Prime Number Theorem with Remainder Term

Shuhao Song and Bowen Yao

March 17, 2025

Abstract

We have formalized the proof of the Prime Number Theorem with remainder term. This is the first formalized version of PNT with an explicit error term.

There are many useful results in this AFP entry.

First, the main result, prime number theorem with remainder:

$$\pi(x) = \mathrm{Li}(x) + O\left(x \exp\left(-\sqrt{\log x}/3653\right)\right)$$

Second, the zero-free region of the Riemann zeta function:

$$\zeta(\beta + i\gamma) \neq 0 \text{ when } \beta \geq 1 - \frac{1}{952320} \left(\log(|\gamma| + 2)\right)^{-1}$$

Moreover, we proved a revised version of Perron's formula, together with the zero-free region we can prove the main result.

Contents

```
theory PNT_Notation
imports
  Prime_Number_Theorem.Prime_Counting_Functions
begin

definition PNT_const_C₁ ≡ 1 / 952320 :: real

abbreviation nat_powr
  (infixr ‹nat'_powr› 80)
where
  n nat_powr x ≡ (of_nat n) powr x

bundle pnt_syntax
begin
notation PNT_const_C₁ (‹C₁›)
notation norm (‹‖_‖›)
notation Suc (‹_₊› [101] 100)
end

end
theory PNT_Remainder_Library
imports
  PNT_Notation
begin
unbundle pnt_syntax
```

1 Auxiliary library for prime number theorem

1.1 Zeta function

```
lemma pre_zeta_1_bound:
  assumes 0 < Re s
  shows ‖pre_zeta 1 s‖ ≤ ‖s‖ / Re s
proof –
  have ‖pre_zeta 1 s‖ ≤ ‖s‖ / (Re s * 1 powr Re s)
    by (rule pre_zeta_bound') (use assms in auto)
  also have ... = ‖s‖ / Re s by auto
  finally show ?thesis .
qed

lemma zeta_pole_eq:
  assumes s ≠ 1
  shows zeta s = pre_zeta 1 s + 1 / (s – 1)
proof –
  have zeta s – 1 / (s – 1) = pre_zeta 1 s by (intro zeta_minus_pole_eq assms)
  thus ?thesis by (simp add: field_simps)
qed

definition zeta' where zeta' s ≡ pre_zeta 1 s * (s – 1) + 1

lemma zeta'_analytic:
  zeta' analytic_on UNIV
  unfolding zeta'_def by (intro analytic_intros) auto
```

lemma *zeta'_analytic_on* [*analytic_intros*]:
zeta' analytic_on A **using** *zeta'_analytic analytic_on_subset* **by** *auto*

lemma *zeta'_holomorphic_on* [*holomorphic_intros*]:
zeta' holomorphic_on A **using** *zeta'_analytic_on* **by** (*intro analytic_imp_holomorphic*)

lemma *zeta_eq_zeta'*:
zeta s = zeta' s / (s − 1)
proof (*cases s = 1*)
case *True* **thus** *?thesis* **using** *zeta_1* **unfolding** *zeta'_def* **by** *auto*
next
case *False* **with** *zeta_pole_eq* [*OF this*]
show *?thesis* **unfolding** *zeta'_def* **by** (*auto simp add*: *field_simps*)
qed

lemma *zeta'_1* [*simp*]: *zeta' 1 = 1* **unfolding** *zeta'_def* **by** *auto*

lemma *zeta_eq_zero_iff_zeta'*:
shows *s ≠ 1 ⟹ zeta' s = 0 ⟷ zeta s = 0*
using *zeta_eq_zeta'* [*of s*] **by** *auto*

lemma *zeta'_eq_zero_iff*:
shows *zeta' s = 0 ⟷ zeta s = 0 ∧ s ≠ 1*
by (*cases s = 1*, *use zeta_eq_zero_iff_zeta'* **in** *auto*)

lemma *zeta_eq_zero_iff*:
shows *zeta s = 0 ⟷ zeta' s = 0 ∨ s = 1*
by (*subst zeta'_eq_zero_iff*, *use zeta_1* **in** *auto*)

