm(w-z))\hat{}2$
have $d > 0$
using *w* **by** (*simp add*: *dist_commute dist_norm d_def*)
have *dle*: $d \le cmod\ ((y-w)^2)$ **if** $r = cmod\ (z-y)$ **for** *y*
by (*smt* (*verit*, *best*) *d_def diff_add_cancel diff_diff_eq2 dist_norm mem_ball norm_minus_commute norm_power norm_triangle_ineq2 power_mono that w*)
have *1*: $\forall_F\ n\ in\ F.\ (\lambda x.\ f\ n\ x\ /\ (x-w)^2)\ contour_integrable_on\ circlepath\ z\ r$
by (*force simp*: *holomorphic_on_open intro*: *w Cauchy_derivative_integral_circlepath eventually_mono* [*OF cont*])
have *2*: *uniform_limit* (*sphere z r*) $(\lambda n\ x.\ f\ n\ x\ /\ (x-w)^2)$ $(\lambda x.\ g\ x\ /\ (x-w)^2)$ *F*
unfolding *uniform_limit_iff*
proof *clarify*
fix *e*::*real*
assume $e > 0$
with ‹$r > 0$›
have $\forall_F\ n\ in\ F.\ \forall x.\ x \neq w \longrightarrow cmod\ (z-x) = r \longrightarrow cmod\ (f\ n\ x - g\ x) < e * cmod\ ((x-w)^2)$
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 $\forall_F\ n\ in\ F.\ \forall x \in sphere\ z\ r.\ dist\ (f\ n\ x\ /\ (x-w)^2)\ (g\ x\ /\ (x-w)^2) < e$
by (*simp add*: *norm_divide field_split_simps sphere_def dist_norm*)
qed
have $((\lambda n.\ contour_integral\ (circlepath\ z\ r)\ (\lambda x.\ f\ n\ x\ /\ (x-w)^2)) \longrightarrow contour_integral\ (circlepath\ z\ r)\ ((\lambda x.\ g\ x\ /\ (x-w)^2)))\ F$
by (*rule contour_integral_uniform_limit_circlepath* [*OF 1 2 F* ‹*0 < r*›])
then have *tendsto_0*: $((\lambda n.\ 1\ /\ (2 * of_real\ pi * \mathrm{i}) * (?conint\ (\lambda x.\ f\ n\ x\ /\ (x-w)^2) - ?conint\ (\lambda x.\ g\ x\ /\ (x-w)^2))) \longrightarrow 0)\ F$
using *Lim_null* **by** (*force intro*!: *tendsto_mult_right_zero*)
have $((\lambda n.\ f'\ n\ w - g'\ w) \longrightarrow 0)\ F$
by (*force simp*: *divide_simps intro*: *eq_f′ eventually_mono* [*OF cont*] *Lim_transform_eventually* [*OF tendsto_0*])
then show *?thesis* **using** *Lim_null* **by** *blast*
qed
obtain *g′* **where** $\bigwedge w.\ w \in ball\ z\ r \Longrightarrow (g\ has_field_derivative\ (g'\ w))\ (at\ w) \wedge ((\lambda n.\ f'\ n\ w) \longrightarrow g'\ w)\ F$
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*: $\bigwedge n.$ $(f\ n)$ *holomorphic_on* S
and *ulim_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists\, d.\ 0 < d \wedge$ *cball* $x\ d \subseteq S \wedge$ *uniform_limit* $(cball\ x\ d)\ f\ g$ *sequentially*
shows g *holomorphic_on* S
proof −
have $\exists f'.$ $(g$ *has_field_derivative* $f')$ $(at\ z)$ **if** $z \in S$ **for** z
proof −
obtain r **where** $0 < r$ **and** r: *cball* $z\ r \subseteq S$
and *ul*: *uniform_limit* $(cball\ z\ r)\ f\ g$ *sequentially*
using *ulim_g* $[OF\ \langle z \in S\rangle]$ **by** *blast*
have $*$: $\forall_F\ n$ *in sequentially. continuous_on* $(cball\ z\ r)\ (f\ n) \wedge f\ n$ *holomorphic_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 $\langle 0 < r\rangle$ *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*: $\bigwedge n\ x.\ x \in S \Longrightarrow ((f\ n)$ *has_field_derivative* $f'\ n\ x)$ $(at\ x)$
and *ulim_g*: $\bigwedge x.\ x \in S$
$\Longrightarrow \exists\, d.\ 0 < d \wedge$ *cball* $x\ d \subseteq S \wedge$ *uniform_limit* $(cball\ x\ d)\ f\ g$ *sequentially*
shows $\exists g'.\ \forall x \in S.\ (g$ *has_field_derivative* $g'\ x)$ $(at\ x) \wedge ((\lambda n.\ f'\ n\ x) \longrightarrow g'\ x)$ *sequentially*
proof −
have y: $\exists y.\ (g$ *has_field_derivative* $y)$ $(at\ z) \wedge (\lambda n.\ f'\ n\ z) \longrightarrow y$ **if** $z \in S$ **for** z
proof −
obtain r **where** $0 < r$ **and** r: *cball* $z\ r \subseteq S$
and *ul*: *uniform_limit* $(cball\ z\ r)\ f\ g$ *sequentially*
using *ulim_g* $[OF\ \langle z \in S\rangle]$ **by** *blast*
have $*$: $\forall_F\ n$ *in sequentially. continuous_on* $(cball\ z\ r)\ (f\ n) \wedge$
$(\forall w \in ball\ z\ r.\ ((f\ n)$ *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 More on winding numbers

```
lemma winding_number_comp:
 assumes open S f holomorphic_on S path_image γ ⊆ S
         valid_path γ z ∉ path_image (f ∘ γ)
 shows winding_number (f ∘ γ) z = 1/(2∗pi∗i) ∗ contour_integral γ (λw. deriv f w / (f w − z))
proof −
 obtain spikes where finite spikes and γ_diff: γ C1_differentiable_on {0..1} − spikes
  using ‹valid_path γ› unfolding valid_path_def piecewise_C1_differentiable_on_def by auto
 have valid_path (f ∘ γ)
  using valid_path_compose_holomorphic assms by blast
 moreover have contour_integral (f ∘ γ) (λw. 1 / (w − z))
     = contour_integral γ (λw. deriv f w / (f w − z))
  unfolding contour_integral_integral
 proof (rule integral_spike[OF negligible_finite[OF ‹finite spikes›]])
  fix t::real assume t:t ∈ {0..1} − spikes
  then have γ differentiable at t
   using γ_diff unfolding C1_differentiable_on_eq by auto
  moreover have f field_differentiable at (γ t)
  proof −
   have γ t ∈ S using ‹path_image γ ⊆ S› t unfolding path_image_def by auto
   thus ?thesis
   using ‹open S› ‹f holomorphic_on S› holomorphic_on_imp_differentiable_at by blast
  qed
  ultimately show deriv f (γ t) / (f (γ t) − z) ∗ vector_derivative γ (at t) =
      1 / ((f ∘ γ) t − z) ∗ vector_derivative (f ∘ γ) (at t)
   by (simp add: vector_derivative_chain_at_general)
 qed
 moreover note ‹z ∉ path_image (f ∘ γ)›
 ultimately show ?thesis
  using winding_number_valid_path by presburger
qed
```

lemma *winding_number_uminus_comp*:
assumes *valid_path* γ $-$ $z \notin$ *path_image* γ
shows *winding_number* (*uminus* $\circ$ γ) z = *winding_number* γ ($-z$)
proof $-$
define *c* **where** *c*= *2* $*$ *pi* $*$ i
have *winding_number* (*uminus* $\circ$ γ) z = *1*/*c* $*$ *contour_integral* γ (λ*w*. *deriv uminus w* / ($-w-z$))
proof (*rule winding_number_comp*[*of UNIV*, *folded c_def*])
show *open UNIV uminus holomorphic_on UNIV path_image* γ $\subseteq$ *UNIV valid_path* γ
using ‹*valid_path* γ› **by** (*auto intro*:*holomorphic_intros*)
show $z \notin$ *path_image* (*uminus* $\circ$ γ)
unfolding *path_image_compose* **using** ‹$-$ $z \notin$ *path_image* γ› **by** *auto*
qed
also have ... = *1*/*c* $*$ *contour_integral* γ (λ*w*. *1* / (*w* $-$ ($-z$)))
by (*auto intro*!:*contour_integral_eq simp add*:*field_simps minus_divide_right*)
also have ... = *winding_number* γ ($-z$)
using *winding_number_valid_path*[*OF* ‹*valid_path* γ› ‹$-$ $z \notin$ *path_image* γ›,*folded c_def*]
by *simp*
finally show *?thesis* **by** *auto*
qed

lemma *winding_number_comp_linear*:
assumes *c*≠*0 valid_path* γ **and** *not_image*: (*z*$-$*b*)/*c* $\notin$ *path_image* γ
shows *winding_number* ((λ*x*. *c*$*$*x*+*b*) $\circ$ γ) z = *winding_number* γ ((*z*$-$*b*)/*c*) (**is** *?L* = *?R*)
proof $-$
define *cc* **where** *cc*=*1* / (*complex_of_real* (*2* $*$ *pi*) $*$ i)
define *zz* **where** *zz*=(*z*$-$*b*)/*c*
have *?L* = *cc* $*$ *contour_integral* γ (λ*w*. *deriv* (λ*x*. *c* $*$ *x* + *b*) *w* / (*c* $*$ *w* + *b* $-$ *z*))
apply (*subst winding_number_comp*[*of UNIV*,*simplified*])
subgoal by (*auto intro*:*holomorphic_intros*)
subgoal using ‹*valid_path* γ› **.**
subgoal using *not_image* ‹*c*≠*0*› **unfolding** *path_image_compose* **by** *auto*
subgoal unfolding *cc_def* **by** *auto*
done
also have ... = *cc* $*$ *contour_integral* γ (λ*w*.*1* / (*w* $-$ *zz*))
proof $-$
have *deriv* (λ*x*. *c* $*$ *x* + *b*) = (λ*x*. *c*)
by (*auto intro*:*derivative_intros*)
then show *?thesis*
unfolding *zz_def cc_def* **using** ‹*c*≠*0*›
by (*auto simp*:*field_simps*)
qed
also have ... = *winding_number* γ *zz*
using *winding_number_valid_path*[*OF* ‹*valid_path* γ› *not_image*,*folded zz_def cc_def*]

by *simp*
finally show *winding_number* $((\lambda x.\ c * x + b) \circ \gamma)\ z = \mathit{winding_number}\ \gamma\ zz$
.
qed

4.11 On analytic functions defined by a series

lemma *series_and_derivative_comparison*:
fixes $S :: \mathit{complex\ set}$
assumes S: *open* S
and h: *summable* h
and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ \mathit{has_field_derivative}\ f'\ n\ x)\ (\mathit{at}\ x)$
and *to_g*: $\forall_F\ n\ \mathit{in\ sequentially}.\ \forall x{\in}S.\ \mathit{norm}\ (f\ n\ x) \leq h\ n$
obtains $g\ g'$ **where** $\forall x \in S.\ ((\lambda n.\ f\ n\ x)\ \mathit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \mathit{sums}\ g'\ x) \wedge (g\ \mathit{has_field_derivative}\ g'\ x)\ (\mathit{at}\ x)$
proof −
obtain g **where** g: *uniform_limit* $S\ (\lambda n\ x.\ \sum i{<}n.\ f\ i\ x)\ g$ *sequentially*
using *Weierstrass_m_test_ev* [*OF to_g h*] **by** *force*
have $*$: $\exists d{>}0.\ \mathit{cball}\ x\ d \subseteq S \wedge \mathit{uniform_limit}\ (\mathit{cball}\ x\ d)\ (\lambda n\ x.\ \sum i{<}n.\ f\ i\ x)\ g$ *sequentially*
if $x \in S$ **for** x
using *open_contains_cball* [*of S*] ‹$x \in S$› S g *uniform_limit_on_subset* **by** *blast*
have $\bigwedge x.\ x \in S \Longrightarrow (\lambda n.\ \sum i{<}n.\ f\ i\ x) \longrightarrow g\ x$
by (*metis tendsto_uniform_limitI* [*OF g*])
moreover have $\exists g'.\ \forall x{\in}S.\ (g\ \mathit{has_field_derivative}\ g'\ x)\ (\mathit{at}\ x) \wedge (\lambda n.\ \sum i{<}n.\ f'\ i\ x) \longrightarrow g'\ x$
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 $S :: \mathit{complex\ set}$
assumes S: *open* S
and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ \mathit{has_field_derivative}\ f'\ n\ x)\ (\mathit{at}\ x)$
and *to_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists d\ h.\ 0 < d \wedge \mathit{summable}\ h \wedge (\forall_F\ n\ \mathit{in\ sequentially}.\ \forall y{\in}\mathit{ball}\ x\ d \cap S.\ \mathit{norm}\ (f\ n\ y) \leq h\ n)$
shows $\exists g\ g'.\ \forall x \in S.\ ((\lambda n.\ f\ n\ x)\ \mathit{sums}\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ \mathit{sums}\ g'\ x) \wedge (g\ \mathit{has_field_derivative}\ g'\ x)\ (\mathit{at}\ x)$
proof −
have $\exists y.\ (\lambda n.\ f\ n\ z)\ \mathit{sums}\ (\sum n.\ f\ n\ z) \wedge (\lambda n.\ f'\ n\ z)\ \mathit{sums}\ y \wedge ((\lambda x.\ \sum n.\ f\ n\ x)\ \mathit{has_field_derivative}\ y)\ (\mathit{at}\ z)$
if $z \in S$ **for** z
proof −
obtain $d\ h$ **where** $0 < d$ *summable* h **and** *le_h*: $\forall_F\ n\ \mathit{in\ sequentially}.\ \forall y{\in}\mathit{ball}\ z\ d \cap S.\ \mathit{norm}\ (f\ n\ y) \leq h\ n$

```
    using to_g ‹z ∈ S› by meson
  then obtain r where r>0 and r: ball z r ⊆ ball z d ∩ S using ‹z ∈ S› S
   by (metis Int_iff open_ball centre_in_ball open_Int open_contains_ball_eq)
  have 1: open (ball z d ∩ S)
    by (simp add: open_Int S)
  have 2: ⋀n x. x ∈ ball z d ∩ S ⟹ (f n has_field_derivative f' n x) (at x)
    by (auto simp: hfd)
  obtain g g' where gg': ∀x ∈ ball z d ∩ S. ((λn. f n x) sums g x) ∧
                                 ((λn. f' n x) sums g' x) ∧ (g has_field_derivative g'
x) (at x)
    by (auto intro: le_h series_and_derivative_comparison [OF 1 ‹summable h›
hfd])
  then have (λn. f' n z) sums g' z
    by (meson ‹0 < r› centre_in_ball contra_subsetD r)
  moreover have (λn. f n z) sums (∑ 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 ((λx. ∑ 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: ⋀n x. x ∈ S ⟹ (f n has_field_derivative f' n x) (at x)
   and to_g: ⋀x. x ∈ S ⟹ ∃ d h. 0 < d ∧ summable h ∧ range h ⊆ ℝ≥0 ∧ (∀F
n in sequentially. ∀ y∈ball x d ∩ S. cmod(f n y) ≤ cmod (h n))
 shows ∃ g g'. ∀ x ∈ S. ((λn. f n x) sums g x) ∧ ((λn. f' n x) sums g' x) ∧ (g
has_field_derivative g' x) (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: ⋀n x. x ∈ S ⟹ f n field_differentiable (at x)
    and to_g: ⋀x. x ∈ S ⟹ ∃ d h. 0 < d ∧ summable h ∧ range h ⊆ ℝ≥0 ∧ (∀F
n in sequentially. ∀ y∈ball x d ∩ S. cmod(f n y) ≤ cmod (h n))
  obtains g where ∀ x ∈ S. ((λn. f n x) sums g x) ∧ g field_differentiable (at x)
proof −
  have hfd': ⋀n x. x ∈ S ⟹ (f n has_field_derivative deriv (f n) x) (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 ⇒ complex and r::real
  assumes summable (λn. a n * r^n)
    shows ∃ g g'. ∀ z. cmod z < 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)
proof (cases 0 < r)
  case True
    have der: ⋀n z. ((λx. a n * x ^ n) has_field_derivative of_nat n * a n * z ^
(n−1)) (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)
```

```
  proof (rule has_field_derivative_transform_within [where d=(r - norm(z-w))/2])
   qed (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)
  using power_series_holomorphic [where a = λn. (deriv ^^ n) f z / (fact n)]
holomorphic_power_series
  by blast

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)

lemma power_series_analytic':
  assumes conv_radius a ≥ r
  shows   (λz. (∑n. a n * (z - w) ^ n)) analytic_on eball w r
proof (cases r)
  case [simp]: (real r')
  have (λz. ∑n. a n * (z - w) ^ n) analytic_on ball w r'
   by (rule power_series_analytic[of w r' a] summable_sums summable_in_conv_radius)+
     (use assms in ‹auto simp: dist_norm norm_minus_commute intro: ereal_le_less›)
  thus ?thesis
    by simp
next
  case [simp]: PInf
  have (λz. ∑n. a n * (z - w) ^ n) analytic_on ball w r' for r'
   by (rule power_series_analytic[of w r' a] summable_sums summable_in_conv_radius)+
     (use assms in ‹auto simp: dist_norm norm_minus_commute intro: ereal_le_less›)
  hence (λz. ∑n. a n * (z - w) ^ n) analytic_on (⋃r'. ball w r')
    by (subst analytic_on_Union) auto
  also have (⋃r'. ball w r') = UNIV
  proof -
    have z ∈ ball w (dist w z + 1) for z
      by auto
    thus ?thesis by blast
  qed
  finally show ?thesis
    by simp
qed (auto intro: analytic_on_empty)
```

4.12 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* λ*n*. (*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* = λ*z*. *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 (⋂*n*. {*w* ∈ *S*. (*deriv* ^^ *n*) *f w* = *0*}) = *S* ⟹ *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 ∈ 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 ⋀n. (deriv ^^ n) f z = (deriv ^^ n) g z
      and z ∈ S w ∈ S
    shows f w = g w
proof (rule holomorphic_fun_eq_0_on_connected [of λx. f x − g x S z, simplified])
  show (λx. f x − g x) holomorphic_on S
    by (intro assms holomorphic_intros)
  show ⋀n. (deriv ^^ n) (λx. 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: ⋀n. 0 < n ⟹ (deriv ^^ n) f z = 0
      and z ∈ S w ∈ S
    shows f w = f z
proof (rule holomorphic_fun_eq_0_on_connected [of λw. f w − f z S z, simplified])
  show (λw. f w − f z) holomorphic_on S
    by (intro assms holomorphic_intros)
  show ⋀n. (deriv ^^ n) (λw. f w − f z) z = 0
    by (subst higher_deriv_diff) (use assms in ‹auto intro: holomorphic_intros›)
qed (use assms in auto)
```

4.13 Some basic lemmas about poles/singularities

```
lemma pole_lemma:
  assumes holf: f holomorphic_on S and a: a ∈ interior S
    shows (λz. 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 ∈ S u ≠ a for u
  proof −
    have fcd: f field_differentiable at u within S
      using holf holomorphic_on_def by (simp add: ‹u ∈ 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_imp_differentiable_at)
  qed
  ultimately show ?thesis
    by (metis (lifting) a at_within_interior holomorphic_onI mem_interior)
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
  then have (λz. (f z − f a) / (z−a)) field_differentiable at x within S
    if x ∈ S for x
    using assms that unfolding holomorphic_on_def
    by (intro derivative_intros) auto
  with False show ?thesis
    using holomorphic_on_def holomorphic_transform by presburger
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 ∈ 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: ⋀w. w ∈ ball x d − {a} ⟹ g w = (w−a) ∗ f w
    using ‹x ∈ S› eq by blast
```

have *?F holomorphic_on ball x (min d e)*
using *d e ‹x ∈ S›* **by** (*fastforce simp*: *holomorphic_on_subset subset_ball intro*!: *pole_theorem_open*)
then show ∃ *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*: ⋀*z*. *z ∈ S* ⟹ ∃ *d*. *0 < d* ∧ (∀ *w ∈ ball z d − {a}*. *g w = (w−a) * f w*)
and [*simp*]: *f a = deriv g a g a = 0*
shows *f analytic_on S*
proof −
have [*simp*]: (λ*z*. *if z = a then deriv g a 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 ⊆ T open T*
and *eq*: ⋀*z*. *z ∈ T − {a}* ⟹ *g z = (z−a) * f z*
shows (λ*z*. *if z = a then deriv g a else f z − g a/(z−a)*) *analytic_on S*
proof (*rule pole_theorem_analytic* [*OF g*])
fix *z*
assume *z ∈ S*
then obtain *e* **where** *0 < e* **and** *e*: *ball z e ⊆ T*
using *assms openE* **by** *blast*
then show ∃ *d>0*. ∀ *w∈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 ⊆ T open T* ⋀*z*. *z ∈ T − {a}* ⟹ *g z = (z−a) * f z*
and [*simp*]: *f a = deriv g a g a = 0*
shows *f analytic_on S*
proof −
have [*simp*]: (λ*z*. *if z = a then deriv g a else f z − g a / (z−a)*) = *f*
by *auto*
have (λ*z*. *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

4.14 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*: $\bigwedge w.\ w \in U \Longrightarrow F\ w$ *contour_integrable_on* (*linepath a b*)
and *cond_uu*: *continuous_on* $(U \times U)\ (\lambda(x,y).\ F\ x\ y)$
and *abu*: *closed_segment a b* $\subseteq U$
shows *continuous_on U* $(\lambda w.$ *contour_integral* (*linepath a b*) $(F\ w))$
proof −
have $\exists d{>}0.\ \forall x'{\in}U.$ *dist* $x'\ w < d \longrightarrow$
dist (*contour_integral* (*linepath a b*) $(F\ x')$)
(*contour_integral* (*linepath a b*) $(F\ w)) \le \varepsilon$
if $w \in U\ 0 < \varepsilon\ a \neq b$ **for** $w\ \varepsilon$
proof −
obtain δ **where** $\delta{>}0$ **and** δ: *cball w* $\delta \subseteq U$ **using** *open_contains_cball* ‹*open U*› ‹$w \in U$› **by** *force*
let *?TZ* = *cball w* δ × *closed_segment a b*
have *uniformly_continuous_on ?TZ* $(\lambda(x,y).\ F\ x\ y)$
by (*metis Sigma_mono* δ *abu compact_Times compact_cball compact_segment compact_uniformly_continuous*
cond_uu continuous_on_subset)
then obtain η **where** $\eta{>}0$
and η: $\bigwedge x\ x'.\ [\![x{\in}?TZ;\ x'{\in}?TZ;$ *dist* $x'\ x < \eta]\!] \Longrightarrow$
dist $((\lambda(x,y).\ F\ x\ y)\ x')\ ((\lambda(x,y).\ F\ x\ y)\ x) < \varepsilon/norm(b-a)$
using ‹$0 < \varepsilon$› ‹$a \neq b$›
by (*auto elim*: *uniformly_continuous_onE* [**where** $\varepsilon = \varepsilon/norm(b-a)$])
have η: $[\![norm\ (w - x1) \le \delta;\ \ x2 \in$ *closed_segment a b*;
$norm\ (w - x1') \le \delta;\ \ x2' \in$ *closed_segment a b*; $norm\ ((x1', x2') - (x1, x2)) < \eta]\!]$
$\Longrightarrow norm\ (F\ x1'\ x2' - F\ x1\ x2) \le \varepsilon\ /\ cmod\ (b-a)$
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*) $(\lambda x.\ F\ x'\ x - F\ w\ x)) \le \varepsilon$
if $x' \in U$ *cmod* $(x' - w) < \delta$ *cmod* $(x' - w) < \eta$ **for** x'
proof −
have $(\lambda x.\ F\ x'\ x - F\ w\ x)$ *contour_integrable_on linepath a b*
by (*simp add*: ‹$w \in U$› *cont_dw contour_integrable_diff that*)
then have *cmod* (*contour_integral* (*linepath a b*) $(\lambda x.\ F\ x'\ x - F\ w\ x)) \le \varepsilon/norm(b-a) * norm(b-a)$
using *has_contour_integral_bound_linepath* [*OF has_contour_integral_integral* _ η]
using ‹$0 < \varepsilon$› ‹$0 < \delta$› *that* **by** (*force simp*: *norm_minus_commute*)
also have $\ldots = \varepsilon$ **using** ‹$a \neq b$› **by** *simp*
finally show *?thesis* **.**
qed
show *?thesis*
using ‹$0 < \delta$› ‹$0 < \eta$› ‹$w \in U$›

apply (*intro exI*[**where** *x*=*min δ η*])
by (*auto simp*: *dist_norm contour_integral_diff* [*OF cont_dw cont_dw*, *symmetric*] *intro*: *le_ee*)
qed
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *continuous_onI continuous_on_iff contour_integral_trivial dist_self*)
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*: $z \in U$ **and** γ: *polynomial_function* γ
and *pasz*: *path_image* $\gamma \subseteq U - \{z\}$ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *zero*: $\bigwedge w.\ w \notin U \Longrightarrow$ *winding_number* $\gamma\ w = 0$
shows ((λ*w*. *f w* / (*w*−*z*)) *has_contour_integral* (2∗*pi* ∗ i ∗ *winding_number* $\gamma\ z$ ∗ *f z*)) γ
proof −
obtain γ' **where** *pfγ'*: *polynomial_function* γ' **and** γ': $\bigwedge x.$ (γ *has_vector_derivative* ($\gamma'\ x$)) (*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*: $\bigwedge x.\ x \in$ *path_image* $\gamma' \Longrightarrow$ *norm* $x \le B$
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* $\gamma \wedge$ *winding_number* $\gamma\ w = 0$}
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 ∗: (λ*c*. *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* (λ*x*. *if x* = *y then deriv f y else* (*f x* − *f y*) / (*x*−*y*)) *y*
using *at_within_open field_differentiable_imp_continuous_at holomorphic_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 (λ*w*. *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 ((λx. 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 ((λx. f z ∗ (1 / (x−z))) has_contour_integral 0) γ
    using has_contour_integral_lmul by fastforce
  then have ((λx. f z / (x−z)) has_contour_integral 0) γ
    by (simp add: field_split_simps)
 moreover have ((λx. (f x − f z) / (x−z)) has_contour_integral contour_integral
γ (d z)) γ
```

by (*metis* (*no_types*, *lifting*) *z cint_fxy contour_integral_eq d_def has_contour_integral_integral mem_Collect_eq v_def*)
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*
by (*metis add.commute diff_add_cancel dist_0_norm dist_commute dist_norm less_eq_real_def*)
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*: $w \in$ *closed_segment* x' $z' \Longrightarrow z' \neq x' \Longrightarrow$ *cmod* (*deriv f w* − *deriv f x*) $\leq e$ **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*: $\bigwedge x.\ x \in U \Longrightarrow$ (*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' $z' \subseteq U$
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 ∗: ((λ*x*. *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*: $\bigwedge T.\ T \in \{0..1\} \Longrightarrow$ *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*: $\bigwedge w.\ w \in v$ − *path_image* $\gamma \Longrightarrow$ ((λ*u*. *f u* / (*u*−*w*) ^ 1) *has_contour_integral h w*) γ

by (*simp add*: *V*)
have $\bigwedge x\ y.\ [\![x \in U;\ y \in U;\ y \neq x]\!] \Longrightarrow (\lambda(x,\ y).\ d\ x\ y)\ -(x,\ y)\rightarrow\ (f\ y - f\ x)\ /\ (y - x)$
unfolding *d_def*
apply (*rule Lim_transform_within_open* [*OF* _ *open_uu_Id*, **where** *f* = $(\lambda(x,y).\ (f\ y - f\ x)\ /\ (y-x))$])
using *con_ff* **by** (*auto simp*: *continuous_within*)
then have *cond_uu*: *continuous_on* $(U \times U)\ (\lambda(x,y).\ d\ x\ y)$
unfolding *continuous_on_eq_continuous_within continuous_within d_def*
by (*fastforce simp add*: *tendsto_f' intro*: *Lim_at_imp_Lim_at_within*)
have *hol_dw*: $(\lambda z.\ d\ z\ w)$ *holomorphic_on U* **if** $w \in U$ **for** *w*
proof −
have *continuous_on U* $((\lambda(x,y).\ d\ x\ y) \circ (\lambda z.\ (w,z)))$
by (*rule continuous_on_compose continuous_intros continuous_on_subset* [*OF cond_uu*] | *force intro*: *that*)+
then have ∗: *continuous_on U* $(\lambda z.\ if\ w = z\ then\ deriv\ f\ z\ else\ (f\ w - f\ z)\ /\ (w-z))$
by (*rule rev_iffD1* [*OF* _ *continuous_on_cong* [*OF refl*]]) (*simp add*: *d_def field_simps*)
have ∗∗: $(\lambda z.\ if\ w = z\ then\ deriv\ f\ z\ else\ (f\ w - f\ z)\ /\ (w-z))$ *field_differentiable at x*
if $x \in U\ x \neq w$ **for** *x*
proof (*rule_tac* $f = \lambda x.\ (f\ w - f\ x)/(w-x)$ **and** $d = dist\ x\ w$ **in** *field_differentiable_transform_within*)
show $(\lambda x.\ (f\ w - f\ x)\ /\ (w-x))$ *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 $(\lambda z.\ if\ w = z\ then\ deriv\ f\ z\ else\ (f\ w - f\ z)\ /\ (w-z))$ *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* $\subseteq U$
have *cont_cint_d*: *continuous_on U* $(\lambda w.\ contour_integral\ (linepath\ a\ b)\ (\lambda z.\ d\ z\ w))$
proof (*rule contour_integral_continuous_on_linepath_2D* [*OF* ‹*open U*› _ _ *abu*])
show $\bigwedge w.\ w \in U \Longrightarrow (\lambda z.\ d\ z\ w)$ *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* $(U \times U)\ (\lambda(x,\ y).\ d\ y\ x)$
by (*auto intro*: *continuous_on_swap_args cond_uu*)
qed

have *cont_cint_dγ*: *continuous_on* {*0..1*} ((λ*w*. *contour_integral* (*linepath a b*) (λ*z*. *d z w*)) ∘ γ)
by (*metis Diff_subset* ‹*path γ*› *cont_cint_d continuous_on_compose continuous_on_subset pasz path_def path_image_def*)
have *continuous_on* {*0..1*} (λ*x*. *vector_derivative* γ (*at x*))
using *pfγ′* **by** (*simp add*: *continuous_on_polymonial_function vector_derivative_at* [*OF γ′*])
then have *cint_cint*: (λ*w*. *contour_integral* (*linepath a b*) (λ*z*. *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*) (λ*z*. *contour_integral* γ (*d z*))
using *abu* **by** (*force simp*: *h_def intro*: *contour_integral_eq*)
also have ... = *contour_integral* γ (λ*w*. *contour_integral* (*linepath a b*) (λ*z*. *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*} (λ*t*. *vector_derivative* (*linepath a b*) (*at t*))
by (*auto intro*!: *continuous_intros*)
show *continuous_on* {*0..1*} (λ*t*. *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 (λ*w*. *contour_integral* (*linepath a b*) (λ*z*. *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) (λ*n*. *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 ε = 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* ⟹ (λ*w*. *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*) (λ*z*. *d z w*) = *0*
proof (*rule contour_integral_unique* [*OF Cauchy_theorem_triangle*])
show ⋀*w*. *w* ∈ *U* ⟹ (λ*z*. *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 ⋀*z*. *z* ∈ *path_image* γ ⟹
contour_integral (*linepath a b*) (λ*x*. *d x z*) +
(*contour_integral* (*linepath b c*) (λ*x*. *d x z*) +
contour_integral (*linepath c a*) (λ*x*. *d x z*)) = *0*
using *abc pasz U* ∗ *eq0* **by** *auto*
then show *?thesis*
by (*simp add*: *contour_integral_eq_0 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 ((λ*w*. *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 ((λ*w*. *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*: ((λ*w*. *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*: ((λ*w*. (*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)) holo-
morphic_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*: $\bigwedge w.\ w \notin S \Longrightarrow$ *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* $\gamma \subseteq S$
and *zero*: $\bigwedge w.\ w \notin S \Longrightarrow$ *winding_number* $\gamma\ w = 0$
shows (*f has_contour_integral 0*) γ
proof −
have *path_image* $\gamma \neq S$
by (*metis compact_valid_path_image vpg compact_open path_image_nonempty S*)
then obtain *z* **where** $z \in S$ **and** *znot*: $z \notin$ *path_image* γ **and** *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 ⊆ *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* ⋀*x*. *x* ∈ *A* ⟹ *f x* ≠ *0*
and *z0*: *z0* ∈ *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 (λ*x*. *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* = (λ*x*. −*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

4.15 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 cmod ((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
    using ‹0 < r› higher_deriv_diff holf by auto
  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 ⟹ (λn. 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 (λi. 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 (λn. (a m * (x−ξ)^m) * (b n * (x−ξ)^n)) sums (f x)
   using am az sm sums_zero_iff_shift [of m (λn. a n * (x−ξ) ^ n) f x]
   by (simp add: b_def monoid_mult_class.power_add algebra_simps)
  then have x ≠ ξ ⟹ (λn. 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 (λn. 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
  show ?thesis
  proof
    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
    show ⋀z. z ∈ cball ξ (min s (r / 2)) − {ξ} ⟹ f z ≠ 0
      by (simp add: * dist_norm norm_minus_commute)
  qed (use ‹0 < r› ‹0 < s› in auto)
qed
```

4.16 The uniform Weierstrass convergence theorem

uniform convergence of the derivatives

```
lemma deriv_uniform_limit_cball:
  fixes f :: 'n ⇒ complex ⇒ complex
  assumes cont: eventually (λn. continuous_on (cball z r) (f n) ∧ f n holomor-
phic_on ball z r) F
      and ulim: uniform_limit (cball z r) f g F
      and F: F ≠ bot
      and s: 0 < s s < r
  shows uniform_limit (cball z s) (λn. deriv (f n)) (deriv g) F
proof −
  define ϱ where ϱ = r − s
  have ϱ: ϱ > 0
    using s by (simp add: ϱ_def)
  have sub_ball: ball w ϱ ⊆ ball z r if w: w ∈ cball z s for w
    using ϱ_def that by (smt (verit, best) ball_subset_ball_iff dist_commute
mem_cball)
  have sub_cball: cball w ϱ ⊆ cball z r if w: w ∈ cball z s for w
    using ϱ_def w by (smt (verit, best) cball_subset_cball_iff dist_commute
```

```
mem_cball)
  — the limit function is itself holomorphic
  have contg: continuous_on (cball z r) g and holg: g holomorphic_on ball z r
    using holomorphic_uniform_limit[OF cont ulim F] by blast+
  show ?thesis
    unfolding uniform_limit_iff
  proof (intro allI impI)
    fix ε :: real assume ε: ε > 0
    define B where B = ε * ϱ / 2
    have B: B > 0
      unfolding B_def using ε ϱ by simp
    from ulim have ∀ F n in F. ∀ x∈cball z r. dist (f n x) (g x) < B
      using B unfolding uniform_limit_iff by blast
    with cont show ∀ F n in F. ∀ w∈cball z s. dist (deriv (f n) w) (deriv g w) < ε
    proof eventually_elim
      case (elim n)
      then have contfn: continuous_on (cball z r) (f n)
        and holfn: f n holomorphic_on ball z r by auto
      from elim have close: norm (f n x − g x) ≤ B if x ∈ cball z r for x
        using that by (simp add: dist_norm less_imp_le)
      show ?case
      proof (intro ballI)
        fix w assume w: w ∈ cball z s
        have wball: w ∈ ball z r
          using w s by auto
        — the derivative of the difference, estimated at the centre w
        have norm ((deriv ^^ 1) (λx. f n x − g x) w) ≤ fact 1 * B / ϱ ^ 1
        proof (rule Cauchy_inequality)
          show (λx. f n x − g x) holomorphic_on ball w ϱ
            using holomorphic_on_subset[OF holfn sub_ball[OF w]]
                  holomorphic_on_subset[OF holg sub_ball[OF w]]
            by (intro holomorphic_intros)
          show continuous_on (cball w ϱ) (λx. f n x − g x)
            using continuous_on_subset[OF contfn sub_cball[OF w]]
                  continuous_on_subset[OF contg sub_cball[OF w]]
            by (intro continuous_intros)
          show 0 < ϱ
            by (rule ϱ)
          fix x assume norm (w − x) = ϱ
          hence x ∈ cball w ϱ
            by (simp add: dist_norm norm_minus_commute)
          with sub_cball[OF w] show norm (f n x − g x) ≤ B
            by (intro close) blast
        qed
        moreover have deriv (λx. f n x − g x) w = deriv (f n) w − deriv g w
          using holfn holg wball
          by (intro deriv_diff holomorphic_on_imp_differentiable_at) auto
        ultimately have norm (deriv (f n) w − deriv g w) ≤ B / ϱ
          by simp
```

```
      also have B / ϱ < ε
        unfolding B_def using ϱ ε by simp
      finally show dist (deriv (f n) w) (deriv g w) < ε
        by (simp add: dist_norm)
    qed
  qed
 qed
qed
```

The version for compact subsets of an open set: locally uniform convergence of the functions gives locally uniform convergence of the derivatives.

```
lemma deriv_uniform_limit_on_compact:
 fixes f :: 'n ⇒ complex ⇒ complex
 assumes ulim: ⋀L. compact L ⟹ L ⊆ A ⟹ uniform_limit L f g F
     and holf: eventually (λn. f n holomorphic_on A) F
     and F: F ≠ bot and A: open A
     and K: compact K K ⊆ A
 shows uniform_limit K (λn. deriv (f n)) (deriv g) F
proof −
 — around each point of K choose a ball whose double still lies in A
 have ∃d>0. cball w (2 * d) ⊆ A if w ∈ K for w
 proof −
  from that K have w ∈ A by blast
  with A obtain e where e: e > 0 ball w e ⊆ A
   using openE by blast
  have cball w (2 * (e/4)) ⊆ ball w e
   using e by auto
  with e show ?thesis
   by (intro exI[of _ e/4]) auto
 qed
 then obtain d where d: ⋀w. w ∈ K ⟹ d w > 0
  and dA: ⋀w. w ∈ K ⟹ cball w (2 * d w) ⊆ A
  by metis
 — finitely many of the small balls already cover K
 have K ⊆ (⋃w∈K. ball w (d w))
  using d by auto
 then obtain T where T: T ⊆ K finite T K ⊆ (⋃w∈T. ball w (d w))
  using K(1)
  by (meson Elementary_Metric_Spaces.open_ball compactE_image)
 — on each of them the previous lemma applies, with the doubled radius as the
outer one
 have step: uniform_limit (cball w (d w)) (λn. deriv (f n)) (deriv g) F if w: w ∈
T for w
 proof (rule deriv_uniform_limit_cball[where r = 2 * d w])
  have wK: w ∈ K
   using w T by blast
  have sub: cball w (2 * d w) ⊆ A
   by (rule dA[OF wK])
  show ∀F n in F. continuous_on (cball w (2 * d w)) (f n) ∧
```

```
                f n holomorphic_on ball w (2 * d w)
    using holf
  proof eventually_elim
    case (elim n)
    have hol: f n holomorphic_on cball w (2 * d w)
      by (rule holomorphic_on_subset[OF elim sub])
    have continuous_on (cball w (2 * d w)) (f n)
      by (rule holomorphic_on_imp_continuous_on[OF hol])
    moreover have f n holomorphic_on ball w (2 * d w)
      by (rule holomorphic_on_subset[OF hol ball_subset_cball])
    ultimately show ?case
      by blast
  qed
  show uniform_limit (cball w (2 * d w)) f g F
    using dA[OF wK] by (intro ulim) auto
  show F ≠ bot
    by (rule F)
  show 0 < d w
    by (rule d[OF wK])
  show d w < 2 * d w
    using d[OF wK] by simp
 qed
 have uniform_limit (⋃ w∈T. cball w (d w)) (λn. deriv (f n)) (deriv g) F
   using T(2) step by (intro uniform_limit_on_UNION) auto
 moreover have K ⊆ (⋃ w∈T. cball w (d w))
   using T(3) by force
 ultimately show ?thesis
   by (rule uniform_limit_on_subset)
qed
```

The logarithmic derivative of a product, over an arbitrary index set (*has_field_derivative_prod'* needs no finiteness: for an infinite set both sides are those of the empty product).

```
lemma deriv_prod_logderiv:
 fixes f :: 'a ⇒ complex ⇒ complex
 assumes holf: ⋀k. k ∈ S ⟹ f k holomorphic_on A and A: open A and z: z ∈
A
   and nz: ⋀k. k ∈ S ⟹ f k z ≠ 0
 shows deriv (λx. ∏ k∈S. f k x) z = (∏ k∈S. f k z) * (∑ k∈S. deriv (f k) z / f
k z)
proof (rule DERIV_imp_deriv, rule has_field_derivative_prod')
 show ⋀k. k ∈ S ⟹ f k z ≠ 0
   using nz by blast
 show ⋀k. k ∈ S ⟹ (f k has_field_derivative deriv (f k) z) (at z)
   using holf A z by (blast intro: holomorphic_derivI)
qed

theorem logderiv_prod_uniform_limit:
 fixes f :: 'a ⇒ complex ⇒ complex and idx :: 'n ⇒ 'a set and F :: 'n filter
```

```
  assumes ulim: ⋀L. compact L ⟹ L ⊆ A ⟹
                uniform_limit L (λn x. ∏k∈idx n. f k x) P F
    and holf: ⋀k. k ∈ I ⟹ f k holomorphic_on A
    and A: open A
    and nzP: ⋀z. z ∈ A ⟹ P z ≠ 0
    and fin: ∀F n in F. finite (idx n) ∧ idx n ⊆ I
    and cover: ⋀k. k ∈ I ⟹ ∀F n in F. k ∈ idx n
    and F: F ≠ bot
    and K: compact K K ⊆ A
  shows uniform_limit K (λn z. ∑k∈idx n. deriv (f k) z / f k z)
                        (λz. deriv P z / P z) F
proof −
  have holg: ∀F n in F. (λx. ∏k∈idx n. f k x) holomorphic_on A
    using fin by eventually_elim (auto intro!: holomorphic_intros holf)
  have ptw: ((λn. ∏k∈idx n. f k z) ⟶ P z) F if z ∈ A for z
  proof −
    have uniform_limit {z} (λn x. ∏k∈idx n. f k x) P F
      using that by (intro ulim) auto
    from tendsto_uniform_limitI[OF this] show ?thesis
      by simp
  qed
  — a non-zero product forces every factor to be non-zero
  have nzf: f k z ≠ 0 if z: z ∈ A and k: k ∈ I for k z
  proof
    assume fz: f k z = 0
    have ∀F n in F. (∏j∈idx n. f j z) = 0
      using fin cover[OF k] by eventually_elim (use fz in ‹auto intro: prod_zero›)
    hence ((λn. ∏j∈idx n. f j z) ⟶ 0) F
      by (rule tendsto_eventually)
    with ptw[OF z] have P z = 0
      using tendsto_unique F by blast
    with nzP[OF z] show False
      by simp
  qed
  — the derivatives of the partial products converge uniformly on K
  have ulim_deriv: uniform_limit K (λn. deriv (λx. ∏k∈idx n. f k x)) (deriv P)
F
    by (rule deriv_uniform_limit_on_compact[OF ulim holg F A K])
  have ulimK: uniform_limit K (λn x. ∏k∈idx n. f k x) P F
    using K by (intro ulim)
  have ntl: ¬ trivial_limit F
    using F by simp
  — P and deriv P are continuous on K, being uniform limits
  have contdg: ∀F n in F. continuous_on K (deriv (λx. ∏k∈idx n. f k x))
    using holg
  proof eventually_elim
    case (elim n)
    show ?case
      using holomorphic_on_imp_continuous_on[OF holomorphic_deriv[OF elim
```

A]] *K*(*2*)
by (*rule continuous_on_subset*)
qed
have *contdP*: *continuous_on K* (*deriv P*)
using *contdg ulim_deriv ntl* **by** (*rule uniform_limit_theorem*)
have *contg*: ∀ *F n in F*. *continuous_on K* (λ*x*. ∏ *k*∈*idx n*. *f k x*)
using *holg*
proof *eventually_elim*
case (*elim n*)
show *?case*
using *holomorphic_on_imp_continuous_on*[*OF elim*] *K*(*2*) **by** (*rule continuous_on_subset*)
qed
have *contP*: *continuous_on K P*
using *contg ulimK ntl* **by** (*rule uniform_limit_theorem*)
have *bddP*: *bounded* (*deriv P* ‘ *K*)
using *contdP K*(*1*) **by** (*intro compact_imp_bounded compact_continuous_image*) *auto*
— and *P* is bounded away from *0* on the compact set *K*
have ∃ *b*>*0*. ∀ *z*∈*K*. *b* ≤ *norm* (*P z*)
proof (*cases K* = {})
case *True*
thus *?thesis*
by (*intro exI*[*of* _ *1*]) *auto*
next
case *False*
have *continuous_on K* (λ*z*. *norm* (*P z*))
using *contP* **by** (*intro continuous_intros*)
then obtain *z0* **where** *z0*: *z0* ∈ *K* ∀ *z*∈*K*. *norm* (*P z0*) ≤ *norm* (*P z*)
using *K*(*1*) *False continuous_attains_inf* **by** *blast*
have *norm* (*P z0*) > *0*
using *z0*(*1*) *K*(*2*) *nzP* **by** *auto*
with *z0* **show** *?thesis* **by** *blast*
qed
then obtain *b* **where** *b*: *b* > *0* ⋀*z*. *z* ∈ *K* ⟹ *b* ≤ *norm* (*P z*)
by *blast*
have *quot*: *uniform_limit K* (λ*n z*. *deriv* (λ*u*. ∏ *k*∈*idx n*. *f k u*) *z* / (∏ *k*∈*idx n*. *f k z*))
(λ*z*. *deriv P z* / *P z*) *F*
by (*rule uniform_lim_divide*[*OF ulim_deriv ulimK bddP b*(*2*) *b*(*1*)])
have *idt*: ∀ *F n in F*. ∀ *z*∈*K*. *deriv* (λ*u*. ∏ *k*∈*idx n*. *f k u*) *z* / (∏ *k*∈*idx n*. *f k z*)
= (∑ *k*∈*idx n*. *deriv* (*f k*) *z* / *f k z*)
using *fin*
proof *eventually_elim*
case (*elim n*)
show *?case*
proof (*intro ballI*)
fix *z* **assume** *z* ∈ *K*
then have *zA*: *z* ∈ *A*

```
      using K(2) by blast
    have nz: ⋀k. k ∈ idx n ⟹ f k z ≠ 0
      using elim nzf[OF zA] by blast
    have deriv (λu. ∏k∈idx n. f k u) z
          = (∏k∈idx n. f k z) * (∑k∈idx n. deriv (f k) z / f k z)
      using elim holf nz by (intro deriv_prod_logderiv[OF _ A zA]) blast+
    moreover have (∏k∈idx n. f k z) ≠ 0
      using elim nz by auto
    ultimately show deriv (λu. ∏k∈idx n. f k u) z / (∏k∈idx n. f k z)
                  = (∑k∈idx n. deriv (f k) z / f k z)
      by simp
  qed
 qed
 have uniform_limit K (λn z. deriv (λu. ∏k∈idx n. f k u) z / (∏k∈idx n. f k
z))
                      (λz. deriv P z / P z) F
    = uniform_limit K (λn z. ∑k∈idx n. deriv (f k) z / f k z)
                      (λz. deriv P z / P z) F
   by (rule uniform_limit_cong[OF idt]) simp
 with quot show ?thesis
   by simp
qed
```

The ordered instance: the schedule of initial segments along *sequentially*.

```
corollary logderiv_prodinf_uniform_limit:
 fixes f :: nat ⇒ complex ⇒ complex
 assumes ulim: ⋀L. compact L ⟹ L ⊆ A ⟹
                 uniform_limit L (λn x. ∏k<n. f k x) P sequentially
   and holf: ⋀k. f k holomorphic_on A
   and A: open A
   and nzP: ⋀z. z ∈ A ⟹ P z ≠ 0
   and K: compact K K ⊆ A
 shows uniform_limit K (λn z. ∑k<n. deriv (f k) z / f k z)
                       (λz. deriv P z / P z) sequentially
proof (rule logderiv_prod_uniform_limit[where idx = λn. {..<n} and I = UNIV])
 show ⋀k. k ∈ UNIV ⟹ ∀F n in sequentially. k ∈ {..<n}
   by (simp add: eventually_gt_at_top)
qed (use assms in auto)
```

The unordered instance, along *finite_subsets_at_top I*: an *unconditionally* uniformly convergent product has an *unconditionally* uniformly convergent logarithmic derivative. This is the version that fits the rest of this theory, since the hypothesis is uniform convergence of *has_setprod* and the conclusion is uniform convergence of *has_sum*; the index type need not be *nat*. It composes directly with *uniform_limit_prodinf'* below, with no detour through the initial segments.

```
corollary logderiv_infprod_uniform_limit:
 fixes f :: 'a ⇒ complex ⇒ complex
```

```
  assumes ulim: ⋀L. compact L ⟹ L ⊆ A ⟹
                uniform_limit L (λX x. ∏k∈X. f k x) P (finite_subsets_at_top I)
    and holf: ⋀k. k ∈ I ⟹ f k holomorphic_on A
    and A: open A
    and nzP: ⋀z. z ∈ A ⟹ P z ≠ 0
    and K: compact K K ⊆ A
  shows uniform_limit K (λX z. ∑k∈X. deriv (f k) z / f k z)
                        (λz. deriv P z / P z) (finite_subsets_at_top I)
proof (rule logderiv_prod_uniform_limit[where idx = λX. X and I = I])
  show ∀F X in finite_subsets_at_top I. finite X ∧ X ⊆ I
    by (intro eventually_finite_subsets_at_top_weakI) auto
  show ∀F X in finite_subsets_at_top I. k ∈ X if k ∈ I for k
    unfolding eventually_finite_subsets_at_top using that by (intro exI[of _
{k}]) auto
qed (use assms in auto)
```

Read at a single point, the unordered version says that the logarithmic derivative is summable in the unordered sense, with sum *deriv P z / P z*.