1.2 Logarithm derivatives

definition *logderiv f x ≡ deriv f x / f x*
definition *log_differentiable*
(**infixr** ‹(*log'_differentiable*)› *50*)
where
f log_differentiable x ≡ (f field_differentiable (at x)) ∧ f x ≠ 0

lemma *logderiv_prod'*:
fixes *f :: 'n ⇒ 'f ⇒ 'f :: real_normed_field*
assumes *fin*: *finite I*
and *lder*: *⋀i. i ∈ I ⟹ f i log_differentiable a*
shows *logderiv (λx. ∏ i∈I. f i x) a = (∑ i∈I. logderiv (f i) a)* (**is** *?P*)
and *(λx. ∏ i∈I. f i x) log_differentiable a* (**is** *?Q*)
proof −
let *?a = λi. deriv (f i) a*
let *?b = λi. ∏ j∈I − {i}. f j a*
let *?c = λi. f i a*
let *?d = ∏ i∈I. ?c i*
have *der*: *⋀i. i ∈ I ⟹ f i field_differentiable (at a)*
and *nz*: *⋀i. i ∈ I ⟹ f i a ≠ 0*
using *lder* **unfolding** *log_differentiable_def* **by** *auto*
have *1*: *(∗) x = (λy. y ∗ x)* **for** *x :: 'f* **by** *auto*
have *((λx. ∏ i∈I. f i x) has_derivative*
(λy. ∑ i∈I. ?a i ∗ y ∗?b i)) (at a within UNIV)
by (*rule has_derivative_prod*, *fold has_field_derivative_def*)
(*rule field_differentiable_derivI*, *elim der*)

```
  hence 2: DERIV (λx. ∏ i∈I. f i x) a :> (∑ i∈I. ?a i * ?b i)
    unfolding has_field_derivative_def
    by (simp add: sum_distrib_left [symmetric] mult_ac)
       (subst 1, blast)
  have prod_nz: (∏ i∈I. ?c i) ≠ 0
    using prod_zero_iff nz fin by auto
  have mult_cong: b = c ⟹ a * b = a * c for a b c :: real by auto
  have logderiv (λx. ∏ i∈I. f i x) a = deriv (λx. ∏ i∈I. f i x) a / ?d
    unfolding logderiv_def by auto
  also have ... = (∑ i∈I. ?a i * ?b i) / ?d
    using 2 DERIV_imp_deriv by auto
  also have ... = (∑ i∈I. ?a i * (?b i / ?d))
    by (auto simp add: sum_divide_distrib)
  also have ... = (∑ i∈I. logderiv (f i) a)
  proof -
    have ⋀a b c :: 'f. a ≠ 0 ⟹ a = b * c ⟹ c / a = inverse b
      by (auto simp add: field_simps)
    moreover have ?d = ?c i * ?b i if i ∈ I for i
      by (intro prod.remove that fin)
    ultimately have ?b i / ?d = inverse (?c i) if i ∈ I for i
      using prod_nz that by auto
    thus ?thesis unfolding logderiv_def using 2
      by (auto simp add: divide_inverse intro: sum.cong)
  qed
  finally show ?P .
  show ?Q by (auto
    simp: log_differentiable_def field_differentiable_def
    intro!: 2 prod_nz)
qed

lemma logderiv_prod:
  fixes f :: 'n ⇒ 'f ⇒ 'f :: real_normed_field
  assumes lder: ⋀i. i ∈ I ⟹ f i log_differentiable a
  shows logderiv (λx. ∏ i∈I. f i x) a = (∑ i∈I. logderiv (f i) a) (is ?P)
    and (λx. ∏ i∈I. f i x) log_differentiable a (is ?Q)
proof -
  consider finite I | infinite I by auto
  hence ?P ∧ ?Q
  proof cases
    assume fin: finite I
    show ?thesis by (auto intro: logderiv_prod' lder fin)
  next
    assume nfin: infinite I
    show ?thesis using nfin
      unfolding logderiv_def log_differentiable_def by auto
  qed
  thus ?P ?Q by auto
qed

lemma logderiv_mult:
  assumes f log_differentiable a
    and g log_differentiable a
  shows logderiv (λz. f z * g z) a = logderiv f a + logderiv g a (is ?P)
    and (λz. f z * g z) log_differentiable a (is ?Q)
proof -
```

```
  have logderiv (λz. f z * g z) a
      = logderiv (λz. ∏ i∈{0, 1}. ([f, g]!i) z) a by auto
  also have ... = (∑ i∈{0, 1}. logderiv ([f, g]!i) a)
    by (rule logderiv_prod(1), use assms in auto)
  also have ... = logderiv f a + logderiv g a
    by auto
  finally show ?P .
  have (λz. ∏ i∈{0, 1}. ([f, g]!i) z) log_differentiable a
    by (rule logderiv_prod(2), use assms in auto)
  thus ?Q by auto
qed

lemma logderiv_cong_ev:
  assumes ∀F x in nhds x. f x = g x
    and x = y
  shows logderiv f x = logderiv g y
proof -
  have deriv f x = deriv g y using assms by (rule deriv_cong_ev)
  moreover have f x = g y using assms by (auto intro: eventually_nhds_x_imp_x)
  ultimately show ?thesis unfolding logderiv_def by auto
qed

lemma logderiv_linear:
  assumes z ≠ a
  shows logderiv (λw. w − a) z = 1 / (z − a)
    and (λw. w − z) log_differentiable a
unfolding logderiv_def log_differentiable_def
  using assms by (auto simp add: derivative_intros)

lemma deriv_shift:
  assumes f field_differentiable at (a + x)
  shows deriv (λt. f (a + t)) x = deriv f (a + x)
proof -
  have deriv (f ∘ (λt. a + t)) x = deriv f (a + x)
    by (subst deriv_chain) (auto intro: assms)
  thus ?thesis unfolding comp_def by auto
qed

lemma logderiv_shift:
  assumes f field_differentiable at (a + x)
  shows logderiv (λt. f (a + t)) x = logderiv f (a + x)
  unfolding logderiv_def by (subst deriv_shift) (auto intro: assms)

lemma logderiv_inverse:
  assumes x ≠ 0
  shows logderiv (λx. 1 / x) x = − 1 / x
proof -
  have deriv (λx. 1 / x) x = (deriv (λx. 1) x * x − 1 * deriv (λx. x) x) / x^2
    by (rule deriv_divide) (use assms in auto)
  hence deriv (λx. 1 / x) x = − 1 / x^2 by auto
  thus ?thesis unfolding logderiv_def power2_eq_square using assms by auto
qed

lemma logderiv_zeta_eq_zeta':
  assumes s ≠ 1 zeta s ≠ 0
```

```
  shows logderiv zeta s = logderiv zeta' s - 1 / (s - 1)
proof -
  have logderiv zeta s = logderiv (λs. zeta' s * (1 / (s - 1))) s
    using zeta_eq_zeta' by auto metis
  also have ... = logderiv zeta' s + logderiv (λs. 1 / (s - 1)) s
  proof -
    have zeta' s ≠ 0 using assms zeta_eq_zero_iff_zeta' by auto
    hence zeta' log_differentiable s
      unfolding log_differentiable_def
      by (intro conjI analytic_on_imp_differentiable_at)
         (rule zeta'_analytic, auto)
    moreover have (λz. 1 / (z - 1)) log_differentiable s
      unfolding log_differentiable_def using assms(1)
      by (intro derivative_intros conjI, auto)
    ultimately show ?thesis using assms by (intro logderiv_mult(1))
  qed
  also have logderiv (λs. 1 / (- 1 + s)) s = logderiv (λs. 1 / s) (- 1 + s)
    by (rule logderiv_shift) (insert assms(1), auto intro: derivative_intros)
  moreover have ... = - 1 / (- 1 + s)
    by (rule logderiv_inverse) (use assms(1) in auto)
  ultimately show ?thesis by auto
qed

lemma analytic_logderiv [analytic_intros]:
  assumes f analytic_on A ⋀z. z ∈ A ⟹ f z ≠ 0
  shows (λs. logderiv f s) analytic_on A
  using assms unfolding logderiv_def by (intro analytic_intros)
```

1.3 Lemmas of integration and integrability

```
lemma powr_has_integral:
  fixes a b w :: real
  assumes Hab: a ≤ b and Hw: w > 0 ∧ w ≠ 1
  shows ((λx. w powr x) has_integral w powr b / ln w - w powr a / ln w) {a..b}
proof (rule fundamental_theorem_of_calculus)
  show a ≤ b using assms by auto
next
  fix x assume x ∈ {a..b}
  have ((λx. exp (x * ln w)) has_vector_derivative exp (x * ln w) * (1 * ln w)) (at x within {a..b})
    by (subst has_real_derivative_iff_has_vector_derivative [symmetric])
       (rule derivative_intros DERIV_cmult_right)+
  hence ((powr) w has_vector_derivative w powr x * ln w) (at x within {a..b})
    unfolding powr_def using Hw by (simp add: DERIV_fun_exp)
  moreover have ln w ≠ 0 using Hw by auto
  ultimately show ((λx. w powr x / ln w) has_vector_derivative w powr x) (at x within {a..b})
    by (auto intro: derivative_eq_intros)
qed

lemma powr_integrable:
  fixes a b w :: real
  assumes Hab: a ≤ b and Hw: w > 0 ∧ w ≠ 1
  shows (λx. w powr x) integrable_on {a..b}
by (rule has_integral_integrable, rule powr_has_integral)
   (use assms in auto)

lemma powr_integral_bound_gt_1:
```