```
corollary has_sum_logderiv_infprod:
  fixes f :: 'a ⇒ complex ⇒ complex
  assumes ulim: ⋀L. compact L ⟹ L ⊆ A ⟹
                uniform_limit L (λX x. ∏k∈X. f k x) P (finite_subsets_at_top I)
    and holf: ⋀k. k ∈ I ⟹ f k holomorphic_on A
    and A: open A
    and nzP: ⋀z. z ∈ A ⟹ P z ≠ 0
    and z: z ∈ A
  shows ((λk. deriv (f k) z / f k z) has_sum deriv P z / P z) I
proof −
  have uniform_limit {z} (λX z. ∑k∈X. deriv (f k) z / f k z)
                         (λz. deriv P z / P z) (finite_subsets_at_top I)
    using z by (intro logderiv_infprod_uniform_limit[OF ulim holf A nzP]) auto
  from tendsto_uniform_limitI[OF this] show ?thesis
    unfolding has_sum_def by simp
qed
```

4.17 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*) $\geq r$
and *eval_fps_expansion*: $\bigwedge z.\ z \in eball\ z0\ r \Longrightarrow eval_fps\ (fps_expansion\ f\ z0)\ (z - z0) = f\ z$
and *eval_fps_expansion'*: $\bigwedge z.\ norm\ z < r \Longrightarrow eval_fps\ (fps_expansion\ f\ z0)\ z = f\ (z0 + z)$
proof −
have $(\lambda n.\ fps_nth\ (fps_expansion\ f\ z0)\ n * (z - z0)\ \hat{}\ n)$ *sums* $f\ z$
if $z \in ball\ z0\ r'$ *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 $*$: $(\lambda n.\ fps_nth\ (fps_expansion\ f\ z0)\ n * (z - z0)\ \hat{}\ n)$ *sums* $f\ z$
if $z \in eball\ z0\ r$ **for** z
using *that* **by** (*subst* (*asm*) *eball_conv_UNION_balls*) *blast*
show *fps_conv_radius* (*fps_expansion f z0*) $\geq r$ **unfolding** *fps_conv_radius_def*
proof (*rule conv_radius_geI_ex*)
fix $r' :: real$ **assume** r': $r' > 0$ *ereal* $r' < r$
thus $\exists z.\ norm\ z = r' \wedge summable\ (\lambda n.\ fps_nth\ (fps_expansion\ f\ z0)\ n * z\ \hat{}\ n)$
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** $z \in eball\ z0\ r$ **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 $\bigwedge z.\ ereal\ (norm\ z) < r \Longrightarrow eval_fps\ f\ z \neq 0$
shows *fps_conv_radius_inverse*: *fps_conv_radius* (*inverse f*) $\geq min\ r\ (fps_conv_radius\ f)$
and *eval_fps_inverse*: $\bigwedge z.\ ereal\ (norm\ z) < fps_conv_radius\ f \Longrightarrow ereal\ (norm\ z) < r \Longrightarrow$
$eval_fps\ (inverse\ f)\ z = inverse\ (eval_fps\ f\ z)$
proof −
define R **where** $R = min\ (fps_conv_radius\ f)\ r$
have $*$: *fps_conv_radius* (*inverse f*) $\geq min\ r\ (fps_conv_radius\ f) \wedge$
$(\forall z \in eball\ 0\ (min\ (fps_conv_radius\ f)\ r).\ eval_fps\ (inverse\ f)\ z = inverse$

```
(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
      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

 show fps_conv_radius (inverse f) ≥ min r (fps_conv_radius f)
  using * by blast
 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 $R \equiv Min\ \{r, fps_conv_radius\ f, fps_conv_radius\ g\}$
assumes $subdegree\ g \leq subdegree\ f$
assumes $fps_conv_radius\ f > 0\ fps_conv_radius\ g > 0\ r > 0$
assumes $\bigwedge z.\ z \in eball\ 0\ r \Longrightarrow z \neq 0 \Longrightarrow eval_fps\ g\ z \neq 0$
shows *fps_conv_radius_divide*: $fps_conv_radius\ (f\ /\ g) \geq R$
and *eval_fps_divide*:
$ereal\ (norm\ z) < R \Longrightarrow c = fps_nth\ f\ (subdegree\ g)\ /\ fps_nth\ g\ (subdegree\ g) \Longrightarrow$
$eval_fps\ (f\ /\ g)\ z = (if\ z = 0\ then\ c\ else\ eval_fps\ f\ z\ /\ eval_fps\ g\ z)$
proof −
define $f'\ g'$ **where** $f' = fps_shift\ (subdegree\ g)\ f$ **and** $g' = fps_shift\ (subdegree\ g)\ g$
have *f_eq*: $f = f' * fps_X\ \hat{}\ subdegree\ g$ **and** *g_eq*: $g = g' * fps_X\ \hat{}\ subdegree\ g$
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*: $g \neq 0$
proof −
define $z :: complex$ **where** $z = (if\ r = \infty\ then\ 1\ else\ of_real\ (real_of_ereal\ r\ /\ 2))$
have $z \in eball\ 0\ r$
using ‹$r > 0$› *ereal_less_real_iff z_def* **by** *fastforce*
moreover have $z \neq 0$ **using** ‹$r > 0$›
by (*cases r*) (*auto simp*: *z_def*)
ultimately have $eval_fps\ g\ z \neq 0$ **by** (*rule assms*(*6*))
thus $g \neq 0$ **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*: $eval_fps\ g'\ z \neq 0$ **if** *z*: $norm\ z < min\ r\ (fps_conv_radius\ g)$ **for** z
proof (*cases* $z = 0$)
case *False*
with *assms* **and** z **have** $eval_fps\ g\ z \neq 0$ **by** *auto*
also from z **have** $eval_fps\ g\ z = eval_fps\ g'\ z * z\ \hat{}\ subdegree\ g$
by (*subst g_eq*) (*auto simp*: *eval_fps_mult*)
finally show *?thesis* **by** *auto*
qed (*use* ‹$g \neq 0$› **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*: *((λn. (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*
proof qed (*use r s* **in** *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 $f\ \xi = 0$
by (*auto simp*: *continuous_constant_on_closure* [*OF contf*])
moreover have $\bigwedge r.\ [\![0 < r; \bigwedge z.\ z \in ball\ \xi\ r - \{\xi\} \Longrightarrow f\ z \neq 0]\!] \Longrightarrow False$
by (*metis open_ball* ‹ξ *islimpt T*› *centre_in_ball fT0 insertE insert_Diff islimptE*)
ultimately show *?thesis*
by (*metis* ‹*open S*› ‹*connected S*› ‹$\xi \in S$› ‹$w \in S$› *holf isolated_zeros*)
qed

corollary *analytic_continuation_open*:
assumes *open s* **and** *open s'* **and** $s \neq \{\}$ **and** *connected s'*
and $s \subseteq s'$
assumes *f holomorphic_on s'* **and** *g holomorphic_on s'*
and $\bigwedge z.\ z \in s \Longrightarrow f\ z = g\ z$
assumes $z \in s'$
shows $f\ z = g\ z$
proof −
from ‹$s \neq \{\}$›
obtain ξ **where** $\xi \in s$ **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* $\lambda z.\ f\ z - g\ z$ *s' s* ξ])
(*insert assms* ‹$\xi \in s$›, *auto intro*: *holomorphic_intros*)
thus *?thesis* **by** *simp*
qed

corollary *analytic_continuation'*:
assumes *f holomorphic_on S open S connected S*
and $U \subseteq S$ $\xi \in S$ ξ *islimpt U*
and *f constant_on U*
shows *f constant_on S*
proof −
obtain *c* **where** *c*: $\bigwedge x.\ x \in U \Longrightarrow f\ x - c = 0$
by (*metis* ‹*f constant_on U*› *constant_on_def diff_self*)
have $(\lambda z.\ f\ z - c)$ *holomorphic_on S*
using *assms* **by** (*intro holomorphic_intros*)
with *c analytic_continuation assms* **have** $\bigwedge x.\ x \in S \Longrightarrow f\ x - c = 0$
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* $K \subseteq S$
and $\neg$ *f constant_on S*
shows *finite* $\{z \in K.\ f\ z = 0\}$
proof (*rule ccontr*)

assume *infinite* $\{z{\in}K.\ f\ z = 0\}$
then obtain z **where** $z \in K$ **and** z: z *islimpt* $\{z{\in}K.\ f\ z = 0\}$
using ‹*compact K*› **by** (*auto simp*: *compact_eq_Bolzano_Weierstrass*)
moreover have $\{z{\in}K.\ f\ z = 0\} \subseteq S$
using ‹$K \subseteq S$› **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* $\{z{\in}S.\ f\ z = 0\}$
proof −
obtain *F*::*nat* ⇒ *complex set*
where *F*: *range F* ⊆ *Collect compact* **and** *Seq*: $S = (\bigcup i.\ F\ i)$
using ‹*fsigma S*› **by** (*meson fsigma_Union_compact*)
have *fin*: *finite* $\{z \in F\ i.\ f\ z = 0\}$ **for** *i*
using *holomorphic_compact_finite_zeros assms F Seq Union_iff* **by** *blast*
have $\{z \in S.\ f\ z = 0\} = (\bigcup i.\ \{z \in F\ i.\ f\ z = 0\})$
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* $\{z{\in}S.\ f\ z = g\ z\}$
shows $S \subseteq \{z{\in}S.\ f\ z = g\ z\}$
proof −
obtain *z* **where** *z*: $z{\in}S\ f\ z = g\ z$
using *eq not_finite_existsD uncountable_infinite* **by** *blast*
have $(\lambda x.\ f\ x - g\ x)$ *holomorphic_on S*
by (*simp add*: *assms holomorphic_on_diff*)
then have $(\lambda x.\ f\ x - g\ x)$ *constant_on S*
using *holomorphic_countable_zeros assms* **by** *force*
with *z* **have** $\bigwedge x.\ x{\in}S \Longrightarrow f\ x - g\ x = 0$
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*: $\bigwedge z.\ norm(\xi - z) = r \Longrightarrow norm(f\ \xi) < norm(f\ z)$
obtains *z* **where** $z \in ball\ \xi\ r\ f\ z = 0$
proof −
{ assume *fnz*: $\bigwedge w.\ w \in ball\ \xi\ r \Longrightarrow f\ w \neq 0$
then have $0 < norm\ (f\ \xi)$
by (*simp add*: ‹$0 < r$›)
have *fnz'*: $\bigwedge w.\ w \in cball\ \xi\ r \Longrightarrow f\ w \neq 0$
using *dist_complex_def fnz norm_less order_le_less* **by** *fastforce*
have *frontier*(*cball* ξ *r*) $\neq \{\}$
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*) $\subseteq$ *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*) $\neq \{\}$
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*: $w \in frontier(cball\ \xi\ r)$
and *now*: $\bigwedge x.\ x \in frontier(cball\ \xi\ r) \Longrightarrow norm\ (f\ w) \leq norm\ (f\ x)$
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*: $v \in frontier(cball\ \xi\ r)$
and *nov*: $\bigwedge x.\ x \in frontier(cball\ \xi\ r) \Longrightarrow norm\ (g\ v) \geq norm\ (g\ x)$
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) $\leq$ *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) $\leq$ *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 (smt (verit, del_insts) ‹0 < cmod (f ξ)› inverse_le_imp_le norm_inverse
now 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 f holomorphic_on S
    and open S and  inj_on f S
```

shows *open* (*f* ‘ *S*)
by (*meson assms inj_on_subset injective_not_constant open_mapping_thm2 order.refl*)

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*: $\bigwedge z.\ z \in \mathit{frontier}\ S \Longrightarrow \mathit{Re}(f\ z) \leq B$
and $\xi \in S$
shows $\mathit{Re}(f\ \xi) \leq B$
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 $\xi \in S\ 0 < n$
and *dnz*: $(\mathit{deriv}\ \hat{}\hat{}\ n)\ f\ \xi \neq 0$
and *dfz*: $\bigwedge i.\ [\![0 < i; i < n]\!] \Longrightarrow (\mathit{deriv}\ \hat{}\hat{}\ i)\ f\ \xi = 0$
obtains *g r* **where** $0 < r$
g holomorphic_on ball ξ r
$\bigwedge w.\ w \in \mathit{ball}\ \xi\ r \Longrightarrow f\ w - f\ \xi = (w - \xi)\hat{}n * g\ w$
$\bigwedge w.\ w \in \mathit{ball}\ \xi\ r \Longrightarrow g\ w \neq 0$
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** $g\ w = \mathit{suminf}\ (\lambda i.\ (\mathit{deriv}\ \hat{}\hat{}\ (i + n))\ f\ \xi\ /\ (\mathit{fact}(i + n)) * (w - \xi)\hat{}i)$ **for** *w*
have *sumsg*: $(\lambda i.\ (\mathit{deriv}\ \hat{}\hat{}\ (i + n))\ f\ \xi\ /\ (\mathit{fact}(i + n)) * (w - \xi)\hat{}i)\ \mathit{sums}\ g\ w$
and *feq*: $f\ w - f\ \xi = (w - \xi)\hat{}n * g\ w$
if *w*: *w* ∈ *ball ξ r* **for** *w*
proof −
define *powf* **where** $\mathit{powf} = (\lambda i.\ (\mathit{deriv}\ \hat{}\hat{}\ i)\ f\ \xi/(\mathit{fact}\ i) * (w - \xi)\hat{}i)$
have [*simp*]: *powf 0* = *f ξ*
by (*simp add*: *powf_def*)
have *sing*: $\{..<n\} - \{i.\ \mathit{powf}\ i = 0\} = (\mathit{if}\ f\ \xi = 0\ \mathit{then}\ \{\}\ \mathit{else}\ \{0\})$
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 $(\sum i<n.\ \mathit{powf}\ i) = f\ \xi$
by (*subst sum.setdiff_irrelevant* [*symmetric*]; *simp add*: *dfz sing*)
ultimately have *fsums*: $(\lambda i.\ \mathit{powf}\ (i+n))\ \mathit{sums}\ (f\ w - f\ \xi)$
using *w sums_iff_shift'* **by** *metis*
then have ∗: $\mathit{summable}\ (\lambda i.\ (w - \xi)\ \hat{}\ n * ((\mathit{deriv}\ \hat{}\hat{}\ (i + n))\ f\ \xi * (w - \xi)\ \hat{}\ i\ /\ \mathit{fact}\ (i + n)))$
unfolding *powf_def* **using** *sums_summable*
by (*auto simp*: *power_add mult_ac*)
have $\mathit{summable}\ (\lambda i.\ (\mathit{deriv}\ \hat{}\hat{}\ (i + n))\ f\ \xi * (w - \xi)\ \hat{}\ i\ /\ \mathit{fact}\ (i + n))$
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 holomor-
```

phic_on_imp_continuous_on)
have *cd*: (λ*z*. *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* {} λ*z*. *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*) (λ*x*. *exp* (*h x*) / *g x*)
by (*auto intro*!: *continuous_intros simp add*: *holg holomorphic_on_imp_continuous_on gne*)
have *gfd*: *dist ξ x* < *r* ⟹ *g field_differentiable at x* **if** *dist ξ x* < *r* **for** *x*
using *holg holomorphic_on_imp_differentiable_at* **by** *auto*
have *0*: *dist ξ x* < *r* ⟹ ((λ*x*. *exp* (*h x*) / *g x*) *has_field_derivative 0*) (*at x*) **for** *x*
by (*rule gfd h derivative_eq_intros DERIV_deriv_iff_field_differentiable* [*THEN iffD2*] | *simp add*: *gne*)+
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* {} λ*x*. *exp*(*h x*) / *g x*]) (*auto simp*: *con 0*)
have *hol*: (λ*z*. *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 $\xi \in S$ $f\ \xi = 0$
and *nonconst*: $\neg$ *f constant_on S*
obtains *g r n*
where $0 < n$ $0 < r$ *ball* ξ *r* $\subseteq S$
g holomorphic_on ball ξ *r*
$\bigwedge w.\ w \in$ *ball* ξ *r* $\Longrightarrow f\ w = (w - \xi)\hat{}n * g\ w$
$\bigwedge w.\ w \in$ *ball* ξ *r* $\Longrightarrow g\ w \neq 0$
proof (*cases* $\forall n{>}0.$ (*deriv* $\hat{}\hat{}$ *n*) *f* $\xi = 0$)
case *True* **then show** *?thesis*
using *holomorphic_fun_eq_const_on_connected* [*OF holf S _* ‹$\xi \in S$›] *nonconst* **by** (*simp add*: *constant_on_def*)
next
case *False*
then obtain *n0* **where** $n0 > 0$ **and** *n0*: (*deriv* $\hat{}\hat{}$ *n0*) *f* $\xi \neq 0$ **by** *blast*
obtain *r0* **where** $r0 > 0$ *ball* ξ *r0* $\subseteq S$ **using** *S openE* ‹$\xi \in S$› **by** *auto*
define *n* **where** $n \equiv$ *LEAST n.* (*deriv* $\hat{}\hat{}$ *n*) *f* $\xi \neq 0$
have *n_ne*: (*deriv* $\hat{}\hat{}$ *n*) *f* $\xi \neq 0$
by (*rule def_LeastI* [*OF n_def*]) (*rule n0*)
then have $0 < n$ **using** ‹*f* $\xi = 0$›
using *funpow_0* **by** *fastforce*
have *n_min*: $\bigwedge k.\ k < n \Longrightarrow$ (*deriv* $\hat{}\hat{}$ *k*) *f* $\xi = 0$
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*: $\bigwedge w.\ w \in$ *ball* ξ *r1* $\Longrightarrow f\ w = (w - \xi)\ \hat{}\ n * g\ w$
and *g0*: $\bigwedge w.\ w \in$ *ball* ξ *r1* $\Longrightarrow g\ w \neq 0$
by (*auto intro*: *holomorphic_factor_order_of_zero* [*OF holf* ‹*open S*› ‹$\xi \in S$› ‹$n > 0$› *n_ne*] *simp*: ‹*f* $\xi = 0$›)
show *?thesis*
proof
show *g holomorphic_on ball* ξ (*min r0 r1*)
using *g* **by** *auto*
show $\bigwedge w.\ w \in$ *ball* ξ (*min r0 r1*) $\Longrightarrow f\ w = (w - \xi)\ \hat{}\ n * g\ w$
by (*simp add*: *geq*)
qed (*use* ‹$0 < n$› ‹$0 < r0$› ‹$0 < r1$› ‹*ball* ξ *r0* $\subseteq S$› *g0* **in** *auto*)
qed

lemma *holomorphic_lower_bound_difference*:
assumes *holf*: *f holomorphic_on S* **and** *S*: *open S connected S*
and $\xi \in S$ **and** $\varphi \in S$
and *fne*: $f\ \varphi \neq f\ \xi$
obtains *k n r*
where $0 < k$ $0 < r$

ball ξ $r \subseteq S$
$\bigwedge w.\ w \in ball\ \xi\ r \Longrightarrow k * norm(w - \xi) \hat{}\ n \leq norm(f\ w - f\ \xi)$
proof −
define n **where** $n = (LEAST\ n.\ 0 < n \wedge (deriv\ \hat{}\hat{}\ n)\ f\ \xi \neq 0)$
obtain $n0$ **where** $0 < n0$ **and** $n0$: $(deriv\ \hat{}\hat{}\ n0)\ f\ \xi \neq 0$
using *fne holomorphic_fun_eq_const_on_connected* [*OF holf S*] ‹$\xi \in S$› ‹$\varphi \in S$› **by** *blast*
then have $0 < n$ **and** n_ne: $(deriv\ \hat{}\hat{}\ n)\ f\ \xi \neq 0$
unfolding n_def **by** (*metis* (*mono_tags*, *lifting*) *LeastI*)+
have n_min: $\bigwedge k.\ [\![0 < k; k < n]\!] \Longrightarrow (deriv\ \hat{}\hat{}\ k)\ f\ \xi = 0$
unfolding n_def **by** (*blast dest*: *not_less_Least*)
then obtain $g\ r$
where $0 < r$ **and** *holg*: g *holomorphic_on ball* $\xi\ r$
and *fne*: $\bigwedge w.\ w \in ball\ \xi\ r \Longrightarrow f\ w - f\ \xi = (w - \xi)\ \hat{}\ n * g\ w$
and *gnz*: $\bigwedge w.\ w \in ball\ \xi\ r \Longrightarrow g\ w \neq 0$
by (*auto intro*: *holomorphic_factor_order_of_zero* [*OF holf* ‹*open S*› ‹$\xi \in S$› ‹$n > 0$› n_ne])
obtain e **where** $e{>}0$ **and** e: *ball* $\xi\ e \subseteq S$ **using** *assms* **by** (*blast elim*!: *openE*)
then have *holfb*: f *holomorphic_on ball* $\xi\ e$
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* $\xi\ d \subseteq ball\ \xi\ r$
by (*auto simp*: *cball_subset_ball_iff*)
then have g *holomorphic_on cball* $\xi\ d$
by (*rule holomorphic_on_subset* [*OF holg*])
then have *closed* $(g\ `\ cball\ \xi\ d)$
by (*simp add*: *compact_imp_closed compact_continuous_image holomorphic_on_imp_continuous_on*)
moreover have $g\ `\ cball\ \xi\ d \neq \{\}$
using ‹$0 < d$› **by** *auto*
ultimately obtain x **where** x: $x \in g\ `\ cball\ \xi\ d$ **and** $\bigwedge y.\ y \in g\ `\ cball\ \xi\ d \Longrightarrow dist\ 0\ x \leq dist\ 0\ y$
by (*rule distance_attains_inf*) *blast*
then have *leg*: $\bigwedge w.\ w \in cball\ \xi\ d \Longrightarrow norm\ x \leq norm\ (g\ w)$
by *auto*
have *ball* $\xi\ d \subseteq cball\ \xi\ d$ **by** *auto*
also have $\ldots \subseteq ball\ \xi\ e$ **using** ‹$0 < d$› d_def **by** *auto*
also have $\ldots \subseteq S$ **by** (*rule e*)
finally have *dS*: *ball* $\xi\ d \subseteq S$ **.**
have $x \neq 0$ **using** *gnz x* ‹$d < r$› **by** *auto*
show *thesis*
proof
show $\bigwedge w.\ w \in ball\ \xi\ d \Longrightarrow cmod\ x * cmod\ (w - \xi)\ \hat{}\ n \leq cmod\ (f\ w - f\ \xi)$
using ‹$d < r$› *leg* **by** (*auto simp*: *fne norm_mult norm_power algebra_simps mult_right_mono*)
qed (*use dS* ‹$x \neq 0$› ‹$d > 0$› **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 have ∀F z in at ξ. cmod (f z) ≤ cmod (g ξ) + 1
      by (rule eventually_mp [OF ∗ tendstoD [where e=1]], auto)
    then show ?thesis
      by blast
  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 (λy. (y - ξ)^2 * f y / (y - ξ)) -ξ→ 0
      by (simp add: LIM_cong power2_eq_square that)
    then have h0: (h has_field_derivative 0) (at ξ)
```

```
    by (simp add: h_def has_field_derivative_iff)
  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 §: ∀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)
  have g holomorphic_on S
    unfolding g_def by (rule pole_lemma [OF holh ξ])
 then have (λz. if z = ξ then deriv g ξ else (g z − g ξ) / (z − ξ)) holomorphic_on
S
    using ξ pole_lemma by blast
  then show ?thesis
    using § by (smt (verit, best) DiffD2 singletonI)
  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
      obtain g where holg: g holomorphic_on ball 0 r
                and geq: ⋀z. z ∈ ball 0 r − {0} ⟹ g z = (inverse ∘ f ∘ inverse) z
        using tendstoD [OF lim [unfolded lim_at_infinity_0] zero_less_one] holo-
morphic_on_extend_bounded [OF 1 2]
        by (smt (verit, del_insts) ‹l = 0› eventually_mono norm_conv_dist)
      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
      proof (rule Lim_transform_within_open [OF ifi0 open_ball])
        show ⋀x. ⟦x ∈ ball 0 r; x ≠ 0⟧ ⟹ (inverse ∘ f ∘ inverse) x = g x
          by (auto simp: geq)
      qed (auto simp: ‹0 < r›)
      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: 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 (clarsimp simp: compact_eq_bounded_closed norm_divide divide_simps
mult.commute elim!: bounded_normE_less)
      by (meson linorder_not_le nle_le)
  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** $0 < n\ c\ n \neq 0\ f = (\lambda z.\ \sum i{\leq}n.\ c\ i * z \mathbin{\hat{}} i)$ **by** *blast*
then have *compact* $\{z.\ f\ z \in k\}$ **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*: $\xi \in S$ *open S*
and *dnz*: *deriv f* $\xi \neq 0$
obtains r **where** $r > 0$ *ball* $\xi\ r \subseteq S$ *inj_on f* (*ball* $\xi\ r$)
proof −
have $*$: $\exists d{>}0.\ \forall x.\ dist\ \xi\ x < d \longrightarrow onorm\ (\lambda v.\ deriv\ f\ x * v - deriv\ f\ \xi * v) < e$ **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** $\delta{>}0$ **and** δ: $\bigwedge x.\ [\![x \in S;\ dist\ x\ \xi \leq \delta]\!] \Longrightarrow cmod\ (deriv\ f\ x - deriv\ f\ \xi) \leq e/2$
using *continuous_onE* [*OF contdf* ‹$\xi \in S$›, *of* $e/2$] ‹$0 < e$›
by (*metis dist_complex_def half_gt_zero less_imp_le*)
have §: $\bigwedge \zeta\ z.\ [\![\zeta \in S;\ dist\ \xi\ \zeta < \delta]\!] \Longrightarrow cmod\ (deriv\ f\ \zeta - deriv\ f\ \xi) * cmod\ z \leq e/2 * cmod\ z$
by (*intro mult_right_mono* [*OF* δ]) (*auto simp*: *dist_commute*)
obtain ε **where** $\varepsilon{>}0$ *ball* $\xi\ \varepsilon \subseteq S$
by (*metis openE* [*OF* ‹*open S*› ‹$\xi \in S$›])
with ‹$\delta{>}0$› **have** $\exists \delta{>}0.\ \forall x.\ dist\ \xi\ x < \delta \longrightarrow onorm\ (\lambda v.\ deriv\ f\ x * v - deriv\ f\ \xi * v) \leq e/2$
using §
apply (*rule_tac* $x{=}min\ \delta\ \varepsilon$ **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' $g' \circ$ (∗) (*deriv f* ξ) $= id$
using *linear_injective_left_inverse* [*of* (∗) (*deriv f* ξ)] **by** *auto*

have *fder*: $\bigwedge x.\ x \in S \Longrightarrow (f\ has_derivative\ (*)\ (deriv\ f\ x))\ (at\ x)$
using ‹*open S*› *has_field_derivative_imp_has_derivative holf holomorphic_derivI* **by** *blast*
show *?thesis*
apply (*rule has_derivative_locally_injective* [*OF S*, **where** $f{=}f$ **and** $f' = \lambda z$

```
h. deriv f z * h and g' = g'])
    using g' * by (simp_all add: fder linear_conv_bounded_linear that)
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 (λw. (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 ((λw. (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 = (λw. (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]
      by (auto simp: y0 y1 power_mult_distrib diff_eq_eq n_ne)
    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} ⇒ 'a
 shows ⟦DERIV f x :> f'; f' ≠ 0; open S; x ∈ S; continuous_on S f;
        ⋀z. z ∈ S ⟹ g (f z) = z⟧
        ⟹ 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} ⇒ 'a
 shows ⟦DERIV f (g y) :> f'; f' ≠ 0; open S; continuous_on S f; g y ∈ S; f(g
y) = y;
         ⋀z. z ∈ S ⟹ g (f z) = z⟧
        ⟹ 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)
                 ⋀z. z ∈ S ⟹ deriv f z * deriv g (f z) = 1
                 ⋀z. z ∈ S ⟹ 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 ∈ 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 holomor-
phic_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 ⋀z. z ∈ S ⟹ 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 $\bigwedge W.\ W \in \mathcal{V} \Longrightarrow$ *openin* (*top_of_set C*) *W*
and *disV*: *pairwise disjnt* $\mathcal{V}$ **and** *homeV*: $\bigwedge W.\ W \in \mathcal{V} \Longrightarrow \exists q.$ *homeomorphism W T p q*
using *cov fim* **unfolding** *covering_space_def* **by** *meson*
then have $\bigwedge W.\ W \in \mathcal{V} \Longrightarrow$ *open* $W \wedge W \subseteq C$
by (*metis* ‹*open C*› *inf_le1 open_Int openin_open*)
then obtain *V* **where** $V \in \mathcal{V}$ *g z* $\in$ *V open V V* $\subseteq$ *C*
by (*metis IntI UnionE image_subset_iff vimageI UV* ‹*f z* $\in$ *T*› ‹*z* $\in$ *U*› *gim pg_f image_subset_iff_funcset*)
have *holp*: *p holomorphic_on V*
using ‹*V* $\subseteq$ *C*› ‹*p holomorphic_on C*› *holomorphic_on_subset* **by** *blast*
moreover have *injp*: *inj_on p V*
by (*metis* ‹*V* $\in$ $\mathcal{V}$› *homeV homeomorphism_def inj_on_inverseI*)
ultimately
obtain *p′* **where** *holp′*: *p′ holomorphic_on* (*p* ‘ *V*) **and** *pp′*: $\bigwedge z.\ z \in V \Longrightarrow p'(p\ z) = z$
using ‹*open V*› *holomorphic_has_inverse* **by** *metis*
have *z* $\in$ *U* $\cap$ *g* −‘ *V*
using ‹*g z* $\in$ *V*› ‹*z* $\in$ *U*› **by** *blast*
moreover have *openin* (*top_of_set U*) (*U* $\cap$ *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 $\cap$ *U* $\subseteq$ *g* −‘ *V*
by (*force simp*: *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* $\in$ *U*
by (*simp add*: ‹*z* $\in$ *U*›)
show (*p′ o f*) *x′* = *g x′* **if** *x′* $\in$ *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* $\in$ *p* ‘ *V*
by (*metis* ‹*z* $\in$ *U*› *image_iff pg_f* ‹*g z* $\in$ *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* $\in$ *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 subset_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)
      using conth' ‹0 < r› by (auto simp add: intro: Schwarz1 [OF holh'])
    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** $d \neq 0$
    and *holf1*: *f holomorphic_on* $(S \cap \{z.\ d \cdot z < k\})$
    and *holf2*: *f holomorphic_on* $(S \cap \{z.\ k < d \cdot z\})$
    and *contf*: *continuous_on S f*
   shows *f holomorphic_on S*
proof −
  have ∗: $\exists t.$ *open t* $\wedge$ $p \in t$ $\wedge$ *continuous_on t f* $\wedge$
          $(\forall a\ b\ c.$ *convex hull* $\{a, b, c\} \subseteq t \longrightarrow$
                contour_integral (*linepath a b*) *f* +
                contour_integral (*linepath b c*) *f* +
                contour_integral (*linepath c a*) *f* = *0*)
       if $p \in S$ **for** *p*
  proof −
    obtain *e* **where** *e*>*0* **and** *e*: *ball p e* $\subseteq S$
      using ‹$p \in S$› *openE S* **by** *blast*
    then have *continuous_on* (*ball p e*) *f*
      using *contf continuous_on_subset* **by** *blast*
    moreover
    have $\{z.$ *dist p z* < *e* $\wedge$ $d \cdot z < k\} \subseteq S \cap \{z.\ d \cdot z < k\}$
        $\{z.$ *dist p z* < *e* $\wedge$ $k < d \cdot z\} \subseteq S \cap \{z.\ k < d \cdot z\}$
      using *e* **by** *auto*
    then have *f holomorphic_on* $\{z.$ *dist p z* < *e* $\wedge$ $d \cdot z < k\}$
          f holomorphic_on $\{z.$ *dist p z* < *e* $\wedge$ $k < d \cdot z\}$
      using *holomorphic_on_subset holf1 holf2* **by** *presburger*+
    ultimately show *?thesis*
      apply (*rule_tac x*=*ball p e* **in** *exI*)
      using ‹*e* > *0*› *e* ‹$d \neq 0$› *hol_pal_lem4* [*of ball p e _ _ _ d _ k*]
      by (*force simp*: *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* $\subseteq S$
    and *holf*: *f holomorphic_on* $(S \cap \{z.\ 0 < Im\ z\})$
    and *contf*: *continuous_on* $(S \cap \{z.\ 0 \leq Im\ z\})$ *f*
    and *f*: $\bigwedge z.\ [\![z \in S;\ z \in \mathbb{R}]\!] \Longrightarrow (f\ z) \in \mathbb{R}$
   shows $(\lambda z.$ *if* $0 \leq Im\ z$ *then f z else cnj*$(f(cnj\ z)))$ *holomorphic_on S*
proof −
  have *1*: $(\lambda z.$ *if* $0 \leq Im\ z$ *then f z else cnj* $(f\ (cnj\ z)))$ *holomorphic_on* $(S \cap \{z.\ 0 < Im\ z\})$
    by (*force intro*: *iffD1* [*OF holomorphic_cong* [*OF refl*] *holf*])
  have *cont_cfc*: *continuous_on* $(S \cap \{z.\ Im\ z \leq 0\})$ (*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})
    by (smt (verit) Int_Collect comp_def hol_cfc holomorphic_cong)
  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
    using 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)
      then have cmod (x *R z) < r
        by (simp add: x)
       then have norm (deriv f (x *R z) − deriv f 0) ≤ norm (x *R z) / (r −
norm (x *R z)) * C
        by (metis Le1)
      also have ... ≤ norm (x *R z) / (r − norm z) * C
        using r x ‹0 < r› ‹0 < C› by (simp add: frac_le mult_left_le_one_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
       using ‹norm z < r›
       by (force simp add: power2_eq_square divide_simps C_def norm_mult §
intro!: le_norm [OF _ int_le])
   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
      proof (rule order_trans [OF _ ∗])
        show (3 − 2 ∗ sqrt 2) ∗ r ∗ cmod (deriv f 0)
           ≤ (cmod z − (cmod z)^2 / (r − cmod z)) ∗ cmod (deriv f 0)
          by (simp add: le_less algebra_simps divide_simps power2_eq_square that)
      qed (use ‹0 < r› that in auto)
      show ?thesis
        using ‹0 < r› sq201 3 C_def ‹0 < C› sq3
        by (intro ball_subset_open_map_image [OF 1 2 _ bounded_ball]) 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)
  moreover have ⋀z. cmod z < r ⟹
        cmod (deriv (λz. f (a + z)) z) ≤ 2 * cmod (deriv (λz. f (a + z)) 0)
    by (simp add: fz deriv_chain dist_norm le)
  ultimately 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
    using ‹0 < r› by (intro Bloch_lemma_0) auto
  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: 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
   by (meson ‹r < 1› ball_subset_cball cpt dual_order.trans image_mono less_le_not_le
subset_ball)
  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
```

```
  obtain f' where (f has_field_derivative f') (at a)
    using dfa field_differentiable_def by blast
  then have ∃c. ((λc. f (a + complex_of_real r * c)) has_field_derivative c) (at
0)
   by (metis (no_types) DERIV_chain2 add_cancel_left_right field_differentiable_add_const

      field_differentiable_def field_differentiable_linear mult_eq_0_iff)
  then have (λw. f (a + complex_of_real r * w)) field_differentiable at 0
    by (simp add: field_differentiable_def)
  then have deriv (λz. f (a + of_real r * z) / (C * of_real r)) 0
        = deriv (λz. f (a + of_real r * z)) 0 / (C * of_real r)
    by (rule deriv_cdivide_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
```

6 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
```

6.1 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 (λz. 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_divide_pos
[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
```

6.2 The Little Picard Theorem

theorem *Landau_Picard*:
obtains R
where $\bigwedge z.\ 0 < R\ z$
$\bigwedge f.\ [\![f\ holomorphic_on\ cball\ 0\ (R(f\ 0));$
$\bigwedge z.\ norm\ z \leq R(f\ 0) \Longrightarrow f\ z \neq 0 \wedge f\ z \neq 1]\!] \Longrightarrow norm(deriv\ f\ 0) < 1$
proof −
define R **where** $R \equiv \lambda z.\ 3 * exp(pi * exp(pi*(2 + 2 * cmod\ z + 12)))$
show *?thesis*
proof
show *Rpos*: $\bigwedge z.\ 0 < R\ z$
by (*auto simp*: *R_def*)
show $norm(deriv\ f\ 0) < 1$
if *holf*: *f holomorphic_on cball 0* $(R(f\ 0))$
and *Rf*: $\bigwedge z.\ norm\ z \leq R(f\ 0) \Longrightarrow f\ z \neq 0 \wedge f\ z \neq 1$ **for** *f*
proof −
let *?r* = $R(f\ 0)$
define *g* **where** $g \equiv f \circ (\lambda z.\ of_real\ ?r * z)$
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 (∗) $(complex_of_real\ (R\ (f\ 0)))$ ‘ $cball\ 0\ 1 \subseteq cball\ 0\ (R\ (f\ 0))$
using *Rpos* **by** (*auto simp*: *less_imp_le norm_mult*)
qed
have ∗: $norm(g\ z) \leq exp(pi * exp(pi * (2 + 2 * norm\ (f\ 0) + 12 * t\ /\ (1 - t))))$
if $0 < t\ t < 1\ norm\ z \leq t$ **for** *t z*
proof (*rule Schottky* [*OF holg*])
show $cmod\ (g\ 0) \leq cmod\ (f\ 0)$
by (*simp add*: *g_def*)
show $\bigwedge z.\ z \in cball\ 0\ 1 \Longrightarrow \neg\ (g\ z = 0 \vee g\ z = 1)$
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*: $cmod\ (g\ z) \leq R\ (f\ 0)\ /\ 3$ **if** $cmod\ (0 - z) = 1/2$ **for** *z*
proof −
have $norm(g\ z) \leq exp(pi * exp(pi * (2 + 2 * norm\ (f\ 0) + 12)))$
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: ⋀z. f z ≠ 0 ∧ f z ≠ 1
  obtains c where f = (λx. c)
proof −
  obtain R
    where Rpos: ⋀z. 0 < R z
      and R:    ⋀h. [[h holomorphic_on cball 0 (R(h 0));
                      ⋀z. norm z ≤ R(h 0) ⟹ h z ≠ 0 ∧ h z ≠ 1]] ⟹ 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 ∀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 ⋀x. f(x) = c
      by (metis UNIV_I convex_UNIV has_field_derivative_zero_constant)
    then show ?thesis
      using that by auto
  next
    case False
    then obtain w where w: deriv f w ≠ 0 by auto
    define fw where fw ≡ (f ∘ (λz. 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 ≠ 0 ∧ fw z ≠ 1 if cmod z ≤ 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

6.3 The Arzelà–Ascoli theorem

lemma *subsequence_diagonalization_lemma*:
fixes $P :: nat \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$
assumes *sub*: $\bigwedge i\ r.\ \exists k.\ strict_mono\ (k :: nat \Rightarrow nat) \wedge P\ i\ (r \circ k)$
and *P_P*: $\bigwedge i\ r{::}nat \Rightarrow 'a.\ \bigwedge k1\ k2\ N.$
$[\![P\ i\ (r \circ k1);\ \bigwedge j.\ N \le j \Longrightarrow \exists j'.\ j \le j' \wedge k2\ j = k1\ j']\!] \Longrightarrow P\ i\ (r \circ k2)$
obtains *k* **where** $strict_mono\ (k :: nat \Rightarrow nat)\ \bigwedge i.\ P\ i\ (r \circ k)$
proof −
obtain *kk* **where** $\bigwedge i\ r.\ strict_mono\ (kk\ i\ r :: nat \Rightarrow nat) \wedge P\ i\ (r \circ (kk\ i\ r))$
using *sub* **by** *metis*
then have *sub_kk*: $\bigwedge i\ r.\ strict_mono\ (kk\ i\ r)$ **and** *P_kk*: $\bigwedge i\ r.\ P\ i\ (r \circ (kk\ i\ r))$
by *auto*
define *rr* **where** $rr \equiv rec_nat\ (kk\ 0\ r)\ (\lambda n\ x.\ x \circ kk\ (Suc\ n)\ (r \circ x))$
then have [*simp*]: $rr\ 0 = kk\ 0\ r\ \bigwedge n.\ rr(Suc\ n) = rr\ n \circ kk\ (Suc\ n)\ (r \circ rr\ n)$
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*: $P\ i\ (r \circ rr\ i)$ **for** *i*
using *P_kk* **by** (*induction i*) (*auto simp*: *o_def*)
have $i \le i+d \Longrightarrow rr\ i\ n \le rr\ (i+d)\ n$ **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 $\bigwedge i\ j\ n.\ i \le j \Longrightarrow rr\ i\ n \le rr\ j\ n$
by (*metis le_iff_add*)
show $strict_mono\ (\lambda n.\ rr\ n\ n)$
unfolding *strict_mono_Suc_iff*
by (*simp add*: *Suc_le_lessD strict_monoD strict_mono_imp_increasing sub_kk sub_rr*)
have $\exists j.\ i \le j \wedge rr\ (n+d)\ i = rr\ n\ j$ **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⟧
```

$\Longrightarrow \exists\, d.\ 0 < d \wedge (\forall\, n\ y.\ y \in S \wedge norm(x - y) < d \longrightarrow norm(\mathcal{F}\ n\ x - \mathcal{F}\ n\ y) < e)$
obtains *g k* **where** *continuous_on S g strict_mono* (*k* :: *nat* ⇒ *nat*)
$\bigwedge e.\ 0 < e \Longrightarrow \exists\, N.\ \forall\, n\ x.\ n \geq N \wedge x \in S \longrightarrow norm(\mathcal{F}(k\ n)\ x - g\ x) < e$
proof −
have *UEQ*: $\bigwedge e.\ 0 < e \Longrightarrow \exists\, d.\ 0 < d \wedge (\forall\, n.\ \forall\, x \in S.\ \forall\, x' \in S.\ dist\ x'\ x < d \longrightarrow dist\ (\mathcal{F}\ n\ x')\ (\mathcal{F}\ n\ x) < e)$
apply (*rule compact_uniformly_equicontinuous* [*OF* ‹*compact S*›, *of range* $\mathcal{F}$])
using *equicont* **by** (*force simp*: *dist_commute dist_norm*)+
have *continuous_on S g*
if $\bigwedge e.\ 0 < e \Longrightarrow \exists\, N.\ \forall\, n\ x.\ n \geq N \wedge x \in S \longrightarrow norm(\mathcal{F}(r\ n)\ x - g\ x) < e$
for *g*:: *'a* ⇒ *'b* **and** *r* :: *nat* ⇒ *nat*
proof (*rule uniform_limit_theorem* [*of* _ $\mathcal{F} \circ r$])
have *continuous_on S* ($\mathcal{F}$ (*r n*)) **for** *n*
using *UEQ* **by** (*force simp*: *continuous_on_iff*)
then show $\forall_F$ *n in sequentially*. *continuous_on S* (($\mathcal{F} \circ r$) *n*)
by (*simp add*: *eventually_sequentially*)
show *uniform_limit S* ($\mathcal{F} \circ r$) *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*: $\bigwedge x.\ x \in R \Longrightarrow \exists\, l.\ (\lambda n.\ \mathcal{F}\ (k\ n)\ x) \longrightarrow l$
using ‹*R* ⊆ *S*› **by** (*force intro*: *function_convergent_subsequence* [*OF* ‹*countable R*› *M*])
then have *Cauchy*: *Cauchy* ((λ*n*. $\mathcal{F}$ (*k n*) *x*)) **if** *x* ∈ *R* **for** *x*
using *convergent_eq_Cauchy that* **by** *blast*
have $\exists\, N.\ \forall\, m\ n\ x.\ N \leq m \wedge N \leq n \wedge x \in S \longrightarrow dist\ ((\mathcal{F} \circ k)\ m\ x)\ ((\mathcal{F} \circ k)\ n\ x) < e$
if *0* < *e* **for** *e*
proof −
obtain *d* **where** *0* < *d*
and *d*: $\bigwedge n.\ \forall\, x \in S.\ \forall\, x' \in S.\ dist\ x'\ x < d \longrightarrow dist\ (\mathcal{F}\ n\ x')\ (\mathcal{F}\ n\ x) < e/3$
by (*metis UEQ* ‹*0* < *e*› *divide_pos_pos zero_less_numeral*)
obtain *T* **where** *T* ⊆ *R* **and** *finite T* **and** *T*: *S* ⊆ ($\bigcup$ *c*∈*T*. *ball c d*)
proof (*rule compactE_image* [*OF* ‹*compact S*›, *of R* (λ*x*. *ball x d*)])
have *closure R* ⊆ ($\bigcup$ *c*∈*R*. *ball c d*)
using ‹*0* < *d*› **by** (*auto simp*: *closure_approachable*)
with *SR* **show** *S* ⊆ ($\bigcup$ *c*∈*R*. *ball c d*)
by *auto*
qed *auto*
have $\exists\, M.\ \forall\, m \geq M.\ \forall\, n \geq M.\ dist\ (\mathcal{F}\ (k\ m)\ x)\ (\mathcal{F}\ (k\ n)\ x) < e/3$ **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*: $\bigwedge x\ m\ n.\ [\![x \in R;\ m \geq MF\ x;\ n \geq MF\ x]\!] \Longrightarrow$

```
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
```

6.3.1 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 (\mathcal{F} n z - \mathcal{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. \mathcal{F} n w / (w - ξ)) has_contour_integral
            (2 * pi) * i * winding_number (circlepath z (2/3 * r)) ξ * \mathcal{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 \mathcal{F} n holomorphic_on S
          by (simp add: \mathcal{H} ‹⋀n. \mathcal{F} n ∈ \mathcal{H}›)
        with z23S show \mathcal{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. \mathcal{F} n w / (w - ξ)) has_contour_integral (2 * pi) * i
* \mathcal{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. \mathcal{F} n w / (w - y)) has_contour_integral (2 * pi) * i * \mathcal{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. \mathcal{F} n w / (w - z)) has_contour_integral (2 * pi) * i * \mathcal{F} n
z)
          (circlepath z (2/3 * r))
        using ‹0 < r› by (force intro!: *)
      have le_er: cmod (\mathcal{F} n x / (x - y) - \mathcal{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 (\mathcal{F} n x) ≤ M
          by (simp add: M dist_commute dist_norm that)
        have cmod (\mathcal{F} n x / (x - y) - \mathcal{F} n x / (x - z)) = cmod (\mathcal{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 (ℱ n y − ℱ n z) = cmod ((2 * pi) * i * ℱ n y −
(2 * pi) * i * ℱ n z)
          by (simp add: right_diff_distrib [symmetric] norm_mult)
        also have cmod ((2 * pi) * i * ℱ n y − (2 * pi) * i * ℱ n z) ≤ e / r *
(2 * pi * (2/3 * r))

      proof (rule has_contour_integral_bound_circlepath [OF has_contour_integral_diff
[OF y z]])
        show ⋀x. cmod (x − z) = 2/3 * r ⟹ cmod (ℱ n x / (x − y) − ℱ n
x / (x − z)) ≤ 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 (ℱ n y − ℱ n z) ≤ 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 ⋀n x.  x ∈ K i ⟹ cmod (ℱ n x) ≤ B i
      using B ‹⋀n. ℱ n ∈ ℋ› by blast
  next
    fix g :: complex ⇒ complex and k :: nat ⇒ nat
    assume *: ⋀(g::complex⇒complex) (k::nat⇒nat). continuous_on (K i) g ⟹
                strict_mono k ⟹
                (⋀e. 0 < e ⟹ ∃N. ∀n≥N. ∀x∈K i. cmod (ℱ (k n) x − g x) <
e) ⟹ thesis
        continuous_on (K i) g
```

```
        strict_mono k
        ⋀e. 0 < e ⟹ ∃N. ∀n x. N ≤ n ∧ x ∈ K i ⟶ cmod (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 Φ where Φ ≡ λg i r. λk::nat⇒nat. ∀e>0. ∃N. ∀n≥N. ∀x∈K i. cmod
((ℱ ∘ (r ∘ k)) n x − g x) < e
  obtain k :: nat ⇒ nat where strict_mono k and k: ⋀i. ∃g. Φ g i id k
  proof (rule subsequence_diagonalization_lemma [where r=id])
    show ∃g. Φ g i id (r ∘ k2)
      if ex: ∃g. Φ g i id (r ∘ k1) and ⋀j. N ≤ j ⟹ ∃j'≥j. k2 j = k1 j'
      for i k1 k2 N and r::nat⇒nat
    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
```

6.4 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*: $\bigwedge n\ z.\ z \in S \Longrightarrow \mathcal{F}\ n\ z \neq 0$
and *z0* $\in S$
shows *g z0* $\neq 0$
proof
assume *g0*: *g z0* $= 0$
obtain *h r m*
where $0 < m\ 0 < r$ **and** *subS*: *ball z0 r* $\subseteq S$
and *holh*: *h holomorphic_on ball z0 r*
and *geq*: $\bigwedge w.\ w \in$ *ball z0 r* $\Longrightarrow$ *g w* $= (w - z0)\hat{}m * h\ w$
and *hnz*: $\bigwedge w.\ w \in$ *ball z0 r* $\Longrightarrow h\ w \neq 0$
by (*blast intro*: *holomorphic_factor_zero_nonconstant* [*OF holg S* ‹*z0* $\in S$› *g0 nonconst*])
then have *holf0*: $\mathcal{F}$ *n holomorphic_on ball z0 r* **for** *n*
by (*meson holf holomorphic_on_subset*)
have $*$: ($\lambda z.$ *deriv* ($\mathcal{F}$ *n*) *z* / $\mathcal{F}$ *n z*) *has_contour_integral 0*) (*circlepath z0* (*r*/2)) **for** *n*
proof (*rule Cauchy_theorem_disc_simple*)
show ($\lambda z.$ *deriv* ($\mathcal{F}$ *n*) *z* / $\mathcal{F}$ *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* $\circ$ *g*)
using ‹0 < *r*› *subS*
by (*intro continuous_intros holomorphic_on_imp_continuous_on holomorphic_on_subset* [*OF holg*]) *auto*
then have *compact* ((*cmod* $\circ$ *g*) ‘ *sphere z0* (*r*/2))
by (*rule compact_continuous_image* [*OF _ compact_sphere*])
moreover have (*cmod* $\circ$ *g*) ‘ *sphere z0* (*r*/2) $\neq \{\}$
using ‹0 < *r*› **by** *auto*
ultimately obtain *b* **where** *b*: $b \in$ (*cmod* $\circ$ *g*) ‘ *sphere z0* (*r*/2)
$\bigwedge t.\ t \in$ (*cmod* $\circ$ *g*) ‘ *sphere z0* (*r*/2) $\Longrightarrow b \leq t$
using *compact_attains_inf* [*of* (*norm* $\circ$ *g*) ‘ (*sphere z0* (*r*/2))] **by** *blast*
have ($\lambda n.$ *contour_integral* (*circlepath z0* (*r*/2)) ($\lambda z.$ *deriv* ($\mathcal{F}$ *n*) *z* / $\mathcal{F}$ *n z*)) $\longrightarrow$
contour_integral (*circlepath z0* (*r*/2)) ($\lambda z.$ *deriv g z* / *g z*)
proof (*rule contour_integral_uniform_limit_circlepath*)
show $\forall_F$ *n in sequentially*. ($\lambda z.$ *deriv* ($\mathcal{F}$ *n*) *z* / $\mathcal{F}$ *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)) (λn z. deriv (ℱ n) z / ℱ n z) (λz. deriv
g z / g z) sequentially
   proof (rule uniform_lim_divide [OF _ _ bo_dg])
    show uniform_limit (sphere z0 (r/2)) (λa. 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
     using isolated_zeros [of λz. g z − g z1 S z2 z0] S ‹z0 ∈ S› z0 z12 by auto
   have g z2 − g z1 ≠ 0
```

proof (*rule Hurwitz_no_zeros* [*of* $S - \{z1\}$ $\lambda n\ z.\ \mathcal{F}\ n\ z - \mathcal{F}\ n\ z1$ $\lambda z.\ g\ z - g\ z1$])
show *open* $(S - \{z1\})$
by (*simp add*: *S open_delete*)
show *connected* $(S - \{z1\})$
by (*simp add*: *connected_open_delete* [*OF S*])
show $\bigwedge n.\ (\lambda z.\ \mathcal{F}\ n\ z - \mathcal{F}\ n\ z1)$ *holomorphic_on* $S - \{z1\}$
by (*intro holomorphic_intros holomorphic_on_subset* [*OF holf*]) *blast*
show $(\lambda z.\ g\ z - g\ z1)$ *holomorphic_on* $S - \{z1\}$
by (*intro holomorphic_intros holomorphic_on_subset* [*OF holg*]) *blast*
show *uniform_limit* K $(\lambda n\ z.\ \mathcal{F}\ n\ z - \mathcal{F}\ n\ z1)$ $(\lambda z.\ g\ z - g\ z1)$ *sequentially*
if *compact* K $K \subseteq S - \{z1\}$ **for** K
proof (*rule uniform_limitI*)
fix *e*::*real*
assume $e > 0$
have *uniform_limit* K $\mathcal{F}$ g *sequentially*
using *that ul_g* **by** *fastforce*
then have K: $\forall_F\ n$ *in sequentially*. $\forall x \in K.$ *dist* $(\mathcal{F}\ n\ x)\ (g\ x) < e/2$
using ‹$0 < e$› **by** (*force simp*: *intro*!: *uniform_limitD*)
have *uniform_limit* $\{z1\}$ $\mathcal{F}$ g *sequentially*
by (*intro ul_g*) (*auto simp*: *z12*)
then have $\forall_F\ n$ *in sequentially*. $\forall x \in \{z1\}.$ *dist* $(\mathcal{F}\ n\ x)\ (g\ x) < e/2$
using ‹$0 < e$› **by** (*force simp*: *intro*!: *uniform_limitD*)
then have *z1*: $\forall_F\ n$ *in sequentially*. *dist* $(\mathcal{F}\ n\ z1)\ (g\ z1) < e/2$
by *simp*
show $\forall_F\ n$ *in sequentially*. $\forall x{\in}K.$ *dist* $(\mathcal{F}\ n\ x - \mathcal{F}\ n\ z1)\ (g\ x - g\ z1) < e$
apply (*intro 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 $\neg$ $(\lambda z.\ g\ z - g\ z1)$ *constant_on* $S - \{z1\}$
unfolding *constant_on_def*
by (*metis Diff_iff* ‹$z0 \in S$› *empty_iff insert_iff right_minus_eq z0 z12*)
show $\bigwedge n\ z.\ z \in S - \{z1\} \Longrightarrow \mathcal{F}\ n\ z - \mathcal{F}\ n\ z1 \neq 0$
by (*metis DiffD1 DiffD2 eq_iff_diff_eq_0 inj inj_onD insertI1* ‹$z1 \in S$›)
show $z2 \in S - \{z1\}$
using ‹$z2 \in S$› ‹$z1 \neq z2$› **by** *auto*
qed
with *z12* **show** *False* **by** *auto*
qed
then show *?thesis* **by** (*auto simp*: *inj_on_def*)
qed

6.5 The Great Picard theorem

lemma *GPicard1*:
assumes S: *open* S *connected* S **and** $w \in S$ $0 < r$ $Y \subseteq X$
and *holX*: $\bigwedge h.\ h \in X \Longrightarrow h$ *holomorphic_on* S
and *X01*: $\bigwedge h\ z.\ [\![h \in X;\ z \in S]\!] \Longrightarrow h\ z \neq 0 \wedge h\ z \neq 1$

```
    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*: $\bigwedge n.\ r\ n \in \{n.\ norm(h\ n\ w) \geq 1\}$
by *blast*
obtain *B* **where** *B*: $\bigwedge j\ z.\ [\![norm\ z = 1/2;\ j \in range\ (\lambda n.\ inverse \circ h\ (r\ n))]\!] \Longrightarrow norm(j\ z) \leq B$
proof (*rule GPicard3* [*OF* _ _ *w*, **where** *K* = *sphere 0* (*1/2*)])
show *range* ($\lambda n.\ inverse \circ h\ (r\ n)$) $\subseteq$
$\{g.\ g\ holomorphic_on\ ball\ 0\ 1 - \{0\} \wedge (\forall z \in ball\ 0\ 1 - \{0\}.\ g\ z \neq 0 \wedge g\ z \neq 1)\}$
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 $\bigwedge j.\ j \in range\ (\lambda n.\ inverse \circ h\ (r\ n)) \Longrightarrow cmod\ (j\ w) \leq 1$
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*: $\bigwedge z.\ z \in ball\ 0\ \varepsilon - \{0\} \Longrightarrow cmod((inverse \circ f)\ z) \leq B$
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
```

7 Moebius functions, Equivalents of Simply Connected Sets, Riemann Mapping Theorem

```
theory Riemann_Mapping
imports Great_Picard
begin
```

7.1 Moebius functions are biholomorphisms of the unit disc

```
definition Moebius_function :: [real,complex,complex] ⇒ complex where
  Moebius_function ≡ λt 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) ^ 2*
using ‹*1 − cnj w * z ≠ 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 ≠ 0* **if** *norm z < 1* **for** *z*
proof −
have *norm (1*::*complex) ≠ norm (z * w)*
using *assms that norm_mult_less* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
show *?thesis*
unfolding *Moebius_function_def*
proof (*intro holomorphic_intros*)
show *⋀z. z ∈ ball 0 1 ⟹ 1 − cnj w * z ≠ 0*
by (*metis * complex_cnj_cnj complex_cnj_mult complex_mod_cnj mem_ball_0 mult.commute mult_1 right_minus_eq*)
qed
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*
using ‹*cmod w2* $<$ *1*› *complex_mod_sqrt_Re_mult_cnj* **by** *force*
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

7.2 A big chain of equivalents of simple connectedness for an open set

lemma *biholomorphic_to_disc_aux*:

```
  assumes open S connected S 0 ∈ S and S01: S ⊆ ball 0 1
      and prev: ⋀f. ⟦f holomorphic_on S; ⋀z. z ∈ S ⟹ f z ≠ 0; inj_on f S⟧
                ⟹ ∃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 −
  define F where F ≡ {h. h holomorphic_on S ∧ h ‘ S ⊆ ball 0 1 ∧ h 0 = 0 ∧
inj_on h S}
  have idF: id ∈ F
    using S01 by (auto simp: F_def)
  then have F ≠ {}
    by blast
  have imF_ne: ((λh. norm(deriv h 0)) ‘ F) ≠ {}
    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
          using DERIV_chain' [where g=f] ‹open S›
            by (meson DERIV_cmult_Id ‹f ∈ F› holF holomorphic_derivI im-
age_subset_iff
              r01 that)
        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
  proof (rule order_antisym [OF _ le])
    show cmod (deriv f 0) ≤ l
      using ‹f ∈ F› bdd cSUP_upper by (fastforce simp: l_def)
  qed
  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:
l_def)
      then have |norm (deriv f 0) − l| < 1 / (Suc n)
        by (fastforce simp: 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
    using ‹open S› ‹0 ∈ S› no_df0 holomorphic_nonconstant [OF holf] eql [OF
_ idF]
    by force
  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 inj_on_subset*)
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*
using *holomorphic_on_subset* [*OF holk*] *f01 h01* **by** *force*
show *g holomorphic_on k* ‘ (*h* ∘ *f*) ‘ *S*
using *holomorphic_on_subset* [*OF holg*] **by** (*force simp*: *kh nf1*)
show *ϑ holomorphic_on g* ‘ *k* ‘ (*h* ∘ *f*) ‘ *S*
using *holomorphic_on_subset* [*OF holϑ*] **by** (*force simp*: *gf kh nf1*)
qed
show ((*ϑ* ∘ *g* ∘ *k*) (*h* (*f z*)))² = *h* (*f z*) **if** *z* ∈ *S* **for** *z*
using *ϑ2 gf kh nf1 that* **by** *fastforce*
qed
qed
have *normψ1*: *norm*(*ψ* (*h* (*f z*))) < *1* **if** *z* ∈ *S* **for** *z*
by (*metis ψ2 h01 image_subset_iff mem_ball_0 nf1 norm_power power_less1_D that*)
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*
using *holomorphic_on_subset* [*OF holh*] *f01* **by** *fastforce*
show *ψ holomorphic_on h* ‘ *f* ‘ *S*
using *holomorphic_on_subset* [*OF holψ*] **by** *fastforce*
show *p holomorphic_on ψ* ‘ *h* ‘ *f* ‘ *S*
using *holomorphic_on_subset* [*OF holp*] **by** (*simp add*: *image_subset_iff normψ1*)
qed
show (*p* ∘ *ψ* ∘ *h* ∘ *f*) ‘ *S* ⊆ *ball 0 1*
using *normψ1 p01* **by** *fastforce*
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
      using q01  holomorphic_on_subset [OF holk]
      by (force 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 fS: f ‘ S = ball 0 1
   using F_def ‹f ∈ F› by blast
 then have ∀ z∈ball 0 1. g z ∈ S ∧ f (g z) = z
   by (metis  gf imageE)
 with fS show ?thesis
   by (metis gf holf holg image_eqI)
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* $\subseteq$ *S pathfinish g* = *pathstart g f holomorphic_on S*
$\bigwedge\gamma$ *z*. ⟦*path* γ; *path_image* $\gamma \subseteq S$; *pathfinish* γ = *pathstart* γ; *z* $\notin$ *S*⟧ $\Longrightarrow$ *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*: $\bigwedge\gamma$ *f*. ⟦*valid_path* γ; *path_image* $\gamma \subseteq S$; *pathfinish* γ = *pathstart* γ; *f holomorphic_on S*⟧ $\Longrightarrow$ (*f has_contour_integral 0*) γ
shows $\exists h.\ \forall z \in S.$ (*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* $\in$ *S*
by *blast*
show *?thesis*
proof (*intro exI ballI*)
fix *x* **assume** *x* $\in$ *S*
then obtain *d* **where** *d* > *0* **and** *d*: *cball x d* $\subseteq$ *S*
using *openS open_contains_cball_eq* **by** *blast*
let *?g* = λ*z*. (*SOME g. polynomial_function g* $\wedge$ *path_image g* $\subseteq$ *S* $\wedge$ *pathstart g* = *a* $\wedge$ *pathfinish g* = *z*)
show ((λ*z*. *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 (λ*y*. *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* $\in$ *S*› *holf continuous_on_eq_continuous_at holomorphic_on_imp_continuous_on* **by** *auto*
ultimately obtain *d1* **where** *d1* > *0*
and *d1*: $\bigwedge$*x'*. *dist x' x* < *d1* $\Longrightarrow$ *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* $\subseteq$ *S*
using *openS* ‹*x* $\in$ *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
        using gxy(2) has_contour_integral_integrable vp by fastforce
      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 (λ*z*. *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* λ*z*. *deriv f z* / *f z*] **by** *metis*
have *Df*: ⋀*x*. *x* ∈ *S* ⟹ *DERIV f x* :> *deriv f x*
using *holf holomorphic_derivI openS* **by** *force*
have *hfd*: ⋀*x*. *x* ∈ *S* ⟹ ((λ*z*. *exp* (*g z*) / *f z*) *has_field_derivative* 0) (*at x*)
by (*rule derivative_eq_intros Df g nz*| *simp*)+
obtain *c* **where** *c*: ⋀*x*. *x* ∈ *S* ⟹ *exp* (*g x*) / *f x* = *c*
proof (*rule DERIV_zero_connected_constant*[*OF* ‹*connected S*› *openS*])
show *continuous_on S* (λ*z*. *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* − {}. ((λ*z*. *exp* (*g z*) / *f z*) *has_field_derivative* 0) (*at x*)
using *hfd* **by** *blast*
qed *auto*
show *?thesis*
proof (*intro exI ballI conjI*)
have *g holomorphic_on S*
using *openS g holomorphic_on_open* **by** *blast*
then show (λ*z*. *Ln*(*inverse c*) + *g z*) *holomorphic_on S*
by (*intro holomorphic_intros*)
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 (λ*z*. *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*: $S \neq \{\}$ $S \neq UNIV$
and *prev*: $\bigwedge f.\ [\![f\ holomorphic_on\ S; \forall z \in S.\ f\ z \neq 0]\!] \Longrightarrow \exists g.\ g\ holomorphic_on\ S \wedge (\forall z \in S.\ f\ z = (g\ z)^2)$
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 −
obtain *a b* **where** $a \in S$ $b \notin S$
using *S* **by** *blast*
then obtain δ **where** $\delta > 0$ **and** δ: *ball a* $\delta \subseteq S$
using *openS openE* **by** *blast*
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *eqg*: $\bigwedge z.\ z \in S \Longrightarrow z - b = (g\ z)^2$
proof (*rule exE* [*OF prev* [*of* $\lambda z.\ z - b$]])
show ($\lambda z.\ z - b$) *holomorphic_on S*
by (*intro holomorphic_intros*)
qed (*use* ‹$b \notin S$› **in** *auto*)
have ¬ *g constant_on S*
proof −
have $(a + \delta/2) \in ball\ a\ \delta$ $a + (\delta/2) \neq a$
using ‹$\delta > 0$› **by** (*simp_all add*: *dist_norm*)
then show *?thesis*
unfolding *constant_on_def*
using *eqg* [*of a*] *eqg* [*of* $a + \delta/2$] ‹$a \in S$› 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* $\subseteq$ (*g* ‘ *ball a* δ)
by (*metis* ‹$0 < \delta$› *centre_in_ball imageI openE*)
have *g_not_r*: *g z* $\notin$ *ball* (−(*g a*)) *r* **if** $z \in S$ **for** *z*
proof
assume *g z* $\in$ *ball* (−(*g a*)) *r*
then have − *g z* $\in$ *ball* (*g a*) *r*
by (*metis add.inverse_inverse dist_minus mem_ball*)
with *r* **have** − *g z* $\in$ (*g* ‘ *ball a* δ)
by *blast*
then obtain *w* **where** *w*: − *g z* = *g w dist a w* < δ
by *auto*
with δ **have** $w \in S$
by *force*
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 that ‹b ∉ S› show False
    by force
  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
     by (smt (verit) comp_def holf holh holomorphic_on_compose image_eqI
nz(1) prev)
   show ?thesis
   proof (intro exI conjI)
    show g ∘ k holomorphic_on h ‘ S
```

by (*smt* (*verit*) *holg holk holomorphic_on_compose holomorphic_on_subset imageE image_subset_iff kh*)
show $\forall z \in h \mathbin{`} S.\ f\ z = ((g \circ k)\ z)^2$
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*: $\forall z{\in}h \mathbin{`} S.\ f\ z \in ball\ 0\ 1 \wedge g\ (f\ z) = z$ **and** *fg*:$\forall z{\in}ball\ 0\ 1.\ g\ z \in h \mathbin{`} S \wedge f\ (g\ z) = z$
using *biholomorphic_to_disc_aux* [*OF 1 2 3 h01 4*] **by** *blast*
show *?thesis*
proof (*intro exI conjI*)
show $f \circ h$ *holomorphic_on S*
by (*simp add*: *f holh holomorphic_on_compose*)
show $k \circ g$ *holomorphic_on ball 0 1*
by (*metis holomorphic_on_subset image_subset_iff fg holk g holomorphic_on_compose*)
qed (*use fg gf kh* **in** *auto*)
qed

lemma *homeomorphic_to_disc*:
assumes $S = UNIV \vee$
$(\exists f\ g.\ f$ *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*
by (*smt* (*verit*, *ccfv_SIG*) *holomorphic_on_imp_continuous_on homeomorphic_ball01_UNIV homeomorphic_minimal assms*)

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 ∧ f holomorphic_on S
⟶ *(f has_contour_integral 0) g*) (**is** *?ci0*)
and *simply_connected_eq_global_primitive*:
simply_connected S ⟷
connected S ∧
(∀f. f holomorphic_on S ⟶
(∃h. ∀z. z ∈ S ⟶ *(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*
assumes *open* S
shows *contractible* $S \longleftrightarrow$ *simply_connected* S
proof
show *contractible* $S \Longrightarrow$ *simply_connected* S
by (*simp add*: *contractible_imp_simply_connected*)
show *simply_connected* $S \Longrightarrow$ *contractible* S
using *assms convex_imp_contractible homeomorphic_contractible_eq*
simply_connected_eq_homeomorphic_to_disc **by** *auto*
qed

7.3 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 $\forall\, C \in$ *components*(*frontier* S). $\neg$ *bounded* C
proof −
have $S = \{\} \vee S$ *homeomorphic ball* (*0*::*complex*) *1*
using *simply_connected_eq_homeomorphic_to_disc assms openS* **by** *blast*
then show *?thesis*
proof
assume $S = \{\}$
then 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*: $\bigwedge x.\ x \in S \Longrightarrow f(g\ x) = x$
and *fim*: f ‘ *ball 0 1* = S **and** *gf*: $\bigwedge y.$ *cmod* $y < 1 \Longrightarrow g(f\ y) = y$
and *contg*: *continuous_on* S g **and** *contf*: *continuous_on* (*ball 0 1*) f
by (*fastforce simp*: *homeomorphism_def homeomorphic_def*)
define D **where** $D \equiv \lambda n.$ *ball* (*0*::*complex*) (*1* − *1*/(*of_nat* n + *2*))
define A **where** $A \equiv \lambda n.$ {z::*complex*. *1* − *1*/(*of_nat* n + *2*) < *norm* z ∧ *norm* z < *1*}
define X **where** $X \equiv \lambda n$::*nat*. *closure*(f ‘ A n)
have *D01*: $D\ n \subseteq$ *ball 0 1* **for** n
by (*simp add*: *D_def ball_subset_ball_iff*)
have *A01*: $A\ n \subseteq$ *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 connected (A n) for n
      using connected_annulus [of _ 0::complex] by (simp add: A_def)
    then have connX: connected(X n) for n
      unfolding X_def
    by (metis A01 connected_continuous_image connected_imp_connected_closure
contf continuous_on_subset)
    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 Seq X_def ‹x ∈ closure S› ‹x ∉ S› clo_fim by fastforce
      qed
    qed
    moreover have (⋂ n. X n) ⊆ closure S − S
    proof −
      have (⋂ n. X n) ⊆ closure S
        using Xsubclo by blast
      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 -
        haveinj_on f (D n)
         unfolding inj_on_def using D01 by (metis gf mem_ball_0 subsetCE)
        then have op_fDn: open(f ‘ (D n))
       by (metis invariance_of_domain D_def Elementary_Metric_Spaces.open_ball

            continuous_on_subset [OF contf D01])
        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
      by (meson True Xsubclo bounded_closure bounded_subset)
    have compaX: compact (X n) for n
      by (simp add: bouX cloX compact_eq_bounded_closed)
    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 open V and compact (closure U)
                  and V: ⋂(range X) − K ⊆ V and U: K ⊆ U U ∩ V = {}
        by (metis Diff_disjoint separation_normal_compact [OF ‹compact K›
clo])
      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 XU: X j ∩ closure U = X j ∩ U
              using J closure_subset empty by fastforce
            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 ≠ {}
        by (metis Cco Ksub UNIV_I ‹C ⊆ K› ‹K ⊆ U› XU bot.extremum_uniqueI
in_components_maximal le_INF_iff le_inf_iff)
          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
        by (metis Diff_iff INT_I U V ‹open V› closure_iff_nhds_not_empty
            order.refl subsetD that)
       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
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)
        moreover have closed C
          using C_ccsw clo_ccs by blast
        ultimately show False
            by (metis C ‹S ≠ {}› ‹S ≠ UNIV› C_ccsw bot_eq_sup_iff con-
nected_component_eq_UNIV frontier_Int_closed
                  frontier_closed frontier_complement frontier_eq_empty fron-
tier_of_components_subset in_components_maximal inf.orderE)
      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 connected (closure S − S)
          by (metis confr frontier_def interior_open openS)
        moreover have ¬ bounded (−S)
          by (simp add: True cobounded_imp_unbounded)
        moreover have bounded (connected_component_set (− S) w)
          using C_ccsw ‹bounded C› by auto
        ultimately have z ∉ S
          using ‹w ∉ S› openS
        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 have connected_component_set (− S) z ≠ {}
          by (metis ComplI connected_component_eq_empty)
        then show frontier (connected_component_set (− S) z) ≠ {}
```

by (*metis False* ‹$S \neq UNIV$› *connected_component_eq_UNIV frontier_complement frontier_eq_empty*)
qed
then show *connected_component_set* $(-S)$ $z \cap$ *frontier* $S \neq \{\}$
by *auto*
qed
then show *False*
by (*metis C_ccsw Compl_iff* ‹$w \notin S$› ‹$z \notin C$› *connected_component_eq_empty connected_component_idemp*)
qed
qed
next
case *False*
obtain w **where** *C_ccsw*: $C =$ *connected_component_set* $(-S)$ w **and** $w \notin S$
using C **by** (*auto simp*: *components_def*)
have *frontier* (*connected_component_set* $(-S)$ w) $\subseteq$ *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) $\subseteq$ *frontier* S
using *frontier_complement frontier_of_connected_component_subset* **by** *blast*
moreover have *frontier* (*connected_component_set* $(-S)$ w) $\neq \{\}$
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*: $z \in$ *frontier* S **and** z: $z \in$ *connected_component_set* $(-S)$ w
by *blast*
have *connected_component_set* (*frontier* S) $z \in$ *components*(*frontier* S)
by (*simp add*: ‹$z \in$ *frontier* S› *componentsI*)
with *prev False* **have** $\neg$ *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 SC_Chain.openS SC_Chain_axioms bounded_subset closed_Compl connected_component_mono frontier_complement frontier_subset_eq*)
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*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*metis DIM_complex assms cobounded_has_bounded_component double_complement dual_order.order_iff_strict*
simply_connected_eq_unbounded_complement_components)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*simp add*: *assms connected_frontier_simple simply_connected_eq_frontier_properties*)
qed

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*)

7.4 Further equivalences based on continuous logs and sqrts

context *SC_Chain*

begin

lemma *continuous_log*:
fixes f :: *complex⇒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⇒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* ∘ *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⇒complex*
assumes *contf*: *continuous_on S f* **and** *nz*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
and *prev*: $\bigwedge f::complex⇒complex.$
$[\![$*continuous_on S f*; $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0]\!]$
$\Longrightarrow \exists g.$ *continuous_on S g* $\wedge$ $(\forall z \in S.\ f\ z = exp(g\ z))$
shows $\exists g.$ *continuous_on S g* $\wedge$ $(\forall z \in S.\ f\ z = (g\ z)^2)$
proof −
obtain *g* **where** *contg*: *continuous_on S g* **and** *geq*: $\bigwedge z.\ z \in S \Longrightarrow f\ z = exp(g\ z)$
using *contf nz prev* **by** *metis*
show *?thesis*
proof (*intro exI ballI conjI*)
show *continuous_on S* $(\lambda z.\ exp(g\ z/2))$

```
    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
     by (metis add.commute add_cancel_right_left dist_commute dist_complex_def
mem_ball
       norm_increases_online norm_not_less_zero norm_zero order_less_asym)
     have ∗: (λx. (f x − f z) / (x − z) / (g x + g z)) −z→ deriv f z / (g z + g z)
     proof (intro tendsto_intros)
       show (λx. (f x − f z) / (x − z)) −z→ deriv f z
         using ‹f holomorphic_on S› ‹z ∈ S› has_field_derivative_iff holomor-
phic_derivI openS by blast
       show g −z→ g z
         using ‹z ∈ S› contg continuous_on_eq_continuous_at isCont_def openS
by blast
     qed (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 ‹z ∈ S›
        apply (simp add: geq field_split_simps power2_eq_square)
        by (metis distrib_left mult_cancel_right)
    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)
 show ?log ?sqrt
    using local.continuous_log winding_number_zero continuous_sqrt continu-
ous_sqrt_imp_simply_connected
   by auto
qed
```

7.5 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* $\vee$ *compact S* $\bigwedge C$. $C \in$ *components S* $\Longrightarrow$ *Borsukian C*
shows *Borsukian S*
by (*metis Borsukian_componentwise_eq assms*)

lemma *simply_connected_eq_Borsukian*:
fixes S :: *complex set*
shows *open S* $\Longrightarrow$ (*simply_connected S* $\longleftrightarrow$ *connected S* $\wedge$ *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* $\Longrightarrow$ *Borsukian S* $\longleftrightarrow$ ($\forall$ $C \in$ *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* $\vee$ *closed S* **and** *bounded S*
shows *Borsukian S* $\longleftrightarrow$ *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

7.6 Finally, the Riemann Mapping Theorem

theorem *Riemann_mapping_theorem*:
open S $\wedge$ *simply_connected S* $\longleftrightarrow$
S = {} $\vee$ *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** _ = *?rhs*)
proof −
have *simply_connected S* $\longleftrightarrow$ *?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*
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

7.7 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*)

7.8 The winding number defines a continuous logarithm for the path itself

lemma *winding_number_as_continuous_log*:
assumes *path p* **and** ζ: ζ ∉ *path_image p*
obtains *q* **where** *path q*
pathfinish q − *pathstart q* = *2* ∗ *of_real pi* ∗ i ∗ *winding_number p* ζ
⋀*t*. *t* ∈ {*0..1*} ⟹ *p t* = ζ + *exp*(*q t*)
proof −
let *?q* = λ*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*}) (λ*x*. *winding_number* (*subpath 0 x p*) ζ)
if *t*: *t* ∈ {*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 ⋀f::complex⇒complex. continuous_on ?B f ∧ (∀ζ ∈ ?B. f ζ ≠
0)
                ⟶ (∃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: ⋀z. z ∈ ?B ⟹ z - ζ = exp (g z) by blast
    obtain d where 0 < d and d:
      ⋀x. ⟦x ∈ {0..1}; dist x t < d⟧ ⟹ 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 ⋀r. ⟦r ∈ closed_segment t u⟧ ⟹ dist (p t) (p r) < cmod (p
t - ζ)
          by (smt (verit, best) d dist_commute dist_in_closed_segment subsetD
```

```
‹dist u t < d›)
        then have piB: path_image(subpath t u p) ⊆ ?B
          by (auto simp: path_image_subpath_gen)
        have ∗: path (g ∘ subpath t u p)
        proof (rule path_continuous_image)
          show path (subpath t u p)
            using ‹path p› t that by auto
          show continuous_on (path_image (subpath t u p)) g
            using piB contg continuous_on_subset by blast
        qed
        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 ζ ≠ 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 = ζ + exp (?q t) if t ∈ {0..1} for t
  proof −
    have path (subpath 0 t p)
      using ‹path p› that by auto
    moreover
```

```
    have ζ ∉ 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] ‹ζ ≠ p 0›
      by (auto simp: exp_add algebra_simps pathfinish_def pathstart_def sub-
path_def)
   qed
  qed
qed
```

7.9 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: ζ ∉ path_image p
  shows winding_number p ζ = 0 ⟷ (∃ a. homotopic_loops (−{ζ}) p (λt. 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: ⋀t. t ∈ {0..1} ⟹ p t = ζ + exp(q t)
    using winding_number_as_continuous_log [OF assms] by blast
  have ∗: homotopic_with_canon (λr. pathfinish r = pathstart r)
                  {0..1} (−{ζ}) ((λw. ζ + exp w) ∘ q) ((λw. ζ + exp w) ∘ (λt. 0))
  proof (rule homotopic_with_compose_continuous_left)
    show homotopic_with_canon (λf. pathfinish ((λw. ζ + exp w) ∘ f) = pathstart
((λw. ζ + exp w) ∘ f))
            {0..1} UNIV q (λt. 0)
   proof (rule homotopic_with_mono, simp_all add: pathfinish_def pathstart_def)
      have homotopic_loops UNIV q (λt. 0)
      by (rule homotopic_loops_linear) (use qeq ‹path q› in ‹auto simp: path_defs›)
      then have homotopic_with (λr. r 1 = r 0) (top_of_set {0..1}) euclidean q
(λt. 0)
        by (simp add: homotopic_loops_def pathfinish_def pathstart_def)
     then show homotopic_with (λh. 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)+
  qed auto
 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* $(-\{\zeta\})$ *p* $(\lambda t.\ a)$ **..**
then have *winding_number p* ζ = *winding_number* $(\lambda t.\ a)$ ζ *a* $\neq$ ζ
using *winding_number_homotopic_loops homotopic_loops_imp_subset* **by** (*force simp*:)+
then show *?lhs*
using *winding_number_zero_const* **by** *auto*
qed

lemma *winding_number_homotopic_paths_null_explicit_eq*:
assumes *path p* **and** ζ: $\zeta \notin$ *path_image p*
shows *winding_number p* ζ = *0* $\longleftrightarrow$ *homotopic_paths* $(-\{\zeta\})$ *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*: *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: $\zeta \notin$ *path_image p*
shows *winding_number p* ζ = *0* $\longleftrightarrow$ ($\exists$ *a. homotopic_paths* $(-\{\zeta\})$ *p* $(\lambda t.\ a)$)
(**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: $\zeta \notin$ *path_image p*
and *path q* **and** ζq: $\zeta \notin$ *path_image q*
and *qp*: *pathstart q* = *pathstart p pathfinish q* = *pathfinish p*
shows *winding_number p* ζ = *winding_number q* ζ $\longleftrightarrow$ *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: 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 *Complex_Singularities*
imports *Conformal_Mappings*
begin

7.10 Non-essential singular points

lemma *at_to_0'*: *NO_MATCH 0 z* ⟹ *at z* = *filtermap* (λ*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 (λ*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
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 (λ*x*. *if x*=*z then 0 else inverse* (*f x*)) *holomorphic_on s*
proof −
define *g* **where** *g* ≡ λ*x*. *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* (λ*z*::*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*]:
assumes *c* ≠ *0*
shows *is_pole* (λ*z*. *c* ∗ *f z* :: *'a* :: *real_normed_field*) *z* ⟷ *is_pole f z*
proof
assume *is_pole* (λ*z*. *c* ∗ *f z*) *z*
with ‹*c*≠*0*› **have** *is_pole* (λ*z*. *inverse c* ∗ (*c* ∗ *f z*)) *z*
unfolding *is_pole_def*
by (*force intro*: *tendsto_mult_filterlim_at_infinity*)
thus *is_pole f z*
using ‹*c*≠*0*› **by** (*simp add*: *field_simps*)
next
assume *is_pole f z*
with ‹*c*≠*0*› **show** *is_pole* (λ*z*. *c* ∗ *f z*) *z*
by (*auto intro*!: *tendsto_mult_filterlim_at_infinity simp*: *is_pole_def*)
qed

lemma *is_pole_uminus_iff* [*simp*]: *is_pole* (λ*z*. −*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* (λ*z*::*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* (λ*z*. *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)
       (use assms in ‹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 ⟷ is_pole (λx. f x + c) z
proof
  assume is_pole f z
  then have filterlim f at_infinity (at z) unfolding is_pole_def .
  moreover have ((λ_. c) ⟶ 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 (λx. f x + c) z unfolding is_pole_def .
next
  assume is_pole (λx. f x + c) z
  then have filterlim (λx. f x + c) at_infinity (at z)
    unfolding is_pole_def .
  moreover have ((λ_. −c) ⟶ −c) (at z) by auto
  ultimately have LIM x (at z). f x :> at_infinity
    using tendsto_add_filterlim_at_infinity'[of (λx. f x + c)
        at z (λ_. − c) −c]
    by auto
```

```
  then show is_pole f z unfolding is_pole_def .
qed

lemma is_pole_minus_const_iff:
  is_pole (λx. f x − c) z ⟷ is_pole f z
  using is_pole_plus_const_iff [of f z −c] by simp

lemma is_pole_alt:
  is_pole f x = (∀ B>0. ∃ U. open U ∧ x∈U ∧ (∀ y∈U. y≠x ⟶ norm (f y)≥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 ≠ 0
  shows   is_pole (λx. f x * g x) x
  unfolding is_pole_def
proof (rule tendsto_mult_filterlim_at_infinity)
  show f −x→ 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 ≠ 0
  shows   is_pole (λx. f x * g x) x
proof −
  have g: g analytic_on {x}
    using assms by auto
  show ?thesis
    using is_pole_mult_analytic_nonzero1 [OF ‹is_pole f x› g] ‹g x ≠ 0›
    by (simp add: mult.commute)
qed

lemma is_pole_mult_analytic_nonzero1_iff:
  assumes f analytic_on {x} f x ≠ 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)
  then have filterlim (f ∘ g) at_infinity sequentially
    unfolding o_def
    using ‹is_pole f z› filterlim_compose is_pole_def by blast
  ultimately show False
     using not_tendsto_and_filterlim_at_infinity trivial_limit_sequentially by
blast
qed
```

7.11 Isolated singularities

The proposition $\exists\, x.\ f\ -z\to\ x\ \vee$ *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 (λx. 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 fast-
force
```

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 (λ*y*. *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 (simp add: LIM_equal)
      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
    using assms by (simp add: no_isolated_singularity)
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 ≡ λn 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'=(λx. 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 x 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 ∀ 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 ∀ F w in (at z). g w = f w
  shows isolated_singularity_at f z
  using assms isolated_singularity_at_cong by blast
```

```
lemma not_essential_powr[singularity_intros]:
  assumes LIM w (at z). f w :> (at x)
  shows not_essential (λw. (f w) powi n) z
proof −
  define fp where fp=(λw. (f w) powi n)
  have ?thesis when n>0
  proof −
    have (λw.  (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 ∀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* $\forall_F$ *w in* (*at z*). *f w*≠*0*
shows *isolated_singularity_at* (λ*w*. (*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 (λ*w*. (*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*. *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*] **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 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 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 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 (λw. f w * g w) z
proof −
  define fg where fg = (λw. 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 fg_def frequently_elim1 not_eventually that by fastforce
    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] 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] 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 that unfolding r1_def by auto
      moreover have g w = gp w * (w−z) powi gn gp w ≠ 0
        using gr 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 by (auto simp add: fg_def power_int_add)
    qed

    obtain [intro]: fp −z→fp z gp −z→gp z
        using fr(1) ‹fr>0› gr(1) ‹gr>0›
        by (metis centre_in_ball continuous_at continuous_on_interior
            holomorphic_on_imp_continuous_on interior_cball)
    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
        using 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 not_eventually that by fastforce
   then have vf −z→0
    unfolding vf_def by (simp add: tendsto_eventually)
   then show ?thesis
    unfolding not_essential_def vf_def by auto
  qed
  moreover have ?thesis when is_pole f z
   by (metis (mono_tags, lifting) filterlim_at filterlim_inverse_at_iff is_pole_def
       not_essential_def that)
  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 show ?thesis unfolding not_essential_def vf_def
     using filterlim_atI filterlim_inverse_at_iff is_pole_def by blast
   qed
   moreover have ?thesis when fz≠0
    using fz not_essential_def tendsto_inverse that by blast
   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* (λ*w*. *inverse* (*f w*)) *z*
proof −
define *vf* **where** *vf* = (λ*w*. *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*}
using *holomorphic_transform*[*of* λ_. *0*]
by (*metis Diff_iff dist_commute holomorphic_on_const insert_iff mem_ball*)
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* (λ*w*. *f w* / *g w*) *z*
proof −
have *not_essential* (λ*w*. *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 *isolated_singularity_at f z* **and** *isolated_singularity_at g z*

shows *isolated_singularity_at* (λ*w*. *f w* − *g w*) *z*
unfolding *diff_conv_add_uminus*
using *assms isolated_singularity_at_add isolated_singularity_at_uminus* **by** *blast*

lemma *isolated_singularity_at_divide*[*singularity_intros*]:
assumes *isolated_singularity_at f z*
and *isolated_singularity_at g z*
and *not_essential g z*
shows *isolated_singularity_at* (λ*w*. *f w* / *g w*) *z*
unfolding *divide_inverse*
by (*simp add*: *assms isolated_singularity_at_inverse isolated_singularity_at_times*)

lemma *isolated_singularity_at_const*[*singularity_intros*]:
isolated_singularity_at (λ*w*. *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* (λ*z*. *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* (λ*w*. *w* ∈ *ball z r* − {*z*}) (*at z*)
using *r*(*1*) **by** (*intro eventually_at_in_open*) *auto*
thus *eventually* (λ*z*. *f analytic_on* {*z*}) (*at z*)
by *eventually_elim* (*use r analytic_on_subset* **in** *auto*)
next
assume *eventually* (λ*z*. *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* (λ*x*. *f* (*x* + *w*)) *z*
shows *isolated_singularity_at f* (*z* + *w*)
proof −
from *assms* **obtain** *r* **where** *r*: *r* > *0* **and** *ana*: (λ*x*. *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 (λx. f (x + w)) z
  using isolated_singularity_at_shift[of f w z]
        isolated_singularity_at_shift[of λx. 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
(λx. 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 (λx. f (x + w)) z
  shows   not_essential f (z + w)
proof −
  from assms consider c where (λx. f (x + w)) −z→ c | is_pole (λx. 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 (λx.
f (x + w)) z
  using not_essential_shift[of f w z]
        not_essential_shift[of λx. f (x + w) −w w + z]
  by (auto simp: algebra_simps)
```

lemma *not_essential_shift_0*:
NO_MATCH 0 z $\Longrightarrow$ *not_essential f z* $\longleftrightarrow$ *not_essential* ($\lambda x.$ *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* $\in$ *A open A*
shows *not_essential f x*
by (*metis assms at_within_open continuous_on holomorphic_on_imp_continuous_on not_essential_def*)

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* ($\lambda w.$ *w*) *z*
by (*simp add*: *not_essential_analytic*)

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

lemma *tendsto_imp_not_essential* [*intro*]: *f* $-z\to$ *c* $\Longrightarrow$ *not_essential f z*
by (*auto simp*: *not_essential_def*)

lemma *eventually_not_pole*:
assumes *isolated_singularity_at f z*
shows *eventually* ($\lambda w.$ $\neg$*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* ($\lambda w.$ *w* $\in$ *ball z r* − {*z*}) (*at z*)
by (*intro eventually_at_in_open*) *auto*
thus *eventually* ($\lambda w.$ $\neg$*is_pole f w*) (*at z*)
by (*metis* (*no_types*, *lifting*) *analytic_at analytic_on_analytic_at eventually_mono not_is_pole_holomorphic r*)
qed

lemma *not_islimpt_poles*:
assumes *isolated_singularity_at f z*
shows $\neg$*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*} $\Longrightarrow$ $\neg$*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*
by (*meson Diff_subset analytic_at assms holomorphic_on_subset isolated_singularity_at_holomorphic*)

lemma *isolated_singularity_sum* [*singularity_intros*]:
assumes ⋀*x*. *x* ∈ *A* ⟹ *isolated_singularity_at* (*f x*) *z*
shows *isolated_singularity_at* (λ*w*. ∑ *x*∈*A*. *f x w*) *z*
using *assms* **by** (*induction A rule*: *infinite_finite_induct*) (*auto intro*!: *singularity_intros*)

lemma *isolated_singularity_prod* [*singularity_intros*]:
assumes ⋀*x*. *x* ∈ *A* ⟹ *isolated_singularity_at* (*f x*) *z*
shows *isolated_singularity_at* (λ*w*. ∏ *x*∈*A*. *f x w*) *z*
using *assms* **by** (*induction A rule*: *infinite_finite_induct*) (*auto intro*!: *singularity_intros*)

lemma *isolated_singularity_sum_list* [*singularity_intros*]:
assumes ⋀*f*. *f* ∈ *set fs* ⟹ *isolated_singularity_at f z*
shows *isolated_singularity_at* (λ*w*. ∑ *f*←*fs*. *f w*) *z*
using *assms* **by** (*induction fs*) (*auto intro*!: *singularity_intros*)

lemma *isolated_singularity_prod_list* [*singularity_intros*]:
assumes ⋀*f*. *f* ∈ *set fs* ⟹ *isolated_singularity_at f z*
shows *isolated_singularity_at* (λ*w*. ∏ *f*←*fs*. *f w*) *z*
using *assms* **by** (*induction fs*) (*auto intro*!: *singularity_intros*)

lemma *isolated_singularity_sum_mset* [*singularity_intros*]:
assumes ⋀*f*. *f* ∈# *F* ⟹ *isolated_singularity_at f z*
shows *isolated_singularity_at* (λ*w*. ∑ *f*∈#*F*. *f w*) *z*
using *assms* **by** (*induction F*) (*auto intro*!: *singularity_intros*)

lemma *isolated_singularity_prod_mset* [*singularity_intros*]:
assumes ⋀*f*. *f* ∈# *F* ⟹ *isolated_singularity_at f z*
shows *isolated_singularity_at* (λ*w*. ∏ *f*∈#*F*. *f w*) *z*
using *assms* **by** (*induction F*) (*auto intro*!: *singularity_intros*)

lemma *analytic_nhd_imp_isolated_singularity*:

assumes *f analytic_on A − {x} x ∈ A open A*
shows *isolated_singularity_at f x*
unfolding *isolated_singularity_at_def* **using** *assms*
using *analytic_imp_holomorphic isolated_singularity_at_def isolated_singularity_at_holomorphic*
by *blast*

lemma *isolated_singularity_at_iff_analytic_nhd*:
isolated_singularity_at f x ⟷ (∃ A. x ∈ A ∧ open A ∧ f analytic_on A − {x})
by (*meson open_ball analytic_nhd_imp_isolated_singularity*
centre_in_ball isolated_singularity_at_def)

7.12 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*: ∃ *F 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* = (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 ∃!*k*. ∃ *g r*. *P k 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** (*rule theI′*)
then have ∃ *r*. *P n g r*
unfolding *P_def zor_poly_def g_def n_def* **by** (*rule someI_ex*)
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 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) 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
    then show ?thesis
      by (metis ‹vfn = − fn› power_int_not_zero right_minus_eq  fr_inverse
vfr(2)
          vector_space_over_itself.scale_right_imp_eq)
  qed
  then show ∀F w in (at z). vfp w = inverse (fp w)
    unfolding eventually_at by (metis DiffI dist_commute mem_ball singletonD
‹r1>0›)
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
      using n_def r_def rball1 that by auto
    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)
      then have ff (w−z) = zor_poly ff 0 (w−z) ∗ (w−z) powi (zorder ff 0)
        using rball2 by blast
      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
   by (metis DiffI ‹0 < r› dist_commute mem_ball singletonD)
 moreover have isCont (zor_poly f z) z
   using holo1[THEN holomorphic_on_imp_continuous_on]
   by (simp add: ‹0 < r1› continuous_on_interior)
 moreover
 have isCont (zor_poly ff 0) 0
  using ‹0 < r2› continuous_on_interior holo2 holomorphic_on_imp_continuous_on
   by fastforce
 then have isCont (λw. zor_poly ff 0 (w−z)) z
    unfolding isCont_iff by simp
 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) r1_def that by auto
    moreover have g w = gp w * (w−z) powi gn gp w ≠ 0
      using gr(2) 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 isolated_singularity_at fg z
      unfolding fg_def using isolated_singularity_at_times[OF f_iso g_iso] .
    moreover have not_essential fg z
      by (simp add: f_iso f_ness fg_def g_iso g_ness not_essential_times)
    moreover have ∃ F w in at z. fg w ≠ 0
      using fg_def fg_nconst by blast
    ultimately show ?thesis
      using that zorder_exist[of fg z] fgn_def fgp_def by fastforce
  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
  proof (rule holomorphic_factor_unique)
    show ∀ w ∈ ball z r2 − {z}. 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
      using fg_times fgp_nz fgr(2) r2_def by fastforce
  next
    show fgp holomorphic_on ball z r2
      using fgr(1) r2_def by auto
  next
    show (λw. fp w * gp w) holomorphic_on ball z r2
      by (metis ball_subset_cball fr(1) gr(1) holomorphic_on_mult holomor-
phic_on_subset
        min.cobounded1 min.cobounded2 r1_def r2_def subset_ball)
  qed (auto simp add: ‹0 < r2›)

  have fgp w = fp w *gp w when w: w ∈ ball z r2−{z} for w
  proof −
```

```
    have w ∈ ball z r1 − {z} w ∈ cball z fgr − {z} w≠z
      using w unfolding r2_def by auto
    then show ?thesis
    by (metis ‹fgn = fn + gn› eq_iff_diff_eq_0 fg_times fgr(2) power_int_eq_0_iff
        mult_right_cancel)
  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: ∃F w in (at z). f w * g w≠ 0
  shows zorder_divide: zorder (λw. f w / g w) z = zorder f z − zorder g z and
        zor_poly_divide: ∀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: ∃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)
  define vg where vg=(λw. inverse (g w))
  have 1: isolated_singularity_at vg z
    by (simp add: g_iso g_ness isolated_singularity_at_inverse vg_def)
  moreover have 2: not_essential vg z
    by (simp add: g_iso g_ness not_essential_inverse vg_def)
  moreover have 3: ∃F w in at z. f w * vg w ≠ 0
    using fg_nconst vg_def by auto
  ultimately have zorder (λw. f w * vg w) z = zorder f z + zorder vg z
    using zorder_times[OF f_iso _ f_ness] by blast
  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 ∀F w in at z. zor_poly (λw. f w * vg w) z w = zor_poly f z w * zor_poly
vg z w
    using zor_poly_times[OF f_iso _ f_ness,of vg] 1 2 3 by blast
  then show ∀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 holomor-
phic_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
      using ‹open S› ‹z∈S› holo holomorphic_on_imp_analytic_at isolated_singularity_at_analytic

        by force
      show not_essential f z unfolding not_essential_def
          using ‹open S› ‹z∈S› at_within_open continuous_on holo holomor-
phic_on_imp_continuous_on
        by fastforce
      have ∀ F w in at z. f w ≠ 0 ∧ w∈S
        using assms(4,5,6) holo non_const non_zero_neighbour_alt by blast
      then show ∃ F w in at z. f w ≠ 0
        by (auto elim: eventually_frequentlyE)
    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› LIM_unique divide_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
    using Lim_transform_within_open by fastforce
  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
      by (auto elim: eventually_frequentlyE)
  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
    using r
   by (metis centre_in_ball continuous_on_interior holomorphic_on_imp_continuous_on
       interior_cball isContD)
  have ∀x ∈ ball z r. x ≠ z ⟶ f x = g x * (x − z) ^ nat n
    by (simp add: ‹¬ n < 0› linorder_not_le power_int_def r)
  then have ∀F x in at z. f x = g x * (x − z) ^ nat n
    using centre_in_ball eventually_at_topological r(1) by blast
  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 ∀w∈cball z r − {z}. f w = g w / (w−z) ^ nat (− n) ∧ g w ≠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 ‹g z≠0› by auto
qed

lemma order_zorder:
  fixes p::complex poly and z::complex
  assumes p≠0
  shows order z p = nat (zorder (poly p) z)
proof −
  define n where n=nat (zorder (poly p) z)
  define h where h=zor_poly (poly p) z
  have ∃ w. poly p w ≠ 0 using assms poly_all_0_iff_0 by auto
  then obtain r where 0 < r cball z r ⊆ UNIV and
      h_holo: h holomorphic_on cball z r and
      poly_prod:(∀ w∈cball z r. poly p w = h w * (w − z) ^ n ∧ h w ≠ 0)
    using zorder_exist_zero[of poly p UNIV z,folded h_def] poly_holomorphic_on
    unfolding n_def by auto
  then have h holomorphic_on ball z r
    and (∀ w∈ball z r. poly p w = h w * (w − z) ^ n)
    and h z≠0
    by auto
  then have order z p = n using ‹p≠0›
  proof (induct n arbitrary:p h)
    case 0
    then have poly p z=h z using ‹r>0› by auto
    then have poly p z≠0 using ‹h z≠0› by auto
    then show ?case using order_root by blast
  next
    case (Suc n)
    define sn where sn=Suc n
    define h' where h'≡ λw. deriv h w * (w−z)+ sn * h w
    have (poly p has_field_derivative poly (pderiv p) w) (at w) for w
      using poly_DERIV[of p w] .
    moreover have (poly p has_field_derivative (h' w)*(w−z)^n ) (at w) when
w∈ball z r for w
     proof (subst DERIV_cong_ev[of w w poly p λw.  h w * (w − z) ^ Suc n
],simp_all)
      show ∀F x in nhds w. poly p x = h x * ((x − z) * (x − z) ^ n)
        unfolding eventually_nhds using Suc(3) ‹w∈ball z r›
        by (metis Elementary_Metric_Spaces.open_ball power_Suc)
      next
        have (h has_field_derivative deriv h w) (at w)
       using ‹h holomorphic_on ball z r› ‹w∈ball z r› holomorphic_on_imp_differentiable_at

          by (simp add: holomorphic_derivI)
        then have ((λw. h w * ((w − z) ^ sn))
```

```
                has_field_derivative h' w * (w - z) ^ (sn - 1)) (at w)
        unfolding h'_def
        apply (auto intro!: derivative_eq_intros simp add:field_simps)
        by (auto simp add:field_simps sn_def)
      then show ((λw. h w * ((w - z) * (w - z) ^ n))
                has_field_derivative h' w * (w - z) ^ n) (at w)
        unfolding sn_def by auto
    qed
   ultimately have ∀ w∈ball z r. poly (pderiv p) w = h' w * (w - z) ^ n
    using DERIV_unique by blast
   moreover have h' holomorphic_on ball z r
    unfolding h'_def using ‹h holomorphic_on ball z r›
    by (auto intro!: holomorphic_intros)
  moreover have h' z≠0 unfolding h'_def sn_def using ‹h z ≠ 0› of_nat_neq_0
by auto
   moreover have pderiv p ≠ 0
   proof
    assume pderiv p = 0
    obtain c where p=[:c:] using ‹pderiv p = 0› using pderiv_iszero by blast
    then have c=0
     using Suc(3)[rule_format,of z] ‹r>0› by auto
    then show False using ‹p≠0› using ‹p=[:c:]› by auto
   qed
   ultimately have order z (pderiv p) = n by (auto elim: Suc.hyps)
   moreover have order z p ≠ 0
    using Suc(3)[rule_format,of z] ‹r>0› order_root ‹p≠0› by auto
   ultimately show ?case
    by (meson ‹pderiv p ≠ 0› order_pderiv2)
 qed
 then show ?thesis unfolding n_def .
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= (λw. g w * (w−z) powi n)
 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 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)
         (use Suc.prems in ‹simp_all add: algebra_simps Suc_diff_le pochham-
mer_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
  using deriv_A[of nat n] that ‹n≥0› by (simp add: nz)
 moreover have n≤zn
 proof -
  have g z ≠ 0
   by (simp add: ‹g z ≠ 0›)
  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 that by (simp add: dist_commute r_def)
      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
    using * assms(1) eq' by blast
 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 (λx. f x * g x) z = zorder f z + zorder g z
 using assms isolated_singularity_at_analytic not_essential_analytic zorder_times
by blast

lemma zorder_cmult:
 assumes c ≠ 0
 shows   zorder (λz. c * f z) z = zorder f z
proof −
 define P where
   P = (λf n h r. 0 < r ∧ 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))
 have *: P (λx. c * f x) n (λx. c * h x) r
   if P f n h r c ≠ 0 for f n h r c
   using that unfolding P_def by (auto intro!: holomorphic_intros)
 have (∃h r. P (λx. c * f x) n h r) ⟷ (∃h r. P f n h r) for n
   using *[of f n _ _ c] *[of λx. c * f x n _ _ inverse c] ‹c ≠ 0›
   by (fastforce simp: field_simps)
 hence (THE n. ∃h r. P (λx. c * f x) n h r) = (THE n. ∃h r. P f n h r)
   by simp
 thus ?thesis
   by (simp add: zorder_def P_def)
qed

lemma zorder_uminus [simp]: zorder (λz. −f z) z = zorder f z
 using zorder_cmult[of −1 f] by simp

lemma zorder_nonzero_div_power:
```