fixes *a b w* :: *real*
assumes *Hab*: $a \leq b$ **and** *Hw*: $w > 1$
shows *integral* $\{a..b\}$ (λx. *w powr x*) $\leq$ *w powr b* / |*ln w*|
proof −
have *integral* $\{a..b\}$ (λx. *w powr x*) = *w powr b* / *ln w* − *w powr a* / *ln w*
by (*intro integral_unique powr_has_integral*) (*use assms* **in** *auto*)
also have ... $\leq$ *w powr b* / |*ln w*| **using** *Hw* **by** *auto*
finally show *?thesis* **.**
qed

lemma *powr_integral_bound_lt_1*:
fixes *a b w* :: *real*
assumes *Hab*: $a \leq b$ **and** *Hw*: $0 < w \wedge w < 1$
shows *integral* $\{a..b\}$ (λx. *w powr x*) $\leq$ *w powr a* / |*ln w*|
proof −
have *integral* $\{a..b\}$ (λx. *w powr x*) = *w powr b* / *ln w* − *w powr a* / *ln w*
by (*intro integral_unique powr_has_integral*) (*use assms* **in** *auto*)
also have ... $\leq$ *w powr a* / |*ln w*| **using** *Hw* **by** (*auto simp add*: *field_simps*)
finally show *?thesis* **.**
qed

lemma *set_integrableI_bounded*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
shows *A* ∈ *sets M*
⟹ (λx. *indicator A x* $*_R$ *f x*) ∈ *borel_measurable M*
⟹ *emeasure M A* < ∞
⟹ (*AE x in M*. *x* ∈ *A* ⟶ *norm* (*f x*) $\leq$ *B*)
⟹ *set_integrable M A f*
unfolding *set_integrable_def*
by (*rule integrableI_bounded_set*[**where** *A*=*A*]) *auto*

lemma *integrable_cut'*:
fixes *a b c* :: *real* **and** *f* :: *real* ⇒ *real*
assumes $a \leq b$ $b \leq c$
and *Hf*: ⋀*x*. $a \leq x$ ⟹ *f integrable_on* $\{a..x\}$
shows *f integrable_on* $\{b..c\}$
proof −
have $a \leq c$ **using** *assms* **by** *linarith*
hence *f integrable_on* $\{a..c\}$ **by** (*rule Hf*)
thus *?thesis* **by**
(*rule integrable_subinterval_real*)
(*subst subset_iff*, (*subst atLeastAtMost_iff*)+,
blast intro: ‹$a \leq b$› *order_trans* [*of a b*])
qed