assumes *sz*: *open S z* ∈ *S f holomorphic_on S f z* ≠ *0* **and** *n* > *0*
shows *zorder* (λ*w*. *f w* / (*w*−*z*) ^ *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* ∃ *F w in at z*. *f w* ≠ *0*
shows *eventually* (λ*w*. *zor_poly f z w* = *f w* ∗ (*w*−*z*) *powi* − *zorder f z*) (*at z*)
proof −
obtain *r* **where** *r*: *r*>*0*
(∀ *w*∈*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 ∗: ∀ *w*∈*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* (λ*w*. *w* ∈ *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* ∈ *S* ∃ *w*∈*S*. *f w* ≠ *0*
shows *eventually* (λ*w*. *zor_poly f z w* = *f w* / (*w*−*z*) ^ *nat* (*zorder f z*)) (*at z*)
proof −
obtain *r* **where** *r*: *r*>*0*
(∀ *w*∈*cball z r* − {*z*}. *f w* = *zor_poly f z w* ∗ (*w*−*z*) ^ *nat* (*zorder f z*))
using *zorder_exist_zero*[*OF assms*] **by** *auto*
then have ∗: ∀ *w*∈*ball z r* − {*z*}. *zor_poly f z w* = *f w* / (*w*−*z*) ^ *nat* (*zorder f z*)
by (*auto simp*: *field_simps powr_minus*)
have *eventually* (λ*w*. *w* ∈ *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* (λ*w*. *zor_poly f z w* = *f w* ∗ (*w*−*z*) ^ *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*
(∀ *w*∈*cball z r* − {*z*}. *f w* = *zor_poly f z w* / (*w*−*z*) ^ *nat* (− *zorder f z*))
using *zorder_exist_pole*[*OF f_holo*,*simplified*,*OF* ‹*is_pole f z*›] **by** *auto*
then have ∗: ∀ *w*∈*ball z r* − {*z*}. *zor_poly f z w* = *f w* ∗ (*w*−*z*) ^ *nat* (− *zorder f z*)
by (*auto simp*: *field_simps*)
have *eventually* (λ*w*. *w* ∈ *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* **have** *isCont* (*zor_poly f z0*) *z0*
by (*auto simp*: *continuous_on_eq_continuous_at*)
hence (*zor_poly f z0* $\longrightarrow$ *zor_poly f z0 z0*) (*at z0*)
unfolding *isCont_def* **.**
ultimately have $((\lambda w.\ f\ w * (w - z0)\ powi - n) \longrightarrow zor_poly\ f\ z0\ z0)$ (*at z0*)
by (*blast intro*: *Lim_transform_eventually*)
hence $((\lambda x.\ f\ (g\ x) * (g\ x - z0)\ powi - n) \longrightarrow zor_poly\ f\ z0\ z0)\ F$
by (*rule filterlim_compose*[*OF _ g*])
from *tendsto_unique*[*OF* ‹*F* $\neq$ *bot*› *this lim*] **show** *?thesis* **.**
qed

lemma *zor_poly_zero_eqI*:
fixes f :: *complex* $\Rightarrow$ *complex* **and** *z0* :: *complex*
defines $n \equiv$ *zorder f z0*
assumes *f holomorphic_on A open A connected A z0* $\in$ *A* $\exists z{\in}A.\ f\ z \neq 0$
assumes *lim*: $((\lambda x.\ f\ (g\ x) / (g\ x - z0)\ \hat{}\ nat\ n) \longrightarrow c)\ F$
assumes *g*: *filterlim g* (*at z0*) *F* **and** $F \neq bot$
shows *zor_poly f z0 z0* $= c$
proof −
from *zorder_exist_zero*[*OF assms*(*2*−*6*)] **obtain** *r* **where**
r: $r > 0$ *cball z0 r* $\subseteq$ *A zor_poly f z0 holomorphic_on cball z0 r*
$\bigwedge w.\ w \in cball\ z0\ r \Longrightarrow f\ w = zor_poly\ f\ z0\ w * (w - z0)\ \hat{}\ nat\ n$
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* $(\lambda w.\ zor_poly\ f\ z0\ w = f\ w / (w - z0)\ \hat{}\ nat\ n)$ (*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*
using *P_def assms holo n_def zorder_exist_pole* **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* = (λ*w*. (*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* (λ*w*. *w* ∈ *ball x r*) (*nhds x*)
using ‹*r* > *0*› **by** (*intro eventually_nhds_in_open*) *auto*
hence *ev''*: *eventually* (λ*w*. *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* (λ*w*. *w* ∈ *ball x r* − {*x*}) (*nhds w*)
using *w* **by** (*intro eventually_nhds_in_open*) *auto*

```
    have §: (deriv P w * (w−x) ^ n − P w * (n * (w−x) ^ (n−1))) / ((w−x) ^ n
* (w−x) ^ n) = D w
      using w n' by (simp add: divide_simps D_def) (simp add: algebra_simps)
    have ((λw. P w / (w−x) ^ n) has_field_derivative D w) (at w)
      by (rule derivative_eq_intros refl | use w § in force)+
    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 (meson * assms(1) holo holomorphic_derivI is_pole_deriv' zorder_deriv')+
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
    thus ?thesis
      using assms is_pole_deriv' by blast
  next
    case (2 c)
    thus ?thesis
      by (meson assms removable_singularity_deriv' tendsto_imp_not_essential)
  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
      using 2 filterlim_compose g(1) is_pole_def by blast
    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)
    by (metis freq non_zero_neighbour not_eventually not_frequently sing)
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 ∗∗ ‹is_pole f x› at_neq_bot is_pole_def
          not_tendsto_and_filterlim_at_infinity by blast
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 r: 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 (metis ‹r>0› r_holo centre_in_ball continuous_on_interior
            holomorphic_on_imp_continuous_on interior_cball isContD)
    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
    using w_Pn ‹r>0› by (force 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]
      by (simp add: fg_def fn_def gn_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
    by eventually_elim auto
  then show False by auto
qed

lemma isolated_pole_imp_neg_zorder:
  assumes isolated_singularity_at f z and is_pole f z
  shows zorder f z < 0
 using analytic_imp_holomorphic assms centre_in_ball isolated_singularity_at_def
zorder_exist_pole by blast


lemma isolated_singularity_at_deriv[singularity_intros]:
  assumes isolated_singularity_at f x
  shows isolated_singularity_at (deriv f) x
  by (meson analytic_deriv assms isolated_singularity_at_def)
```

```
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)
      by (auto simp: algebra_simps simp flip:power_int_add_1' wz_def)
    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
   using isolated_singularity_at_deriv not_essential_deriv zorder_divide by blast
  moreover have zorder (deriv f) z = zorder f z - 1
     using f_iso f_ness f_ord fg_nconst frequently_elim1 zorder_deriv_minus_1
by fastforce
  ultimately show zorder ff z < 0 by auto

  show ∃ 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* (λ*z*. *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 ∃$_F$ *w in at z*. *deriv f w* ∗ *f w* ≠ *0*
using *assms f_iso isolated_pole_imp_nzero_times* **by** *blast*
show *zorder f z* ≠ *0*
using *isolated_pole_imp_neg_zorder assms* **by** *fastforce*
qed

7.13 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*)

lemma *zorder_zero_eqI'*:
assumes *f analytic_on* {*z*}
assumes ⋀*i*. *i* < *nat n* ⟹ (*deriv* ^^ *i*) *f z* = *0*
assumes (*deriv* ^^ *nat n*) *f z* ≠ *0* **and** *n* ≥ *0*
shows *zorder f z* = *n*
proof −
from *assms*(*1*) **obtain** *A* **where** *open A z* ∈ *A f holomorphic_on A*
using *analytic_at* **by** *blast*
thus *?thesis*
using *zorder_zero_eqI*[*of f A z n*] *assms* **by** *blast*
qed

7.14 Isolated zeros

definition *isolated_zero* :: (*'a*::*topological_space* ⇒ *'b*::*real_normed_div_algebra*) ⇒ *'a* ⇒ *bool* **where**
isolated_zero f a ⟷ *f* −*a*→ *0* ∧ *eventually* (λ*x*. *f x* ≠ *0*) (*at a*)

lemma *isolated_zero_shift*:
fixes *z* :: *'a* :: *real_normed_vector*
shows *isolated_zero f z* ⟷ *isolated_zero* (λ*w*. *f* (*z* + *w*)) *0*
unfolding *isolated_zero_def*
by (*simp add*: *at_to_0'* *eventually_filtermap* *filterlim_filtermap* *add_ac*)

lemma *isolated_zero_shift'*:
fixes *z* :: *'a* :: *real_normed_vector*
assumes *NO_MATCH 0 z*
shows *isolated_zero f z* ⟷ *isolated_zero* (λ*w*. *f* (*z* + *w*)) *0*
by (*rule isolated_zero_shift*)

lemma *isolated_zero_imp_not_essential* [*intro*]:
isolated_zero f z ⟹ *not_essential f z*
unfolding *isolated_zero_def not_essential_def*
using *tendsto_nhds_iff* **by** *blast*

lemma *pole_is_not_zero*:
fixes *f*:: *'a*::*perfect_space* ⇒ *'b*::*real_normed_field*
assumes *is_pole f z*
shows ¬*isolated_zero f z*
proof
assume *isolated_zero f z*
then have *filterlim f* (*nhds 0*) (*at z*)
unfolding *isolated_zero_def* **using** *tendsto_nhds_iff* **by** *blast*
moreover have *filterlim f at_infinity* (*at z*)
using ‹*is_pole f z*› **unfolding** *is_pole_def* **.**
ultimately show *False*
using *not_tendsto_and_filterlim_at_infinity*[*OF at_neq_bot*]
by *auto*
qed

lemma *isolated_zero_imp_pole_inverse*:
fixes *f* :: _ ⇒ *'b*::{*real_normed_div_algebra*, *division_ring*}
assumes *isolated_zero f z*
shows *is_pole* (λ*z*. *inverse* (*f z*)) *z*
proof −
from *assms* **have** *ev*: *eventually* (λ*z*. *f z* ≠ *0*) (*at z*)
by (*auto simp*: *isolated_zero_def*)
have *filterlim f* (*nhds 0*) (*at z*)
using *assms* **by** (*simp add*: *isolated_zero_def*)
with *ev* **have** *filterlim f* (*at 0*) (*at z*)
using *filterlim_atI* **by** *blast*
also have *?this* ⟷ *filterlim* (λ*z*. *inverse* (*inverse* (*f z*))) (*at 0*) (*at z*)

```
    by (rule filterlim_cong) (use ev in ‹auto elim!: eventually_mono›)
  finally have filterlim (λz. inverse (f z)) at_infinity (at z)
    by (subst filterlim_inverse_at_iff [symmetric])
  thus ?thesis
    by (simp add: is_pole_def)
qed

lemma is_pole_imp_isolated_zero_inverse:
  fixes f :: _ ⇒ 'b::{real_normed_div_algebra, division_ring}
  assumes is_pole f z
  shows   isolated_zero (λz. inverse (f z)) z
proof −
  from assms have ev: eventually (λz. f z ≠ 0) (at z)
    by (simp add: non_zero_neighbour_pole)
  have filterlim f at_infinity (at z)
    using assms by (simp add: is_pole_def)
  also have ?this ⟷ filterlim (λz. inverse (inverse (f z))) at_infinity (at z)
    by (rule filterlim_cong) (use ev in ‹auto elim!: eventually_mono›)
  finally have filterlim (λz. inverse (f z)) (at 0) (at z)
    by (subst (asm) filterlim_inverse_at_iff [symmetric]) auto
  hence filterlim (λz. inverse (f z)) (nhds 0) (at z)
    using filterlim_at by blast
  moreover have eventually (λz. inverse (f z) ≠ 0) (at z)
    using ev by eventually_elim auto
  ultimately show ?thesis
    by (simp add: isolated_zero_def)
qed

lemma is_pole_inverse_iff: is_pole (λz. inverse (f z)) z ⟷ isolated_zero f z
  using is_pole_imp_isolated_zero_inverse isolated_zero_imp_pole_inverse by
fastforce

lemma isolated_zero_inverse_iff: isolated_zero (λz. inverse (f z)) z ⟷ is_pole
f z
  using is_pole_imp_isolated_zero_inverse isolated_zero_imp_pole_inverse by
fastforce

lemma zero_isolated_zero:
  fixes f :: 'a :: {t2_space, perfect_space} ⇒ _
  assumes isolated_zero f z isCont f z
  shows f z = 0
proof (rule tendsto_unique)
  show f −z→ f z
    using assms(2) by (rule isContD)
  show f −z→ 0
    using assms(1) by (simp add: isolated_zero_def)
qed auto

lemma zero_isolated_zero_analytic:
```

assumes *isolated_zero f z f analytic_on* {*z*}
shows *f z = 0*
using *assms*(*1*) *analytic_at_imp_isCont*[*OF assms*(*2*)] **by** (*rule zero_isolated_zero*)

lemma *isolated_zero_analytic_iff*:
assumes *f analytic_on* {*z*}
shows *isolated_zero f z* ⟷ *f z = 0* ∧ *eventually* (λ*z*. *f z* ≠ *0*) (*at z*)
proof *safe*
assume *f z = 0 eventually* (λ*z*. *f z* ≠ *0*) (*at z*)
with *assms* **show** *isolated_zero f z*
unfolding *isolated_zero_def* **by** (*metis analytic_at_imp_isCont isCont_def*)
qed (*use zero_isolated_zero_analytic*[*OF _ assms*] **in** ‹*auto simp*: *isolated_zero_def*›)

lemma *non_isolated_zero_imp_eventually_zero*:
assumes *f analytic_on* {*z*} *f z = 0* ¬*isolated_zero f z*
shows *eventually* (λ*z*. *f z = 0*) (*at z*)
proof (*rule not_essential_frequently_0_imp_eventually_0*)
from *assms*(*1*) **show** *isolated_singularity_at f z not_essential f z*
by (*simp_all add*: *isolated_singularity_at_analytic not_essential_analytic*)
from *assms*(*1*,*2*) **have** *f* −*z*→ *0*
by (*metis analytic_at_imp_isCont continuous_within*)
thus *frequently* (λ*z*. *f z = 0*) (*at z*)
using *assms*(*2*,*3*) **by** (*auto simp*: *isolated_zero_def frequently_def*)
qed

lemma *non_isolated_zero_imp_eventually_zero′*:
assumes *f analytic_on* {*z*} *f z = 0* ¬*isolated_zero f z*
shows *eventually* (λ*z*. *f z = 0*) (*nhds z*)
using *non_isolated_zero_imp_eventually_zero*[*OF assms*] *assms*(*2*)
using *eventually_nhds_conv_at* **by** *blast*

end
theory *Complex_Residues*
imports *Complex_Singularities*
begin

7.15 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 = λf 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 (λz. 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 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 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* (λ*z*. (*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* (λ*z*. (*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 (λw. 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* $(\lambda z.\ f\ z\ /\ z\ \hat{}\ Suc\ n)\ 0 = fps_nth\ F\ n$
proof −
from *has_fps_expansion_imp_holomorphic*[*OF assms*] **obtain** *s*
where *open s* $0 \in s$ *f holomorphic_on s* $\bigwedge z.\ z \in s \Longrightarrow f\ z = eval_fps\ F\ z$
by *auto*
with *assms* **have** *residue* $(\lambda z.\ f\ z\ /\ (z - 0)\ \hat{}\ Suc\ n)\ 0 = (deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n$
unfolding *has_fps_expansion_def*
by (*intro residue_holomorphic_over_power*[*of s*]) (*auto simp*: *zero_ereal_def*)
also from *assms* **have** $\ldots = fps_nth\ F\ n$
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* $\hat{\hat{}}$ (*n* − *1*)) *h z* / *fact* (*n*−*1*))
proof −
define *g* **where** *g* ≡ $\lambda x.$ *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*) $\hat{}$ *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 holomorphic_on cball z r*
(∀ *w*∈*cball z r* − {*z*}. *f w* = *h w* / (*w* − *z*) $\hat{}$ *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*) $\hat{}$ *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* $\hat{\hat{}}$ (*n* − *1*)) *h z* / *fact* (*n*−*1*))
define *h'* **where** *h'* ≡ $\lambda u.$ *h u* / (*u* − *z*) $\hat{}$ *n*
have (*h' has_contour_integral c* / *fact* (*n* − *1*) ∗ (*deriv* $\hat{\hat{}}$ (*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:∃\<^sub>F w in at z. g w ≠0
  proof (rule ccontr)
    assume ¬ (∃\<^sub>F w in at z. g w ≠ 0)
    then have ∀\<^sub>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 ∀\<^sub>F w in at z. f w ≠0 ∧ w∈s
      apply (rule non_zero_neighbour_alt)
      using assms by auto
    with g_nconst have ∃\<^sub>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 holomor-
phic_on_imp_continuous_on
        by auto
      show f z ≠ 0 by fact
    qed
    show filterlim id (at z) (at z) by (simp add: filterlim_iff)
    have ((λw. (f w ∗ (w − z)) / g w) ⟶ f z / g′) (at z)
    proof (rule lhopital_complex_simple)
      show ((λw. 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 ((λw. (f w / g w) ∗ (w − z)) ⟶ f z / g′) (at z)
      by (simp add: field_split_simps)
  qed
qed
```

7.16 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 (− (ℤ≤0 − {−of_nat n}) :: complex set)
    by (intro open_Compl closed_subset_Ints) auto
  show Gamma holomorphic_on (− (ℤ≤0 − {−of_nat n}) − {− of_nat n})
    by (rule holomorphic_Gamma) auto
  show (λw. Gamma w ∗ (w − (−of_nat n))) −(−of_nat n)→ (− 1) ^ n / fact n
    using Gamma_residues[of n] by simp
qed auto

end
```

8 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* ∈ *O*[*at_top*](λ_. *1*)
assumes *cf*: *continuous_on UNIV f*
shows *bounded* (*range f*)
proof −
obtain *c1 c2*
where *eventually* (λ*x*. *norm* (*f x*) ≤ *c1*) *at_bot eventually* (λ*x*. *norm* (*f x*) ≤ *c2*) *at_top*
using *assms* **by** (*auto elim*!: *landau_o.bigE*)
then obtain *x1 x2* **where** *x1*: ⋀*x*. *x* ≤ *x1* ⟹ *norm* (*f x*) ≤ *c1* **and** *x2*: ⋀*x*. *x* ≥ *x2* ⟹ *norm* (*f x*) ≤ *c2*
by (*auto simp*: *eventually_at_bot_linorder eventually_at_top_linorder*)
have *compact* (*f* ‘ {*x1*..*x2*})
by (*intro compact_continuous_image continuous_on_subset*[*OF cf*]) *auto*
hence *bounded* (*f* ‘ {*x1*..*x2*})
by (*rule compact_imp_bounded*)
then obtain *c3* **where** *c3*: ⋀*x*. *x* ∈ {*x1*..*x2*} ⟹ *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* ξ](λ_. *1*)
shows (∃*g*. *g holomorphic_on S* ∧ (∀*z*∈*S* − {ξ}. *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 ⋀*z*. *z* ∈ *X* ⟹ *f* ∈ *O*[*at z*](λ_. *1*)
shows ∃*g*. *g holomorphic_on S* ∧ (∀*z*∈*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 ∃ *g*. *g holomorphic_on* (*S* − *X*) ∧ (∀ *z*∈*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*) ∀ *z*∈*S* − *X* − {*z0*}. *g z* = *f z* **by** *blast*
have ∃ *h*. *h holomorphic_on S* ∧ (∀ *z*∈*S* − *X*. *h z* = *g z*)
proof (*rule insert.IH*)
fix *z0′* **assume** *z0′*: *z0′* ∈ *X*
hence *eventually* (λ*z*. *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* (λ*z*. *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 (*use insert.prems g* **in** *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** ((λ*x*. *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 (λ*x*. *f x* − *f x0*) ∈ *O*[*at x0 within A*](λ*x*. *x* − *x0*)
proof −
from *assms* **obtain** *f′* **where** (*f has_field_derivative f′*) (*at x0 within A*)
by (*auto simp*: *field_differentiable_def*)
hence ((λ*x*. (*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

8.1 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 *closed_segment a a′ ⊆ S − insert p pts*
using *e idt.prems*(*6*) **by** *auto*

```
    then have continuous_on (closed_segment a a') f
    using holomorphic_on_imp_continuous_on holomorphic_on_subset idt.prems(5)
by presburger
    then show ?thesis
      using contour_integrable_continuous_linepath by (simp add: g_def)
  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
           by (intro winding_number_zero_outside) auto
         qed
        ultimately show ?thesis
         unfolding n_Suc using Suc.hyps pcirc
       by (metis add.right_neutral not_in_path_image_join that valid_path_imp_path
winding_number_join)
       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
      by (simp add: Rings.ring_distribs(2) Suc.hyps n_Suc pcirc pcirc_integrable)
   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'
     proof −
      have winding_number (reversepath (n_circ (nat n))) p' = 0
       using n_circ that
     by (metis add.inverse_neutral valid_path_imp_path winding_number_reversepath)
      then show ?thesis
       using cp_def n_circ(7) that by force
```

```
    qed
  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)
          by (simp add: g'_def s'_def subset_path_image_join)
      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 z_notin_cp: z ∉ path_image cp
          using cp(6) cp_def n_circ(6) z by auto
        have z_notin_pg: z ∉ path_image pg
          by (metis Diff_iff Diff_insert2 pg(4) subsetD z)
          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))
         by (simp add: valid_path_imp_path winding_number_join z_notin_cp
z_notin_pg)
        also have ... = winding_number g z + winding_number cp z
            by (simp add: valid_path_imp_path winding_number_reversepath
z_notin_pg)
          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 ∗: f contour_integrable_on g f contour_integrable_on pg f contour_integrable_on
cp
       by (auto simp add: open_Diff[OF ‹open S›,OF finite_imp_closed[OF fin]]
           intro!: contour_integrable_holomorphic_simple[OF holo _ _ path_img])
     have contour_integral g' f = contour_integral g f + contour_integral pg f
       + contour_integral cp f + contour_integral (reversepath pg) f
      using ∗ by (simp add: g'_def contour_integrable_reversepath_eq)
     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 −
     obtain [simp]: p' ∉ path_image g p' ∉ path_image pg p'∉path_image cp
       using 2.prems(8) that by (metis Diff_iff Diff_insert2 ‹p' ∈ pts› cp(4)
pg(4) subsetD)
     have winding_number g' p' = winding_number g p' + winding_number pg p'
       + winding_number (cp +++ reversepath pg) p'
      by (simp add: g'_def not_in_path_image_join valid_path_imp_path wind-
ing_number_join)
     also have ... = winding_number g p' using that
     by (simp add: valid_path_imp_path winding_number_join winding_number_reversepath)
     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: 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 ≡ λp. 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
   by (metis ‹pts2 ∩ S = {}› circ_def disjoint_iff_not_equal homo mult_zero_left
sum.neutral)
  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
    by simp
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
```

8.2 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'* ≡ λ*w*. *pp w* ∗ (*w* − *p*) *powi po*
define *ff'* **where** *ff'* ≡ (λ*x*. *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 *ball p e1* − {*p*} ⊆ *S* − *poles*
using *avoid_def e1_avoid pz_def* **by** *fastforce*
then have *f holomorphic_on ball p e1* − {*p*}
by (*intro holomorphic_on_subset*[*OF f_holo*])
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*] *continuous_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 f'_def ff'_def prin_def anal_def
        apply (simp add: field_simps flip: wp_def)
        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
          using ‹r<e1› by force
        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
     using ‹p ∈ S› ‹p ∈ pz› cont_def contour_integral_unique get_e by fastforce
```

```
    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
```

8.3 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 ⇒ complex and n :: nat and r :: real
 defines g ≡ (λw. f w / w ^ Suc n) and γ ≡ circlepath 0 r
 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
 shows   fps_coeff_conv_residues:
         (deriv ^^ n) f 0 / fact n =
           contour_integral γ g / (2 * pi * i) − (∑ z∈S. residue g z) (is ?thesis1)
   and   fps_coeff_residues_bound:
         (⋀z. norm z = r ⟹ z ∉ k ⟹ norm (f z) ≤ C) ⟹ C ≥ 0 ⟹ finite
k ⟹
           norm ((deriv ^^ n) f 0 / fact n + (∑ z∈S. residue g z)) ≤ C / r ^ n
proof −
 have holo: g holomorphic_on A − insert 0 S
  unfolding g_def using assms by (auto intro!: holomorphic_intros)
 have contour_integral γ g = 2 * pi * i * (∑ z∈insert 0 S. winding_number γ
z * residue g z)
 proof (rule Residue_theorem)
  show g holomorphic_on A − insert 0 S by fact
  from assms show ∀ z. z ∉ A ⟶ winding_number γ 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 (∑z∈insert 0 S. winding_number γ z * residue g z) = (∑z∈insert 0 S.
residue g z)
    by (intro sum.cong) simp_all
  also have ... = residue g 0 + (∑z∈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 ^^ 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: ⋀z. norm z = r ⟹ z ∉ k ⟹ norm (f z) ≤ C C ≥ 0 and k: finite
k
  have (deriv ^^ n) f 0 / fact n + (∑z∈S. residue g z) = contour_integral γ g /
(2 * pi * i)
    using ‹?thesis1› by (simp add: algebra_simps)
  also have norm ... = norm (contour_integral γ g) / (2 * pi)
    by (simp add: norm_divide norm_mult)
  also have norm (contour_integral γ g) ≤ C / r ^ 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 $f :: complex \Rightarrow complex$ **and** $r :: real$
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* $\notin S$
assumes *g*: *eventually* $(\lambda n.\ g\ n = -(\sum z{\in}S.\ residue\ (\lambda z.\ f\ z\ /\ z\ \hat{}\ Suc\ n)\ z))$ *sequentially*
(**is** *eventually* $(\lambda n.\ _ = -?g'\ n)$ _)
shows $(\lambda n.\ (deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n - g\ n) \in O(\lambda n.\ 1\ /\ r\ \hat{}\ n)$ (**is** $(\lambda n.\ ?c\ n - _) \in O(_)$)
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: $\bigwedge z.\ z \in sphere\ 0\ r \Longrightarrow norm\ (f\ z) \leq C$
by (*auto simp*: *bounded_iff sphere_def*)
have $0 \leq norm\ (f\ (of_real\ r))$ **by** *simp*
also from C[*of of_real r*] **and** ‹$r > 0$› **have** $\ldots \leq C$ **by** *simp*
finally have *C_nonneg*: $C \geq 0$ **.**

have $(\lambda n.\ ?c\ n + ?g'\ n) \in O(\lambda n.\ of_real\ (1\ /\ r\ \hat{}\ n))$
proof (*intro bigoI*[*of _ C*] *always_eventually allI*)
fix $n :: nat$
from *assms* **and** C **and** *C_nonneg* **have** $norm\ (?c\ n + ?g'\ n) \leq C\ /\ r\ \hat{}\ n$
by (*intro fps_coeff_residues_bound*[**where** $A = A$ **and** $k = \{\}$]) *auto*
also have $\ldots = C * norm\ (complex_of_real\ (1\ /\ r\ \hat{}\ n))$
using ‹$r > 0$› **by** (*simp add*: *norm_divide norm_power*)
finally show $norm\ (?c\ n + ?g'\ n) \leq \ldots$ **.**
qed
also have *?this* $\longleftrightarrow$ $(\lambda n.\ ?c\ n - g\ n) \in O(\lambda n.\ of_real\ (1\ /\ r\ \hat{}\ n))$
by (*intro landau_o.big.in_cong eventually_mono*[*OF g*]) *simp_all*
finally show *?thesis* **.**
qed

corollary *fps_coeff_residues_bigo′*:
fixes $f :: complex \Rightarrow complex$ **and** $r :: real$
assumes *exp*: *f has_fps_expansion F*
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* $\notin S$
assumes *eventually* $(\lambda n.\ g\ n = -(\sum z{\in}S.\ residue\ (\lambda z.\ f\ z\ /\ z\ \hat{}\ Suc\ n)\ z))$ *sequentially*
(**is** *eventually* $(\lambda n.\ _ = -?g'\ n)$ _)
shows $(\lambda n.\ fps_nth\ F\ n - g\ n) \in O(\lambda n.\ 1\ /\ r\ \hat{}\ n)$ (**is** $(\lambda n.\ ?c\ n - _) \in O(_)$)
proof −
have $fps_nth\ F = (\lambda n.\ (deriv\ \hat{\hat{}}\ n)\ f\ 0\ /\ fact\ n)$
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

8.4 Rouche's theorem

theorem *Rouche_theorem*:
fixes *f g*::*complex* ⇒ *complex* **and** *s*:: *complex set*
defines *fg*≡(λ*p*. *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* ≡ λ*p*. *winding_number* γ *p*
define *c* **where** *c* ≡ *2* ∗ *complex_of_real pi* ∗ i
define *h* **where** *h* ≡ λ*p*. *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*:((λ*x*. *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
    by (force intro!: has_integral_spike_eq[OF negligible_finite, OF ‹finite spikes›])
    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 cmod (g p/f p) = 1
        by (simp add: add_eq_0_iff2 h_def)
      then show False
        by (smt (verit) divide_eq_1_iff norm_divide path_less that)
    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
      using ‹h p≠0› ‹f p≠0› ‹fg p≠0›
     unfolding der_fg der_h
      apply (simp add: divide_simps h_def fg_def)
    by (simp add: mult.commute mult.left_commute ring_class.ring_distribs(1))
  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 holomorphic_on_add by auto
  moreover
  have path_image γ ⊆ s − {p∈s. fg p = 0}
    using path_fg unfolding zeros_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


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 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*]: $F \neq 0$ $G \neq 0$
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* ... $\geq$ *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* $(F \hat{} n) \geq$ *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 ≥ 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 eval_fls_eq:
  assumes N ≤ fls_subdegree F fls_subdegree F ≥ 0 ∨ z ≠ 0
  assumes (λn. fls_nth F (int n + N) * z powi (int n + N)) sums S
  shows   eval_fls F z = S
proof (cases z = 0)
  case [simp]: True
  have (λn. fls_nth F (int n + N) * z powi (int n + N)) =
        (λn. if n ∈ (if N ≤ 0 then {nat (−N)} else {}) then fls_nth F (int n + N)
else 0)
    by (auto simp: fun_eq_iff split: if_splits)
  also have ... sums (∑ n∈(if N ≤ 0 then {nat (−N)} else {}). fls_nth F (int n
+ N))
    by (rule sums_If_finite_set) auto
  also have ... = fls_nth F 0
    using assms by auto
  also have ... = eval_fls F z
   using assms by (auto simp: eval_fls_def eval_fps_at_0 power_int_0_left_if)
  finally show ?thesis
    using assms by (simp add: sums_iff)
next
  case [simp]: False
  define N' where N' = fls_subdegree F
  define d where d = nat (N' − N)

  have (λn. fls_nth F (int n + N) * z powi (int n + N)) sums S
    by fact
  also have ?this ⟷ (λn. fls_nth F (int (n+d) + N) * z powi (int (n+d) +
N)) sums S
    by (rule sums_zero_iff_shift [symmetric]) (use assms in ‹auto simp: d_def
N'_def›)
  also have (λn. int (n+d) + N) = (λn. int n + N')
    using assms by (auto simp: N'_def d_def)
  finally have (λn. fls_nth F (int n + N') * z powi (int n + N')) sums S .
  hence (λn. z powi (−N') * (fls_nth F (int n + N') * z powi (int n + N')))
sums (z powi (−N') * S)
    by (intro sums_mult)
  hence (λn. fls_nth F (int n + N') * z ^ n) sums (z powi (−N') * S)
    by (simp add: power_int_add power_int_minus field_simps)
  thus ?thesis
    by (simp add: eval_fls_def eval_fps_def sums_iff power_int_minus N'_def)
qed

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   (λx. eval_fls f (g x)) holomorphic_on A
  by (meson assms holomorphic_on_compose holomorphic_on_eval_fls holomor-
phic_transform o_def)

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)
   using assms holomorphic_on_eval_fls holomorphic_on_imp_continuous_on
by blast

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 (λx. eval_fls f (g x))
  by (metis assms continuous_on_compose2 continuous_on_eval_fls order.refl)

lemmas has_field_derivative_eval_fps' [derivative_intros] =
  DERIV_chain2[OF has_field_derivative_eval_fps]


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 (metis DERIV_imp_deriv assms has_field_derivative_eval_fls)

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 *(λx. 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*] *(λz. 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 ... = (λ*z*. *eval_fps G z* ∗ *z powi n*)
by (*intro ext*) (*simp_all add*: *eval_fls_def G_def n_def*)
also have ... ∼[*at 0*] (λ*z*. *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*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
defines *n* ≡ *fls_subdegree F*
shows *f* ∼[*at z*] (λ*w*. *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 (λ*z*. *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 (λ*z*. *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*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F* **and** *fls_subdegree F* ≥ *0*
shows *f* −*z*→ *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 ≠ 0
    using assms(2) by auto
  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 filterlim (λz. 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:¬ fls_subdegree F < 0
  define ff where ff=(λz. fls_nth F (fls_subdegree F)
                          * z powi fls_subdegree F)

  have (ff ⟶ (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 ∼[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: (λw. 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*
proof −
have *is_pole* (λ*x*. *f* (*z* + *x*)) *0*
using *assms*(*2*) *is_pole_shift_0* **by** *blast*
then show *?thesis*
using *F is_pole_0_imp_neg_fls_subdegree* **by** *blast*
qed

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 *has_laurent_expansion_analytic_imp_fls_nth_0_eq*:
assumes *f analytic_on* {*z*} (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
shows *fls_nth F 0* = *f z*
proof (*rule tendsto_unique*)
from *assms* **have** *fls_subdegree F* ≥ *0*
using *analytic_at_imp_no_pole is_pole_fls_subdegree_iff not_le* **by** *blast*
thus *f* −*z*→ *fls_nth F 0*
by (*intro has_laurent_expansion_imp_tendsto*) (*use assms* **in** *auto*)
show *f* −*z*→ *f z*
by (*intro analytic_at_imp_isCont isContD assms*)
qed *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'*: $\forall_F\ z$ *in at 0*. *eval_fps F z* = *f z*
using *eventually_at_filter eventually_mono* **by** *fastforce*
moreover have *eventually* ($\lambda z.\ z \in$ *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* ($\lambda z.$ *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* $\longleftrightarrow$ *eventually* ($\lambda z.$ *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*]:
($\lambda x.\ x\ \hat{}\ n$) *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*]:
($\lambda x.\ x$ *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*]:
($\lambda x.\ x$) *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 ($\lambda x.\ c * f\ x$) *has_laurent_expansion fls_const c* * *F*
proof −
from *assms* **have** *eventually* ($\lambda z.\ z \in$ *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 (λz. eval_fls F z = f z) (at 0)
    by (auto simp: has_laurent_expansion_def)
  ultimately have eventually (λz. 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 (λx. eval_fls F x = f x) (at 0)
           and eventually (λx. eval_fls G x = g x) (at 0)
    using assms by (auto simp: has_laurent_expansion_def)
  ultimately have eventually (λx. 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   (λx. f x − g x) has_laurent_expansion F − G
  using has_laurent_expansion_add[of f F λx. − 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   (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_mult)
  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 (λx. eval_fls F x = f x) (at 0)
           and eventually (λx. eval_fls G x = g x) (at 0)
    using assms by (auto simp: has_laurent_expansion_def)
  ultimately have eventually (λx. 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 ⟹ (λx. 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 $(\lambda x.\ f\ x\ \hat{}\ n)$ *has_laurent_expansion* $F\ \hat{}\ n$
by (*induction n*) (*auto intro*!: *laurent_expansion_intros assms*)

lemma *has_laurent_expansion_sum* [*laurent_expansion_intros*]:
assumes $\bigwedge x.\ x \in I \Longrightarrow f\ x$ *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 $(\lambda x.\ f\ x * x\ powi\ (-n))\ has_laurent_expansion\ (fls_shift\ n\ F)$
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* $0 \in A$ $\bigwedge x.\ x \in A - \{0\} \Longrightarrow (f\ has_field_derivative\ f'\ x)\ (at\ x)$
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* $\longleftrightarrow$ *?thesis*
proof (*intro has_laurent_expansion_cong refl*)
have *eventually* $(\lambda z.\ z \in A - \{0\})\ (at\ 0)$
by (*intro eventually_at_in_open assms*)
thus *eventually* $(\lambda z.\ deriv\ f\ z = f'\ z)\ (at\ 0)$
by *eventually_elim* (*auto intro*!: *DERIV_imp_deriv assms*)
qed
finally show *?thesis* **.**
qed

definition *laurent_expansion* :: $(complex \Rightarrow complex) \Rightarrow complex \Rightarrow complex\ fls$
where
laurent_expansion f z =
(*if eventually* $(\lambda z.\ f\ z = 0)\ (at\ z)$ *then 0*
else fls_shift $(-zorder\ f\ z)$ (*fps_to_fls* (*fps_expansion* (*zor_poly f z*) *z*)))

lemma *laurent_expansion_cong*:
assumes *eventually* $(\lambda w.\ f\ w = g\ w)\ (at\ z)$ $z = z'$
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* $\exists_F$ *w in at 0. f w* $\neq$ *0*)
case *False*
have (*λ_. 0*) *has_laurent_expansion 0*
by *simp*
also have *?this* $\longleftrightarrow$ *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* $\neq$ *0 zor_poly f 0 holomorphic_on cball 0 r r* > *0*
$\forall$ *w*$\in$*cball 0 r* − {*0*}. *f w* = *zor_poly f 0 w* ∗ *w powi n* ∧ *zor_poly f 0 w* $\neq$ *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* $\longleftrightarrow$ *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 ∃\<^sub>F w in at z. f w ≠ 0)
    case False
    then have ∀\<^sub>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 ‹∀\<^sub>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 o g) z
    proof −
      have ∃\<^sub>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 ∀\<^sub>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 ∼[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: (λx. 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 (λw. 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 (λw. (w − z) powi fls_subdegree F) z
    by simp
  finally have pole': ... .
```

```
  have False if fls_subdegree F ≥ 0
  proof −
    have (λw. (w − z) powi fls_subdegree F) holomorphic_on UNIV
      using that by (intro holomorphic_intros) auto
    hence ¬is_pole (λw. (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 analytic_at_imp_has_fps_expansion:
  assumes f analytic_on {z}
  shows   (λx. f (z + x)) has_fps_expansion fps_expansion f z
proof −
  have f ∘ (λx. z + x) analytic_on {0}
    by (intro analytic_on_compose_gen[OF _ assms] analytic_intros) auto
  hence (f ∘ (λx. z + x)) has_fps_expansion fps_expansion (f ∘ (λx. 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 (λw. 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 ≠ 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 ≠ 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 $(\lambda_.\ 0)$ *has_laurent_expansion F*
shows $F = 0$
proof −
have *at* $(0 :: complex) \neq bot$
by *auto*
moreover have $(\lambda z.\ if\ z = 0\ then\ fls_nth\ F\ (fls_subdegree\ F)\ else\ 0)$ *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 $?f\ -0\rightarrow\ fls_nth\ F\ (fls_subdegree\ F)$
by (*auto simp*: *isCont_def*)
moreover have $?f\ -0\rightarrow\ 0 \longleftrightarrow (\lambda_::complex.\ 0 :: complex)\ -0\rightarrow\ 0$
by (*intro filterlim_cong*) (*auto simp*: *eventually_at_filter*)
hence $?f\ -0\rightarrow\ 0$
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** $(\lambda x.\ f\ x - f\ x)$ *has_laurent_expansion* $F - G$
by (*intro laurent_expansion_intros*)
hence $(\lambda_.\ 0)$ *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 $(\lambda x.\ f\ (z + x))$ *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* (λ*x*. *f x* ≠ *0*) (*at 0*)
proof (*rule ccontr*)
assume ¬*eventually* (λ*x*. *f x* ≠ *0*) (*at 0*)
with *assms* **have** *eventually* (λ*x*. *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* (λ*x*. *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 (λ*w*. *f* (*z*+*w*)) *has_laurent_expansion F*
shows *eventually* (λ*x*. *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 (λ*x*. *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* (λ*z*. *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 (λz. 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 (λz. 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 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 *holomorphic_on_imp_fps_conv_radius_ge*:
assumes *f has_fps_expansion F f holomorphic_on eball 0 r*
shows *fps_conv_radius F ≥ r*
proof −
define *n* **where** *n = subdegree F*
have *fps_conv_radius* (*fps_expansion f 0*) *≥ r*
by (*intro conv_radius_fps_expansion assms*)
also have *fps_expansion f 0 = F*
using *assms* **by** (*intro fps_expansion_eqI*)
finally show *?thesis*
by *simp*
qed

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 *has_fps_expansion_imp_sums_complex*:
fixes *F* :: *complex fps*
assumes *f has_fps_expansion F f holomorphic_on eball 0 r ereal* (*norm z*) *< r*
shows (*λn. fps_nth F n ∗ z ^ n*) *sums f z*
proof −
have *r*: *fps_conv_radius F ≥ r*
using *assms*(*1*,*2*) **by** (*rule holomorphic_on_imp_fps_conv_radius_ge*)
from *assms* **obtain** *R* **where** *R*: *norm z < R ereal R < r*
using *ereal_dense2 less_ereal.simps*(*1*) **by** *blast*
have *z*: *norm z < fps_conv_radius F*
using *r R assms*(*3*) **by** *order*

```
  have summable (λn. fps_nth F n * z ^ n)
    by (rule summable_fps) (use z in auto)
  moreover have eval_fps F z = f z
  proof (rule has_fps_expansion_imp_eval_fps_eq[where r = R])
    have *: ereal (norm z) < r if norm z < R for z :: complex
      using that R ereal_le_less less_imp_le by blast
    show f holomorphic_on ball 0 R
      using assms(2) by (rule holomorphic_on_subset) (use * in auto)
  qed (use R assms(1) in auto)
  ultimately show ?thesis
    unfolding eval_fps_def sums_iff 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 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) * i * 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) * i * 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: $(\lambda w.\ f\ (z+w))$ *has_laurent_expansion* F **and** $F \neq 0$
shows $(f\ -z\to\ 0) \longleftrightarrow$ *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** $F \neq 0$
shows *is_pole* $(\lambda x.\ deriv\ f\ x\ /\ f\ x)\ 0 \longleftrightarrow$ *is_pole* $f\ 0 \vee (f\ -0\to\ 0)$
proof −
have $(\lambda x.\ deriv\ f\ x\ /\ f\ x)$ *has_laurent_expansion fls_deriv* F / F
using F **by** (*auto intro*:*laurent_expansion_intros*)

have *is_pole* $(\lambda x.\ deriv\ f\ x\ /\ f\ x)\ 0 \longleftrightarrow$
fls_subdegree (*fls_deriv* F / F) < 0
apply (*rule is_pole_fls_subdegree_iff*)
using F **by** (*auto intro*:*laurent_expansion_intros*)
also have ... $\longleftrightarrow$ *is_pole* $f\ 0 \vee (f\ -0\to 0)$
proof (*cases fls_subdegree* $F = 0$)
case *True*
then have *fls_subdegree* (*fls_deriv* F / F) ≥ 0
by (*metis diff_zero div_0* ‹$F\neq 0$› *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 ¬ $(f\ -0\to 0)$
using *tendsto_0_subdegree_iff_0*[*OF F* ‹$F\neq 0$›] *True* **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
then have *fls_deriv* $F \neq 0$
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 _* ‹$F\neq 0$›])
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 $f\ -0\to\ 0 = (0 <$ *fls_subdegree* $F)$
using *tendsto_0_subdegree_iff_0*[*OF F* ‹$F \neq 0$›] .
moreover have *is_pole* $f\ 0 = (\mathit{fls_subdegree}\ F < 0)$
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*:(λ*w*. *f* (*z*+*w*)) *has_laurent_expansion F* **and** *F*≠*0*
  shows *is_pole* (λ*x*. *deriv f x* / *f x*) *z* ⟷ *is_pole f z* ∨ (*f* −*z*→*0*)
proof −
  define *ff df* **where** *ff*=(λ*w*. *f* (*z*+*w*)) **and** *df*=(λ*w*. *deriv f* (*z* + *w*))
  have *is_pole* (λ*x*. *deriv f x* / *f x*) *z*
        ⟷ *is_pole* (λ*x*. *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* (λ*x*. *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* (λ*z*. *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*:(λ*w*. *f* (*z*+*w*)) *has_laurent_expansion F*
      and *fls_subdegree F*≠*0*
  shows *eventually* (λ*w*. *deriv f w* ≠ *0*) (*at z*)
proof −
  have ∀ *F x in at 0*. *deriv* (λ*w*. *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


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 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 zorder_const [simp]: c ≠ 0 ⟹ zorder (λ_. c) z = 0
  by (intro zorder_eqI[where S = UNIV]) auto

lemma zorder_prod_analytic:
  assumes ⋀x. x ∈ A ⟹ f x analytic_on {z}
  assumes eventually (λz. (∏x∈A. f x z) ≠ 0) (at z)
  shows   zorder (λz. ∏x∈A. f x z) z = (∑x∈A. zorder (f x) z)
  using assms
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
  have zorder (λz. f x z * (∏x∈A. f x z)) z = zorder (f x) z + zorder (λz. ∏x∈A.
f x z) z
    using insert.prems insert.hyps by (intro zorder_times_analytic analytic_intros)
auto
  also have zorder (λz. ∏x∈A. f x z) z = (∑x∈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 ≠ 0
```

shows *zorder g z = 0*
using *analytic_at assms zorder_eqI* **by** *fastforce*

lemma *zorder_pos_iff*:
assumes *f holomorphic_on A open A z ∈ A frequently* (λz. *f z ≠ 0*) (*at z*)
shows *zorder f z > 0 ⟷ f z = 0*
proof −
have *f analytic_on* {*z*}
using *assms analytic_at* **by** *blast*
hence ∗: (λw. *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 ≠ 0*
proof
assume *fps_expansion f z = 0*
hence *eventually* (λz. *f z = 0*) (*nhds z*)
using ∗ **by** (*auto simp*: *has_fps_expansion_def nhds_to_0′ eventually_filtermap add_ac*)
hence *eventually* (λz. *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* (λz. *f z ≠ 0*) (*at z*)
shows *zorder f z > 0 ⟷ f z = 0*
using *analytic_at assms zorder_pos_iff* **by** *blast*

lemma *zorder_ge_0*:
assumes *f analytic_on* {*z*} *frequently* (λz. *f z ≠ 0*) (*at z*)
shows *zorder f z ≥ 0*
proof −
have ∗: (λw. *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 (λw. f w ≠ 0) (at z)
  shows   zorder f z = 0 ⟷ f z ≠ 0
  using assms zorder_eq_0I zorder_pos_iff' by fastforce

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 zorder_compose_aux:
  assumes isolated_singularity_at f 0 not_essential f 0
  assumes G: g has_fps_expansion G G ≠ 0 g 0 = 0
  assumes eventually (λw. f w ≠ 0) (at 0)
  shows   zorder (f ∘ 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 ≠ 0› ‹g 0 = 0›
  have [simp]: F ≠ 0
    using has_laurent_expansion_eventually_nonzero_iff[of f 0 F] F assms by
simp
  have FG: (f ∘ 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 ∘ 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 ... = 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: $(\lambda x.\ g\ (z + x) - g\ z)$ *has_fps_expansion* G $G \neq 0$
assumes *eventually* $(\lambda w.\ f\ w \neq 0)$ $(at\ (g\ z))$
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* $(\lambda x.\ g\ x - g\ z)\ z =$ *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 $\ldots =$ *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* $c \longleftrightarrow F =$ *fps_const* c
using *fps_to_fls_eq_iff* **by** *fastforce*

lemma *zorder_compose'*:
assumes *isolated_singularity_at* $f\ (g\ z)$ *not_essential* $f\ (g\ z)$
assumes g *analytic_on* $\{z\}$
assumes *eventually* $(\lambda w.\ f\ w \neq 0)$ $(at\ (g\ z))$
assumes *eventually* $(\lambda w.\ g\ w \neq g\ z)$ $(at\ z)$
shows *zorder* $(f \circ g)\ z =$ *zorder* $f\ (g\ z) *$ *zorder* $(\lambda x.\ g\ x - g\ z)\ z$
proof −
obtain G **where** G [*fps_expansion_intros*]: $(\lambda x.\ g\ (z + x))$ *has_fps_expansion* G
using *assms analytic_at_imp_has_fps_expansion* **by** *blast*
have G': $(\lambda x.\ g\ (z + x) - g\ z)$ *has_fps_expansion* $G -$ *fps_const* $(g\ z)$
by (*intro fps_expansion_intros*)
hence G'': $(\lambda x.\ g\ (z + x) - g\ z)$ *has_laurent_expansion fps_to_fls* $(G -$ *fps_const* $(g\ z))$
using *has_laurent_expansion_fps* **by** *blast*
have *nz*: $G -$ *fps_const* $(g\ z) \neq 0$
using *has_laurent_expansion_eventually_nonzero_iff*[*OF* G''] *assms* **by** *auto*
have *zorder* $(f \circ g)\ z =$ *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*]:
(λz. *tan* (*c* ∗ *z*)) *has_laurent_expansion fps_to_fls* (*fps_tan c*)
by (*intro has_laurent_expansion_fps has_fps_expansion_tan*)

8.5 More Laurent expansions

lemma *has_laurent_expansion_frequently_zero_iff*:
assumes (λw. *f* (*z* + *w*)) *has_laurent_expansion F*
shows *frequently* (λz. *f z* = *0*) (*at z*) ⟷ *F* = *0*
using *assms* **by** (*simp add*: *frequently_def has_laurent_expansion_eventually_nonzero_iff*)

lemma *has_laurent_expansion_eventually_zero_iff*:
assumes (λw. *f* (*z* + *w*)) *has_laurent_expansion F*
shows *eventually* (λz. *f z* = *0*) (*at z*) ⟷ *F* = *0*
using *assms*
by (*metis has_laurent_expansion_frequently_zero_iff has_laurent_expansion_isolated*

has_laurent_expansion_not_essential laurent_expansion_def
not_essential_frequently_0_imp_eventually_0 not_essential_has_laurent_expansion)

lemma *has_laurent_expansion_frequently_nonzero_iff*:
assumes (λw. *f* (*z* + *w*)) *has_laurent_expansion F*
shows *frequently* (λz. *f z* ≠ *0*) (*at z*) ⟷ *F* ≠ *0*
using *assms* **by** (*metis has_laurent_expansion_eventually_zero_iff not_eventually*)

lemma *has_laurent_expansion_sum_list* [*laurent_expansion_intros*]:
assumes ⋀*x*. *x* ∈ *set xs* ⟹ *f x has_laurent_expansion F x*
shows (λy. ∑ *x*←*xs*. *f x y*) *has_laurent_expansion* (∑ *x*←*xs*. *F x*)
using *assms* **by** (*induction xs*) (*auto intro*!: *laurent_expansion_intros*)

lemma *has_laurent_expansion_prod_list* [*laurent_expansion_intros*]:
assumes ⋀*x*. *x* ∈ *set xs* ⟹ *f x has_laurent_expansion F x*
shows (λy. ∏ *x*←*xs*. *f x y*) *has_laurent_expansion* (∏ *x*←*xs*. *F x*)
using *assms* **by** (*induction xs*) (*auto intro*!: *laurent_expansion_intros*)

lemma *has_laurent_expansion_sum_mset* [*laurent_expansion_intros*]:
assumes ⋀*x*. *x* ∈# *I* ⟹ *f x has_laurent_expansion F x*
shows (λy. ∑ *x*∈#*I*. *f x y*) *has_laurent_expansion* (∑ *x*∈#*I*. *F x*)
using *assms* **by** (*induction I*) (*auto intro*!: *laurent_expansion_intros*)

lemma *has_laurent_expansion_prod_mset* [*laurent_expansion_intros*]:
assumes ⋀*x*. *x* ∈# *I* ⟹ *f x has_laurent_expansion F x*
shows (λy. ∏ *x*∈#*I*. *f x y*) *has_laurent_expansion* (∏ *x*∈#*I*. *F x*)
using *assms* **by** (*induction I*) (*auto intro*!: *laurent_expansion_intros*)

8.6 Formal convergence versus analytic convergence

The convergence of a sequence of formal power series and the convergence of the functions in the complex plane do not imply each other:

- If we have the sequence of constant power series $(1/n)_{n\geq 0}$, this clearly converges to the zero function analytically, but as a series of formal power series it is divergent (since the 0-th coefficient never stabilises).
- Conversely, the sequence of series $(n!x^n)_{n\geq 0}$ converges formally to 0, but the corresponding sequence of functions diverges for every $x \neq 0$.

  However, if the sequence of series converges to some limit series h and the corresponding series of functions converges uniformly to some limit function $g(x)$, then h is also a series expansion of $g(x)$, i.e. in that case, formal and analytic convergence agree.

proposition *uniform_limit_imp_fps_expansion_eq*:
fixes *f* :: *'a* ⇒ *complex fps*
assumes *lim1*: (*f* ⟶ *h*) *F*
assumes *lim2*: *uniform_limit A* (λ*x z*. *f' x z*) *g' F*
assumes *expansions*: *eventually* (λ*x*. *f' x has_fps_expansion f x*) *F g' has_fps_expansion g*
assumes *holo*: *eventually* (λ*x*. *f' x holomorphic_on A*) *F*
assumes *A*: *open A 0* ∈ *A*
assumes *nontriv* [*simp*]: *F* ≠ *bot*
shows *g* = *h*
proof (*rule fps_ext*)
fix *n* :: *nat*
have *eventually* (λ*x*. *fps_nth* (*f x*) *n* = *fps_nth h n*) *F*
using *lim1* **unfolding** *tendsto_fps_iff* **by** *blast*
hence *eventually* (λ*x*. (*deriv* ^^ *n*) (*f' x*) *0* / *fact n* = *fps_nth h n*) *F*
using *expansions*(*1*)
proof *eventually_elim*
case (*elim x*)
have *fps_nth* (*f x*) *n* = (*deriv* ^^ *n*) (*f' x*) *0* / *fact n*
by (*rule fps_nth_fps_expansion*) (*use elim* **in** *auto*)
with *elim* **show** *?case*
by *simp*
qed
hence ((λ*x*. (*deriv* ^^ *n*) (*f' x*) *0* / *fact n*) ⟶ *fps_nth h n*) *F*
by (*simp add*: *tendsto_eventually*)

moreover have ((λ*x*. (*deriv* ^^ *n*) (*f' x*) *0*) ⟶ (*deriv* ^^ *n*) *g' 0*) *F*
using *lim2*
proof (*rule higher_deriv_complex_uniform_limit*)
show *eventually* (λ*x*. *f' x holomorphic_on A*) *F*
using *holo* **by** *eventually_elim auto*
qed (*use A* **in** *auto*)

hence $((\lambda x.\ (deriv\ \widehat{\ }\widehat{\ }\ n)\ (f'\ x)\ 0\ /\ fact\ n) \longrightarrow (deriv\ \widehat{\ }\widehat{\ }\ n)\ g'\ 0\ /\ fact\ n)\ F$
by (*intro tendsto_divide*) *auto*

ultimately have $fps_nth\ h\ n = (deriv\ \widehat{\ }\widehat{\ }\ n)\ g'\ 0\ /\ fact\ n$
using *tendsto_unique*[*OF nontriv*] **by** *blast*
also have $\ldots = fps_nth\ g\ n$
by (*rule fps_nth_fps_expansion* [*symmetric*]) *fact*
finally show $fps_nth\ g\ n = fps_nth\ h\ n$ **..**
qed

end

theory *Meromorphic* **imports**
Laurent_Convergence
Cauchy_Integral_Formula
HOL−Analysis.Sparse_In
begin

8.7 Remove singular points

This function takes a complex function and returns a version of it where all removable singularities have been removed and all other singularities (to be more precise, unremovable discontinuities) are set to 0.

This is very useful since it is sometimes difficult to avoid introducing removable singularities. For example, consider the meromorphic functions $f(z) = z$ and $g(z) = 1/z$. Then a mathematician would write $f(z)g(z) = 1$, but in Isabelle of course this is not so.

Using the *remove_sings* function, we indeed have $remove_sings\ (\lambda z.\ f\ z * g\ z) = (\lambda_.\ 1)$.

definition $remove_sings :: (complex \Rightarrow complex) \Rightarrow complex \Rightarrow complex$ **where**
$remove_sings\ f\ z = (if\ \exists c.\ f\ -z\to\ c\ then\ Lim\ (at\ z)\ f\ else\ 0)$

lemma *remove_sings_eqI* [*intro*]:
assumes $f\ -z\to\ c$
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 $\neg(\exists\, c.\ (f \circ g)\ -z\to\ c)$
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* $(\lambda x.\ f\ x = g\ x)$ $(at\ z)$ $z = z'$
shows *remove_sings f z* = *remove_sings g z′*
proof (*cases* $\exists\, c.\ f\ -z\to\ c$)
case *True*
then obtain *c* **where** *c*: $f\ -z\to\ c$ **by** *blast*
hence *remove_sings f z* = *c*
by *blast*
moreover have $f\ -z\to\ c \longleftrightarrow g\ -z'\to\ c$
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 $f\ -z\to\ c \longleftrightarrow g\ -z'\to\ c$ **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* $\longleftrightarrow$ *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 *remove_sings_has_laurent_expansion* [*laurent_expansion_intros*]:
assumes *f has_laurent_expansion F*
shows *remove_sings f has_laurent_expansion F*
proof −
have *remove_sings f has_laurent_expansion F* ⟷ *f has_laurent_expansion F*
proof (*rule has_laurent_expansion_cong*)
have *isolated_singularity_at f 0*
using *assms* **by** (*metis has_laurent_expansion_isolated_0*)
thus *eventually* (λ*x*. *remove_sings f x* = *f x*) (*at 0*)
by (*rule eventually_remove_sings_eq_at*)
qed *auto*
with *assms* **show** *?thesis*
by *simp*
qed

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 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 simp

lemma remove_sings_analytic_at:
  assumes isolated_singularity_at f z f -z→ c
  shows   remove_sings f analytic_on {z}
proof -
  from assms(1) obtain A where A: open A z ∈ A f holomorphic_on (A - {z})
   using analytic_imp_holomorphic isolated_singularity_at_iff_analytic_nhd by
auto
  have ana: f analytic_on (A - {z})
    by (subst analytic_on_open) (use A in auto)

  have remove_sings f holomorphic_on A
  proof (rule no_isolated_singularity)
    have f holomorphic_on (A - {z})
      by fact
    moreover have remove_sings f holomorphic_on (A - {z}) ⟷ f holomor-
phic_on (A - {z})
      by (intro holomorphic_cong remove_sings_at_analytic) (auto intro!: ana-
lytic_on_subset[OF ana])
    ultimately show remove_sings f holomorphic_on (A - {z})
      by blast
    hence continuous_on (A-{z}) (remove_sings f)
      by (intro holomorphic_on_imp_continuous_on)
    moreover have isCont (remove_sings f) z
```

using *assms isCont_def remove_sings_eqI tendsto_remove_sings_iff* **by** *blast*
ultimately show *continuous_on A (remove_sings f)*
by (*metis A(1) DiffI closed_singleton continuous_on_eq_continuous_at open_Diff singletonD*)
qed (*use A(1)* **in** *auto*)
thus *?thesis*
using *A(1,2) analytic_at* **by** *blast*
qed

lemma *remove_sings_analytic_on*:
assumes *f analytic_on A*
shows *remove_sings f analytic_on A*
proof −
from *assms* **obtain** *B* **where** *B*: *open B A* ⊆ *B f holomorphic_on B*
by (*metis analytic_on_holomorphic*)
have *remove_sings f holomorphic_on B* ⟷ *f holomorphic_on B*
proof (*rule holomorphic_cong*)
fix *z* **assume** *z* ∈ *B*
have *f analytic_on* {*z*}
using ‹*z* ∈ *B*› *B holomorphic_on_imp_analytic_at* **by** *blast*
thus *remove_sings f z* = *f z*
by (*rule remove_sings_at_analytic*)
qed *auto*
thus *?thesis*
using *B analytic_on_holomorphic* **by** *blast*
qed

lemma *residue_remove_sings* [*simp*]:
assumes *isolated_singularity_at f z*
shows *residue* (*remove_sings f*) *z* = *residue f z*
proof −
from *assms* **have** *eventually* (λ*w*. *remove_sings f w* = *f w*) (*at z*)
using *eventually_remove_sings_eq_at* **by** *blast*
then obtain *A* **where** *A*: *open A z* ∈ *A* ⋀*w*. *w* ∈ *A* − {*z*} ⟹ *remove_sings f w* = *f w*
by (*subst* (*asm*) *eventually_at_topological*) *blast*
from *A(1,2)* **obtain** ε **where** ε: ε > *0 cball z* ε ⊆ *A*
using *open_contains_cball_eq* **by** *blast*
hence *eq*: *remove_sings f w* = *f w* **if** *w* ∈ *cball z* ε − {*z*} **for** *w*
using *that A* ε **by** *blast*

define *P* **where** *P* = (λ*f c* ε. (*f has_contour_integral of_real* (2 ∗ *pi*) ∗ i ∗ *c*) (*circlepath z* ε))
have *P* (*remove_sings f*) *c* δ ⟷ *P f c* δ **if** *0* < δ δ < ε **for** *c* δ
unfolding *P_def* **using** ‹ε > *0*› *that* **by** (*intro has_contour_integral_cong*) (*auto simp*: *eq*)
hence ∗: (∀ε>*0*. ε < *e* ⟶ *P* (*remove_sings f*) *c* ε) ⟷ (∀ε>*0*. ε < *e* ⟶ *P f c* ε) **if** *e* ≤ ε **for** *c e*

```
    using that by force
  have **: (∃ e>0. ∀ε>0. ε < e ⟶ P (remove_sings f) c ε) ⟷ (∃ e>0. ∀ε>0.
ε < e ⟶ P f c ε) for c
  proof
    assume (∃ e>0. ∀ε>0. ε < e ⟶ P (remove_sings f) c ε)
    then obtain e where e > 0 ∀ε>0. ε < e ⟶ P (remove_sings f) c ε
      by blast
    thus (∃ e>0. ∀ε>0. ε < e ⟶ P f c ε)
      by (intro exI[of _ min e ε]) (use *[of min e ε c] ε(1) in auto)
  next
    assume (∃ e>0. ∀ε>0. ε < e ⟶ P f c ε)
    then obtain e where e > 0 ∀ε>0. ε < e ⟶ P f c ε
      by blast
    thus (∃ e>0. ∀ε>0. ε < e ⟶ P (remove_sings f) c ε)
      by (intro exI[of _ min e ε]) (use *[of min e ε c] ε(1) in auto)
  qed
  show ?thesis
   unfolding residue_def by (intro arg_cong[of _ _ Eps] ext **[unfolded P_def])
qed

lemma remove_sings_shift_0:
  remove_sings f z = remove_sings (λw. f (z + w)) 0
proof (cases ∃ c. f −z→ c)
  case True
  then obtain c where c: f −z→ c
    by blast
  from c have remove_sings f z = c
    by (rule remove_sings_eqI)
  moreover have remove_sings (λw. f (z + w)) 0 = c
   by (rule remove_sings_eqI) (use c in ‹simp_all add: at_to_0' filterlim_filtermap
add_ac›)
  ultimately show ?thesis
    by simp
next
  case False
  hence ¬(∃ c. (λw. f (z + w)) −0→ c)
    by (simp add: at_to_0' filterlim_filtermap add_ac)
  with False show ?thesis
    by (simp add: remove_sings_def)
qed

lemma remove_sings_shift_0':
  NO_MATCH 0 z ⟹ remove_sings f z = remove_sings (λw. f (z + w)) 0
  by (rule remove_sings_shift_0)
```

8.8 Meromorphicity

We define meromorphicity in terms of Laurent series expansions. This has the advantage of giving us a particularly simple definition that makes many

of the lemmas below trivial because they reduce to similar statements about Laurent series that are already in the library.

On open domains, this definition coincides with the usual one from the literature, namely that the function be holomorphic on its domain except for a set of non-essential singularities that is *sparse*, i.e. that has no limit point inside the domain.

However, unlike the definitions found in the literature, our definition also makes sense for non-open domains: similarly to (*analytic_on*), we consider a function meromorphic on a non-open domain if it is meromorphic on some open superset of that domain.

We will prove all of this below.

definition *meromorphic_on* :: (*complex* ⇒ *complex*) ⇒ *complex set* ⇒ *bool*
(**infixl** ‹(*meromorphic′_on*)› *50*) **where**
f meromorphic_on A ⟷ (∀ *z*∈*A*. ∃ *F*. (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion* *F*)

lemma *meromorphic_at_iff*: *f meromorphic_on* {*z*} ⟷ *isolated_singularity_at* *f z* ∧ *not_essential f z*
unfolding *meromorphic_on_def*
by (*metis has_laurent_expansion_isolated has_laurent_expansion_not_essential* *insertI1 singletonD not_essential_has_laurent_expansion*)

lemma *meromorphic_at_cong_ev*:
assumes *eventually* (λ*w*. *f w* = *g w*) (*at z*)
shows *f meromorphic_on* {*z*} ⟷ *g meromorphic_on* {*z*}
unfolding *meromorphic_on_def* **using** *assms has_laurent_expansion_cong′* **by** *blast*

named_theorems *meromorphic_intros*

lemma *meromorphic_on_empty* [*simp*, *intro*]: *f meromorphic_on* {}
by (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_def′*:
f meromorphic_on A ⟷ (∀ *z*∈*A*. (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion laurent_expansion f z*)
unfolding *meromorphic_on_def* **using** *laurent_expansion_eqI* **by** *blast*

lemma *meromorphic_on_meromorphic_at*: *f meromorphic_on A* ⟷ (∀ *x*∈*A*. *f* *meromorphic_on* {*x*})
by (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_altdef*:
f meromorphic_on A ⟷ (∀ *z*∈*A*. *isolated_singularity_at f z* ∧ *not_essential f z*)
by (*subst meromorphic_on_meromorphic_at*) (*auto simp*: *meromorphic_at_iff*)

lemma *meromorphic_on_cong*:
assumes $\bigwedge z.\ z \in A \Longrightarrow$ *eventually* $(\lambda w.\ f\ w = g\ w)$ $(at\ z)$ $A = B$
shows f *meromorphic_on* $A \longleftrightarrow g$ *meromorphic_on* B
unfolding *meromorphic_on_def* **using** *assms*
by (*intro ball_cong refl arg_cong*[*of* _ _ *Ex*] *has_laurent_expansion_cong ext*)
(*simp_all add*: *at_to_0'' eventually_filtermap add_ac*)

lemma *meromorphic_on_subset*: f *meromorphic_on* $A \Longrightarrow B \subseteq A \Longrightarrow f$ *meromorphic_on* B
by (*auto simp*: *meromorphic_on_def*)

lemma *meromorphic_on_Un*:
assumes f *meromorphic_on* A f *meromorphic_on* B
shows f *meromorphic_on* $(A \cup B)$
using *assms* **unfolding** *meromorphic_on_def* **by** *blast*

lemma *meromorphic_on_Union*:
assumes $\bigwedge A.\ A \in B \Longrightarrow f$ *meromorphic_on* A
shows f *meromorphic_on* $(\bigcup B)$
using *assms* **unfolding** *meromorphic_on_def* **by** *blast*

lemma *meromorphic_on_UN*:
assumes $\bigwedge x.\ x \in X \Longrightarrow f$ *meromorphic_on* $(A\ x)$
shows f *meromorphic_on* $(\bigcup x{\in}X.\ A\ x)$
using *assms* **unfolding** *meromorphic_on_def* **by** *blast*

lemma *meromorphic_on_imp_has_laurent_expansion*:
assumes f *meromorphic_on* A $z \in A$
shows $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion laurent_expansion* $f\ z$
using *assms laurent_expansion_eqI* **unfolding** *meromorphic_on_def* **by** *blast*

lemma *meromorphic_on_open_nhd*:
assumes f *meromorphic_on* A
obtains B **where** *open* B $A \subseteq B$ f *meromorphic_on* B
proof −
obtain F **where** F: $\bigwedge z.\ z \in A \Longrightarrow (\lambda w.\ f\ (z + w))$ *has_laurent_expansion* $F\ z$
using *assms* **unfolding** *meromorphic_on_def* **by** *metis*
have $\exists Z.$ *open* $Z \wedge z \in Z \wedge (\forall\, w{\in}Z{-}\{z\}.$ *eval_fls* $(F\ z)\ (w - z) = f\ w)$ **if** z: $z \in A$ **for** z
proof −
obtain Z **where** Z: *open* Z $0 \in Z$ $\bigwedge w.\ w \in Z - \{0\} \Longrightarrow$ *eval_fls* $(F\ z)\ w = f\ (z + w)$
using $F[OF\ z]$ **unfolding** *has_laurent_expansion_def eventually_at_topological* **by** *blast*
hence *open* $((+)\ z\ `\ Z)$ **and** $z \in (+)\ z\ `\ Z$
using *open_translation* **by** *auto*
moreover have *eval_fls* $(F\ z)\ (w - z) = f\ w$ **if** $w \in (+)\ z\ `\ Z - \{z\}$ **for** w
using $Z(3)[of\ w - z]$ *that* **by** *auto*
ultimately show *?thesis* **by** *blast*

```
  qed
  then obtain Z where Z:
    ⋀z. z ∈ A ⟹ open (Z z) ∧ z ∈ Z z ∧ (∀w∈Z z−{z}. eval_fls (F z) (w − z)
= f w)
    by metis

  define B where B = (⋃z∈A. Z z ∩ eball z (fls_conv_radius (F z)))
  show ?thesis
  proof (rule that[of B])
    show open B
      using Z unfolding B_def by auto
    show A ⊆ B
       unfolding B_def using F Z by (auto simp: has_laurent_expansion_def
zero_ereal_def)
    show f meromorphic_on B
      unfolding meromorphic_on_def
    proof
      fix z assume z: z ∈ B
      show ∃F. (λw. f (z + w)) has_laurent_expansion F
      proof (cases z ∈ A)
        case True
        thus ?thesis using F by blast
      next
        case False
           then obtain z0 where z0: z0 ∈ A z ∈ Z z0 − {z0} dist z0 z <
fls_conv_radius (F z0)
          using z False Z unfolding B_def by auto
        hence (λw. eval_fls (F z0) (w − z0)) analytic_on {z}
          by (intro analytic_on_eval_fls′ analytic_intros) (auto simp: dist_norm)
        also have ?this ⟷ f analytic_on {z}
        proof (intro analytic_at_cong refl)
          have eventually (λw. w ∈ Z z0 − {z0}) (nhds z)
            using Z[of z0] z0 by (intro eventually_nhds_in_open) auto
          thus ∀F x in nhds z. eval_fls (F z0) (x − z0) = f x
            by eventually_elim (use Z[of z0] z0 in auto)
        qed
        finally show ?thesis
       using analytic_at_imp_has_fps_expansion has_fps_expansion_to_laurent
by blast
      qed
    qed
  qed
qed

lemma meromorphic_on_not_essential:
  assumes f meromorphic_on {z}
  shows   not_essential f z
 using assms has_laurent_expansion_not_essential unfolding meromorphic_on_def
by blast
```

lemma *meromorphic_on_isolated_singularity*:
assumes *f meromorphic_on* {*z*}
shows *isolated_singularity_at f z*
using *assms has_laurent_expansion_isolated* **unfolding** *meromorphic_on_def*
by *blast*

lemma *meromorphic_on_imp_not_islimpt_singularities*:
assumes *f meromorphic_on A z* ∈ *A*
shows ¬*z islimpt* {*w*. ¬*f analytic_on* {*w*}}
proof −
obtain *B* **where** *B*: *open B A* ⊆ *B f meromorphic_on B*
using *assms meromorphic_on_open_nhd* **by** *blast*
obtain *F* **where** *F*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
using *B assms*(2) **unfolding** *meromorphic_on_def* **by** *blast*
from *F* **have** *isolated_singularity_at f z*
using *has_laurent_expansion_isolated assms*(2) **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* ∈ *ball z r* − {*z*} **for** *w*
by (*rule analytic_on_subset*[*OF r*(2)]) (*use that* **in** *auto*)
hence *eventually* (λ*w*. *f analytic_on* {*w*}) (*at z*)
using *eventually_at_in_open*[*of ball z r z*] ‹*r* > *0*› **by** (*auto elim*!: *eventually_mono*)
thus ¬*z islimpt* {*w*. ¬*f analytic_on* {*w*}}
by (*auto simp*: *islimpt_conv_frequently_at frequently_def*)
qed

lemma *meromorphic_on_imp_sparse_singularities*:
assumes *f meromorphic_on A*
shows {*w*. ¬*f analytic_on* {*w*}} *sparse_in A*
by (*metis assms meromorphic_on_imp_not_islimpt_singularities*
meromorphic_on_open_nhd sparse_in_open sparse_in_subset)

lemma *meromorphic_on_imp_sparse_singularities′*:
assumes *f meromorphic_on A*
shows {*w*∈*A*. ¬*f analytic_on* {*w*}} *sparse_in A*
using *meromorphic_on_imp_sparse_singularities*[*OF assms*]
by (*rule sparse_in_subset2*) *auto*

lemma *meromorphic_onE*:
assumes *f meromorphic_on A*
obtains *pts* **where** *pts* ⊆ *A pts sparse_in A f analytic_on A* − *pts*
⋀*z*. *z* ∈ *A* ⟹ *not_essential f z* ⋀*z*. *z* ∈ *A* ⟹ *isolated_singularity_at f z*
proof (*rule that*)
show {*z* ∈ *A*. ¬ *f analytic_on* {*z*}} *sparse_in A*
using *assms* **by** (*rule meromorphic_on_imp_sparse_singularities′*)
show *f analytic_on A* − {*z* ∈ *A*. ¬ *f analytic_on* {*z*}}
by (*subst analytic_on_analytic_at*) *auto*

qed (*use assms* **in** ‹*auto intro*: *meromorphic_on_isolated_singularity meromorphic_on_not_essential meromorphic_on_subset*›)

lemma *meromorphic_onI_weak*:
assumes *f analytic_on A − pts* ⋀*z*. *z* ∈ *pts* ⟹ *not_essential f z pts sparse_in A*
pts ∩ *frontier A* = {}
shows *f meromorphic_on A*
unfolding *meromorphic_on_def*
proof
fix *z* **assume** *z*: *z* ∈ *A*
show (∃*F*. (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*)
proof (*cases z* ∈ *pts*)
case *False*
have *f analytic_on* {*z*}
using *assms*(*1*) **by** (*rule analytic_on_subset*) (*use False z* **in** *auto*)
thus *?thesis*
using *isolated_singularity_at_analytic not_essential_analytic*
not_essential_has_laurent_expansion **by** *blast*
next
case *True*
show *?thesis*
proof (*rule exI*, *rule not_essential_has_laurent_expansion*)
show *not_essential f z*
using *assms*(*2*) *True* **by** *blast*
next
show *isolated_singularity_at f z*
proof (*rule isolated_singularity_at_holomorphic*)
show *open* (*interior A − pts* − {*z*}))
proof (*rule open_diff_sparse_pts*)
show *pts* − {*z*} *sparse_in interior A*
using *sparse_in_subset sparse_in_subset2 assms interior_subset Diff_subset* **by** *metis*
qed *auto*
next
have *f analytic_on interior A* − (*pts* − {*z*}) − {*z*}
using *assms*(*1*) **by** (*rule analytic_on_subset*) (*use interior_subset* **in** *blast*)
thus *f holomorphic_on interior A* − (*pts* − {*z*}) − {*z*}
by (*rule analytic_imp_holomorphic*)
next
from *assms*(*4*) **and** *True* **have** *z* ∈ *interior A*
unfolding *frontier_def* **using** *closure_subset z* **by** *blast*
thus *z* ∈ *interior A* − (*pts* − {*z*})
by *blast*
qed
qed
qed

lemma *meromorphic_onI_open*:
assumes *open A f analytic_on A − pts* ⋀*z*. *z* ∈ *pts* ⟹ *not_essential f z*
assumes ⋀*z*. *z* ∈ *A* ⟹ ¬*z islimpt pts* ∩ *A*
shows *f meromorphic_on A*
proof (*rule meromorphic_onI_weak*)
have ∗: *A − pts* ∩ *A* = *A − pts*
by *blast*
show *f analytic_on A − pts* ∩ *A*
unfolding ∗ **by** *fact*
show *pts* ∩ *A sparse_in A*
using *assms*(*1*,*4*) **by** (*subst sparse_in_open*) *auto*
show *not_essential f z* **if** *z* ∈ *pts* ∩ *A* **for** *z*
using *assms*(*3*) *that* **by** *blast*
show *pts* ∩ *A* ∩ *frontier A* = {}
using ‹*open A*› *frontier_disjoint_eq* **by** *blast*
qed

lemma *meromorphic_at_isCont_imp_analytic*:
assumes *f meromorphic_on* {*z*} *isCont f z*
shows *f analytic_on* {*z*}
proof −
have ∗: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion laurent_expansion f z*
using *assms* **by** (*auto intro*: *meromorphic_on_imp_has_laurent_expansion*)
from *assms* **have** ¬*is_pole f z*
using *is_pole_def not_tendsto_and_filterlim_at_infinity trivial_limit_at* **by**
(*metis isContD*)
with ∗ **have** *fls_subdegree* (*laurent_expansion f z*) ≥ *0*
using *has_laurent_expansion_imp_is_pole linorder_not_le* **by** *blast*
hence ∗∗: (λ*w*. *eval_fls* (*laurent_expansion f z*) (*w* − *z*)) *analytic_on* {*z*}
by (*intro analytic_intros*)+ (*use* ∗ **in** ‹*auto simp*: *has_laurent_expansion_def*
zero_ereal_def›)
have (λ*w*. *eval_fls* (*laurent_expansion f z*) (*w* − *z*)) −*z*→ *eval_fls* (*laurent_expansion*
f z) (*z* − *z*)
by (*intro isContD analytic_at_imp_isCont* ∗∗)
also have *?this* ⟷ *f* −*z*→ *eval_fls* (*laurent_expansion f z*) (*z* − *z*)
by (*intro filterlim_cong refl*)
(*use* ∗ **in** ‹*auto simp*: *has_laurent_expansion_def at_to_0′ eventually_filtermap*
add_ac›)
finally have *f* −*z*→ *eval_fls* (*laurent_expansion f z*) *0*
by *simp*
moreover from *assms* **have** *f* −*z*→ *f z*
by (*auto intro*: *isContD*)
ultimately have ∗∗∗: *eval_fls* (*laurent_expansion f z*) *0* = *f z*
by (*rule LIM_unique*)

have *eventually* (λ*w*. *f w* = *eval_fls* (*laurent_expansion f z*) (*w* − *z*)) (*at z*)
using ∗ **by** (*simp add*: *has_laurent_expansion_def at_to_0′ eventually_filtermap*
add_ac eq_commute)

```
  hence eventually (λw. f w = eval_fls (laurent_expansion f z) (w − z)) (nhds z)
    unfolding eventually_at_filter by eventually_elim (use ∗∗∗ in auto)
  hence f analytic_on {z} ⟷ (λw. eval_fls (laurent_expansion f z) (w − z))
analytic_on {z}
    by (intro analytic_at_cong refl)
  with ∗∗ show ?thesis
    by simp
qed

lemma analytic_on_imp_meromorphic_on:
  assumes f analytic_on A
  shows   f meromorphic_on A
  by (rule meromorphic_onI_weak[of _ _ {}]) (use assms in auto)

lemma meromorphic_on_compose:
  assumes g meromorphic_on A f analytic_on B f ‘ B ⊆ A
  shows   (λw. g (f w)) meromorphic_on B
  unfolding meromorphic_on_def
proof safe
  fix z assume z: z ∈ B
  have f analytic_on {z}
    using assms(2) by (rule analytic_on_subset) (use assms(3) z in auto)
  hence (λw. f w − f z) analytic_on {z}
    by (intro analytic_intros)
  then obtain F where F: (λw. f (z + w) − f z) has_fps_expansion F
    using analytic_at_imp_has_fps_expansion by blast

  from assms(3) and z have f z ∈ A
    by auto
  with assms(1) obtain G where G: (λw. g (f z + w)) has_laurent_expansion
G
    using z by (auto simp: meromorphic_on_def)

  have ∃H. ((λw. g (f z + w)) ∘ (λw. f (z + w) − f z)) has_laurent_expansion H
  proof (cases F = 0)
    case False
    show ?thesis
    proof (rule exI, rule has_laurent_expansion_compose)
      show (λw. f (z + w) − f z) has_laurent_expansion fps_to_fls F
        using F by (rule has_laurent_expansion_fps)
      show fps_nth F 0 = 0
        using has_fps_expansion_imp_0_eq_fps_nth_0[OF F] by simp
    qed fact+
  next
    case True
    have (λw. g (f z)) has_laurent_expansion fls_const (g (f z))
      by auto
    also have ?this ⟷ (λw. ((λw. g (f z + w)) ∘ (λw. f (z + w) − f z)) w)
                        has_laurent_expansion fls_const (g (f z))
```

```
    proof (rule has_laurent_expansion_cong, goal_cases)
      case 1
      from F and True have eventually (λw. f (z + w) − f z = 0) (nhds 0)
        by (simp add: has_fps_expansion_0_iff)
      hence eventually (λw. f (z + w) − f z = 0) (at 0)
        by (simp add: eventually_nhds_conv_at)
      thus ?case
        by eventually_elim auto
    qed auto
    finally show ?thesis
      by blast
  qed
  thus ∃H. (λw. g (f (z + w))) has_laurent_expansion H
    by (simp add: o_def)
qed

lemma constant_on_imp_meromorphic_on:
  assumes f constant_on A open A
  shows f meromorphic_on A
  using assms analytic_on_imp_meromorphic_on
    constant_on_imp_analytic_on
  by blast
```

8.9 Nice meromorphicity

This is probably very non-standard, but we call a function "nicely meromorphic" if it is meromorphic and has no removable singularities. That means that the only singularities are poles. We furthermore require that the function return 0 at any pole, for convenience.

```
definition nicely_meromorphic_on :: (complex ⇒ complex) ⇒ complex set ⇒ bool
    (infixl ‹(nicely'_meromorphic'_on)› 50)
    where f nicely_meromorphic_on A ⟷ f meromorphic_on A
        ∧ (∀z∈A. (is_pole f z ∧ f z=0) ∨ f −z→ f z)

lemma nicely_meromorphic_on_subset:
  f nicely_meromorphic_on A ⟹ B ⊆ A ⟹ f nicely_meromorphic_on B
  using meromorphic_on_subset unfolding nicely_meromorphic_on_def by blast

lemma constant_on_imp_nicely_meromorphic_on:
  assumes f constant_on A open A
  shows f nicely_meromorphic_on A
  by (meson analytic_at_imp_isCont assms
      constant_on_imp_holomorphic_on
      constant_on_imp_meromorphic_on
      holomorphic_on_imp_analytic_at isCont_def
      nicely_meromorphic_on_def)

lemma nicely_meromorphic_on_imp_analytic_at:
```

```
  assumes f nicely_meromorphic_on A z ∈ A ¬is_pole f z
  shows   f analytic_on {z}
proof (rule meromorphic_at_isCont_imp_analytic)
  show f meromorphic_on {z}
   by (rule meromorphic_on_subset[of _ A]) (use assms in ‹auto simp: nicely_meromorphic_on_def›)
next
  from assms have f −z→ f z
    by (auto simp: nicely_meromorphic_on_def)
  thus isCont f z
    by (auto simp: isCont_def)
qed

lemma analytic_on_imp_nicely_meromorphic_on:
  f analytic_on A ⟹ f nicely_meromorphic_on A
  by (meson analytic_at_imp_isCont analytic_on_analytic_at
        analytic_on_imp_meromorphic_on isContD nicely_meromorphic_on_def)

lemma remove_sings_meromorphic [meromorphic_intros]:
  assumes f meromorphic_on A
  shows   remove_sings f meromorphic_on A
  unfolding meromorphic_on_def
proof safe
  fix z assume z: z ∈ A
  show ∃ F. (λw. remove_sings f (z + w)) has_laurent_expansion F
   using assms meromorphic_on_isolated_singularity meromorphic_on_not_essential
         not_essential_has_laurent_expansion z meromorphic_on_subset by blast
qed

lemma remove_sings_nicely_meromorphic:
  assumes f meromorphic_on A
  shows   remove_sings f nicely_meromorphic_on A
proof −
  have remove_sings f meromorphic_on A
    by (simp add: assms remove_sings_meromorphic)
  moreover have is_pole (remove_sings f) z
        ∧ remove_sings f z = 0 ∨
            remove_sings f −z→ remove_sings f z
    if z∈A for z
  proof (cases ∃ c. f −z→ c)
    case True
    then have remove_sings f −z→ remove_sings f z
      by (metis remove_sings_eqI tendsto_remove_sings_iff
          assms meromorphic_onE that)
    then show ?thesis by simp
  next
    case False
    then have is_pole (remove_sings f) z
        ∧ remove_sings f z = 0
      by (meson is_pole_remove_sings_iff remove_sings_def
```

remove_sings_eq_0_iff assms meromorphic_onE that)
then show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
unfolding *nicely_meromorphic_on_def* **by** *simp*
qed

A nicely meromorphic function that frequently takes the same value in the neighbourhood of some point is constant.

lemma *frequently_eq_meromorphic_imp_constant*:
assumes *frequently* $(\lambda z.\ f\ z = c)$ $(at\ z)$
assumes *f nicely_meromorphic_on A open A connected A* $z \in A$
shows $\bigwedge w.\ w \in A \Longrightarrow f\ w = c$
proof −
from *assms*(*2*) **have** *mero*: *f meromorphic_on A*
by (*auto simp*: *nicely_meromorphic_on_def*)
have *sparse*: $\{z.\ is_pole\ f\ z\}$ *sparse_in A*
using *assms*(*2*) *mero*
by (*meson assms*(*3*) *meromorphic_on_isolated_singularity meromorphic_on_meromorphic_at not_islimpt_poles sparse_in_open*)

have *eq*: *f w* = *c* **if** *w*: $w \in A$ ¬*is_pole f w* **for** *w*
proof −
have *f w* − *c* = *0*
proof (*rule analytic_continuation*[*of* $\lambda w.\ f\ w - c$])
show $(\lambda w.\ f\ w - c)$ *holomorphic_on* $\{z{\in}A.\ \neg is_pole\ f\ z\}$ **using** *assms*(*2*)
by (*intro holomorphic_intros*)
(*metis* (*mono_tags*, *lifting*) *analytic_imp_holomorphic analytic_on_analytic_at*

mem_Collect_eq nicely_meromorphic_on_imp_analytic_at)
next
from *sparse* **and** *assms*(*3*) **have** *open* $(A - \{z.\ is_pole\ f\ z\})$
by (*simp add*: *open_diff_sparse_pts*)
also have $A - \{z.\ is_pole\ f\ z\} = \{z{\in}A.\ \neg is_pole\ f\ z\}$
by *blast*
finally show *open* ... **.**
next
from *sparse* **have** *connected* $(A - \{z.\ is_pole\ f\ z\})$
using *assms*(*3*,*4*) **by** (*intro sparse_imp_connected*) *auto*
also have $A - \{z.\ is_pole\ f\ z\} = \{z{\in}A.\ \neg is_pole\ f\ z\}$
by *blast*
finally show *connected* ... **.**
next
have *eventually* $(\lambda w.\ w \in A)$ $(at\ z)$
using *assms* **by** (*intro eventually_at_in_open'*) *auto*
moreover have *eventually* $(\lambda w.\ \neg is_pole\ f\ w)$ $(at\ z)$ **using** *mero*
by (*metis assms*(*5*) *eventually_not_pole meromorphic_onE*)
ultimately have *ev*: *eventually* $(\lambda w.\ w \in A \wedge \neg is_pole\ f\ w)$ $(at\ z)$
by *eventually_elim auto*

```
      show z islimpt {z∈A. ¬is_pole f z ∧ f z = c}
        using frequently_eventually_frequently[OF assms(1) ev]
        unfolding islimpt_conv_frequently_at by (rule frequently_elim1) auto
    next
      from assms(1) have ¬is_pole f z
        by (simp add: frequently_const_imp_not_is_pole)
      with ‹z ∈ A› show z ∈ {z ∈ A. ¬ is_pole f z}
        by auto
    qed (use w in auto)
    thus f w = c
      by simp
  qed

  have not_pole: ¬is_pole f w if w: w ∈ A for w
  proof −
    have eventually (λw. ¬is_pole f w) (at w)
      using mero by (metis eventually_not_pole meromorphic_onE that)
    moreover have eventually (λw. w ∈ A) (at w)
      using w ‹open A› by (intro eventually_at_in_open')
    ultimately have eventually (λw. f w = c) (at w)
      by eventually_elim (auto simp: eq)
    hence is_pole f w ⟷ is_pole (λ_. c) w
      by (intro is_pole_cong refl)
    thus ?thesis
      by simp
  qed

  show f w = c if w: w ∈ A for w
    using eq[OF w not_pole[OF w]] .
qed

lemma nicely_meromorphic_on_unop:
  assumes f nicely_meromorphic_on A
  assumes f meromorphic_on A ⟹ (λz. h (f z)) meromorphic_on A
  assumes ⋀z. z ∈ A ⟹ is_pole f z ⟹ is_pole (λz. h (f z)) z
  assumes ⋀z. z ∈ f ‘ A ⟹ isCont h z
  assumes h 0 = 0
  shows   (λz. h (f z)) nicely_meromorphic_on A
  unfolding nicely_meromorphic_on_def
proof (intro conjI ballI)
  show (λz. h (f z)) meromorphic_on A
    using assms(1,2) by (auto simp: nicely_meromorphic_on_def)
next
  fix z assume z: z ∈ A
  hence is_pole f z ∧ f z = 0 ∨ f −z→ f z
    using assms by (auto simp: nicely_meromorphic_on_def)
  thus is_pole (λz. h (f z)) z ∧ h (f z) = 0 ∨ (λz. h (f z)) −z→ h (f z)
  proof (rule disj_forward)
    assume is_pole f z ∧ f z = 0
```

```
    thus is_pole (λz. h (f z)) z ∧ h (f z) = 0
      using assms z by auto
  next
    assume ∗: f −z→ f z
    from z assms have isCont h (f z)
      by auto
    with ∗ show (λz. h (f z)) −z→ h (f z)
      using continuous_within continuous_within_compose3 by blast
  qed
qed
```

8.10 Closure properties and proofs for individual functions

lemma *meromorphic_on_const* [*intro*, *meromorphic_intros*]: (λ_. *c*) *meromorphic_on A*
by (*rule analytic_on_imp_meromorphic_on*) *auto*

lemma *meromorphic_on_id* [*intro*, *meromorphic_intros*]: (λ*w*. *w*) *meromorphic_on A*
by (*auto simp*: *meromorphic_on_def intro*!: *exI laurent_expansion_intros*)

lemma *meromorphic_on_id'* [*intro*, *meromorphic_intros*]: *id meromorphic_on A*
by (*auto simp*: *meromorphic_on_def intro*!: *exI laurent_expansion_intros*)

lemma *meromorphic_on_add* [*meromorphic_intros*]:
assumes *f meromorphic_on A g meromorphic_on A*
shows (λ*w*. *f w* + *g w*) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_uminus* [*meromorphic_intros*]:
assumes *f meromorphic_on A*
shows (λ*w*. −*f w*) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_diff* [*meromorphic_intros*]:
assumes *f meromorphic_on A g meromorphic_on A*
shows (λ*w*. *f w* − *g w*) *meromorphic_on A*
using *meromorphic_on_add*[*OF assms*(*1*) *meromorphic_on_uminus*[*OF assms*(*2*)]]
by *simp*

lemma *meromorphic_on_mult* [*meromorphic_intros*]:
assumes *f meromorphic_on A g meromorphic_on A*
shows (λ*w*. *f w* ∗ *g w*) *meromorphic_on A*

unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_power* [*meromorphic_intros*]:
assumes *f meromorphic_on A*
shows (λ*w*. *f w* ^ *n*) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_powi* [*meromorphic_intros*]:
assumes *f meromorphic_on A*
shows (λ*w*. *f w powi n*) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_scaleR* [*meromorphic_intros*]:
assumes *f meromorphic_on A*
shows (λ*w*. *scaleR x* (*f w*)) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_inverse* [*meromorphic_intros*]:
assumes *f meromorphic_on A*
shows (λ*w*. *inverse* (*f w*)) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assumption*)+

lemma *meromorphic_on_divide* [*meromorphic_intros*]:
assumes *f meromorphic_on A g meromorphic_on A*
shows (λ*w*. *f w* / *g w*) *meromorphic_on A*
using *meromorphic_on_mult*[*OF assms*(*1*) *meromorphic_on_inverse*[*OF assms*(*2*)]]
by (*simp add*: *field_simps*)

lemma *meromorphic_on_sum* [*meromorphic_intros*]:
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i meromorphic_on A*
shows (λ*w*. ∑ *i*∈*I*. *f i w*) *meromorphic_on A*
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*

tion)+

lemma *meromorphic_on_sum_list* [*meromorphic_intros*]:
assumes $\bigwedge i.\ i \in set\ fs \Longrightarrow f\ i\ meromorphic_on\ A$
shows $(\lambda w.\ \sum i{\leftarrow}fs.\ f\ i\ w)\ meromorphic_on\ A$
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*
tion)+

lemma *meromorphic_on_sum_mset* [*meromorphic_intros*]:
assumes $\bigwedge i.\ i \in\# I \Longrightarrow f\ i\ meromorphic_on\ A$
shows $(\lambda w.\ \sum i{\in}\#I.\ f\ i\ w)\ meromorphic_on\ A$
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*
tion)+

lemma *meromorphic_on_prod* [*meromorphic_intros*]:
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i\ meromorphic_on\ A$
shows $(\lambda w.\ \prod i{\in}I.\ f\ i\ w)\ meromorphic_on\ A$
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*
tion)+

lemma *meromorphic_on_prod_list* [*meromorphic_intros*]:
assumes $\bigwedge i.\ i \in set\ fs \Longrightarrow f\ i\ meromorphic_on\ A$
shows $(\lambda w.\ \prod i{\leftarrow}fs.\ f\ i\ w)\ meromorphic_on\ A$
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*
tion)+

lemma *meromorphic_on_prod_mset* [*meromorphic_intros*]:
assumes $\bigwedge i.\ i \in\# I \Longrightarrow f\ i\ meromorphic_on\ A$
shows $(\lambda w.\ \prod i{\in}\#I.\ f\ i\ w)\ meromorphic_on\ A$
unfolding *meromorphic_on_def*
by (*rule laurent_expansion_intros exI ballI*
assms[*THEN meromorphic_on_imp_has_laurent_expansion*] | *assump-*
tion)+

lemma *nicely_meromorphic_on_const* [*intro*]: $(\lambda_.\ c)\ nicely_meromorphic_on\ A$
unfolding *nicely_meromorphic_on_def* **by** *auto*

lemma *nicely_meromorphic_on_cmult_left* [*intro*]:
assumes $f\ nicely_meromorphic_on\ A$
shows $(\lambda z.\ c * f\ z)\ nicely_meromorphic_on\ A$

proof (*cases* $c = 0$)
case [*simp*]: *False*
show *?thesis*
using *assms* **by** (*rule nicely_meromorphic_on_unop*) (*auto intro*!: *meromorphic_intros*)
qed (*auto intro*!: *nicely_meromorphic_on_const*)

lemma *nicely_meromorphic_on_cmult_right* [*intro*]:
assumes *f nicely_meromorphic_on A*
shows $(\lambda z.\ f\ z * c)$ *nicely_meromorphic_on A*
using *nicely_meromorphic_on_cmult_left*[*OF assms*, *of c*] **by** (*simp add*: *mult.commute*)

lemma *nicely_meromorphic_on_scaleR* [*intro*]:
assumes *f nicely_meromorphic_on A*
shows $(\lambda z.\ c *_R f\ z)$ *nicely_meromorphic_on A*
using *assms* **by** (*auto simp*: *scaleR_conv_of_real*)

lemma *nicely_meromorphic_on_uminus* [*intro*]:
assumes *f nicely_meromorphic_on A*
shows $(\lambda z.\ -f\ z)$ *nicely_meromorphic_on A*
using *nicely_meromorphic_on_cmult_left*[*OF assms*, *of* -1] **by** *simp*

lemma *meromorphic_on_If* [*meromorphic_intros*]:
assumes *f meromorphic_on A g meromorphic_on B*
assumes $\bigwedge z.\ z \in A \Longrightarrow z \in B \Longrightarrow f\ z = g\ z$ *open A open B* $C \subseteq A \cup B$
shows $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *meromorphic_on C*
proof (*rule meromorphic_on_subset*)
show $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *meromorphic_on* $(A \cup B)$
proof (*rule meromorphic_on_Un*)
have $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *meromorphic_on A* $\longleftrightarrow$ *f meromorphic_on A*
proof (*rule meromorphic_on_cong*)
fix *z* **assume** $z \in A$
hence *eventually* $(\lambda z.\ z \in A)$ *(at z)*
using ‹*open A*› **by** (*intro eventually_at_in_open'*) *auto*
thus $\forall_F\ w$ *in at z.* $(\textit{if}\ w \in A\ \textit{then}\ f\ w\ \textit{else}\ g\ w) = f\ w$
by *eventually_elim auto*
qed *auto*
with *assms*(*1*) **show** $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *meromorphic_on A*
by *blast*
next
have $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *meromorphic_on B* $\longleftrightarrow$ *g meromorphic_on B*
proof (*rule meromorphic_on_cong*)
fix *z* **assume** $z \in B$
hence *eventually* $(\lambda z.\ z \in B)$ *(at z)*
using ‹*open B*› **by** (*intro eventually_at_in_open'*) *auto*
thus $\forall_F\ w$ *in at z.* $(\textit{if}\ w \in A\ \textit{then}\ f\ w\ \textit{else}\ g\ w) = g\ w$
by *eventually_elim* (*use assms*(*3*) **in** *auto*)

```
    qed auto
    with assms(2) show (λz. if z ∈ A then f z else g z) meromorphic_on B
      by blast
  qed
qed fact

lemma meromorphic_on_deriv [meromorphic_intros]:
  f meromorphic_on A ⟹ deriv f meromorphic_on A
 by (metis meromorphic_on_def isolated_singularity_at_deriv meromorphic_on_isolated_singularity

              meromorphic_on_meromorphic_at meromorphic_on_not_essential
not_essential_deriv
          not_essential_has_laurent_expansion)

lemma meromorphic_on_higher_deriv [meromorphic_intros]:
  f meromorphic_on A ⟹ (deriv ^^ n) f meromorphic_on A
  by (induction n) (auto intro!: meromorphic_intros)

lemma analytic_on_eval_fps [analytic_intros]:
  assumes f analytic_on A
  assumes ⋀z. z ∈ A ⟹ norm (f z) < fps_conv_radius F
  shows   (λw. eval_fps F (f w)) analytic_on A
   by (rule analytic_on_compose[OF assms(1) analytic_on_eval_fps, unfolded
o_def])
     (use assms(2) in auto)

lemma meromorphic_on_eval_fps [meromorphic_intros]:
  assumes f analytic_on A
  assumes ⋀z. z ∈ A ⟹ norm (f z) < fps_conv_radius F
  shows   (λw. eval_fps F (f w)) meromorphic_on A
 by (rule analytic_on_imp_meromorphic_on analytic_intros analytic_on_eval_fps
assms)+

lemma meromorphic_on_eval_fls [meromorphic_intros]:
  assumes f analytic_on A
  assumes ⋀z. z ∈ A ⟹ norm (f z) < fls_conv_radius F
  shows   (λw. eval_fls F (f w)) meromorphic_on A
proof (cases fls_conv_radius F > 0)
  case False
  with assms(2) have A = {}
    by (metis all_not_in_conv ereal_less(2) norm_eq_zero order.strict_trans
          zero_ereal_def zero_less_norm_iff)
  thus ?thesis
    by auto
next
  case True
  have F: eval_fls F has_laurent_expansion F
    using True by (rule eval_fls_has_laurent_expansion)
  show ?thesis
```

```
  proof (rule meromorphic_on_compose[OF _ assms(1)])
    show eval_fls F meromorphic_on eball 0 (fls_conv_radius F)
    proof (rule meromorphic_onI_open)
      show eval_fls F analytic_on eball 0 (fls_conv_radius F) − {0}
        by (rule analytic_on_eval_fls) auto
      show not_essential (eval_fls F) z if z ∈ {0} for z
        using that F has_laurent_expansion_not_essential_0 by blast
    qed (auto simp: islimpt_finite)
  qed (use assms(2) in auto)
qed

lemma meromorphic_on_imp_analytic_cosparse:
  assumes f meromorphic_on A
  shows   eventually (λz. f analytic_on {z}) (cosparse A)
 unfolding eventually_cosparse using assms meromorphic_on_imp_sparse_singularities
by auto

lemma meromorphic_on_imp_not_pole_cosparse:
  assumes f meromorphic_on A
  shows   eventually (λz. ¬is_pole f z) (cosparse A)
proof −
  have eventually (λz. f analytic_on {z}) (cosparse A)
    by (rule meromorphic_on_imp_analytic_cosparse) fact
  thus ?thesis
    by eventually_elim (blast dest: analytic_at_imp_no_pole)
qed

lemma eventually_remove_sings_eq:
  assumes f meromorphic_on A
  shows   eventually (λz. remove_sings f z = f z) (cosparse A)
proof −
  have eventually (λz. f analytic_on {z}) (cosparse A)
    using assms by (rule meromorphic_on_imp_analytic_cosparse)
  thus ?thesis
    by eventually_elim auto
qed

lemma remove_sings_constant_on_open_iff:
  assumes f meromorphic_on A open A
  shows   remove_sings f constant_on A ⟷ (∃ c. ∀≈x∈A. f x = c)
proof
  assume remove_sings f constant_on A
  then obtain c where c: remove_sings f z = c if z ∈ A for z
    using that by (auto simp: constant_on_def)
  have ∀≈x∈A. x ∈ A
    using ‹open A› by (simp add: eventually_in_cosparse)
  hence ∀≈x∈A. f x = c
    using eventually_remove_sings_eq[OF assms(1)] by eventually_elim (use c
in auto)
```

thus $\exists\, c.\ \forall_{\approx} x{\in}A.\ f\ x = c$
by *blast*
next
assume $\exists\, c.\ \forall_{\approx} x{\in}A.\ f\ x = c$
then obtain c **where** c: $\forall_{\approx} x{\in}A.\ f\ x = c$
by *blast*
have $\forall_{\approx} x{\in}A.$ *remove_sings* $f\ x = c$
using *eventually_remove_sings_eq*[*OF assms*(*1*)] c **by** *eventually_elim auto*
hence *remove_sings* $f\ z = c$ **if** $z \in A$ **for** z **using** *that*
by (*meson assms*(*2*) c *eventually_cosparse_open_eq remove_sings_eqI tendsto_eventually*)
thus *remove_sings f constant_on A*
unfolding *constant_on_def* **by** *blast*
qed

A meromorphic function on a connected domain takes any given value either almost everywhere or almost nowhere.

lemma *meromorphic_imp_constant_or_avoid*:
assumes *mero*: *f meromorphic_on A* **and** *A*: *open A connected A*
shows *eventually* (λ*z*. *f z* = *c*) (*cosparse A*) ∨ *eventually* (λ*z*. *f z* ≠ *c*) (*cosparse A*)
proof −
have *eventually* (λ*z*. *f z* = *c*) (*cosparse A*) **if** *freq*: *frequently* (λ*z*. *f z* = *c*) (*cosparse A*)
proof −
let *?f* = *remove_sings f*
have *ev*: *eventually* (λ*z*. *?f z* = *f z*) (*cosparse A*)
by (*rule eventually_remove_sings_eq*) *fact*
have *frequently* (λ*z*. *?f z* = *c*) (*cosparse A*)
using *frequently_eventually_frequently*[*OF freq ev*] **by** (*rule frequently_elim1*) *auto*
then obtain *z0* **where** *z0*: *z0* ∈ *A frequently* (λ*z*. *?f z* = *c*) (*at z0*)
using *A* **by** (*auto simp*: *eventually_cosparse_open_eq frequently_def*)
have *mero'*: *?f nicely_meromorphic_on A*
using *mero remove_sings_nicely_meromorphic* **by** *blast*
have *eq*: *?f w* = *c* **if** *w*: *w* ∈ *A* **for** *w*
using *frequently_eq_meromorphic_imp_constant*[*OF z0*(*2*) *mero'*] *A z0*(*1*) *w* **by** *blast*
have *eventually* (λ*z*. *z* ∈ *A*) (*cosparse A*)
by (*rule eventually_in_cosparse*) (*use A* **in** *auto*)
thus *eventually* (λ*z*. *f z* = *c*) (*cosparse A*)
using *ev* **by** *eventually_elim* (*use eq* **in** *auto*)
qed
thus *?thesis*
by (*auto simp*: *frequently_def*)
qed

lemma *nicely_meromorphic_imp_constant_or_avoid*:
assumes *f nicely_meromorphic_on A open A connected A*

```
  shows (∀ x∈A. f x = c) ∨ (∀≈x∈A. f x ≠ c)
proof −
  have (∀≈x∈A. f x = c) ∨ (∀≈x∈A. f x ≠ c)
    by (intro meromorphic_imp_constant_or_avoid)
       (use assms in ‹auto simp: nicely_meromorphic_on_def›)
  thus ?thesis
  proof
    assume ev: ∀≈x∈A. f x = c
    have ∀ x∈A. f x = c
    proof
      fix x assume x: x ∈ A
      have not_essential f x
        using assms x unfolding nicely_meromorphic_on_def by blast
      moreover have is_pole f x ⟷ is_pole (λ_. c) x
      by (intro is_pole_cong) (use ev x in ‹auto simp: eventually_cosparse_open_eq
assms›)
      hence ¬is_pole f x
        by auto
      ultimately have f analytic_on {x}
        using assms(1) nicely_meromorphic_on_imp_analytic_at x by blast
      hence f −x→ f x
        by (intro isContD analytic_at_imp_isCont)
      also have ?this ⟷ (λ_. c) −x→ f x
      by (intro tendsto_cong) (use ev x in ‹auto simp: eventually_cosparse_open_eq
assms›)
      finally have (λ_. c) −x→ f x .
      moreover have (λ_. c) −x→ c
        by simp
      ultimately show f x = c
        using LIM_unique by blast
    qed
    thus ?thesis
      by blast
  qed blast
qed

lemma nicely_meromorphic_onE:
  assumes f nicely_meromorphic_on A
  obtains pts where pts ⊆ A pts sparse_in A
    f analytic_on A − pts
    ⋀z. z ∈ pts ⟹ is_pole f z ∧ f z=0
proof −
  define pts where pts = {z ∈ A. ¬ f analytic_on {z}}
  have pts ⊆ A pts sparse_in A
    using assms unfolding pts_def nicely_meromorphic_on_def
    by (auto intro:meromorphic_on_imp_sparse_singularities′)
  moreover have f analytic_on A − pts unfolding pts_def
    by (subst analytic_on_analytic_at) auto
  moreover have ⋀z. z ∈ pts ⟹ is_pole f z ∧ f z=0
```

by (*metis* (*no_types*, *lifting*) *remove_sings_eqI*
remove_sings_eq_0_iff assms is_pole_imp_not_essential
mem_Collect_eq nicely_meromorphic_on_def
nicely_meromorphic_on_imp_analytic_at pts_def)
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemma *nicely_meromorphic_onI_open*:
assumes *open A* **and**
analytic:*f analytic_on A − pts* **and**
pole:⋀*x*. *x*∈*pts* ⟹ *is_pole f x* ∧ *f x* = *0* **and**
isolated:⋀*x*. *x*∈*A* ⟹ *isolated_singularity_at f x*
shows *f nicely_meromorphic_on A*
proof −
have *f meromorphic_on A*
proof (*rule meromorphic_onI_open*)
show ⋀*z*. *z* ∈ *pts* ⟹ *not_essential f z*
using *pole* **unfolding** *not_essential_def* **by** *auto*
show ⋀*z*. *z* ∈ *A* ⟹ ¬ *z islimpt pts* ∩ *A*
by (*metis assms*(*3*) *assms*(*4*) *inf_commute inf_le2*
islimpt_subset mem_Collect_eq not_islimpt_poles subsetI)
qed *fact*+
moreover have (∀ *z*∈*A*. (*is_pole f z* ∧ *f z*=*0*) ∨ *f* −*z*→ *f z*)
by (*meson DiffI analytic analytic_at_imp_isCont*
analytic_on_analytic_at assms(*3*) *isContD*)
ultimately show *?thesis* **unfolding** *nicely_meromorphic_on_def*
by *auto*
qed

lemma *nicely_meromorphic_without_singularities*:
assumes *f nicely_meromorphic_on A* ∀ *z*∈*A*. ¬ *is_pole f z*
shows *f analytic_on A*
by (*meson analytic_on_analytic_at assms*
nicely_meromorphic_on_imp_analytic_at)

lemma *meromorphic_on_cong′*:
assumes *eventually* (λ*z*. *f z* = *g z*) (*cosparse A*) *A* = *B*
shows *f meromorphic_on A* ⟷ *g meromorphic_on B*
unfolding *assms*(*2*)[*symmetric*]
by (*rule meromorphic_on_cong eventually_cosparse_imp_eventually_at assms*)+
auto

lemma *meromorphic_on_Gamma* [*meromorphic_intros*]: *Gamma meromorphic_on A*
proof −
have *1*: *Gamma meromorphic_on* {*1* − *of_nat n*} **for** *n*
proof (*induction n*)
case *0*

```
  thus ?case by (auto intro!: analytic_on_imp_meromorphic_on analytic_intros)
 next
  case (Suc n)
  have *: (λw. w + 1) analytic_on {-complex_of_nat n}
   by (auto intro!: analytic_intros)
  have (λw. Gamma (w + 1)) meromorphic_on {-complex_of_nat n}
   using meromorphic_on_compose[OF Suc.IH *] by auto
  hence (λw. Gamma (w + 1) / w) meromorphic_on {1 - of_nat (Suc n)}
   by (auto intro!: meromorphic_intros)
  also have ?this ⟷ Gamma meromorphic_on {1 - of_nat (Suc n)}
  proof (rule meromorphic_at_cong_ev)
   have ℤ_{≤0} sparse_in (UNIV :: complex set)
    by (rule sparse_subset_Ints) auto
   hence eventually (λw. w ∉ ℤ_{≤0}) (at (1 - complex_of_nat (Suc n)))
    by (auto simp: sparse_in_eventually_iff)
   thus ∀_F w in at (1 - complex_of_nat (Suc n)). Gamma (w + 1) / w =
Gamma w
    by eventually_elim (auto simp: Gamma_plus1)
  qed
  finally show ?case .
 qed

 have Gamma meromorphic_on {z} for z
 proof (cases z ∈ ℤ_{≤0})
  case False
  thus ?thesis by (auto intro!: analytic_on_imp_meromorphic_on analytic_intros)
 next
  case True
  then obtain n where [simp]: z = -of_nat n
   by (elim nonpos_Ints_cases')
  show ?thesis
   using 1[of n + 1] by simp
 qed
 thus ?thesis
  using meromorphic_on_meromorphic_at by blast
qed

lemma meromorphic_on_Gamma' [meromorphic_intros]:
 assumes f analytic_on A
 shows   (λz. Gamma (f z)) meromorphic_on A
 using meromorphic_on_compose[OF meromorphic_on_Gamma[of UNIV] assms]
by simp
```

8.11 Meromorphic functions and zero order

```
lemma zorder_power_int:
 assumes f meromorphic_on {z} frequently (λz. f z ≠ 0) (at z)
 shows   zorder (λz. f z powi n) z = n * zorder f z
proof -
```

```
  from assms(1) obtain L where L: (λw. f (z + w)) has_laurent_expansion L
    by (auto simp: meromorphic_on_def)
  from assms(2) and L have [simp]: L ≠ 0
    by (metis assms(1) has_laurent_expansion_eventually_nonzero_iff meromor-
phic_at_iff
        not_essential_frequently_0_imp_eventually_0 not_eventually not_frequently)
  from L have L': (λw. f (z + w) powi n) has_laurent_expansion L powi n
    by (intro laurent_expansion_intros)
  have zorder f z = fls_subdegree L
    using L assms(2) ‹L ≠ 0› by (simp add: has_laurent_expansion_zorder)
  moreover have zorder (λz. f z powi n) z = fls_subdegree (L powi n)
    using L' assms(2) ‹L ≠ 0› by (simp add: has_laurent_expansion_zorder)
  moreover have fls_subdegree (L powi n) = n * fls_subdegree L
    by simp
  ultimately show ?thesis
    by simp
qed

lemma zorder_power:
  assumes f meromorphic_on {z} frequently (λz. f z ≠ 0) (at z)
  shows   zorder (λz. f z ^ n) z = n * zorder f z
  using zorder_power_int[OF assms, of int n] by simp

lemma zorder_add1:
  assumes f meromorphic_on {z} frequently (λz. f z ≠ 0) (at z)
  assumes g meromorphic_on {z} frequently (λz. g z ≠ 0) (at z)
  assumes zorder f z < zorder g z
  shows   zorder (λz. f z + g z) z = zorder f z
proof −
  from assms(1) obtain F where F: (λw. f (z + w)) has_laurent_expansion F
    by (auto simp: meromorphic_on_def)
  from assms(3) obtain G where G: (λw. g (z + w)) has_laurent_expansion G
    by (auto simp: meromorphic_on_def)
  have [simp]: F ≠ 0 G ≠ 0
     by (metis assms has_laurent_expansion_eventually_nonzero_iff meromor-
phic_at_iff
        not_essential_frequently_0_imp_eventually_0 not_eventually not_frequently
F G)+
  have ∗: zorder f z = fls_subdegree F zorder g z = fls_subdegree G
    using F G assms by (simp_all add: has_laurent_expansion_zorder)
  from assms ∗ have F ≠ −G
    by auto
  hence [simp]: F + G ≠ 0
    by (simp add: add_eq_0_iff2)
  moreover have zorder (λz. f z + g z) z = fls_subdegree (F + G)
    using has_laurent_expansion_zorder[OF has_laurent_expansion_add[OF F
G]] ‹F ≠ −G› by simp
  moreover have fls_subdegree (F + G) = fls_subdegree F
    using assms by (simp add: ∗ fls_subdegree_add_eq1)
```

ultimately show *?thesis*
by (*simp add*: ∗)
qed

lemma *zorder_add2*:
assumes *f meromorphic_on* {*z*} *frequently* (λ*z*. *f z* ≠ *0*) (*at z*)
assumes *g meromorphic_on* {*z*} *frequently* (λ*z*. *g z* ≠ *0*) (*at z*)
assumes *zorder f z* > *zorder g z*
shows *zorder* (λ*z*. *f z* + *g z*) *z* = *zorder g z*
using *zorder_add1*[*OF assms*(*3*,*4*) *assms*(*1*,*2*)] *assms*(*5*−) **by** (*simp add*: *add.commute*)

lemma *zorder_add_ge*:
fixes *f g* :: *complex* ⇒ *complex*
assumes *f meromorphic_on* {*z*} *frequently* (λ*z*. *f z* ≠ *0*) (*at z*)
assumes *g meromorphic_on* {*z*} *frequently* (λ*z*. *g z* ≠ *0*) (*at z*)
assumes *frequently* (λ*z*. *f z* + *g z* ≠ *0*) (*at z*) *zorder f z* ≥ *c zorder g z* ≥ *c*
shows *zorder* (λ*z*. *f z* + *g z*) *z* ≥ *c*
proof −
from *assms*(*1*) **obtain** *F* **where** *F*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
by (*auto simp*: *meromorphic_on_def*)
from *assms*(*3*) **obtain** *G* **where** *G*: (λ*w*. *g* (*z* + *w*)) *has_laurent_expansion G*
by (*auto simp*: *meromorphic_on_def*)
have [*simp*]: *F* ≠ *0 G* ≠ *0*
using *assms F G has_laurent_expansion_frequently_nonzero_iff* **by** *blast*+
have *FG*: (λ*w*. *f* (*z* + *w*) + *g* (*z* + *w*)) *has_laurent_expansion F* + *G*
by (*intro laurent_expansion_intros F G*)
have [*simp*]: *F* + *G* ≠ *0*
using *assms*(*5*) *has_laurent_expansion_frequently_nonzero_iff*[*OF FG*] **by** *blast*

have ∗: *zorder f z* = *fls_subdegree F zorder g z* = *fls_subdegree G*
zorder (λ*z*. *f z* + *g z*) *z* = *fls_subdegree* (*F* + *G*)
using *F G FG has_laurent_expansion_zorder* **by** *simp_all*
moreover have *zorder* (λ*z*. *f z* + *g z*) *z* = *fls_subdegree* (*F* + *G*)
using *has_laurent_expansion_zorder*[*OF has_laurent_expansion_add*[*OF F G*]] **by** *simp*
moreover have *fls_subdegree* (*F* + *G*) ≥ *min* (*fls_subdegree F*) (*fls_subdegree G*)
by (*intro fls_plus_subdegree*) *simp*
ultimately show *?thesis*
using *assms*(*6*,*7*) **unfolding** ∗ **by** *linarith*
qed

lemma *zorder_diff_ge*:
fixes *f g* :: *complex* ⇒ *complex*
assumes *f meromorphic_on* {*z*} *frequently* (λ*z*. *f z* ≠ *0*) (*at z*)
assumes *g meromorphic_on* {*z*} *frequently* (λ*z*. *g z* ≠ *0*) (*at z*)
assumes *frequently* (λ*z*. *f z* ≠ *g z*) (*at z*) *zorder f z* ≥ *c zorder g z* ≥ *c*

shows *zorder* $(\lambda z.\ f\ z - g\ z)\ z \geq c$
proof −
have $(\lambda z.\ -\ g\ z)$ *meromorphic_on* $\{z\}$
by (*auto intro*: *meromorphic_intros assms*)
thus *?thesis*
using *zorder_add_ge*[*of f z* $\lambda z.\ -g\ z\ c$] *assms* **by** *simp*
qed

lemma *zorder_diff1*:
assumes *f meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ f\ z \neq 0)$ *(at z)*
assumes *g meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ g\ z \neq 0)$ *(at z)*
assumes *zorder f z* < *zorder g z*
shows *zorder* $(\lambda z.\ f\ z - g\ z)\ z =$ *zorder f z*
proof −
have *zorder* $(\lambda z.\ f\ z + (-g\ z))\ z =$ *zorder f z*
by (*intro zorder_add1 meromorphic_intros assms*) (*use assms* **in** *auto*)
thus *?thesis*
by *simp*
qed

lemma *zorder_diff2*:
assumes *f meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ f\ z \neq 0)$ *(at z)*
assumes *g meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ g\ z \neq 0)$ *(at z)*
assumes *zorder f z* > *zorder g z*
shows *zorder* $(\lambda z.\ f\ z - g\ z)\ z =$ *zorder g z*
proof −
have *zorder* $(\lambda z.\ f\ z + (-g\ z))\ z =$ *zorder* $(\lambda z.\ -g\ z)\ z$
by (*intro zorder_add2 meromorphic_intros assms*) (*use assms* **in** *auto*)
thus *?thesis*
by *simp*
qed

lemma *zorder_mult*:
assumes *f meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ f\ z \neq 0)$ *(at z)*
assumes *g meromorphic_on* $\{z\}$ *frequently* $(\lambda z.\ g\ z \neq 0)$ *(at z)*
shows *zorder* $(\lambda z.\ f\ z * g\ z)\ z =$ *zorder f z* + *zorder g z*
proof −
from *assms*(*1*) **obtain** *F* **where** *F*: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion F*
by (*auto simp*: *meromorphic_on_def*)
from *assms*(*3*) **obtain** *G* **where** *G*: $(\lambda w.\ g\ (z + w))$ *has_laurent_expansion G*
by (*auto simp*: *meromorphic_on_def*)
have [*simp*]: $F \neq 0\ G \neq 0$
by (*metis assms has_laurent_expansion_eventually_nonzero_iff meromorphic_at_iff*
not_essential_frequently_0_imp_eventually_0 not_eventually not_frequently F G)+
have ∗: *zorder f z* = *fls_subdegree F zorder g z* = *fls_subdegree G*
using *F G assms* **by** (*simp_all add*: *has_laurent_expansion_zorder*)
moreover have *zorder* $(\lambda z.\ f\ z * g\ z)\ z =$ *fls_subdegree* $(F * G)$

```
    using has_laurent_expansion_zorder[OF has_laurent_expansion_mult[OF F
G]] by simp
  moreover have fls_subdegree (F * G) = fls_subdegree F + fls_subdegree G
    using assms by simp
  ultimately show ?thesis
    by (simp add: *)
qed

lemma zorder_divide:
  assumes f meromorphic_on {z} frequently (λz. f z ≠ 0) (at z)
  assumes g meromorphic_on {z} frequently (λz. g z ≠ 0) (at z)
  shows   zorder (λz. f z / g z) z = zorder f z − zorder g z
proof −
  from assms(1) obtain F where F: (λw. f (z + w)) has_laurent_expansion F
    by (auto simp: meromorphic_on_def)
  from assms(3) obtain G where G: (λw. g (z + w)) has_laurent_expansion G
    by (auto simp: meromorphic_on_def)
  have [simp]: F ≠ 0 G ≠ 0
     by (metis assms has_laurent_expansion_eventually_nonzero_iff meromor-
phic_at_iff
       not_essential_frequently_0_imp_eventually_0 not_eventually not_frequently
F G)+
  have *: zorder f z = fls_subdegree F zorder g z = fls_subdegree G
    using F G assms by (simp_all add: has_laurent_expansion_zorder)
  moreover have zorder (λz. f z / g z) z = fls_subdegree (F / G)
   using has_laurent_expansion_zorder[OF has_laurent_expansion_divide[OF F
G]] by simp
  moreover have fls_subdegree (F / G) = fls_subdegree F − fls_subdegree G
    using assms by (simp add: fls_divide_subdegree)
  ultimately show ?thesis
    by (simp add: *)
qed

lemma constant_on_extend_nicely_meromorphic_on:
  assumes f nicely_meromorphic_on B f constant_on A
  assumes open A open B connected B A ≠ {} A ⊆ B
  shows   f constant_on B
proof −
  from assms obtain c where c: ⋀z. z ∈ A ⟹ f z = c
    by (auto simp: constant_on_def)
  have eventually (λz. z ∈ A) (cosparse A)
    by (intro eventually_in_cosparse assms order.refl)
  hence eventually (λz. f z = c) (cosparse A)
    by eventually_elim (use c in auto)
  hence freq: frequently (λz. f z = c) (cosparse A)
    by (intro eventually_frequently) (use assms in auto)
  then obtain z0 where z0: z0 ∈ A frequently (λz. f z = c) (at z0)
    using assms by (auto simp: frequently_def eventually_cosparse_open_eq)
```

```
  have f z = c if z ∈ B for z
  proof (rule frequently_eq_meromorphic_imp_constant[OF _ assms(1)])
    show z0 ∈ B frequently (λz. f z = c) (at z0)
      using z0 assms by auto
  qed (use assms that in auto)
  thus f constant_on B
    by (auto simp: constant_on_def)
qed
```

8.12 More on poles and zeros

```
lemma zorder_prod:
  assumes ⋀x. x ∈ A ⟹ f x meromorphic_on {z}
  assumes eventually (λz. (∏ x∈A. f x z) ≠ 0) (at z)
  shows   zorder (λz. ∏ x∈A. f x z) z = (∑ x∈A. zorder (f x) z)
  using assms
proof (induction A rule: infinite_finite_induct)
  case (insert a A)
  have zorder (λz. ∏ x∈insert a A. f x z) z = zorder (λz. f a z * (∏ x∈A. f x z)) z
    using insert.hyps by simp
  also have ... = zorder (f a) z + zorder (λz. ∏ x∈A. f x z) z
  proof (subst zorder_mult)
    have ∀⇩F z in at z. f a z ≠ 0
      using insert.prems(2) by eventually_elim (use insert.hyps in auto)
    thus ∃⇩F z in at z. f a z ≠ 0
      using eventually_frequently at_neq_bot by blast
  next
    have ∀⇩F z in at z. (∏ x∈A. f x z) ≠ 0
      using insert.prems(2) by eventually_elim (use insert.hyps in auto)
    thus ∃⇩F z in at z. (∏ x∈A. f x z) ≠ 0
      using eventually_frequently at_neq_bot by blast
  qed (use insert.prems in ‹auto intro!: meromorphic_intros›)
  also have zorder (λz. ∏ x∈A. f x z) z = (∑ x∈A. zorder (f x) z)
    by (intro insert.IH) (use insert.prems insert.hyps in ‹auto elim!: eventu-
ally_mono›)
  also have zorder (f a) z + ... = (∑ x∈insert a A. zorder (f x) z)
    using insert.hyps by simp
  finally show ?case .
qed auto

lemma zorder_scale:
  assumes f meromorphic_on {a * z} a ≠ 0
  shows zorder (λw. f (a * w)) z = zorder f (a * z)
proof (cases eventually (λz. f z = 0) (at (a * z)))
  case True
  hence ev: eventually (λz. f (a * z) = 0) (at z)
  proof (rule eventually_compose_filterlim)
    show filterlim ((*) a) (at (a * z)) (at z)
    proof (rule filterlim_atI)
```

```
      show ∀F x in at z. a * x ≠ a * z
        using eventually_neq_at_within[of z z] by eventually_elim (use ‹a ≠ 0› in auto)
    qed (auto intro!: tendsto_intros)
  qed

  have zorder (λw. f (a * w)) z = zorder (λ_. 0) z
    by (rule zorder_cong) (use ev in auto)
  also have ... = zorder (λ_. 0) (a * z)
    by (simp add: zorder_shift')
  also have ... = zorder f (a * z)
    by (rule zorder_cong) (use True in auto)
  finally show ?thesis .
next
  case False
  define G where G = fps_const a * fps_X
  have zorder (f ∘ (λz. a * z)) z = zorder f (a * z) * int (subdegree G)
  proof (rule zorder_compose)
    show isolated_singularity_at f (a * z) ∧ not_essential f (a * z)
      using assms(1) by (auto simp: meromorphic_on_altdef)
  next
    have (λx. a * x) has_fps_expansion G
      unfolding G_def by (intro fps_expansion_intros)
    thus (λx. a * (z + x) - a * z) has_fps_expansion G
      by (simp add: algebra_simps)
  next
    show ∀F w in at (a * z). f w ≠ 0 using False
    by (metis assms(1) has_laurent_expansion_isolated has_laurent_expansion_not_essential
          meromorphic_on_def non_zero_neighbour not_eventually singletonI)
  qed (use ‹a ≠ 0› in ‹auto simp: G_def›)
  also have subdegree G = 1
    using ‹a ≠ 0› by (simp add: G_def)
  finally show ?thesis
    by (simp add: o_def)
qed

lemma zorder_uminus:
  assumes f meromorphic_on {-z}
  shows zorder (λw. f (-w)) z = zorder f (-z)
  using assms zorder_scale[of f -1 z] by simp

lemma is_pole_deriv_iff:
  assumes f meromorphic_on {z}
  shows   is_pole (deriv f) z ⟷ is_pole f z
proof -
  from assms obtain F where F: (λw. f (z + w)) has_laurent_expansion F
    by (auto simp: meromorphic_on_def)
  have deriv (λw. f (z + w)) has_laurent_expansion fls_deriv F
    using F by (rule has_laurent_expansion_deriv)
```

```
  also have deriv (λw. f (z + w)) = (λw. deriv f (z + w))
    by (simp add: deriv_shift_0' add_ac o_def fun_eq_iff)
  finally have F': (λw. deriv f (z + w)) has_laurent_expansion fls_deriv F .
  have is_pole (deriv f) z ⟷ fls_subdegree (fls_deriv F) < 0
    using is_pole_fls_subdegree_iff[OF F'] by simp
  also have ... ⟷ fls_subdegree F < 0
    using fls_deriv_subdegree0 fls_subdegree_deriv linorder_less_linear by fast-
force
  also have ... ⟷ is_pole f z
    using F by (simp add: has_laurent_expansion_imp_is_pole_iff)
  finally show ?thesis .
qed

lemma isolated_zero_remove_sings_iff [simp]:
  assumes isolated_singularity_at f z
  shows   isolated_zero (remove_sings f) z ⟷ isolated_zero f z
proof −
  have ∗: (∀F x in at z. remove_sings f x ≠ 0) ⟷ (∀F x in at z. f x ≠ 0)
  proof
    assume (∀F x in at z. f x ≠ 0)
    thus (∀F x in at z. remove_sings f x ≠ 0)
      using eventually_remove_sings_eq_at[OF assms]
      by eventually_elim auto
  next
    assume (∀F x in at z. remove_sings f x ≠ 0)
    thus (∀F x in at z. f x ≠ 0)
      using eventually_remove_sings_eq_at[OF assms]
      by eventually_elim auto
  qed
  show ?thesis
    unfolding isolated_zero_def using assms ∗ by simp
qed

lemma zorder_isolated_zero_pos:
  assumes isolated_zero f z f analytic_on {z}
  shows   zorder f z > 0
proof (subst zorder_pos_iff' [OF assms(2)])
  show f z = 0
    using assms by (simp add: zero_isolated_zero_analytic)
next
  have ∀F z in at z. f z ≠ 0
    using assms by (auto simp: isolated_zero_def)
  thus ∃F z in at z. f z ≠ 0
    by (simp add: eventually_frequently)
qed

lemma zorder_isolated_zero_pos':
  assumes isolated_zero f z isolated_singularity_at f z
  shows   zorder f z > 0
```

proof −
from *assms*(*1*) **have** *f* −*z*→ *0*
by (*simp add*: *isolated_zero_def*)
with *assms*(*2*) **have** *remove_sings f analytic_on* {*z*}
by (*intro remove_sings_analytic_at*)
hence *zorder* (*remove_sings f*) *z* > *0*
using *assms* **by** (*intro zorder_isolated_zero_pos*) *auto*
thus *?thesis*
using *assms* **by** *simp*
qed

lemma *zero_isolated_zero_nicely_meromorphic*:
assumes *isolated_zero f z f nicely_meromorphic_on* {*z*}
shows *f z* = *0*
proof −
have ¬*is_pole f z*
using *assms pole_is_not_zero* **by** *blast*
with *assms*(*2*) **have** *f analytic_on* {*z*}
by (*simp add*: *nicely_meromorphic_on_imp_analytic_at*)
with *zero_isolated_zero_analytic assms*(*1*) **show** *?thesis*
by *blast*
qed

lemma *meromorphic_on_imp_not_zero_cosparse*:
assumes *f meromorphic_on A*
shows *eventually* (λ*z*. ¬*isolated_zero f z*) (*cosparse A*)
proof −
have *eventually* (λ*z*. ¬*is_pole* (λ*z*. *inverse* (*f z*)) *z*) (*cosparse A*)
by (*intro meromorphic_on_imp_not_pole_cosparse meromorphic_intros assms*)
thus *?thesis*
by (*simp add*: *is_pole_inverse_iff*)
qed

lemma *nicely_meromorphic_on_inverse* [*meromorphic_intros*]:
assumes *f nicely_meromorphic_on A*
shows (λ*x*. *inverse* (*f x*)) *nicely_meromorphic_on A*
unfolding *nicely_meromorphic_on_def*
proof (*intro conjI ballI*)
fix *z* **assume** *z*: *z* ∈ *A*
have *is_pole f z* ∧ *f z* = *0* ∨ *f* −*z*→ *f z*
using *assms z* **by** (*auto simp*: *nicely_meromorphic_on_def*)
thus *is_pole* (λ*x*. *inverse* (*f x*)) *z* ∧ *inverse* (*f z*) = *0* ∨
(λ*x*. *inverse* (*f x*)) −*z*→ *inverse* (*f z*)
proof
assume *is_pole f z* ∧ *f z* = *0*
hence *isolated_zero* (λ*z*. *inverse* (*f z*)) *z* ∧ *inverse* (*f z*) = *0*
by (*auto simp*: *isolated_zero_inverse_iff*)
hence (λ*x*. *inverse* (*f x*)) −*z*→ *inverse* (*f z*)
by (*simp add*: *isolated_zero_def*)

```
    thus ?thesis ..
  next
    assume lim: f −z→ f z
    hence ana: f analytic_on {z}
      using assms is_pole_def nicely_meromorphic_on_imp_analytic_at
          not_tendsto_and_filterlim_at_infinity trivial_limit_at z by blast
    show ?thesis
    proof (cases isolated_zero f z)
      case True
      with lim have f z = 0
        using continuous_within zero_isolated_zero by blast
      with True have is_pole (λz. inverse (f z)) z ∧ inverse (f z) = 0
        by (auto simp: is_pole_inverse_iff)
      thus ?thesis ..
    next
      case False
      hence f z ≠ 0 ∨ (f z = 0 ∧ eventually (λz. f z = 0) (at z))
        using non_isolated_zero_imp_eventually_zero[OF ana] by blast
      thus ?thesis
      proof (elim disjE conjE)
        assume f z ≠ 0
        hence (λz. inverse (f z)) −z→ inverse (f z)
          by (intro tendsto_intros lim)
        thus ?thesis ..
      next
        assume ∗: f z = 0 eventually (λz. f z = 0) (at z)
        have eventually (λz. inverse (f z) = 0) (at z)
          using ∗(2) by eventually_elim auto
        hence (λz. inverse (f z)) −z→ 0
          by (simp add: tendsto_eventually)
        with ∗(1) show ?thesis
          by auto
      qed
    qed
  qed
qed (use assms in ‹auto simp: nicely_meromorphic_on_def intro!: meromorphic_intros›)

lemma is_pole_zero_at_nicely_mero:
  assumes f nicely_meromorphic_on A is_pole f z z ∈ A
  shows f z = 0
  by (meson assms at_neq_bot
      is_pole_def nicely_meromorphic_on_def
      not_tendsto_and_filterlim_at_infinity)

lemma zero_or_pole:
  assumes mero: f nicely_meromorphic_on A
    and z ∈ A f z = 0 and event: ∀F x in at z. f x ≠ 0
  shows isolated_zero f z ∨ is_pole f z
proof −
```

```
  from mero ‹z∈A›
  have (is_pole f z ∧ f z=0) ∨ f −z→ f z
    unfolding nicely_meromorphic_on_def by simp
  moreover have isolated_zero f z if f −z→ f z
    unfolding isolated_zero_def
    using ‹f z=0› that event tendsto_nhds_iff by auto
  ultimately show ?thesis by auto
qed

lemma isolated_zero_fls_subdegree_iff:
  assumes (λx. f (z + x)) has_laurent_expansion F
  shows isolated_zero f z ⟷ fls_subdegree F > 0
  using assms unfolding isolated_zero_def
 by (metis Lim_at_zero fls_zero_subdegree has_laurent_expansion_eventually_nonzero_iff
not_le
       order.refl tendsto_0_subdegree_iff_0)

lemma zorder_pos_imp_isolated_zero:
  assumes f meromorphic_on {z} eventually (λz. f z ≠ 0) (at z) zorder f z > 0
  shows   isolated_zero f z
  using assms isolated_zero_fls_subdegree_iff
  by (metis has_laurent_expansion_eventually_nonzero_iff
      has_laurent_expansion_zorder insertI1
      meromorphic_on_def)

lemma zorder_neg_imp_is_pole:
  assumes f meromorphic_on {z} eventually (λz. f z ≠ 0) (at z) zorder f z < 0
  shows   is_pole f z
  using assms is_pole_fls_subdegree_iff at_neq_bot eventually_frequently mero-
morphic_at_iff
       neg_zorder_imp_is_pole by blast

lemma not_pole_not_isolated_zero_imp_zorder_eq_0:
  assumes f meromorphic_on {z} ¬is_pole f z ¬isolated_zero f z frequently (λz.
f z ≠ 0) (at z)
  shows   zorder f z = 0
proof −
  have remove_sings f analytic_on {z}
   using assms meromorphic_at_iff not_essential_def remove_sings_analytic_at
by blast
  moreover from this and assms have remove_sings f z ≠ 0
   using isolated_zero_def meromorphic_at_iff non_zero_neighbour remove_sings_eq_0_iff
by blast
  moreover have frequently (λz. remove_sings f z ≠ 0) (at z)
    using assms analytic_at_neq_imp_eventually_neq calculation(1,2)
      eventually_frequently trivial_limit_at by blast
  ultimately have zorder (remove_sings f) z = 0
    using zorder_eq_0_iff by blast
  thus ?thesis
```

```
    using assms(1) meromorphic_at_iff by auto
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 (intro non_isolated_zero_imp_eventually_zero′ analytic_intros assms) auto
  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 (λw. 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 (λx. 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 (λx. f (g x)) z
      by (rule is_pole_compose) fact+
    thus ?thesis
      by (auto simp: not_essential_def)
  qed
qed


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* (λ*w*. *g w* − *g z* = *0*) (*nhds z*)
by (*intro non_isolated_zero_imp_eventually_zero′*) (*use assms* **in** ‹*auto intro*!: *analytic_intros*›)
hence *isolated_singularity_at* (λ*x*. *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* (λ*w*. *g w* ∈ *ball* (*g z*) *r*) (*at z*)
using ‹*r* > *0*› **unfolding** *tendsto_def* **by** *force*
moreover have *eventually* (λ*w*. *g w* ≠ *g z*) (*at z*) **using** *True*
by (*auto simp*: *isolated_zero_def elim*!: *eventually_mono*)
ultimately have *eventually* (λ*w*. *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* (λ*x*. *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*:
fixes *f* :: *'a* :: *perfect_space* ⇒ *'b* :: *real_normed_div_algebra*
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*)
have *eventually* (λ*z*. *z* ∈ *A* − {*x*}) (*at x*)
by (*intro eventually_at_in_open assms*)
hence *eventually* (λ*z*. *f z* = *c*) (*at x*)
by *eventually_elim* (*use c* **in** *auto*)
hence *f* −*x*→ *c*
using *tendsto_eventually* **by** *blast*
moreover from *assms* **have** *f* −*x*→ *0*
by (*simp add*: *isolated_zero_def*)
ultimately have [*simp*]: *c* = *0*
using *tendsto_unique*[*of at x f c 0*] **by** (*simp add*: *at_neq_bot*)

have *eventually* (λ*x*. *f x* ≠ *0*) (*at x*)
using *assms* **by** (*auto simp*: *isolated_zero_def*)
moreover have *eventually* (λ*x*. *x* ∈ *A*) (*at x*)
using *assms* **by** (*intro eventually_at_in_open'*) *auto*
ultimately have *eventually* (λ*x*. *False*) (*at x*)
by *eventually_elim* (*use c* **in** *auto*)
thus *False*
by *simp*
qed

lemma *nicely_meromorphic_on_cosparse_eq_imp_eq*:
assumes ∀ ≈*x*∈*A*. *f x* = *g x f nicely_meromorphic_on A g nicely_meromorphic_on A*
assumes *open A x* ∈ *A*
shows *f x* = *g x*
proof −
have ∗: *is_pole f x* ⟷ *is_pole g x*
by (*intro is_pole_cong*) (*use assms* **in** ‹*auto simp*: *eventually_cosparse_open_eq*›)

show *?thesis*
proof (*cases is_pole f x*)
case *True*
thus *?thesis* **using** ∗ *assms*(*2*,*3*)[*THEN is_pole_zero_at_nicely_mero*[*of* _ *A x*]] *assms*(*3*−) **by** *auto*
next
case *False*
with ∗ **have** ∗∗: *f analytic_on* {*x*} *g analytic_on* {*x*}
using *assms nicely_meromorphic_on_imp_analytic_at* **by** *blast*+
show *?thesis*
using *assms*(*1*) ∗∗ **by** (*rule analytic_on_continuation*) (*use assms*(*5*) **in** *auto*)
qed
qed

8.13 A relation for the zero order of a function

The following is a relational version of the statement *zorder f z = n*, i.e. the fact that a complex-valued function f has a zero of multiplicity n at z, where $n = 0$ means that f does not in fact have a zero there, and $n < 0$ means that f has a pole of multiplicity $-n$ there.

The difference to the mere statement that *zorder f z = n* is that this also includes the fact that f is in fact meromorphci at z and that it is not identically zero in the pointed neighbourhood of z, i.e. that the *zorder* is in fact well-defined. This is very useful when proving that the *zorder* of a function built by multiplying together other functions has a certain value.

Note that (just like *zorder*), our relation does not care about the value $f(z)$ itself, but only about the behaviour of f in the pointed neighbourhood of z.

definition *has_zorder* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ *int* ⇒ *bool* **where**
has_zorder f z n ⟷ *f meromorphic_on* {*z*} ∧ *eventually* (λ*z*. *f z* ≠ *0*) (*at z*) ∧ *zorder f z* = *n*

lemma *has_zorderI*:
assumes *f meromorphic_on* {*z*} *eventually* (λ*z*. *f z* ≠ *0*) (*at z*) *n* = *zorder f z*
shows *has_zorder f z n*
using *assms* **unfolding** *has_zorder_def* **by** *auto*

lemma
assumes *has_zorder f z n*
shows *has_zorder_imp_meromorphic_at*: *f meromorphic_on* {*z*}
and *has_zorder_imp_eventually_nonzero*: *eventually* (λ*z*. *f z* ≠ *0*) (*at z*)
and *has_zorder_imp_zorder_eq*: *zorder f z* = *n*
using *assms* **unfolding** *has_zorder_def* **by** *blast*+

lemma *has_laurent_expansion_imp_has_zorder*:
assumes (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F F* ≠ *0 fls_subdegree F* = *n*
shows *has_zorder f z n*

unfolding *has_zorder_def*
proof *safe*
show *f meromorphic_on* {*z*}
using *assms*(*1*) **unfolding** *meromorphic_on_def* **by** *auto*
have ¬*frequently* (λ*z*. *f z* = *0*) (*at z*)
using *assms* **by** (*subst has_laurent_expansion_frequently_zero_iff*)
thus *eventually* (λ*z*. *f z* ≠ *0*) (*at z*)
by (*auto simp*: *frequently_def*)
show *zorder f z* = *n*
using *assms has_laurent_expansion_zorder* **by** *blast*
qed

lemma *has_zorder_altdef*:
has_zorder f z n ⟷ (∃ *F*. (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F* ∧ *F* ≠ *0*
∧ *fls_subdegree F* = *n*)
proof
assume *has_zorder f z n*
then obtain *F* **where** *F*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F*
and *f*: *eventually* (λ*z*. *f z* ≠ *0*) (*at z*) *zorder f z* = *n*
by (*auto simp*: *has_zorder_def meromorphic_on_def*)
from *f*(*1*) **and** *F* **have** *F* ≠ *0*
using *has_laurent_expansion_eventually_nonzero_iff* **by** *blast*
have *fls_subdegree F* = *n* **using** *F f*
by (*simp add*: *has_laurent_expansion_eventually_nonzero_iff has_laurent_expansion_zorder*)
thus (∃ *F*. (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F* ∧ *F* ≠ *0* ∧ *fls_subdegree*
F = *n*)
using *F* **and** ‹*F* ≠ *0*› **by** *blast*
qed (*use has_laurent_expansion_imp_has_zorder* **in** *auto*)

lemma *has_zorder_shift*:
has_zorder f z n ⟷ *has_zorder* (λ*w*. *f* (*z* + *w*)) *0 n*
unfolding *has_zorder_def*
by (*simp add*: *has_zorder_def meromorphic_on_def at_to_0′ zorder_shift′*
eventually_filtermap add_ac)

lemma *has_zorder_shift′*:
NO_MATCH 0 z ⟹ *has_zorder f z n* ⟷ *has_zorder* (λ*w*. *f* (*z* + *w*)) *0 n*
by (*rule has_zorder_shift*)

lemma *has_zorder_transfer_eventually_at*:
assumes *has_zorder f z n eventually* (λ*w*. *f w* = *g w*) (*at z*)
shows *has_zorder g z n*
proof −
from *assms* **obtain** *F* **where** *F*: (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F F*
≠ *0 fls_subdegree F* = *n*
by (*auto simp*: *has_zorder_altdef*)
show *?thesis*
proof (*rule has_laurent_expansion_imp_has_zorder*)
have (λ*w*. *f* (*z* + *w*)) *has_laurent_expansion F* ⟷ (λ*w*. *g* (*z* + *w*)) *has_laurent_expansion*

```
F
      by (rule has_laurent_expansion_cong') (use assms in auto)
    with F show (λw. g (z + w)) has_laurent_expansion F
      by auto
  qed (use F in auto)
qed

lemma has_zorder_cong_ev:
  assumes eventually (λw. f w = g w) (at z) z = z' m = n
  shows   has_zorder f z m ⟷ has_zorder g z' n
 using has_zorder_transfer_eventually_at[of f z n g] has_zorder_transfer_eventually_at[of
g z n f] assms
  by (auto simp: eq_commute)


lemma has_zorder_imp_is_pole_iff:
  assumes has_zorder f z n
  shows   is_pole f z ⟷ n < 0
proof −
 obtain F where F: (λw. f (z + w)) has_laurent_expansion F F ≠ 0 fls_subdegree
F = n
    using assms by (auto simp: has_zorder_altdef)
  thus ?thesis
    by (simp add: has_laurent_expansion_imp_is_pole_iff)
qed

lemma has_zorder_imp_isolated_zero_iff:
  assumes has_zorder f z n
  shows   isolated_zero f z ⟷ n > 0
proof −
 obtain F where F: (λw. f (z + w)) has_laurent_expansion F F ≠ 0 fls_subdegree
F = n
    using assms by (auto simp: has_zorder_altdef)
  thus ?thesis
    by (metis isolated_zero_fls_subdegree_iff)
qed

lemma has_zorder_analytic_imp_isolated_zero_iff:
  assumes has_zorder f z n f analytic_on {z}
  shows   f z = 0 ⟷ n > 0
proof −
 obtain F where F: (λw. f (z + w)) has_laurent_expansion F F ≠ 0 fls_subdegree
F = n
    using assms by (auto simp: has_zorder_altdef)
  from F and assms have fls_subdegree F ≥ 0
    using analytic_at_imp_no_pole has_zorder_imp_is_pole_iff by fastforce
  with ‹F ≠ 0› have fls_nth F 0 = 0 ⟷ fls_subdegree F > 0
   by (metis fls_eq0_below_subdegree nth_fls_subdegree_nonzero not_le order_antisym_conv)
  also have fls_nth F 0 = f z
```

```
    by (rule has_laurent_expansion_analytic_imp_fls_nth_0_eq[OF assms(2)])
fact
  finally show ?thesis
    by (auto simp: F(3))
qed

lemma has_zorder_compose:
  assumes has_zorder f (g z) n (λw. g (z + w) − g z) has_fps_expansion G G
≠ 0
  assumes k = int (subdegree G) ∗ n
  shows   has_zorder (f ∘ g) z k
proof −
 obtain F where F: (λw. f (g z + w)) has_laurent_expansion F F ≠ 0 fls_subdegree
F = n
    using assms(1) by (auto simp: has_zorder_altdef)
  have [simp]: fps_nth G 0 = 0
    using assms(2) has_fps_expansion_imp_0_eq_fps_nth_0 by fastforce

  show ?thesis
  proof (rule has_laurent_expansion_imp_has_zorder)
    have ((λw. f (g z + w)) ∘ (λw. g (z + w) − g z)) has_laurent_expansion
          fls_compose_fps F G using assms
      by (intro laurent_expansion_intros F has_laurent_expansion_fps assms
fps_expansion_intros) auto
    thus (λx. (f ∘ g) (z + x)) has_laurent_expansion fls_compose_fps F G
      by (simp add: o_def)
  next
    show fls_compose_fps F G ≠ 0
      by (subst fls_compose_fps_eq_0_iff) (use assms F in auto)
  next
    show fls_subdegree (fls_compose_fps F G) = k
      by (auto simp: ‹k = _› ‹fls_subdegree F = _›)
  qed
qed

lemma has_zorder_compose_scale:
  assumes has_zorder f (c ∗ z) n c ≠ 0
  shows   has_zorder (λz. f (c ∗ z)) z n
proof −
  have has_zorder (f ∘ (λz. c ∗ z)) z n
    using assms(1)
  proof (rule has_zorder_compose)
    have (λw. c ∗ w) has_fps_expansion (fps_const c ∗ fps_X)
      by (intro fps_expansion_intros)
    thus (λw. c ∗ (z + w) − c ∗ z) has_fps_expansion (fps_const c ∗ fps_X)
      by (auto simp: algebra_simps)
  qed (auto simp: ‹c ≠ 0›)
  thus ?thesis
    by (simp add: o_def)
```

qed

lemma *has_zorder_compose_shift*:
assumes *has_zorder f* (*z* + *c*) *n*
shows *has_zorder* (λ*z*. *f* (*z* + *c*)) *z n*
using *assms* **by** (*simp add*: *has_zorder_shift' add_ac*)

named_theorems *zorder_intros*

lemma *has_zorder_remove_sings* [*zorder_intros*]:
assumes *has_zorder f z n*
shows *has_zorder* (*remove_sings f*) *z n*
using *assms*
proof (*rule has_zorder_transfer_eventually_at*)
from *assms* **have** *mero*: *f meromorphic_on* {*z*}
by (*rule has_zorder_imp_meromorphic_at*)
show *eventually* (λ*w*. *f w* = *remove_sings f w*) (*at z*)
using *eventually_remove_sings_eq*[*OF mero*] **by** (*simp add*: *eq_commute*)
qed

lemma *analytic_imp_has_zorder_0*:
assumes *f analytic_on* {*z*} *f z* ≠ *0*
shows *has_zorder f z 0*
unfolding *has_zorder_def*
proof *safe*
from *assms*(*1*) **show** *f meromorphic_on* {*z*}
by (*simp add*: *analytic_on_imp_meromorphic_on*)
show *eventually* (λ*w*. *f w* ≠ *0*) (*at z*)
using *assms analytic_at_neq_imp_eventually_neq* **by** *blast*
show *zorder f z* = *0*
using *assms zorder_eq_0I* **by** *blast*
qed

lemma *has_zorder_const* [*zorder_intros*]:
assumes *c* ≠ *0*
shows *has_zorder* (λ_. *c*) *z 0*
by (*rule has_laurent_expansion_imp_has_zorder*[*of* _ _ *fls_const c*]) (*use assms* **in** *auto*)

lemma *has_zorder_ident_0* [*zorder_intros*]: *has_zorder* (λ*z*. *z*) *0 1*
by (*rule has_laurent_expansion_imp_has_zorder*[*of* _ _ *fls_X*])
(*auto intro*!: *laurent_expansion_intros*)

lemma *has_zorder_linear* [*zorder_intros*]:
assumes *n* = (*if z* = *w then 1 else 0*)
shows *has_zorder* (λ*z*. *z* − *w*) *z n*
proof (*cases z* = *w*)
case *False*

thus *?thesis*
by (*auto intro*!: *analytic_imp_has_zorder_0 analytic_intros simp*: *assms*)
next
case [*simp*]: *True*
show *?thesis*
by (*simp add*: *assms has_zorder_shift' has_zorder_ident_0*)
qed

lemma *has_zorder_ident* [*zorder_intros*]: $n = (\mathit{if}\ w = 0\ \mathit{then}\ 1\ \mathit{else}\ 0) \Longrightarrow$ *has_zorder* $(\lambda z.\ z)\ w\ n$
using *has_zorder_linear*[*of n w 0*] **by** *simp*

lemma *has_zorder_mult* [*zorder_intros*]:
assumes *has_zorder f z m has_zorder g z n k* $= m + n$
shows *has_zorder* $(\lambda z.\ f\ z * g\ z)\ z\ k$
proof −
obtain *F* **where** *F*: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion F F* $\neq 0$ *fls_subdegree F* $= m$
using *assms*(*1*) **by** (*auto simp*: *has_zorder_altdef*)
obtain *G* **where** *G*: $(\lambda w.\ g\ (z + w))$ *has_laurent_expansion G G* $\neq 0$ *fls_subdegree G* $= n$
using *assms*(*2*) **by** (*auto simp*: *has_zorder_altdef*)
show *?thesis*
proof (*rule has_laurent_expansion_imp_has_zorder*)
show $(\lambda w.\ f\ (z + w) * g\ (z + w))$ *has_laurent_expansion* $(F * G)$
by (*intro laurent_expansion_intros F G*)
qed (*use F G* ‹$k = m + n$› **in** *simp_all*)
qed

lemma *has_zorder_cmult_iff*: $c \neq 0 \Longrightarrow$ *has_zorder* $(\lambda z.\ c * f\ z)\ z\ n \longleftrightarrow$ *has_zorder f z n*
using *has_zorder_mult*[*OF _ has_zorder_const refl*, *of f z n c*]
has_zorder_mult[*OF _ has_zorder_const refl*, *of* $\lambda z.\ c * f\ z\ z\ n\ \mathit{inverse}\ c$]
by (*auto simp*: *field_simps*)

lemma *has_zorder_uminus* [*zorder_intros*]:
assumes *has_zorder f z n*
shows *has_zorder* $(\lambda z.\ -f\ z)\ z\ n$
using *has_zorder_mult*[*OF assms has_zorder_const*[*of* -1] *refl*] **by** *simp*

lemma *has_zorder_uminus_iff*: *has_zorder* $(\lambda z.\ -f\ z)\ z\ n \longleftrightarrow$ *has_zorder f z n*
using *has_zorder_cmult_iff*[*of* -1 *f z n*] **by** *simp*

lemma *has_zorder_power_int* [*zorder_intros*]:
assumes *has_zorder f z n k* $= m * n$
shows *has_zorder* $(\lambda z.\ f\ z\ \mathit{powi}\ m)\ z\ k$
proof −
obtain *F* **where** *F*: $(\lambda w.\ f\ (z + w))$ *has_laurent_expansion F F* $\neq 0$ *fls_subdegree F* $= n$

```
    using assms(1) by (auto simp: has_zorder_altdef)
  show ?thesis
  proof (rule has_laurent_expansion_imp_has_zorder)
    show (λw. f (z + w) powi m) has_laurent_expansion (F powi m)
      by (intro laurent_expansion_intros F)
  qed (use F ‹k = m * n› in simp_all)
qed

lemma has_zorder_power [zorder_intros]:
  assumes has_zorder f z n k = int m * n
  shows   has_zorder (λz. f z ^ m) z k
  using has_zorder_power_int[OF assms] by simp

lemma has_zorder_inverse [zorder_intros]:
  assumes has_zorder f z n k = -n
  shows   has_zorder (λz. inverse (f z)) z k
  using has_zorder_power_int[OF assms(1), of -n -1] assms(2) by simp

lemma has_zorder_divide [zorder_intros]:
  assumes has_zorder f z m has_zorder g z n k = m - n
  shows   has_zorder (λz. f z / g z) z k
 using has_zorder_mult[OF assms(1) has_zorder_inverse[OF assms(2) refl] refl]
  by (simp add: field_simps assms(3))

lemma has_zorder_prod [zorder_intros]:
  assumes ⋀x. x ∈ A ⟹ has_zorder (f x) z (g x)
  shows   has_zorder (λz. ∏x∈A. f x z) z (∑x∈A. g x)
  using assms
  by (induction A rule: infinite_finite_induct) (auto intro!: zorder_intros)

lemma has_zorder_prod_list [zorder_intros]:
  assumes ⋀x. x ∈ set xs ⟹ has_zorder (f x) z (g x)
  shows   has_zorder (λz. ∏x←xs. f x z) z (∑x←xs. g x)
  using assms by (induction xs) (auto intro!: zorder_intros)

lemma has_zorder_prod_mset [zorder_intros]:
  assumes ⋀x. x ∈# A ⟹ has_zorder (f x) z (g x)
  shows   has_zorder (λz. ∏x∈#A. f x z) z (∑x∈#A. g x)
  using assms by (induction A) (auto intro!: zorder_intros)

lemma has_zorder_deriv:
  assumes has_zorder f z n n ≠ 0
  shows   has_zorder (deriv f) z (n - 1)
proof -
  from assms obtain F
    where F: (λw. f (z + w)) has_laurent_expansion F F ≠ 0 fls_subdegree F =
n
    by (auto simp: has_zorder_altdef)
  have (λw. deriv (λw. f (z + w)) w) has_laurent_expansion fls_deriv F
```

```
    by (intro laurent_expansion_intros F)
  hence F': (λw. deriv f (z + w)) has_laurent_expansion fls_deriv F
    by (simp add: deriv_shift_0' o_def add_ac)

  have fls_nth F n ≠ 0
    using F by auto
  hence fls_nth (fls_deriv F) (n - 1) ≠ fls_nth 0 (n - 1)
    using ‹n ≠ 0› by auto
  hence fls_deriv F ≠ 0
    by metis

  show ?thesis
  proof (rule has_laurent_expansion_imp_has_zorder)
    have deriv (λw. f (z + w)) has_laurent_expansion fls_deriv F
      by (intro laurent_expansion_intros F(1))
    also have deriv (λw. f (z + w)) = (λw. deriv f (z + w))
      by (rule ext) (simp add: deriv_shift_0' o_def add_ac)
    finally show (λw. deriv f (z + w)) has_laurent_expansion fls_deriv F .
  next
    show fls_subdegree (fls_deriv F) = n - 1
      using F assms(2) by (subst fls_subdegree_deriv) auto
  qed fact
qed

lemma has_zorder_logderiv:
  assumes has_zorder f z n n ≠ 0
  shows   has_zorder (λz. deriv f z / f z) z (-1)
  by (rule has_zorder_divide has_zorder_deriv assms)+ auto


lemma has_zorder_sin_0: has_zorder sin 0 1
proof (rule has_laurent_expansion_imp_has_zorder[of _ _ fps_to_fls (fps_sin
1)])
  have subdegree (fps_sin 1 :: complex fps) = 1
    by (rule subdegreeI) auto
  thus fls_subdegree (fps_to_fls (fps_sin 1 :: complex fps)) = 1
    by (simp add: fls_subdegree_fls_to_fps)
next
  have fps_nth (fps_sin 1 :: complex fps) 1 ≠ fps_nth (0 :: complex fps) 1
    by auto
  hence (fps_sin 1 :: complex fps) ≠ 0
    by metis
  thus fps_to_fls (fps_sin 1 :: complex fps) ≠ 0
    by simp
qed (auto intro!: has_laurent_expansion_fps fps_expansion_intros)

lemma has_zorder_sin_1: has_zorder sin (of_int n * of_real pi) 1
proof -
  have [simp]: sin (complex_of_int n * complex_of_real pi) = 0
```

```
    by (induction n) (auto simp: algebra_simps sin_diff)
  have [simp]: cos (complex_of_int n * complex_of_real pi) = (-1) powi n
   by (induction n) (auto simp: algebra_simps cos_diff power_int_diff field_simps)
  have has_zorder (λw. (-1) powi n * sin w) 0 1
    by (auto intro!: zorder_intros has_zorder_sin_0)
  thus ?thesis
    by (simp add: has_zorder_shift' sin_add)
qed

lemma has_zorder_cos_1: has_zorder cos (of_real (pi/2) + of_int n * of_real
pi) 1
proof -
  have has_zorder (λw. -sin w) (of_int n * of_real pi) 1
    unfolding has_zorder_uminus_iff by (rule has_zorder_sin_1)
  also have ?this ⟷ has_zorder cos (of_real (pi/2) + of_int n * of_real pi) 1
    by (simp add: has_zorder_shift' cos_sin_eq flip: sin_minus)
  finally show ?thesis .
qed

lemma has_zorder_Gamma [zorder_intros]:
  assumes n = (if z ∈ ℤ≤0 then -1 else 0)
  shows   has_zorder Gamma z n
proof (cases z ∈ ℤ≤0)
  case False
  thus ?thesis
     by (auto simp: assms Gamma_nonzero intro!: analytic_imp_has_zorder_0
analytic_intros)
next
  case True
  then obtain n where z_eq: z = -of_nat n
    by (elim nonpos_Ints_cases')
  have [simp]: 1 - z ∉ ℤ≤0
    by (auto elim!: nonpos_Ints_cases simp: z_eq complex_eq_iff)

  have 1: complex_of_real pi ≠ 0
    by auto
  have 2: has_zorder (λz. rGamma (1 - z)) z 0
    by (rule analytic_imp_has_zorder_0)
       (use True in ‹auto intro!: analytic_intros rGamma_nonzero simp: non-
pos_Ints_altdef›)
  have 3: has_zorder (λz. sin (complex_of_real pi * z)) z 1
    by (rule has_zorder_compose_scale)
      (use has_zorder_sin_1[of -n] in ‹simp_all add: z_eq mult_ac›)
  have has_zorder (λz. of_real pi * rGamma (1 - z) / sin (of_real pi * z)) z
(-1)
    by (rule zorder_intros 1 2 3 refl)+ auto
  also have ?this ⟷ has_zorder Gamma z (-1)
  proof (rule has_zorder_cong_ev)
    have ℤ sparse_in (UNIV :: complex set)
```

```
    by (rule sparse_subset_Ints) auto
  hence eventually (λw. w ∉ ℤ) (at z)
    by (auto simp: sparse_in_eventually_iff)
  thus ∀F w in at z. of_real pi * rGamma (1 - w) / sin (of_real pi * w) =
Gamma w
  proof eventually_elim
    case (elim w)
    have 1 - w ∉ ℤ≤0
      using elim by auto
    hence Gamma (1 - w) ≠ 0
      by (auto simp: Gamma_nonzero)
    moreover have sin (w * complex_of_real pi) ≠ 0
      using elim sin_eq_0 by auto
    ultimately show ?case
    using Gamma_reflection_complex[of w] by (auto simp: field_simps rGamma_inverse_Gamma)
  qed
 qed auto
 finally show ?thesis
  using True by (auto simp: assms)
qed

lemma has_zorder_rGamma [zorder_intros]:
 assumes n = (if z ∈ ℤ≤0 then 1 else 0)
 shows   has_zorder rGamma z n
 unfolding rGamma_inverse_Gamma by (rule zorder_intros refl)+ (use assms
in auto)

end
```

9 The Weierstraß Factorisation Theorem

```
theory Weierstrass_Factorization
 imports Meromorphic
begin

no_notation Infinite_Set_Sum.abs_summable_on (infix ‹abs'_summable'_on›
50)
```

9.1 The elementary factors

The Weierstraß elementary factors are the family of entire functions

$$E_n(z) = (1-z)\exp\left(z + \frac{z^2}{2} + \ldots + \frac{z^n}{n}\right)$$

with the key properties that they have a single zero at $z = 1$ and satisfy $E_n(z) = 1 + O(z^{n+1})$ around the origin.

```
definition weierstrass_factor :: nat ⇒ complex ⇒ complex where
```

weierstrass_factor n z = (1 − z) ∗ exp (∑ k=1..n. z ^ k / of_nat k)

lemma *weierstrass_factor_continuous_on* [*continuous_intros*]:
continuous_on A f ⟹ continuous_on A (λz. weierstrass_factor n (f z))
by (*auto simp*: *weierstrass_factor_def intro*!: *continuous_intros*)

lemma *weierstrass_factor_holomorphic* [*holomorphic_intros*]:
f holomorphic_on A ⟹ (λz. weierstrass_factor n (f z)) holomorphic_on A
by (*auto simp*: *weierstrass_factor_def intro*!: *holomorphic_intros*)

lemma *weierstrass_factor_analytic* [*analytic_intros*]:
f analytic_on A ⟹ (λz. weierstrass_factor n (f z)) analytic_on A
by (*auto simp*: *weierstrass_factor_def intro*!: *analytic_intros*)

lemma *weierstrass_factor_0* [*simp*]: *weierstrass_factor n 0 = 1*
by (*auto simp*: *weierstrass_factor_def power_0_left*)

lemma *weierstrass_factor_1* [*simp*]: *weierstrass_factor n 1 = 0*
by (*simp add*: *weierstrass_factor_def*)

lemma *weierstrass_factor_eq_0_iff* [*simp*]: *weierstrass_factor n z = 0 ⟷ z = 1*
by (*simp add*: *weierstrass_factor_def*)

lemma *zorder_weierstrass_factor* [*simp*]: *zorder (weierstrass_factor n) 1 = 1*
proof (*rule zorder_eqI*)
show *(λz. −exp (∑ k=1..n. z ^ k / of_nat k)) holomorphic_on UNIV*
by (*intro holomorphic_intros*) *auto*
qed (*auto simp*: *weierstrass_factor_def algebra_simps*)

lemma *has_zorder_weierstrass_factor* [*zorder_intros*]: *has_zorder (weierstrass_factor n) 1 1*
by (*rule has_zorderI*)
(*auto intro*!: *analytic_on_imp_meromorphic_on analytic_intros eventually_neq_at_within*)

lemma *has_field_derivative_weierstrass_factor*:
assumes *(f has_field_derivative f′) (at z within A)*
assumes *f z ≠ 1*
shows *((λz. weierstrass_factor n (f z)) has_field_derivative*
(f′ ∗ weierstrass_factor n (f z) ∗ f z ^ n / (f z − 1)))
(at z within A)
proof −
define *W* **where** *W = weierstrass_factor n (f z)*
have *((λz. weierstrass_factor n (f z)) has_field_derivative*
(f′ ∗ ((∑ n=1..n. f z ^ (n − 1)) ∗ W − exp (∑ k=1..n. f z ^ k / of_nat k)))) (at z within A)
unfolding *weierstrass_factor_def W_def*
by (*auto intro*!: *derivative_eq_intros assms*(*1*) *simp*: *algebra_simps*
simp flip: *sum_distrib_left sum_distrib_right*)

```
  also have exp (∑k=1..n. f z ^ k / of_nat k) = W / (1 - f z)
    using ‹f z ≠ 1› by (auto simp: weierstrass_factor_def W_def)
  also have (∑ n=1..n. f z ^ (n - 1)) = (∑ k<n. f z ^ k)
    by (rule sum.reindex_bij_witness[of _ λn. n+1 λn. n-1]) auto
  also have ... = (f z ^ n - 1) / (f z - 1)
    by (subst geometric_sum) (use ‹f z ≠ 1› in auto)
  also have (f z ^ n - 1) / (f z - 1) * W - W / (1 - f z) = W * f z ^ n / (f z
- 1)
    using ‹f z ≠ 1› by (auto simp: divide_simps) (auto simp: algebra_simps)?
  finally show ?thesis
    unfolding W_def by (simp add: mult_ac)
qed

lemma weierstrass_factor_bound:
  assumes norm z ≤ 1 / 2
  shows   norm (weierstrass_factor n z - 1) ≤ 3 * norm z ^ Suc n
proof (cases n = 0 ∨ z = 0)
  case True
  thus ?thesis
  proof
    assume n = 0
    thus ?thesis by (auto simp: weierstrass_factor_def)
  qed auto
next
  case False
  with assms have z ≠ 1 n > 0 z ≠ 0
    by auto

  have summable (λk. cmod z ^ (k + Suc n) / real (k + Suc n))
    using ln_series'[of -norm z] assms
    by (intro summable_norm summable_ignore_initial_segment)
       (simp_all add: sums_iff summable_minus_iff power_minus' norm_divide
norm_power)
  also have ?this ⟷ summable (λk. norm z ^ Suc n * (norm z ^ k / real (k +
Suc n)))
    by (simp add: power_add mult_ac)
  also have ... ⟷ summable (λk. norm z ^ k / real (k + Suc n))
    by (subst summable_cmult_iff) (use ‹z ≠ 0› in auto)
  finally have summable: summable (λk. norm z ^ k / real (k + Suc n)) .

  have (λk. z ^ k / of_nat k) sums - Ln (1 - z)
    using sums_minus[OF Ln_series[of -z]] assms by (simp add: power_minus')
  hence (λk. z ^ (k + Suc n) / of_nat (k + Suc n)) sums (- Ln (1 - z) -
(∑ k<Suc n. z ^ k / of_nat k))
    by (intro sums_split_initial_segment)
  also have (∑ k<Suc n. z ^ k / of_nat k) = (∑ k=1..n. z ^ k / of_nat k)
    by (intro sum.mono_neutral_right) auto
  finally have norm (ln (1 - z) + (∑ k=1..n. z ^ k / of_nat k)) =
                 norm (∑ k. z ^ (k + Suc n) / of_nat (k + Suc n))
```

```
    by (simp add: sums_iff norm_uminus_minus)

  also have ... ≤ (∑k. norm (z ^ (k + Suc n) / of_nat (k + Suc n)))
    using ln_series'[of -norm z] assms
    by (intro summable_norm summable_ignore_initial_segment)
       (simp_all add: sums_iff summable_minus_iff power_minus' norm_divide
norm_power)
  also have ... = (∑k. norm z ^ Suc n * (norm z ^ k / real (k + Suc n)))
    by (simp add: algebra_simps norm_mult norm_power norm_divide power_add
del: of_nat_add of_nat_Suc)
  also have ... = norm z ^ Suc n * (∑k. norm z ^ k / real (k + Suc n))
    by (intro suminf_mult summable)
  also have ... ≤ norm z ^ Suc n * (1 / (1 - norm z))
  proof (intro mult_left_mono[OF sums_le])
    show (λk. norm z ^ k / real (k + Suc n)) sums (∑k. norm z ^ k / real (k +
Suc n))
      using summable by blast
    show (λk. norm z ^ k) sums (1 / (1 - norm z))
      by (rule geometric_sums) (use assms in auto)
  qed (auto simp: field_simps)
  also have norm z ^ Suc n * (1 / (1 - norm z)) ≤ norm z ^ Suc n * (1 / (1 -
(1 / 2)))
    using assms by (intro mult_mono power_mono divide_left_mono diff_mono
mult_pos_pos) auto
  also have ... = 2 * norm z ^ Suc n
    by simp
  finally have norm_le: norm (ln (1 - z) + (∑k=1..n. z ^ k / of_nat k)) ≤ 2
* norm z ^ Suc n .

  also have ... ≤ 2 * norm z ^ 2
    using ‹n > 0› assms by (intro mult_left_mono power_decreasing) auto
  also have ... ≤ 2 * (1 / 2) ^ 2
    by (intro mult_left_mono assms power_mono) auto
  finally have norm_le': norm (ln (1 - z) + (∑k=1..n. z ^ k / of_nat k)) ≤ 1
/ 2
    by (simp add: power2_eq_square)

  have weierstrass_factor n z = exp (ln (1 - z) + (∑k=1..n. z ^ k / of_nat k))
    using ‹z ≠ 1› by (simp add: exp_add weierstrass_factor_def)
  also have norm (... - 1) ≤ (3 / 2) * norm (ln (1 - z) + (∑k=1..n. z ^ k /
of_nat k))
    by (intro norm_exp_bounds norm_le')
  also have ... ≤ (3 / 2) * (2 * norm z ^ Suc n)
    by (intro mult_left_mono norm_le) auto
  finally show ?thesis
    by simp
qed
```

9.2 Infinite products of elementary factors

We now show that the elementary factors can be used to construct an entire function with its zeros at a certain set of points (given by a sequence of non-zero numbers a_n with no accumulation point).

locale *weierstrass_product* =
  fixes *a* :: *'a* ⇒ *complex*
  fixes *p* :: *'a* ⇒ *nat*
  fixes *I* :: *'a set*
  assumes *a_nonzero*: ⋀*x*. *x* ∈ *I* ⟹ *a x* ≠ *0*
  assumes *filterlim_a*: *filterlim a at_infinity* (*inf cofinite* (*principal I*))
  assumes *summable_a_p*: ⋀*r*. *r* > *0* ⟹ (λ*x*. (*r* / *norm* (*a x*)) ^ *Suc* (*p x*)) *summable_on I*
begin

Due to the assumptions on a, every point can only occur a finite number of times. Moreover, the range of a is a sparse set (i.e. it has no limit points) and is therefore in particular a closed set.

lemma *finite_occs_a*: *finite* (*a* −‘ {*z*} ∩ *I*)
proof −
  have *eventually* (λ*n*. *norm* (*a n*) > *norm z*) (*inf cofinite* (*principal I*))
    using *filterlim_a* **by** (*metis filterlim_at_infinity_imp_norm_at_top filterlim_at_top_dense*)
  hence *finite* {*x* ∈ *I*. *norm* (*a x*) ≤ *norm z*}
    by (*auto simp*: *eventually_inf_principal eventually_cofinite not_less*)
  thus *?thesis*
    by (*rule finite_subset* [*rotated*]) *auto*
qed

lemma *not_islimpt_range_a*: ¬*z islimpt a* ‘ *I*
proof −
  define *r* **where** *r* = *norm z* + *1*
  define *U* **where** *U* = *ball* (*0*::*complex*) *r*
  have *finite* (*U* ∩ *a* ‘ *I*)
  proof −
    have *eventually* (λ*n*. *norm* (*a n*) ≥ *r*) (*inf cofinite* (*principal I*))
      using *filterlim_a* **by** (*metis filterlim_at_infinity_imp_norm_at_top filterlim_at_top*)
    hence *finite* {*x* ∈ *I*. *norm* (*a x*) < *r*}
      by (*auto simp*: *eventually_inf_principal eventually_cofinite not_le*)
    also have {*x*∈*I*. *norm* (*a x*) < *r*} = *a* −‘ *U* ∩ *I*
      by (*auto simp*: *U_def*)
    finally have *finite* (*a* −‘ *U* ∩ *I*) **.**
    hence *finite* (*a* ‘ (*a* −‘ *U* ∩ *I*))
      by (*rule finite_imageI*)
    also have *a* ‘ (*a* −‘ *U* ∩ *I*) = *U* ∩ *a* ‘ *I*
      by *auto*
    finally show *?thesis* **.**

```
  qed
  moreover have open U z ∈ U
    by (auto simp: U_def r_def)
  ultimately show ?thesis
    unfolding islimpt_eq_acc_point by blast
qed

lemma closed_range_a: closed (a ‘ I)
  using not_islimpt_range_a unfolding closed_limpt by blast
```

We construct the product by multiplying the Weierstraß factors with the appropriate weight.

```
definition f :: complex ⇒ complex where
  f z = (∏∞x∈I. weierstrass_factor (p x) (z / a x))
```

The product converges absolutely.

```
lemma abs_multipliable: (λx. weierstrass_factor (p x) (z / a x)) abs_multipliable_on
I
  unfolding abs_multipliable_on_iff_summable_on
proof −
  define I1 where I1 = {x∈I. norm (z / a x) ≤ 1 / 2}
  define I2 where I2 = {x∈I. norm (z / a x) > 1 / 2}
  have finite I2
  proof −
    have eventually (λx. norm (a x) ≥ 2 ∗ norm z) (inf cofinite (principal I))
      using filterlim_a unfolding filterlim_at_infinity_conv_norm_at_top filter-
lim_at_top
      by blast
    hence finite {x ∈ I. norm (a x) < 2 ∗ norm z}
      by (auto simp: eventually_inf_principal eventually_cofinite not_le)
    thus finite I2
    by (rule finite_subset [rotated]) (auto simp: I2_def norm_divide divide_simps)
  qed

  have (λn. weierstrass_factor (p n) (z / a n) − 1) abs_summable_on (I1 ∪ I2)
  proof (rule summable_on_Un_disjoint)
    show (λx. weierstrass_factor (p x) (z / a x) − 1) abs_summable_on I2
      by (rule summable_on_finite) fact
  next
    show (λx. weierstrass_factor (p x) (z / a x) − 1) abs_summable_on I1
    proof (rule summable_on_comparison_test)
      have (λx. 3 ∗ norm (z / a x) ^ Suc (p x)) summable_on I
        using summable_on_cmult_right[OF summable_a_p[of norm z], of 3]
        by (cases z = 0) (auto simp: norm_divide)
      thus (λx. 3 ∗ norm (z / a x) ^ Suc (p x)) summable_on I1
        by (rule summable_on_subset) (auto simp: I1_def)
    next
      fix x assume x: x ∈ I1
      hence norm (z / a x) ≤ 1 / 2
```

```
      by (auto simp: norm_divide I1_def)
    thus norm (weierstrass_factor (p x) (z / a x) - 1) ≤ 3 * cmod (z / a x) ^
Suc (p x)
      by (rule weierstrass_factor_bound)
  qed auto
 qed (auto simp: I1_def I2_def)
 also have I1 ∪ I2 = I
  by (auto simp: I1_def I2_def)
 finally show (λn. weierstrass_factor (p n) (z / a n) - 1) abs_summable_on I
.
qed

lemma multipliable: (λx. weierstrass_factor (p x) (z / a x)) multipliable_on I
 using abs_multipliable by (rule abs_multipliable_multipliable)

lemma has_setprod: ((λx. weierstrass_factor (p x) (z / a x)) has_setprod f z) I
 using multipliable[of z] unfolding f_def by auto
```

On compact sets, it converges uniformly.

```
context
 fixes P
 defines P ≡ (λJ z. ∏x∈J. weierstrass_factor (p x) (z / a x))
begin

lemma uniform_limit:
 assumes compact K
 shows   uniform_limit K P f (finite_subsets_at_top I)
proof -
 have ∗: uniform_limit (cball 0 R) P f (finite_subsets_at_top I) if R > 0 for R
 proof -
  define I1 where I1 = {x∈I. R ≤ norm (a x) / 2}
  show ?thesis
   unfolding P_def f_def
  proof (rule uniform_limit_infprod_M_test_strong)
   show finite (I - I1)
   proof -
    have eventually (λx. norm (a x) ≥ 2 * R) (inf cofinite (principal I))
      using filterlim_a unfolding filterlim_at_infinity_conv_norm_at_top
filterlim_at_top
     by blast
    hence finite {x ∈ I. norm (a x) < 2 * R}
     by (auto simp: eventually_inf_principal eventually_cofinite not_le)
    thus finite (I - I1)
      by (rule finite_subset [rotated]) (auto simp: I1_def norm_divide di-
vide_simps)
   qed
  next
   fix x z assume x: x ∈ I1 and z: z ∈ cball (0::complex) R
   from x z have norm (weierstrass_factor (p x) (z / a x) - 1) ≤ 3 * (norm
```

```
(z / a x)) ^ Suc (p x)
      by (intro weierstrass_factor_bound) (auto simp: norm_divide divide_simps
I1_def)
    also have ... = 3 * (norm z / norm (a x)) ^ Suc (p x)
      by (simp add: norm_divide)
    also have ... ≤ 3 * (R / norm (a x)) ^ Suc (p x)
     by (intro mult_left_mono power_mono divide_right_mono) (use z in auto)
    finally show norm (weierstrass_factor (p x) (z / a x) - 1) ≤
                 3 * (R / norm (a x)) ^ Suc (p x) by simp
  next
    show (λx. 3 * (R / norm (a x)) ^ Suc (p x)) summable_on I1
    by (rule summable_on_cmult_right, rule summable_on_subset, rule summable_a_p)
        (auto simp: I1_def ‹R > 0›)
  next
    fix x assume x: x ∈ I - I1
    have compact ((λz. weierstrass_factor (p x) (z / a x) - 1) ‘ cball 0 R)
      by (intro compact_continuous_image continuous_intros) (use x in ‹auto
simp: a_nonzero›)
    thus bounded ((λz. weierstrass_factor (p x) (z / a x) - 1) ‘ cball 0 R)
      by (rule compact_imp_bounded)
  qed
qed

from assms have bounded K
  by (rule compact_imp_bounded)
then obtain R where R: K ⊆ cball 0 R
  by (auto simp: bounded_iff cball_def)
show ?thesis
  by (rule uniform_limit_on_subset[OF *[of max R 1]]) (use R in auto)
qed
```

Since the factors are entire, so is the product.

```
lemma analytic [analytic_intros]: f analytic_on A
proof -
  have f analytic_on {z} for z
  proof -
    define r where r = norm z + 1
    have r: r > 0
      by (auto simp: r_def add_nonneg_pos)
    have ∀F J in finite_subsets_at_top I. finite J ∧ J ⊆ I
      by (rule eventually_finite_subsets_at_top_weakI) auto
    hence *: ∀F J in finite_subsets_at_top I. continuous_on (cball 0 r) (P J) ∧
P J holomorphic_on ball 0 r
      by eventually_elim (auto simp: P_def a_nonzero intro!: continuous_intros
holomorphic_intros)
    have f holomorphic_on ball 0 r using r holomorphic_uniform_limit[OF *
uniform_limit]
      by auto
    hence f analytic_on ball 0 r
```

```
    by (simp add: analytic_on_open)
  thus f analytic_on {z}
    by (rule analytic_on_subset) (auto simp: r_def)
 qed
 thus f analytic_on A
  by (metis analytic_on_analytic_at)
qed

lemma holomorphic [holomorphic_intros]: f holomorphic_on A
 using analytic by (rule analytic_imp_holomorphic)

end
```

The zeros of the product are precisely the numbers in the range of a.

```
lemma zero: f z = 0 ⟷ z ∈ a ` I
proof −
 have ∗: (λx. weierstrass_factor (p x) (z / a x)) strongly_multipliable_on I
  using abs_multipliable by (rule abs_multipliable_on_imp_strongly_multipliable_on)
 show ?thesis
  using has_setprod_eq_0_iff[OF ∗ has_setprod] a_nonzero by auto
qed

lemma isolated_zero:
 assumes z ∈ a ` I
 shows   isolated_zero f z
proof −
 have eventually (λz. f z ≠ 0) (at z)
  using not_islimpt_range_a[of z] by (auto simp: islimpt_iff_eventually zero)
 moreover have f −z→ f z
  by (intro isContD analytic_at_imp_isCont analytic)
 hence f −z→ 0
  using assms zero[of z] by auto
 ultimately show ?thesis
  by (auto simp: isolated_zero_def)
qed
```

The multiplicity of each zero is the number of times it appears in the range of a.

```
theorem has_zorder: has_zorder f z (card (a −` {z} ∩ I))
proof −
 define g where g = (λz x. weierstrass_factor (p x) (z / a x))
 define I1 where I1 = {x∈I. a x ≠ z}
 define I2 where I2 = a −` {z} ∩ I
 have finite I2
  using finite_occs_a[of z] unfolding I2_def .
 hence prod2: (g w has_setprod (∏ x∈I2. g w x)) I2 for w
  by (rule has_setprod_finite)
 have has_zorder (λw. ∏ x∈I2. g w x) z (∑ x∈I2. 1)
 proof (intro zorder_intros)
```

```
    fix x assume x: x ∈ I2
    have [simp]: a x ≠ 0
      using a_nonzero[of x] x by (auto simp: I2_def)
    have has_zorder (λz. weierstrass_factor (p x) z) 1 1
      by (rule zorder_intros)
    hence *: has_zorder (λz. weierstrass_factor (p x) z) (1 / a x * z) 1
      using x by (auto simp: I2_def)
    show has_zorder (λz. g z x) z 1
      using has_zorder_compose_scale[OF *] by (simp add: g_def)
  qed
  hence zorder2: has_zorder (λw. ∏ x∈I2. g w x) z (card I2)
    by simp

  have abs_mult1: g w abs_multipliable_on I1 for w
    by (rule abs_multipliable_on_subset[of _ I])
       (use abs_multipliable in ‹auto simp: g_def I1_def›)
  hence prod1: (g w has_setprod (∏∞x∈I1. g w x)) I1 for w
    by (intro has_setprod_infprod abs_multipliable_multipliable)
  have zorder1: has_zorder (λw. ∏∞x∈I1. g w x) z 0
  proof (rule analytic_imp_has_zorder_0)
    show (λw. infprod (g w) I1) analytic_on {z}
    proof −
      interpret I1: weierstrass_product a p I1
      proof
        show filterlim a at_infinity (inf cofinite (principal I1))
          by (intro filterlim_mono[OF filterlim_a] inf_mono) (auto simp: I1_def)
      next
        fix r :: real assume r: r > 0
        show (λx. (r / cmod (a x)) ^ Suc (p x)) summable_on I1
          by (rule summable_on_subset [OF summable_a_p[OF r]]) (auto simp:
I1_def)
      qed (auto simp: a_nonzero I1_def)
      show ?thesis
        using I1.analytic[of {z}] by (simp add: I1.f_def [abs_def] g_def)
    qed
  next
    have (∏∞x∈I1. g z x) = 0 ⟷ (∃x∈I1. g z x = 0)
      by (rule has_setprod_eq_0_iff)
      (use prod1 abs_mult1 in ‹auto intro: abs_multipliable_on_imp_strongly_multipliable_on›)
    thus infprod (g z) I1 ≠ 0
      by (auto simp: g_def I1_def)
  qed

  have has_zorder (λw. (∏∞x∈I1. g w x) * (∏ x∈I2. g w x)) z (card I2)
    using has_zorder_mult[OF zorder1 zorder2 refl] by simp
  also have (λw. (∏∞x∈I1. g w x) * (∏ x∈I2. g w x)) = f
  proof
    fix w :: complex
    have ((λx. g w x) has_setprod ((∏∞x∈I1. g w x) * (∏ x∈I2. g w x))) (I1 ∪
```

```
I2)
    by (intro has_setprod_Un_disjoint prod1 prod2) (auto simp: I1_def I2_def)
  also have I1 ∪ I2 = I
    by (auto simp: I1_def I2_def)
  finally show (∏∞x∈I1. g w x) * (∏x∈I2. g w x) = f w
    using has_setprod_unique[OF _ has_setprod[of w]] unfolding g_def by
blast
 qed
 finally show ?thesis
  by (simp add: I2_def)
qed

lemma zorder: zorder f z = card (a -` {z} ∩ I)
 using has_zorder by (simp add: has_zorder_def)
```

We also derive a nice form for the logarithmic derivative, which is a uniformly convergent sum.

```
lemma uniform_limit_logderiv:
 assumes K: compact K K ∩ a ` I = {}
 shows uniform_limit K (λJ z. ∑x∈J. (z / a x) ^ p x / (z - a x)) (λz. deriv f
z / f z)
       (finite_subsets_at_top I)
proof -
 define g where g = (λx z. weierstrass_factor (p x) (z / a x))
 have uniform_limit K (λJ z. ∑x∈J. deriv (g x) z / g x z) (λz. deriv f z / f z)
       (finite_subsets_at_top I)
 proof (rule logderiv_infprod_uniform_limit)
  show K ⊆ -(a ` I)
   using K by auto
  show compact K
   by fact
  show open (-a ` I)
   using closed_range_a by blast
  show g x holomorphic_on - a ` I if x ∈ I for x
   by (auto simp: g_def intro!: holomorphic_intros)
  show f z ≠ 0 if z ∈ -a ` I for z
   using that by (auto simp: zero)
  show uniform_limit L (λJ x. ∏xa∈J. g xa x) f (finite_subsets_at_top I) if
compact L for L
   unfolding g_def by (rule uniform_limit) fact
 qed
 also have ?this ⟷ uniform_limit K (λJ z. ∑x∈J. (z / a x) ^ p x / (z - a
x)) (λz. deriv f z / f z)
                  (finite_subsets_at_top I)
 proof (rule uniform_limit_cong)
  have ∀F J in finite_subsets_at_top I. finite J ∧ J ⊆ I
   by (rule eventually_finite_subsets_at_top_weakI) auto
  show ∀F J in finite_subsets_at_top I.
       ∀z∈K. (∑x∈J. deriv (g x) z / g x z) = (∑x∈J. (z / a x) ^ p x / (z -
```

```
a x))
    proof (intro eventually_finite_subsets_at_top_weakI ballI sum.cong refl)
      fix J z x
      assume J: finite J J ⊆ I and z: z ∈ K and x: x ∈ J
      have neq: z ≠ a x
        using J z x K by auto
      have [simp]: a x ≠ 0
        by (rule a_nonzero) (use J x in auto)
      note [derivative_intros] = has_field_derivative_weierstrass_factor
      have ((λz. g x z) has_field_derivative (g x z * (z / a x) ^ p x / (a x * (z /
a x - 1)))) (at z)
        unfolding g_def using neq by (auto intro!: derivative_eq_intros simp:
a_nonzero)
      also have (a x * (z / a x - 1)) = z - a x
        using neq by (simp add: field_simps)
      finally have deriv (g x) z = g x z * (z / a x) ^ p x / (z - a x)
        by (rule DERIV_imp_deriv)
      also have ... / g x z = (z / a x) ^ p x / (z - a x)
        by (simp add: g_def)
      finally show deriv (g x) z / g x z = (z / a x) ^ p x / (z - a x) .
    qed
  qed auto
  finally show ?thesis .
qed

lemma has_sum_logderiv:
  assumes z ∉ a ' I
  shows ((λx. (z / a x) ^ p x / (z - a x)) has_sum (deriv f z / f z)) I
proof -
  have uniform_limit {z} (λJ z. ∑x∈J. (z / a x) ^ p x / (z - a x)) (λz. deriv f
z / f z)
         (finite_subsets_at_top I)
    by (rule uniform_limit_logderiv) (use assms in auto)
  thus ?thesis
    by (simp add: has_sum_def)
qed

theorem logderiv_eq:
  assumes z ∉ a ' I
  shows   deriv f z / f z = (∑∞x∈I. (z / a x) ^ p x / (z - a x))
  using has_sum_logderiv[OF assms] by (simp add: has_sum_iff)

end
```

The following locale is the case where we take the product over a sequence of points $(a_n)_{n\geq 0}$ with $|a_n| \to \infty$ and we choose $p(n) = n$, which is guaranteed to work.

```
locale weierstrass_product' =
  fixes a :: nat ⇒ complex
```

```
  assumes a_nonzero': ⋀n. a n ≠ 0
  assumes filterlim_a': filterlim a at_infinity at_top
begin

sublocale weierstrass_product a λn. n UNIV
proof
  show a n ≠ 0 for n :: nat
    using a_nonzero' by auto
next
  show filterlim a at_infinity (inf cofinite (principal UNIV))
    using filterlim_a' by (simp add: cofinite_eq_sequentially)
next
  fix r :: real assume r: r > 0

  have summable (λn. (r / norm (a n)) ^ Suc n)
  proof (rule summable_comparison_test_ev)
    have eventually (λn. norm (a n) > 2 * r) at_top
      using filterlim_a' by (metis filterlim_at_infinity_imp_norm_at_top filter-
lim_at_top_dense)
    thus eventually (λn. norm ((r / norm (a n)) ^ Suc n) ≤ (1 / 2) ^ Suc n)
at_top
    proof eventually_elim
      case (elim n)
      have norm ((r / norm (a n)) ^ Suc n) = (r / norm (a n)) ^ Suc n
        using ‹r > 0› by (simp add: abs_mult)
      also have ... ≤ (1 / 2) ^ Suc n
        using ‹r > 0› elim by (intro power_mono) (auto simp: divide_simps)
      finally show ?case .
    qed
  next
    show summable (λn. (1 / 2) ^ Suc n :: real)
      unfolding summable_Suc_iff by (intro summable_geometric) auto
  qed
  thus (λx. (r / norm (a x)) ^ Suc x) summable_on UNIV
    by (rule summable_nonneg_imp_summable_on) (use r in auto)
qed

end
```

9.3 Writing a quotient as an exponential

If two holomorphic functions f and g on a simply connected domain have the same zeros with the same multiplicities, they can be written as $g(x) = e^{h(x)} f(x)$ for some holomorphic function $h(x)$.

```
lemma holomorphic_zorder_factorization:
  assumes g holomorphic_on A open A connected A
  assumes f holomorphic_on A ⋀z. z ∈ A ⟹ f z = 0 ⟷ g z = 0
          ⋀z. z ∈ A ⟹ zorder f z = zorder g z
```

```
  obtains h where h holomorphic_on A ⋀z. z ∈ A ⟹ h z ≠ 0 ⋀z. z ∈ A ⟹
g z = h z * f z
proof (cases ∃z∈A. g z ≠ 0)
  case False
  show ?thesis
    by (rule that[of λ_. 1]) (use False assms in auto)
next
  case True
  define F where F = fps_expansion f
  define G where G = fps_expansion g
   define c where c = (λz. fps_nth (G z) (subdegree (G z)) / fps_nth (F z)
(subdegree (F z)))
  define h where h = (λz. if f z = 0 then c z else g z / f z)

  have ev_nonzero: eventually (λw. g w ≠ 0 ∧ w ∈ A) (at z) if z ∈ A for z
  proof −
    from True obtain z0 where z0: z0 ∈ A g z0 ≠ 0
      by auto
    show ?thesis
      by (rule non_zero_neighbour_alt[where ?β = z0])
        (use assms that z0 in ‹auto intro: simply_connected_imp_connected›)
  qed

  have g_ana: g analytic_on {z} if z ∈ A for z
    using assms ‹open A› analytic_at that by blast
  have f_ana: f analytic_on {z} if z ∈ A for z
    using assms ‹open A› analytic_at that by blast

  have F: (λw. f (z + w)) has_fps_expansion F z if z ∈ A for z
   unfolding F_def by (rule analytic_at_imp_has_fps_expansion[OF f_ana[OF
that]])
  have G: (λw. g (z + w)) has_fps_expansion G z if z ∈ A for z
   unfolding G_def by (rule analytic_at_imp_has_fps_expansion[OF g_ana[OF
that]])

  have [simp]: G z ≠ 0 if z ∈ A for z
  proof
    assume G z = 0
    hence eventually (λw. g w = 0) (at z) using G[OF that]
        by (auto simp: has_fps_expansion_0_iff at_to_0' eventually_filtermap
add_ac
                  eventually_at_filter nhds_to_0' elim: eventually_mono)
    hence eventually (λ_. False) (at z)
     using ev_nonzero[OF that] unfolding eventually_at_filter by eventually_elim
auto
    thus False
      by simp
  qed
  have [simp]: F z ≠ 0 if z ∈ A for z
```

```
  proof
    assume F z = 0
    hence eventually (λw. f w = 0) (at z) using F[of z] that
        by (auto simp: has_fps_expansion_0_iff at_to_0' eventually_filtermap
add_ac
                    eventually_at_filter nhds_to_0' elim: eventually_mono)
    hence eventually (λ_. False) (at z)
      using ev_nonzero[OF that] unfolding eventually_at_filter
      by eventually_elim (use assms in auto)
    thus False
      by simp
  qed
  have [simp]: c z ≠ 0 if z ∈ A for z
    using that by (simp add: c_def)

  have h: h analytic_on {z} ∧ h z ≠ 0 if z ∈ A for z
  proof −
    show ?thesis
    proof (cases f z = 0)
      case False
      from False that have (λz. g z / f z) analytic_on {z}
        by (intro analytic_intros g_ana f_ana) auto
      also have ?this ⟷ h analytic_on {z}
      proof (rule analytic_at_cong)
        have eventually (λw. f w ≠ 0) (nhds z)
          using ev_nonzero[OF ‹z ∈ A›] unfolding eventually_at_filter
          by eventually_elim (use False ‹z ∈ A› assms in auto)
        thus eventually (λz. g z / f z = h z) (nhds z)
          by eventually_elim (auto simp: h_def)
      qed auto
      finally have h analytic_on {z} .
      moreover have h z ≠ 0
        using that assms by (simp add: h_def)
      ultimately show ?thesis by blast
    next
      case True
      with that have z: z ∈ A f z = 0
        by auto
      have zorder f z = int (subdegree (F z))
        using F by (rule has_fps_expansion_zorder) (use z in auto)
      also have zorder f z = zorder g z
        using z assms by auto
      also have zorder g z = subdegree (G z)
        using G by (rule has_fps_expansion_zorder) (use z in auto)
      finally have subdegree_eq: subdegree (F z) = subdegree (G z)
        by simp

      have (λw. if w = 0 then c z else g (z + w) / f (z + w)) has_fps_expansion
G z / F z (is ?P)
```

```
      using subdegree_eq z by (intro has_fps_expansion_divide F G) (auto simp:
c_def)
    also have ?this ⟷ (λw. h (z + w)) has_fps_expansion G z / F z
    proof (intro has_fps_expansion_cong)
      have eventually (λw. w ≠ z ⟶ f w ≠ 0) (nhds z)
        using ev_nonzero[OF ‹z ∈ A›] unfolding eventually_at_filter
        by eventually_elim (use ‹z ∈ A› assms in auto)
      hence eventually (λw. w ≠ 0 ⟶ f (z + w) ≠ 0) (nhds 0)
        by (simp add: nhds_to_0' eventually_filtermap)
      thus eventually (λw. (if w = 0 then c z else g (z + w) / f (z + w)) = h (z
+ w)) (nhds 0)
        unfolding h_def by eventually_elim (use z in ‹auto simp: c_def h_def›)
    qed auto
    finally have h analytic_on {z}
      using has_fps_expansion_imp_analytic by blast
    moreover have h z ≠ 0
      using that z by (auto simp: h_def c_def)
    ultimately show ?thesis by blast
  qed
qed

from h have h_ana: h analytic_on A and h_nz: ∀z∈A. h z ≠ 0
  using analytic_on_analytic_at by blast+
moreover have g z = h z * f z if z ∈ A for z
  using assms that by (auto simp: h_def)
ultimately show ?thesis
  using ‹open A› by (intro that[of h]) (auto simp: analytic_on_open)
qed
```

9.4 Constructing the sequence of zeros

The form of the Weierstraß Factorisation Theorem that we derived above requires an explicit sequence of the zeros that tends to infinity. We will now show that under mild conditions, such a sequence always exists.

More precisely: if A is an infinite closed set that is sparse in the sense that its intersection with any compact set is finite, then there exists an injective sequence f enumerating the values of A in ascending order by absolute value, and f tends to infinity for $n \to \infty$.

```
lemma sequence_of_sparse_set_exists:
  fixes A :: complex set
  assumes infinite A closed A ⋀r. r ≥ 0 ⟹ finite (A ∩ cball 0 r)
  obtains f :: nat ⇒ complex
    where mono (norm ∘ f) inj f range f = A filterlim f at_infinity at_top
proof −
  have ∃f::nat ⇒ complex. ∀n.
        f n ∈ A ∧
        f n ∉ f ‘ {..<n} ∧
        {z∈A. norm z < norm (f n)} ⊆ f ‘ {..<n} ∧
```

```
      (∀k<n. norm (f k) ≤ norm (f n))
  proof (rule dependent_wf_choice[OF wf], goal_cases)
    case (1 f g n r)
    thus ?case by auto
  next
    case (2 n f)
    have f: f k ∈ A {z ∈ A. norm z < norm (f k)} ⊆ f ` {..<k} ∀k'<k. cmod (f
k') ≤ cmod (f k)
      if k < n for k
      using 2[of k] that by simp_all

    have infinite (A − f ` {..<n})
      using assms(1) by (intro Diff_infinite_finite) auto
    then obtain z0 where z0: z0 ∈ A − f ` {..<n}
      by (meson finite.intros(1) finite_subset subsetI)
    have finite (A ∩ cball 0 (norm z0))
      by (intro assms(3)) auto
    hence finite (A ∩ cball 0 (norm z0) − f ` {..<n})
      using finite_subset by blast
    moreover from z0 have A ∩ cball 0 (norm z0) − f ` {..<n} ≠ {}
      by auto
    ultimately obtain z where is_arg_min norm (λz. z ∈ A ∩ cball 0 (norm
z0) − f ` {..<n}) z
      using ex_is_arg_min_if_finite by blast
    hence z: z ∈ A norm z ≤ norm z0 z ∉ f ` {..<n}
          ⋀w. w ∈ A ⟹ norm w ≤ norm z0 ⟹ w ∉ f ` {..<n} ⟹ norm w ≥
norm z
      by (auto simp: is_arg_min_def)

    show ?case
    proof (rule exI[of _ z], safe)
      fix w assume w: w ∈ A norm w < norm z
      with z(4)[of w] z w show w ∈ f ` {..<n}
        by linarith
    next
      fix k assume k: k < n
      show norm (f k) ≤ norm z
        using f(2)[of k] z(1,3) k by auto
    qed (use z in auto)
  qed
  then obtain f :: nat ⇒ complex where f:
    ⋀n. f n ∈ A
    ⋀n. f n ∉ f ` {..<n}
    ⋀n. {z∈A. norm z < norm (f n)} ⊆ f ` {..<n}
    ⋀k n. k < n ⟹ norm (f k) ≤ norm (f n)
    by meson
  from f(2) have f_neq: f n ≠ f k if k < n for k n
    using that by blast
```

have *inj*: *inj f*
proof (*rule injI*)
  fix *m n* :: *nat*
  assume *f m* = *f n*
  thus *m* = *n*
    using *f_neq*[*of m n*] *f_neq*[*of n m*] **by** (*cases m n rule*: *linorder_cases*) *auto*
qed

have *range*: *range f* = *A*
proof *safe*
  fix *z* **assume** *z*: *z* ∈ *A*
  show *z* ∈ *range f*
  proof (*rule ccontr*)
    assume *z* ∉ *range f*
    hence *norm* (*f n*) ≤ *norm z* **for** *n*
      using *f*(*3*)[*of n*] *z* **by** *auto*
    hence *range f* ⊆ *A* ∩ *cball 0* (*norm z*)
      using *f*(*1*) **by** *auto*
    moreover have *finite* (*A* ∩ *cball 0* (*norm z*))
      by (*intro assms*) *auto*
    ultimately have *finite* (*range f*)
      using *finite_subset* **by** *blast*
    moreover have *infinite* (*range f*)
      using *inj* **by** (*subst finite_image_iff*) *auto*
    ultimately show *False* **by** *contradiction*
  qed
qed (*use f*(*1*) **in** *auto*)

have *mono*: *mono* (*norm* ∘ *f*)
proof (*rule monoI*, *unfold o_def*)
  fix *m n* :: *nat*
  assume *m* ≤ *n*
  thus *norm* (*f m*) ≤ *norm* (*f n*)
    using *f*(*4*)[*of m n*] **by** (*cases m* < *n*) *auto*
qed

have ¬*bounded A*
proof
  assume *bounded A*
  hence *bdd_above* (*norm* ‘ *A*)
    by (*meson bdd_above_norm*)
  hence *norm z* ≤ *Sup* (*norm* ‘ *A*) **if** *z* ∈ *A* **for** *z*
    using *that* **by** (*meson cSUP_upper*)
  hence *A* ⊆ *cball 0* (*Sup* (*norm* ‘ *A*))
    by *auto*
  also have ... ⊆ *cball 0* (*max 1* (*Sup* (*norm* ‘ *A*)))
    by *auto*
  finally have *A* ⊆ *A* ∩ *cball 0* (*max 1* (*Sup* (*norm* ‘ *A*)))
    by *blast*

```
    moreover have finite (A ∩ cball 0 (max 1 (Sup (norm ‘ A))))
      by (intro assms) auto
    ultimately have finite A
      using finite_subset by blast
    hence finite (range f)
      by (simp add: range)
    thus False
      using inj by (subst (asm) finite_image_iff) auto
  qed

  have lim: filterlim f at_infinity at_top
    unfolding filterlim_at_infinity_conv_norm_at_top filterlim_at_top
  proof
    fix C :: real
    from ‹¬bounded A› obtain z where z ∈ A norm z > C
      unfolding bounded_iff by (auto simp: not_le)
    obtain n where [simp]: z = f n
      using range ‹z ∈ A› by auto
    show eventually (λn. norm (f n) ≥ C) at_top
      using eventually_ge_at_top[of n]
    proof eventually_elim
      case (elim k)
      have C ≤ norm (f n)
        using ‹norm z > C› by simp
      also have ... ≤ norm (f k)
        using monoD[OF ‹mono (norm ∘ f)›, of n k] elim by auto
      finally show ?case .
    qed
  qed

  show ?thesis
    by (intro that[of f] inj range mono lim)
qed


lemma strict_mono_sequence_partition:
  assumes strict_mono (f :: nat ⇒ 'a :: {linorder, no_top})
  assumes x ≥ f 0
  assumes filterlim f at_top at_top
  shows   ∃k. x ∈ {f k..<f (Suc k)}
proof −
  define k where k = (LEAST k. f (Suc k) > x)
  {
    obtain n where x ≤ f n
      using assms by (auto simp: filterlim_at_top eventually_at_top_linorder)
    also have f n < f (Suc n)
      using assms by (auto simp: strict_mono_Suc_iff)
    finally have ∃n. f (Suc n) > x by auto
  }
```

```
  from LeastI_ex[OF this] have x < f (Suc k)
    by (simp add: k_def)
  moreover have f k ≤ x
  proof (cases k)
    case (Suc k')
    have k ≤ k' if f (Suc k') > x
      using that unfolding k_def by (rule Least_le)
    with Suc show f k ≤ x by (cases f k ≤ x) (auto simp: not_le)
  qed (use assms in auto)
  ultimately show ?thesis by auto
qed


lemma strict_mono_sequence_partition':
  assumes strict_mono (f :: nat ⇒ 'a :: {linorder, no_top})
  assumes x ≥ f 0
  assumes filterlim f at_top at_top
  shows   ∃!k. x ∈ {f k..<f (Suc k)}
proof (rule ex_ex1I)
  show ∃k. x ∈ {f k..<f (Suc k)}
    using strict_mono_sequence_partition[OF assms] .
  fix k1 k2 assume x ∈ {f k1..<f (Suc k1)} x ∈ {f k2..<f (Suc k2)}
  thus k1 = k2
  proof (induction k1 k2 rule: linorder_wlog)
    case (le k1 k2)
    hence f k2 < f (Suc k1)
      by auto
    hence k2 < Suc k1
      using assms(1) strict_mono_less by blast
    with le show k1 = k2
      by linarith
  qed auto
qed

lemma sequence_of_sparse_set_exists':
  fixes A :: complex set and c :: complex ⇒ nat
  assumes infinite A closed A ⋀r. r ≥ 0 ⟹ finite (A ∩ cball 0 r)
  assumes c_pos: ⋀x. x ∈ A ⟹ c x > 0
  obtains f :: nat ⇒ complex where
    mono (norm ∘ f) range f = A filterlim f at_infinity at_top
    ⋀z. z ∈ A ⟹ finite (f -` {z}) ∧ card (f -` {z}) = c z
proof -
  obtain f :: nat ⇒ complex where f:
    mono (norm ∘ f) inj f range f = A filterlim f at_infinity at_top
    using assms sequence_of_sparse_set_exists by blast
  have f_eq_iff [simp]: f m = f n ⟷ m = n for m n
    using ‹inj f› by (auto simp: inj_def)

  define h :: nat ⇒ nat where h = (λn. ∑k<n. c (f k))
```

have [*simp*]: $h\ 0 = 0$
by (*simp add*: *h_def*)
have *h_ge*: $h\ n \geq n$ **for** n
proof −
have $h\ n \geq (\sum k{<}n.\ 1)$
unfolding *h_def* **by** (*intro sum_mono*) (*use c_pos f* **in** ‹*auto simp*: *Suc_le_eq*›)
thus *?thesis* **by** *simp*
qed

have *strict_mono h*
unfolding *strict_mono_Suc_iff* **using** *f* **by** (*auto simp*: *h_def c_pos*)
moreover from *this* **have** *filterlim h at_top at_top*
using *filterlim_subseq* **by** *blast*
ultimately have *Ex1*: $\exists!k.\ n \in \{h\ k..{<}h\ (Suc\ k)\}$ **for** n
by (*intro strict_mono_sequence_partition'*) *auto*

define $g :: nat \Rightarrow nat$ **where** $g = (\lambda n.\ THE\ k.\ n \in \{h\ k..{<}h\ (Suc\ k)\})$
have *g*: $n \in \{h\ (g\ n)..{<}h\ (Suc\ (g\ n))\}$ **for** n
using *theI'*[*OF Ex1*[*of n*]] **by** (*simp add*: *g_def*)
have *g_eqI*: $g\ n = k$ **if** $n \in \{h\ k..{<}h\ (Suc\ k)\}$ **for** $n\ k$
using *the1_equality*[*OF Ex1 that*] **by** (*simp add*: *g_def*)
have *g_h*: $g\ (h\ n) = n$ **for** n
by (*rule g_eqI*) (*auto intro*: *strict_monoD*[*OF* ‹*strict_mono h*›])

have *mono g*
unfolding *incseq_Suc_iff*
proof *safe*
fix $n :: nat$
have $h\ (g\ n) + 1 \leq Suc\ n$
using *g*[*of n*] **by** *auto*
also have $Suc\ n < h\ (Suc\ (g\ (Suc\ n)))$
using *g*[*of Suc n*] **by** *auto*
finally show $g\ n \leq g\ (Suc\ n)$
by (*metis* ‹*strict_mono h*› *add_lessD1 less_Suc_eq_le strict_mono_less*)
qed

have *filterlim g at_top at_top*
unfolding *filterlim_at_top*
proof
fix $n :: nat$
show *eventually* $(\lambda m.\ g\ m \geq n)$ *at_top*
using *eventually_ge_at_top*[*of h n*]
proof *eventually_elim*
case (*elim m*)
have $n \leq g\ (h\ n)$
by (*simp add*: *g_h*)
also have $g\ (h\ n) \leq g\ m$
by (*intro monoD*[*OF* ‹*mono g*›] *elim*)

```
    finally show ?case .
  qed
 qed

 have vimage: (f ∘ g) -` {f n} = {h n..<h (Suc n)} for n
  using g by (auto intro!: g_eqI)

 show ?thesis
 proof (rule that[of f ∘ g])
  have incseq (λn. (norm ∘ f) (g n))
   by (intro monoI monoD[OF f(1)] monoD[OF ‹incseq g›])
  thus incseq (norm ∘ (f ∘ g))
   by (simp add: o_def)
 next
  have range (f ∘ g) ⊆ A
   using f(3) by auto
  moreover have A ⊆ range (f ∘ g)
  proof
   fix z assume z ∈ A
   then obtain n where [simp]: z = f n
    using f(3) by auto
   have f (g (h n)) ∈ range (f ∘ g)
    unfolding o_def by blast
   thus z ∈ range (f ∘ g)
    by (simp add: g_h)
  qed
  ultimately show range (f ∘ g) = A by blast
 next
  fix z assume z ∈ A
  then obtain n where [simp]: z = f n
   using f(3) by auto
  have finite {h n..<h (Suc n)}
   by auto
  moreover have card {h n..<h (Suc n)} = c z
   by (simp add: h_def)
  ultimately show finite ((f ∘ g) -` {z}) ∧ card ((f ∘ g) -` {z}) = c z
   using vimage[of n] by simp
 next
  show filterlim (f ∘ g) at_infinity at_top
   unfolding o_def by (rule filterlim_compose[OF f(4) ‹filterlim g at_top
at_top›])
 qed
qed
```

9.5 The factorisation theorem for holomorphic functions

If g is a holomorphic function on an open connected domain whose zeros do not have an accumulation point on the frontier of A, then we can write g as a product of a function h holomorphic on A and an entire function f such

that h is non-zero everywhere in A and the zeros of f are precisely the zeros of A with the same multiplicity.

In other words, we can get rid of all the zeros of g by dividing it with a suitable entire function f.

```
theorem weierstrass_factorization:
  assumes g holomorphic_on A open A connected A
  assumes ⋀z. z ∈ frontier A ⟹ ¬z islimpt {w∈A. g w = 0}
  obtains h f where
    h holomorphic_on A f holomorphic_on UNIV
    ∀z. f z = 0 ⟷ (∀z∈A. g z = 0) ∨ (z ∈ A ∧ g z = 0)
    ∀z∈A. zorder f z = zorder g z
    ∀z∈A. h z ≠ 0
    ∀z∈A. g z = h z * f z
proof (cases ∀z∈A. g z = 0)
  case True
  show ?thesis
  proof (rule that[of λ_. 1 λ_. 0]; (intro ballI allI impI)?)
    fix z assume z: z ∈ A
    have ev: eventually (λw. w ∈ A) (at z)
      using z assms by (intro eventually_at_in_open') auto
    show zorder (λ_::complex. 0 :: complex) z = zorder g z
      by (intro zorder_cong eventually_mono[OF ev] refl) (use True in auto)
  qed (use assms True in auto)
next
  case not_identically_zero: False
  define Z where Z = {z∈A. g z = 0}
  have freq_nz: frequently (λz. g z ≠ 0) (at z) if z ∈ A for z
  proof -
    have ∀F w in at z. g w ≠ 0 ∧ w ∈ A
    using non_zero_neighbour_alt[OF assms(1,2,3) that(1)] not_identically_zero
by auto
    hence ∀F w in at z. g w ≠ 0
      by eventually_elim auto
    thus ?thesis
      using eventually_frequently by force
  qed

  have zorder_pos_iff: zorder g z > 0 ⟷ g z = 0 if z ∈ A for z
   by (subst zorder_pos_iff[OF assms(1,2) that]) (use freq_nz[of z] that in auto)

  show ?thesis
  proof (cases finite Z)
    case True
    define f where f = (λz. ∏w∈Z. (z - w) powi (zorder g w))

    have eq_zero_iff: f z = 0 ⟷ z ∈ A ∧ g z = 0 for z
      using True local.zorder_pos_iff
      unfolding f_def Z_def by fastforce
```

have *zorder_eq*: *zorder f z = zorder g z* **if** *z ∈ A* **for** *z*
proof (*cases g z = 0*)
case *False*
have *g analytic_on {z}*
using *that assms analytic_at* **by** *blast*
hence [*simp*]: *zorder g z = 0*
using *False* **by** (*intro zorder_eq_0I*) *auto*
moreover have *f analytic_on {z}*
unfolding *f_def* **by** (*auto intro*!: *analytic_intros*)
hence *zorder f z = 0*
using *False* **by** (*intro zorder_eq_0I*) (*auto simp*: *eq_zero_iff*)
ultimately show *?thesis*
by *simp*
next
case *zero*: *True*
have *g analytic_on {z}*
using *that assms*(*1*,*2*) *analytic_at* **by** *blast*
hence *zorder g z ≥ 0*
using *that* **by** (*intro zorder_ge_0 freq_nz*) *auto*
define *f'* **where** *f' = (λz'. (∏ w∈Z−{z}. (z' − w) powi (zorder g w)))*
have *zorder* (*λz'. (z' − z) powi (zorder g z) ∗ f' z'*) *z = zorder g z*
proof (*rule zorder_eqI*)
show *open* (*UNIV* :: *complex set*) *f' holomorphic_on UNIV z ∈ UNIV*
using *local.zorder_pos_iff*
by (*fastforce intro*!: *holomorphic_intros simp*: *f'_def Z_def*)+
show *f' z ≠ 0*
using *True* **unfolding** *f'_def* **by** (*subst prod_zero_iff*) *auto*
qed (*use ‹zorder g z ≥ 0›* **in** ‹*auto simp*: *powr_of_int*›)
also have (*λz'. (z' − z) powi (zorder g z) ∗ f' z'*) *= f*
proof
fix *z'* :: *complex*
have *Z = insert z* (*Z − {z}*)
using *that zero* **by** (*auto simp*: *Z_def*)
also have (∏ *w∈*.... (*z' − w*) *powi* (*zorder g w*)) = (*z' − z*) *powi* (*zorder g z*) ∗ *f' z'*
using *True* **by** (*subst prod.insert*) (*auto simp*: *f'_def*)
finally show (*z' − z*) *powi* (*zorder g z*) ∗ *f' z' = f z'*
by (*simp add*: *f_def*)
qed
finally show *?thesis* **.**
qed

obtain *h* :: *complex ⇒ complex* **where** *h*:
h holomorphic_on A ⋀*z. z ∈ A ⟹ h z ≠ 0* ⋀*z. z ∈ A ⟹ g z = h z ∗ f z*
proof (*rule holomorphic_zorder_factorization*[*OF assms*(*1−3*)])
show *f holomorphic_on A*
using *local.zorder_pos_iff*
unfolding *f_def Z_def* **by** (*fastforce intro*: *holomorphic_intros*)
show *f z = 0 ⟷ g z = 0* **if** *z ∈ A* **for** *z*

```
      using that by (subst eq_zero_iff) auto
    show zorder f z = zorder g z if z ∈ A for z
      by (rule zorder_eq) fact
  qed metis

  show ?thesis
  proof (rule that[of h f]; (intro ballI)?)
    show h holomorphic_on A
      by fact
    show f holomorphic_on UNIV
      using local.zorder_pos_iff
      unfolding f_def Z_def by (fastforce intro: holomorphic_intros)
  qed (use True not_identically_zero in ‹auto simp: eq_zero_iff zorder_eq h(2)
h(3)[symmetric]›)

next
  case False
  note infinite_zeroes = not_identically_zero False
  define c where c = (λz. nat (zorder g z))

  have ev_nz: eventually (λw. g w ≠ 0) (at z) if z ∈ A for z
  proof −
    from infinite_zeroes(1) obtain z0 where z0: z0 ∈ A g z0 ≠ 0
      by auto
    have eventually (λw. g w ≠ 0 ∧ w ∈ A) (at z)
      by (rule non_zero_neighbour_alt[where ?β = z0]) (use assms z0 that in
auto)
    thus ?thesis
      by eventually_elim auto
  qed

  have no_limpt_Z: ¬z islimpt Z for z
  proof
    assume z islimpt Z
    show False
    proof (cases z ∈ A)
      case False
      have z islimpt A
        by (rule islimpt_subset[OF ‹z islimpt Z›]) (auto simp: Z_def)
      hence z ∈ closure A
        by (simp add: closure_def)
      with ‹z ∉ A› have z ∈ frontier A
        by (simp add: closure_Un_frontier)
      with assms and ‹z islimpt Z› show False
        by (auto simp: Z_def)
    next
      case True
      from True have eventually (λw. g w ≠ 0) (at z)
        using ev_nz by blast
```

```
    hence ¬z islimpt Z
      by (auto simp: islimpt_iff_eventually Z_def elim!: eventually_mono)
    with ‹z islimpt Z› show False by blast
  qed
qed
have closed Z
  using no_limpt_Z unfolding closed_limpt by blast

obtain a where a:
    incseq (norm ∘ a) range a = Z − {0}
    ⋀z. z ∈ Z − {0} ⟹ finite (a −‘ {z}) ∧ card (a −‘ {z}) = c z
    filterlim a at_infinity at_top
proof (rule sequence_of_sparse_set_exists′)
  show infinite (Z − {0})
    using infinite_zeroes(2) by auto
next
  show closed (Z − {0})
    unfolding closed_limpt using no_limpt_Z islimpt_subset by blast
next
  show finite ((Z − {0}) ∩ cball 0 R) if R ≥ 0 for R
  proof (rule ccontr)
    assume ∗: infinite ((Z − {0}) ∩ cball 0 R)
    have ∃ z∈cball 0 R. z islimpt ((Z − {0}) ∩ cball 0 R)
      by (rule Heine_Borel_imp_Bolzano_Weierstrass) (use ∗ in auto)
    then obtain z where z islimpt ((Z − {0}) ∩ cball 0 R)
      by blast
    hence z islimpt Z
      using islimpt_subset by blast
    thus False
      using no_limpt_Z by blast
  qed
next
  show c z > 0 if z ∈ Z − {0} for z
    using zorder_pos_iff[of z] that by (auto simp: c_def Z_def)
qed metis

interpret f: weierstrass_product′ a
proof
  show a n ≠ 0 for n
    using a(2) by auto
qed fact+

define m where m = (if 0 ∈ A then nat (zorder g 0) else  0)

have zorder_eq: zorder (λz. z ^ m ∗ f.f z) z = zorder g z if z ∈ A for z
proof (cases g z = 0)
  case False
  have g analytic_on {z}
    using ‹z ∈ A› analytic_at assms by blast
```

```
    hence zorder g z = 0
      by (intro zorder_eq_0I False)
    have z ∉ range a
      using False Z_def a(2) by blast
    hence zorder (λz. z ^ m * f.f z) z = 0
      using False ‹zorder g z = 0›
      by (intro zorder_eq_0I analytic_intros) (auto simp: f.zero m_def)
    with ‹zorder g z = 0› show ?thesis
      by simp
  next
    case True
    define F where F = fps_expansion f.f z
    have frequently (λw. g w ≠ 0) (at z)
      using ev_nz[OF that] eventually_frequently by force
    hence zorder g z ≥ 0
      by (intro zorder_ge_0) (use assms that in ‹auto simp: analytic_at›)

    have ev: eventually (λz. z ∈ A) (nhds z)
      using that assms by (intro eventually_nhds_in_open) auto
    have exp1: (λw. f.f (z + w)) has_fps_expansion F
        unfolding F_def by (intro analytic_at_imp_has_fps_expansion[OF
f.analytic])
    have exp2: (λw. (z + w) ^ m * f.f (z + w)) has_fps_expansion (fps_const
z + fps_X) ^ m * F
      by (intro fps_expansion_intros exp1)
    have [simp]: F ≠ 0
    proof
      assume F = 0
      hence eventually (λz. f.f z = 0) (nhds z)
        using exp1 by (auto simp: has_fps_expansion_def nhds_to_0' eventu-
ally_filtermap)
      hence eventually (λz. g z = 0) (at z)
       by (auto simp: f.zero a Z_def eventually_at_filter elim!: eventually_mono)
      hence eventually (λz::complex. False) (at z)
        using ev_nz[OF ‹z ∈ A›] by eventually_elim auto
      thus False by simp
    qed

    have zorder (λw. w ^ m * f.f w) z = int (subdegree ((fps_const z + fps_X)
^ m * F))
      using has_fps_expansion_zorder[OF exp2] by simp
    also have ... = int (subdegree F) + (if z = 0 then m else 0)
      by auto
    also have int (subdegree F) = zorder f.f z
      using has_fps_expansion_zorder[OF exp1] by simp
    also have ... = int (card (a -‘ {z}))
      using f.zorder by simp
    also have card (a -‘ {z}) = (if z = 0 then 0 else c z)
    proof (cases z = 0)
```

```
      case True
      hence a -` {z} = {}
        using a(2) by auto
      thus ?thesis using True by simp
    next
      case False
      thus ?thesis
        by (subst a(3)) (use True that in ‹auto simp: Z_def›)
    qed
    also have int ... + (if z = 0 then m else 0) = zorder g z
      using ‹zorder g z ≥ 0› that by (auto simp: c_def m_def)
    finally show ?thesis .
  qed

  have eq_zero_iff: z ^ m * f.f z = 0 ⟷ g z = 0 if z ∈ A for z
    using that by (auto simp add: f.zero a m_def zorder_pos_iff Z_def)

  obtain h :: complex ⇒ complex where h:
    h holomorphic_on A ⋀z. z ∈ A ⟹ h z ≠ 0 ⋀z. z ∈ A ⟹ g z = h z * (z
^ m * f.f z)
  proof (rule holomorphic_zorder_factorization[OF assms(1−3)])
    show (λz. z ^ m * f.f z) holomorphic_on A
      by (intro holomorphic_intros)
    show z ^ m * f.f z = 0 ⟷ g z = 0 if z ∈ A for z
      by (rule eq_zero_iff) fact+
    show zorder (λz. z ^ m * f.f z) z = zorder g z if z ∈ A for z
      by (rule zorder_eq) fact+
  qed metis

  show ?thesis
  proof (rule that[of h λz. z ^ m * f.f z]; (intro ballI allI impI)?)
    show h holomorphic_on A
      by fact
    show (λz. z ^ m * f.f z) holomorphic_on UNIV
      by (intro holomorphic_intros)
  next
    fix z :: complex
    show (z ^ m * f.f z = 0) = ((∀z∈A. g z = 0) ∨ z ∈ A ∧ g z = 0)
      using infinite_zeroes(1) a(2)
      by (auto simp: m_def zorder_eq eq_zero_iff zorder_pos_iff Z_def f.zero)
  qed (use zorder_eq eq_zero_iff h in auto)
 qed
qed
```

The following is a simpler version for entire functions.

```
theorem weierstrass_factorization_UNIV:
 assumes g holomorphic_on UNIV
 obtains h f where
   h holomorphic_on UNIV f holomorphic_on UNIV
```

$\forall z.\ f\ z = 0 \longleftrightarrow g\ z = 0$
$\forall z.\ zorder\ f\ z = zorder\ g\ z$
$\forall z.\ h\ z \neq 0$
$\forall z.\ g\ z = h\ z * f\ z$
using *assms* **by** (*rule weierstrass_factorization*, *goal_cases*) *auto*

9.6 The factorisation theorem for meromorphic functions

Let g be a meromorphic function on a connected open domain A. Assume that the poles and zeros of g have no accumulation point on the border of A. Then g can be written in the form $g(z) = h(z)f_1(z)/f_2(z)$ where h is holomorphic on A with no zeroes and f_1 and f_2 are entire.

Moreover, as direct consequences of that, the zeroes of f_1 are precisely the zeroes of g and the zeros of f_2 are precisely the poles of g (with the corresponding multiplicity).

theorem *weierstrass_factorization_meromorphic*:
assumes *mero*: *g nicely_meromorphic_on A* **and** *A*: *open A connected A*
assumes *no_limpt*: $\bigwedge z.\ z \in frontier\ A \Longrightarrow \neg z\ islimpt\ \{w{\in}A.\ g\ w = 0 \vee is_pole\ g\ w\}$
obtains *h f1 f2* **where**
h holomorphic_on A f1 holomorphic_on UNIV f2 holomorphic_on UNIV
$\forall z{\in}A.\ f1\ z = 0 \longleftrightarrow \neg is_pole\ g\ z \wedge g\ z = 0$
$\forall z{\in}A.\ f2\ z = 0 \longleftrightarrow is_pole\ g\ z$
$\forall z{\in}A.\ \neg is_pole\ g\ z \longrightarrow zorder\ f1\ z = zorder\ g\ z$
$\forall z{\in}A.\ is_pole\ g\ z \longrightarrow zorder\ f2\ z = -zorder\ g\ z$
$\forall z{\in}A.\ h\ z \neq 0$
$\forall z{\in}A.\ g\ z = h\ z * f1\ z\ /\ f2\ z$
proof −
have *mero'*: *g meromorphic_on A*
using *mero* **unfolding** *nicely_meromorphic_on_def* **by** *auto*
define *pts* **where** *pts* = $\{z{\in}A.\ is_pole\ g\ z\}$
have $\{z.\ is_pole\ g\ z\}$ *sparse_in A*
using *meromorphic_on_imp_not_pole_cosparse*[*OF mero'*]
by (*auto simp*: *eventually_cosparse*)
hence *pts sparse_in A*
unfolding *pts_def* **by** (*rule sparse_in_subset2*) *auto*
have *open_diff_pts*: *open* $(A - pts')$ **if** $pts' \subseteq pts$ **for** *pts'*
proof (*rule open_diff_sparse_pts*)
show *pts' sparse_in A*
using ‹*pts sparse_in A*› **by** (*rule sparse_in_subset2*) *fact*
qed (*use* ‹*open A*› **in** *auto*)

have *ev*: *eventually* $(\lambda w.\ w \in A - pts)$ (*at z*) **if** $z \in A$ **for** *z*
proof (*cases* $z \in pts$)
case *False*
thus *?thesis*
using *that open_diff_pts*[*of pts*] **by** (*intro eventually_at_in_open'*) *auto*
next

```
    case True
    have eventually (λw. w ∈ (A − (pts − {z})) − {z}) (at z)
      using that by (intro eventually_at_in_open open_diff_pts) auto
    also have A − (pts − {z}) − {z} = A − pts
      using True by auto
    finally show ?thesis .
  qed

  show ?thesis
  proof (cases ∀ z∈A−pts. g z = 0)
    case True
    have no_poles: ¬is_pole g z if z ∈ A for z
    proof −
      have is_pole g z ⟷ is_pole (λ_::complex. 0 :: complex) z
        by (intro is_pole_cong[OF eventually_mono[OF ev]]) (use that True in
auto)
      thus ?thesis
        by simp
    qed
    hence [simp]: pts = {}
      by (auto simp: pts_def)
    have [simp]: zorder g z = zorder (λ_::complex. 0 :: complex) z if z ∈ A for z
      by (intro zorder_cong[OF eventually_mono[OF ev]]) (use True that in auto)
    show ?thesis
      by (rule that[of λ_. 1 λ_. 0 λ_. 1]) (use True in ‹auto simp: no_poles›)

  next

    case False
    have is_pole_iff: is_pole g z ⟷ z ∈ pts if z ∈ A for z
      using that by (auto simp: pts_def)

    obtain h f1 where h_f1:
      h holomorphic_on A − pts f1 holomorphic_on UNIV
      ∀ z. f1 z = 0 ⟷ (∀ z∈A−pts. g z = 0) ∨ (z ∈ A − pts ∧ g z = 0)
      ∀ z∈A−pts. zorder f1 z = zorder g z
      ∀ z∈A−pts. h z ≠ 0
      ∀ z∈A−pts. g z = h z ∗ f1 z
    proof (rule weierstrass_factorization)
      have g analytic_on A − pts
        by (rule nicely_meromorphic_without_singularities)
          (use mero in ‹auto simp: pts_def dest: nicely_meromorphic_on_subset›)
      thus holo: g holomorphic_on A − pts
        by (rule analytic_imp_holomorphic)
      show open (A − pts)
        by (rule open_diff_pts) auto
      show connected (A − pts)
        by (rule sparse_imp_connected) (use A ‹pts sparse_in A› in auto)
      show ¬ z islimpt {w ∈ A − pts. g w = 0} if z ∈ frontier (A − pts) for z
```

```
  proof −
    from that have z ∈ frontier A − pts ∪ pts
      using ‹open (A − pts)› ‹open A› closure_mono[of A − pts A]
      by (auto simp: frontier_def interior_open)
    thus ?thesis
    proof
      assume z ∈ pts
      hence is_pole g z
        by (auto simp: pts_def)
      hence eventually (λz. g z ≠ 0) (at z)
        using non_zero_neighbour_pole by blast
      hence ¬z islimpt {w. g w = 0}
        by (auto simp: islimpt_iff_eventually)
      thus ?thesis
        using islimpt_subset[of z {w∈A−pts. g w = 0} {w. g w = 0}] by blast
    next
      assume z: z ∈ frontier A − pts
      show ¬ z islimpt {w ∈ A − pts. g w = 0}
      proof
        assume z islimpt {w ∈ A − pts. g w = 0}
        hence z islimpt {w ∈ A. g w = 0 ∨ is_pole g w}
          by (rule islimpt_subset) (auto simp: pts_def)
        thus False using no_limpt z by blast
      qed
    qed
  qed
qed

have f1_eq_0_iff: f1 z = 0 ⟷ (z ∈ A − pts ∧ g z = 0) for z
  using h_f1(3) False by auto

define h' where h' = (λz. if z ∈ pts then 0 else inverse (h z))

have isolated_h: isolated_singularity_at h z if z ∈ pts for z
proof −
  have open (A − (pts − {z}))
    by (rule open_diff_pts) auto
  moreover have z ∈ (A − (pts − {z}))
    using that by (auto simp: pts_def)
  moreover have h holomorphic_on (A − (pts − {z})) − {z}
    by (rule holomorphic_on_subset[OF h_f1(1)]) (use that in auto)
  ultimately show isolated_singularity_at h z
    using isolated_singularity_at_holomorphic by blast
qed

have is_pole_h: is_pole h z if z ∈ A is_pole g z for z
proof −
  have f1: f1 analytic_on {z}
    by (meson analytic_on_holomorphic h_f1(2) open_UNIV top_greatest)
```

```
    have eventually (λw. g w ≠ 0) (at z)
      using ‹is_pole g z› non_zero_neighbour_pole by blast
    with ev[OF that(1)] have ev': eventually (λw. g w * f1 w ≠ 0) (at z)
      by eventually_elim (use h_f1(3) in auto)

    have is_pole (λw. g w / f1 w) z
    proof (rule is_pole_divide_zorder)
      show isolated_singularity_at f1 z not_essential f1 z
     using f1 by (simp_all add: isolated_singularity_at_analytic not_essential_analytic)
      show isolated_singularity_at g z not_essential g z
        using mero' that unfolding meromorphic_on_altdef by blast+
      show freq: frequently (λw. g w * f1 w ≠ 0) (at z)
        using ev' by (rule eventually_frequently[rotated]) auto
      from freq have freq': frequently (λw. f1 w ≠ 0) (at z)
        using frequently_elim1 by fastforce
      have zorder g z < 0
     using ‹is_pole g z› ‹isolated_singularity_at g z› isolated_pole_imp_neg_zorder
by auto
      also have 0 ≤ zorder f1 z
        by (rule zorder_ge_0[OF f1 freq'])
      finally show zorder g z < zorder f1 z .
    qed
    also have ?this ⟷ is_pole h z
      proof (intro is_pole_cong refl eventually_mono[OF eventually_conj[OF
ev[OF that(1)] ev']])
      fix w assume w ∈ A − pts ∧ g w * f1 w ≠ 0
      thus g w / f1 w = h w using h_f1(6)
        by (auto simp: divide_simps)
    qed
    finally show is_pole h z .
  qed

  have h' analytic_on {z} if z ∈ A for z
  proof (cases z ∈ pts)
    case False
    moreover have open (A − pts)
      by (rule open_diff_pts) auto
    ultimately have (λz. inverse (h z)) analytic_on {z}
      using that h_f1(1,2,5) ‹open (A − pts)› analytic_at False
      by (intro analytic_intros) (auto simp: f1_eq_0_iff)
    also have eventually (λz. z ∈ A − pts) (nhds z)
     using that False ‹open (A − pts)› by (intro eventually_nhds_in_open) auto
    hence (λz. inverse (h z)) analytic_on {z} ⟷ h' analytic_on {z}
      by (intro analytic_at_cong) (auto elim!: eventually_mono simp: h'_def)
    finally show ?thesis .
  next
    case True
    have (λw. if w = z then 0 else inverse (h w)) holomorphic_on (A − (pts −
{z}))
```

```
  proof (rule is_pole_inverse_holomorphic)
    from True have A − (pts − {z}) − {z} = A − pts
      by auto
    thus h holomorphic_on A − (pts − {z}) − {z}
      using h_f1(1) by simp
  next
    show open (A − (pts − {z}))
      by (rule open_diff_pts) auto
  next
    have is_pole g z
      using True that by (simp add: is_pole_iff)
    thus is_pole h z
      using is_pole_h that by auto
  next
    show ∀ w∈A−(pts−{z})−{z}. h w ≠ 0
      using h_f1(5) by auto
  qed
  also have ?this ⟷ h' holomorphic_on A − (pts − {z})
  proof (intro holomorphic_cong refl)
    fix w assume w: w ∈ A − (pts − {z})
    show (if w = z then 0 else inverse (h w)) = h' w
      using True w by (cases w = z) (auto simp: h'_def)
  qed
  finally have h' holomorphic_on A − (pts − {z}) .
  moreover have z ∈ A − (pts − {z}) open (A − (pts − {z}))
    using True that by (auto intro!: open_diff_pts)
  ultimately show h' analytic_on {z}
    using analytic_at by blast
qed
hence h': h' analytic_on A
  using analytic_on_analytic_at by blast

have h'_eq_0_iff: h' w = 0 ⟷ is_pole g w if w ∈ A for w
  using that h_f1(5) is_pole_iff[of w] by (auto simp: h'_def)

obtain h'' f2 where h''_f2:
   h'' holomorphic_on A f2 holomorphic_on UNIV
   ∀ z. f2 z = 0 ⟷ (∀ z∈A. h' z = 0) ∨ (z ∈ A ∧ h' z = 0)
   ∀ z∈A. h' z = 0 ⟶ zorder f2 z = zorder h' z
   ∀ z∈A. h'' z ≠ 0 ∀ z∈A. h' z = h'' z * f2 z
proof (rule weierstrass_factorization[of h' A])
  show open A connected A
    by fact+
  show h' holomorphic_on A
    using h' ‹open A› by (simp add: analytic_on_open)
  show ¬z islimpt {w∈A. h' w = 0} if z ∈ frontier A for z
  proof
    assume z islimpt {w∈A. h' w = 0}
    also have {w∈A. h' w = 0} = pts
```

```
      by (auto simp: h'_eq_0_iff pts_def)
    finally have z islimpt {w ∈ A. g w = 0 ∨ is_pole g w}
      by (rule islimpt_subset) (auto simp: pts_def)
    thus False using no_limpt[of z] that
      by blast
  qed
qed blast

show ?thesis
proof (rule that[of λw. inverse (h'' w) f1 f2]; (intro ballI allI impI)?)
  show (λw. inverse (h'' w)) holomorphic_on A
    using h''_f2(1,2,5) by (intro holomorphic_intros) auto
next
  show f1 holomorphic_on UNIV f2 holomorphic_on UNIV
    by fact+
next
  show f1 z = 0 ⟷ ¬ is_pole g z ∧ g z = 0 if z ∈ A for z
    using that by (subst f1_eq_0_iff) (auto simp: pts_def)
next
  show f2 z = 0 ⟷ is_pole g z if z ∈ A for z
  proof −
    have ¬(∀ z∈A. h' z = 0)
      using False h''_f2(6) h_f1(6) h'_eq_0_iff is_pole_iff by auto
    hence f2 z = 0 ⟷ h' z = 0
      using h''_f2(3) that by auto
    also have ... ⟷ is_pole g z
      using that by (simp add: is_pole_iff h'_eq_0_iff)
    finally show ?thesis .
  qed
next
  show zorder f1 z = zorder g z if z ∈ A ¬is_pole g z for z
    using h_f1(4) that by (auto simp: pts_def)
next
  show zorder f2 z = −zorder g z if z ∈ A is_pole g z for z
  proof −
    have zorder f2 z = zorder h' z
      using h''_f2(4) that h'_eq_0_iff[of z] is_pole_iff[of z] by auto
    also have ... = zorder (λw. inverse (h w)) z
      using that by (intro zorder_cong eventually_mono[OF ev]) (auto simp:
h'_def)
    also have ... = −zorder h z
    proof (intro zorder_inverse)
      have is_pole h z
        using that is_pole_iff[of z] is_pole_h[of z] by auto
      thus not_essential h z
        by force
      show frequently (λw. h w ≠ 0) (at z)
       using non_zero_neighbour_pole[OF ‹is_pole h z›] eventually_frequently
by force
```

```
    qed (use that in ‹auto intro!: isolated_h simp: pts_def›)
    also have zorder f1 z = 0
    proof (rule zorder_eq_0I)
      show f1 analytic_on {z}
        using that h_f1(2) holomorphic_on_imp_analytic_at by blast
      show f1 z ≠ 0
        using that h_f1(3) False by (auto simp: pts_def)
    qed
    hence zorder h z = zorder f1 z + zorder h z
      by simp
    also have ... = zorder (λw. f1 w * h w) z
    proof (rule zorder_times [symmetric])
      have f1 analytic_on {z}
        using that h_f1(2) holomorphic_on_imp_analytic_at by blast
      thus isolated_singularity_at f1 z not_essential f1 z
       using isolated_singularity_at_analytic not_essential_analytic by blast+
      show isolated_singularity_at h z
        using that by (intro isolated_h) (auto simp: pts_def)
      have is_pole h z
        using is_pole_iff[of z] that by (intro is_pole_h) auto
      thus not_essential h z
        by force
      have z ∈ A
        using that by auto
      have eventually (λw. g w ≠ 0) (at z)
        using non_zero_neighbour_pole[of g z] that by auto
      hence eventually (λw. f1 w * h w ≠ 0) (at z)
        using ev[OF ‹z ∈ A›]  by eventually_elim (use h_f1(6) in auto)
      thus frequently (λw. f1 w * h w ≠ 0) (at z)
        using eventually_frequently by force
    qed
    also have ... = zorder g z
    proof (rule zorder_cong)
      have eventually (λw. w ∈ A - pts) (at z)
        using ev[of z] that by auto
      thus eventually (λw. f1 w * h w = g w) (at z)
        by eventually_elim (use h_f1(6) in auto)
    qed auto
    finally show ?thesis .
  qed
next
  show inverse (h'' z) ≠ 0 if z ∈ A for z
    using h''_f2(5) that by auto
next
  show g z = inverse (h'' z) * f1 z / f2 z if z: z ∈ A for z
  proof (cases is_pole g z)
    case False
    have *: g z = h z * f1 z h' z = h'' z * f2 z h'' z ≠ 0 h z ≠ 0
      using that h''_f2(5,6) h_f1(5,6) False unfolding pts_def by blast+
```

```
      have g z = h z * f1 z
        by fact
      also have ... = f1 z / h' z
        using * that False by (simp add: h'_def field_simps pts_def)
      also have h' z = h'' z * f2 z
        by fact
      also have f1 z / (h'' z * f2 z) = inverse (h'' z) * f1 z / f2 z
        by (simp add: divide_inverse_commute)
      finally show ?thesis .
    next
      case True
      have ¬g -z→ g z
      using True at_neq_bot is_pole_def not_tendsto_and_filterlim_at_infinity
by blast
      with mero and z and True have g z = 0
        by (auto simp: nicely_meromorphic_on_def)
      moreover have f2 z = 0
        using True z by (simp add: h''_f2(3) h'_eq_0_iff)
      ultimately show ?thesis by simp
    qed
  qed
 qed
qed
```

Again, we derive an easier version for functions meromorphic on the entire complex plane.

```
theorem weierstrass_factorization_meromorphic_UNIV:
 assumes g nicely_meromorphic_on UNIV
 obtains h f1 f2 where
   h holomorphic_on UNIV f1 holomorphic_on UNIV f2 holomorphic_on UNIV
   ∀z. f1 z = 0 ⟷ ¬is_pole g z ∧ g z = 0
   ∀z. f2 z = 0 ⟷ is_pole g z
   ∀z. ¬is_pole g z ⟶ zorder f1 z = zorder g z
   ∀z. is_pole g z ⟶ zorder f2 z = -zorder g z
   ∀z. h z ≠ 0
   ∀z. g z = h z * f1 z / f2 z
proof (rule weierstrass_factorization_meromorphic)
 show g nicely_meromorphic_on UNIV
   by fact
 show connected (UNIV :: complex set)
   by (simp add: Convex.connected_UNIV)
 show ¬ z islimpt {w ∈ UNIV. g w = 0 ∨ is_pole g w} if z ∈ frontier UNIV for
z
   using that by simp
 show open (UNIV :: complex set)
   by simp
qed auto

end
```

10 Complex Analyticity and the Complexification of Real-Analytic Functions

Connections with complex analysis: a Cauchy–Riemann criterion for holomorphy; holomorphic functions are C^∞; and a real function is real-analytic at a point iff it extends holomorphically to a complex ball around that point.

theory *Complex_Analytic*
imports *HOL−Analysis.Real_Analytic Cauchy_Integral_Formula*
begin

10.1 A CR-linear self-map of the plane is multiplication by a scalar

definition *CR_linear* :: (*complex* ⇒ *complex*) ⇒ *bool* **where**
CR_linear L ⟷ *bounded_linear L* ∧ (∀ *v*. *L* (i ∗ *v*) = i ∗ *L v*)

lemma *CR_linear_is_mult*:
assumes *CR_linear L*
shows ∃ *a*. ∀ *v*. *L v* = *a* ∗ *v*
proof −
from *assms* **have** *bl*: *bounded_linear L* **and** *cr*: ⋀*v*. *L* (i ∗ *v*) = i ∗ *L v*
by (*auto simp*: *CR_linear_def*)
interpret *bounded_linear L* **by** (*rule bl*)
have *Lof*: *L* (*complex_of_real r*) = *complex_of_real r* ∗ *L 1* **for** *r*
proof −
have *L* (*complex_of_real r*) = *L* (*r* ∗*R* *1*)
by (*simp add*: *scaleR_conv_of_real*)
also have ... = *r* ∗*R* *L 1*
by (*rule scaleR*)
also have ... = *complex_of_real r* ∗ *L 1*
by (*simp add*: *scaleR_conv_of_real*)
finally show *?thesis* **.**
qed
have *L v* = *L 1* ∗ *v* **for** *v*
proof −
have *decomp*: *v* = *complex_of_real* (*Re v*) + i ∗ *complex_of_real* (*Im v*)
by (*simp add*: *complex_eq_iff*)
have *L v* = *L* (*complex_of_real* (*Re v*) + i ∗ *complex_of_real* (*Im v*))
using *decomp* **by** *simp*
also have ... = *L* (*complex_of_real* (*Re v*)) + *L* (i ∗ *complex_of_real* (*Im v*))
by (*rule add*)
also have ... = *L* (*complex_of_real* (*Re v*)) + i ∗ *L* (*complex_of_real* (*Im v*))
using *cr* **by** *simp*
also have ... = *complex_of_real* (*Re v*) ∗ *L 1* + i ∗ (*complex_of_real* (*Im v*) ∗ *L 1*)
by (*simp add*: *Lof*)

```
    also have ... = L 1 * (complex_of_real (Re v) + i * complex_of_real (Im
v))
      by (simp add: algebra_simps)
    also have ... = L 1 * v
      using decomp by simp
    finally show ?thesis .
  qed
  thus ?thesis by blast
qed
```

10.2 Several-variable Cauchy–Riemann criterion

```
theorem CauchyRiemann_imp_holomorphic:
  fixes f :: complex ⇒ complex
  assumes Sopen: open S
    and diff: ⋀x. x ∈ S ⟹ (f has_derivative (L x)) (at x)
    and CR:   ⋀x. x ∈ S ⟹ CR_linear (L x)
  shows f holomorphic_on S
proof −
  have f field_differentiable (at x) if xS: x ∈ S for x
  proof −
    from CR[OF xS] obtain a where a: ⋀v. L x v = a * v
      using CR_linear_is_mult by blast
    from diff[OF xS] have (f has_derivative (L x)) (at x) .
    moreover have L x = (*) a
      using a by auto
    ultimately have (f has_derivative (*) a) (at x) by simp
    hence (f has_field_derivative a) (at x)
      by (simp add: has_field_derivative_def mult.commute)
    thus ?thesis
      using field_differentiable_def by blast
  qed
  thus ?thesis
    using Sopen by (simp add: holomorphic_on_open field_differentiable_def)
qed
```

A constant multiple of a holomorphic function is C^k at each point, for every k.

```
lemma holomorphic_const_mult_Ck_at:
  fixes f :: complex ⇒ complex
  assumes f holomorphic_on U and open U and x ∈ U
  shows Ck_at k (λy. c * f y) x
  using assms
proof (induction k arbitrary: f x c)
  case 0
  have continuous (at x) (λy. c * f y)
  proof −
    have continuous_on U f
      using 0 holomorphic_on_imp_continuous_on by blast
```

```
    hence continuous (at x) f
      using 0 by (simp add: continuous_on_eq_continuous_at)
    thus ?thesis by (intro continuous_intros)
  qed
  thus ?case by simp
next
  case (Suc k)
  note holf = Suc.prems(1) and openU = Suc.prems(2) and xU = Suc.prems(3)

  — The scaled map is holomorphic, hence differentiable on U.
  have holcf: (λy. c * f y) holomorphic_on U
    using holf by (intro holomorphic_intros)

  — (i) Neighbourhood: Ck_at k holds throughout U by the induction hypothesis.
  have nbhd: ∃A. open A ∧ x ∈ A ∧ (∀y∈A. Ck_at k (λy. c * f y) y)
  proof (intro exI[of _ U] conjI ballI)
    fix y assume y ∈ U
    show Ck_at k (λz. c * f z) y
      using Suc.IH[OF holf openU ‹y ∈ U›] .
  qed (use openU xU in auto)

  — (ii) Differentiability at x.
  have diff: (λy. c * f y) differentiable (at x)
    using holomorphic_imp_differentiable_real[OF holcf openU xU] .

  — (iii) The directional derivative map is C^k at x.
  have derivs: ∀v. Ck_at k (λy. frechet_derivative (λz. c * f z) (at y) v) x
  proof
    fix v
    — On U the directional derivative equals (c * v) * deriv f y.
    have eqd: ⋀y. y ∈ U ⟹
                frechet_derivative (λz. c * f z) (at y) v = (c * v) * deriv f y
    proof −
      fix y assume yU: y ∈ U
      have frechet_derivative (λz. c * f z) (at y) v = deriv (λz. c * f z) y * v
        using frechet_derivative_holomorphic[OF holcf openU yU] by simp
      also have deriv (λz. c * f z) y = c * deriv f y
        using holf openU yU
        by (simp add: deriv_cmult holomorphic_on_imp_differentiable_at)
      finally show frechet_derivative (λz. c * f z) (at y) v = (c * v) * deriv f y
        by (simp add: algebra_simps)
    qed
    — deriv f is holomorphic on U, so (c*v) * deriv f is C^k by the IH.
    have holderiv: deriv f holomorphic_on U
      using holf openU by (rule holomorphic_deriv)
    have base: Ck_at k (λy. (c * v) * deriv f y) x
      using Suc.IH[OF holderiv openU xU] .
    show Ck_at k (λy. frechet_derivative (λz. c * f z) (at y) v) x
      by (rule Ck_at_transfer_open[OF openU xU _ base]) (simp add: eqd)
```

```
  qed

  show ?case
    unfolding Ck_at.simps(2)
    using nbhd diff derivs by blast
qed

theorem holomorphic_imp_Cinfinity_on:
  assumes f holomorphic_on U and open U
  shows Cinfinity_on f U
  unfolding Cinfinity_on_def Cinfinity_at_def
proof (intro conjI ballI allI)
  show open U by (rule assms(2))
next
  fix x :: complex and k assume xU: x ∈ U
  have Ck_at k (λy. 1 * f y) x
    using holomorphic_const_mult_Ck_at[OF assms(1) assms(2) xU] .
  thus Ck_at k f x by simp
qed
```

10.3 Real analyticity via holomorphic extension

```
definition has_holo_extension_at :: (real ⇒ real) ⇒ real ⇒ bool where
  has_holo_extension_at f c ⟷
     (∃r>0. ∃g. g holomorphic_on ball (complex_of_real c) r
              ∧ (∀x. |x − c| < r ⟶ g (complex_of_real x) = complex_of_real (f
x)))


lemma real_analytic_at_1d_imp_holo_extension:
  fixes f :: real ⇒ real
  assumes real_analytic_at_1d f c
  shows has_holo_extension_at f c
proof −
  from assms obtain r where r: 0 < r
    and TS: ⋀x. |x − c| < r ⟹
          (λn. (deriv ^^ n) f c / fact n * (x − c) ^ n) sums f x
    unfolding real_analytic_at_1d_def by blast
  — the complex coefficients (same as the real Taylor coefficients)
  define b :: nat ⇒ complex where b = (λn. complex_of_real ((deriv ^^ n) f c /
fact n))
  — the complex sum function on the ball; well-defined by summability
  define g :: complex ⇒ complex where
    g = (λw. ∑n. b n * (w − complex_of_real c) ^ n)
  — The complex power series is summable at every complex point of the ball.
  have summ_complex: summable (λn. b n * (w − complex_of_real c) ^ n)
    if w: w ∈ ball (complex_of_real c) r for w
  proof −
    have nw: norm (w − complex_of_real c) < r
```

```
    using w by (simp add: dist_norm norm_minus_commute)
  — pick an intermediate real radius s
  define s where s = (norm (w − complex_of_real c) + r) / 2
  have s_pos: 0 < s
  proof −
    have 0 < norm (w − complex_of_real c) + r
      using r norm_ge_zero[of w − complex_of_real c] by linarith
    thus ?thesis by (simp add: s_def)
  qed
  have s_lt_r: s < r using nw by (simp add: s_def)
  have nw_lt_s: norm (w − complex_of_real c) < s
    using nw by (simp add: s_def)
  — real series converges at c + s, since |s| < r
  have |(c + s) − c| < r using s_pos s_lt_r by simp
  from TS[OF this] have realsum:
    summable (λn. (deriv ^^ n) f c / fact n * ((c + s) − c) ^ n)
    by (rule sums_summable)
  have realsum': summable (λn. (deriv ^^ n) f c / fact n * s ^ n)
    using realsum by simp
  — cast to complex: b n * (complex_of_real s)^n is summable
  have cast: summable (λn. b n * (complex_of_real s) ^ n)
  proof −
    have (λn. of_real ((deriv ^^ n) f c / fact n * s ^ n) :: complex)
          = (λn. b n * (complex_of_real s) ^ n)
      by (simp only: b_def of_real_mult of_real_power)
    moreover have summable (λn. of_real ((deriv ^^ n) f c / fact n * s ^ n) ::
complex)
      using realsum' by (rule summable_of_real)
    ultimately show ?thesis by simp
  qed
  — powser_inside upgrades to summability strictly inside
  have norm (w − complex_of_real c) < norm (complex_of_real s)
    using nw_lt_s s_pos by simp
  from powser_inside[OF cast this]
  show ?thesis .
qed
— hence at each ball point the series sums to g w
have sums_g: (λn. b n * (w − complex_of_real c) ^ n) sums g w
  if w: w ∈ ball (complex_of_real c) r for w
  using summ_complex[OF w] by (simp add: g_def summable_sums)
— holomorphy from power_series_holomorphic
have holo: g holomorphic_on ball (complex_of_real c) r
proof (rule power_series_holomorphic)
  fix w :: complex assume w ∈ ball (complex_of_real c) r
  thus (λn. b n * (w − complex_of_real c) ^ n) sums g w
    by (rule sums_g)
qed
— on the real axis, the series sums to of_real (f x)
have realaxis: g (complex_of_real x) = complex_of_real (f x)
```

```
    if x: |x − c| < r for x
  proof −
    have wball: complex_of_real x ∈ ball (complex_of_real c) r
      using x by (simp add: dist_norm norm_minus_commute flip: of_real_diff)
      have (λn. b n ∗ (complex_of_real x − complex_of_real c) ^ n) sums g
(complex_of_real x)
      by (rule sums_g[OF wball])
    moreover have
      (λn. b n ∗ (complex_of_real x − complex_of_real c) ^ n)
        = (λn. complex_of_real ((deriv ^^ n) f c / fact n ∗ (x − c) ^ n))
      by (simp only: b_def of_real_mult of_real_power flip: of_real_diff)
    ultimately have
      (λn. complex_of_real ((deriv ^^ n) f c / fact n ∗ (x − c) ^ n))
        sums g (complex_of_real x) by simp
    moreover have
      (λn. complex_of_real ((deriv ^^ n) f c / fact n ∗ (x − c) ^ n))
        sums complex_of_real (f x)
      using TS[OF x] by (rule sums_of_real)
    ultimately show ?thesis by (rule sums_unique2)
  qed
  show ?thesis
    unfolding has_holo_extension_at_def
    using r holo realaxis by blast
qed
```

A real function with a holomorphic extension around c is real-analytic at c.

```
lemma holo_extension_imp_real_analytic_at_1d:
  fixes f :: real ⇒ real
  assumes has_holo_extension_at f c
  shows real_analytic_at_1d f c
proof −
  from assms obtain r g where r: 0 < r
    and holo: g holomorphic_on ball (complex_of_real c) r
    and onaxis: ⋀x. |x − c| < r ⟹ g (complex_of_real x) = complex_of_real (f
x)
    unfolding has_holo_extension_at_def by blast
  — the real coefficients are the real parts of the complex Taylor coefficients
  define A :: nat ⇒ complex where
    A = (λn. (deriv ^^ n) g (complex_of_real c) / fact n)
  define a :: nat ⇒ real where a = (λn. Re (A n))
  — the real power series with coefficients a sums to f x on the ball
  have PS: (λn. a n ∗ (x − c) ^ n) sums f x if x: |x − c| < r for x
  proof −
    have wball: complex_of_real x ∈ ball (complex_of_real c) r
      using x by (simp add: dist_norm norm_minus_commute flip: of_real_diff)
    — complex Taylor series of g at the real point
    have cseries: (λn. A n ∗ (complex_of_real x − complex_of_real c) ^ n)
                  sums g (complex_of_real x)
      unfolding A_def by (rule holomorphic_power_series[OF holo wball])
```

```
    have eqf: g (complex_of_real x) = complex_of_real (f x) by (rule onaxis[OF
x])
    — rewrite the complex terms and take real parts
    have term_eq: A n * (complex_of_real x − complex_of_real c) ^ n
                  = complex_of_real (a n * (x − c) ^ n)
                    + i * complex_of_real (Im (A n) * (x − c) ^ n) for n
    proof −
      have pw: (complex_of_real x − complex_of_real c) ^ n
               = complex_of_real ((x − c) ^ n)
        by (simp flip: of_real_diff of_real_power)
      have A n * (complex_of_real x − complex_of_real c) ^ n
           = A n * complex_of_real ((x − c) ^ n) by (simp only: pw)
      also have ... = complex_of_real (Re (A n) * (x − c) ^ n)
                      + i * complex_of_real (Im (A n) * (x − c) ^ n)
        by (simp add: complex_eq_iff)
      finally show ?thesis by (simp add: a_def)
    qed
    have cseries': (λn. complex_of_real (a n * (x − c) ^ n)
                        + i * complex_of_real (Im (A n) * (x − c) ^ n))
                   sums complex_of_real (f x)
      using cseries by (simp add: term_eq eqf)
    — take real parts
    have (λn. Re (complex_of_real (a n * (x − c) ^ n)
                  + i * complex_of_real (Im (A n) * (x − c) ^ n)))
         sums Re (complex_of_real (f x))
      by (rule sums_Re[OF cseries'])
    thus ?thesis by simp
  qed
  show ?thesis by (rule real_powser_imp_real_analytic_at_1d[OF r PS])
qed


theorem real_analytic_at_1d_iff_holo_extension:
  fixes f :: real ⇒ real
  shows real_analytic_at_1d f c ⟷ has_holo_extension_at f c
 using real_analytic_at_1d_imp_holo_extension holo_extension_imp_real_analytic_at_1d
  by blast

end
theory Complex_Analysis
  imports
  Riemann_Mapping
  Residue_Theorem
  Weierstrass_Factorization
  Complex_Analytic
begin

end
```

References

[1]