lemma *integration_by_part'*:
fixes *a b* :: *real*
and *f g* :: *real* ⇒ *'a* :: {*real_normed_field*, *banach*}
and *f' g'* :: *real* ⇒ *'a*
assumes $a \leq b$
and ⋀*x*. *x* ∈ $\{a..b\}$ ⟹ (*f has_vector_derivative f' x*) (*at x*)
and ⋀*x*. *x* ∈ $\{a..b\}$ ⟹ (*g has_vector_derivative g' x*) (*at x*)
and *int*: (λx. *f x* * *g' x*) *integrable_on* $\{a..b\}$
shows ((λx. *f' x* * *g x*) *has_integral*
f b * *g b* − *f a* * *g a* − *integral*$\{a..b\}$ (λx. *f x* * *g' x*)) $\{a..b\}$

```
proof −
  define prod where prod ≡ (∗) :: 'a ⇒ 'a ⇒ 'a
  define y where y ≡ f b ∗ g b − f a ∗ g a − integral{a..b} (λx. f x ∗ g' x)
  have 0: bounded_bilinear prod unfolding prod_def
    by (rule bounded_bilinear_mult)
  have 1: ((λx. f x ∗ g' x) has_integral f b ∗ g b − f a ∗ g a − y) {a..b}
  using y_def and int and integrable_integral by auto
  note 2 = integration_by_parts
    [where y = y and prod = prod, OF 0, unfolded prod_def]
  have continuous_on {a..b} f continuous_on {a..b} g
    by (auto intro: has_vector_derivative_continuous
                has_vector_derivative_at_within assms
          simp: continuous_on_eq_continuous_within)
  with assms and 1 show ?thesis by (fold y_def, intro 2) auto
qed

lemma integral_bigo:
  fixes a :: real and f g :: real ⇒ real
  assumes f_bound: f ∈ O(g)
    and Hf: ⋀x. a ≤ x ⟹ f integrable_on {a..x}
    and Hf': ⋀x. a ≤ x ⟹ (λx. |f x|) integrable_on {a..x}
    and Hg': ⋀x. a ≤ x ⟹ (λx. |g x|) integrable_on {a..x}
  shows (λx. integral{a..x} f) ∈ O(λx. 1 + integral{a..x} (λx. |g x|))
proof −
  from ‹f ∈ O(g)› obtain c where
    ∀F x in at_top. |f x| ≤ c ∗ |g x| and Hc: c ≥ 0
    unfolding bigo_def by auto
  then obtain N' :: real where asymp: ⋀n. n ≥ N' ⟹ |f n| ≤ c ∗ |g n|
    by (subst (asm) eventually_at_top_linorder) (blast)
  define N where N ≡ max a N'
  define I where I ≡ |integral {a..N} f|
  define c' where c' ≡ max I c
  have ⋀x. N ≤ x ⟹ |integral {a..x} f|
      ≤ c' ∗ |1 + integral {a..x} (λx. |g x|)|
  proof −
    fix x :: real
    assume 1: N ≤ x
    define J where J ≡ integral {a..x} (λx. |g x|)
    have 2: a ≤ N unfolding N_def by linarith
    hence 3: a ≤ x using 1 by linarith
    have nnegs: 0 ≤ I 0 ≤ J
      unfolding I_def J_def using 1 2 Hg' by (auto intro!: integral_nonneg)
    hence abs_eq: |I| = I |J| = J
      using nnegs by simp+
    have int|f|: (λx. |f x|) integrable_on {N..x}
      using 2 1 Hf' by (rule integrable_cut')
    have intf: f integrable_on {N..x}
      using 2 1 Hf by (rule integrable_cut')
    have ⋀x. a ≤ x ⟹ (λx. c ∗ |g x|) integrable_on {a..x}
      by (blast intro: Hg' integrable_cmul [OF Hg', simplified])
    hence intc|g|: (λx. c ∗ |g x|) integrable_on {N..x}
      using 2 1 by (blast intro: integrable_cut')
    have |integral {a..x} f| ≤ I + |integral {N..x} f|
      unfolding I_def
      by (subst Henstock_Kurzweil_Integration.integral_combine
```

```
      [OF 2 1 Hf [of x], THEN sym])
      (rule 3, rule abs_triangle_ineq)
  also have ... ≤ I + integral {N..x} (λx. |f x|)
  proof -
    note integral_norm_bound_integral [OF intf int|f|]
    then have |integral {N..x} f| ≤ integral {N..x} (λx. |f x|) by auto
    then show ?thesis by linarith
  qed
  also have ... ≤ I + c ∗ integral {N..x} (λx. |g x|)
  proof -
    have 1: N' ≤ N unfolding N_def by linarith
    hence ⋀y :: real. N ≤ y ⟹ |f y| ≤ c ∗ |g y|
    proof -
      fix y :: real
      assume N ≤ y
      thus |f y| ≤ c ∗ |g y|
        by (rule asymp [OF order_trans [OF 1]])
    qed
    hence integral {N..x} (λx. |f x|) ≤ integral {N..x} (λx. c ∗ |g x|)
      by (rule integral_le [OF int|f| intc|g|]) simp
    thus ?thesis by simp
  qed
  also have ... ≤ I + c ∗ integral {a..x} (λx. |g x|)
  proof -
    note Henstock_Kurzweil_Integration.integral_combine [OF 2 1 Hg' [OF 3]]
    moreover have 0 ≤ integral {a..N} (λx. |g x|)
      by (metis abs_ge_zero Hg' 2 integral_nonneg)
    ultimately show ?thesis
      using Hc by (simp add: landau_omega.R_mult_left_mono)
  qed
  also have ... ≤ c' + c' ∗ integral {a..x} (λx. |g x|)
    unfolding c'_def using Hc
    by (auto intro!: add_mono mult_mono integral_nonneg Hg' 3)
  finally show |integral {a..x} f|
    ≤ c' ∗ |1 + integral {a..x} (λx. |g x|)|
    by (simp add: integral_nonneg Hg' 3 field_simps)
 qed
 note 0 = this
 show ?thesis proof (rule eventually_mono [THEN bigoI])
  show ∀F x in at_top. N ≤ x by simp
  show ⋀x. N ≤ x ⟹
    ‖integral {a..x} f‖ ≤ c' ∗ ‖1 + integral {a..x} (λx. |g x|)‖
    by (auto intro: 0)
 qed
qed

lemma integral_linepath_same_Re:
 assumes Ha: Re a = Re b
  and Hb: Im a < Im b
  and Hf: (f has_contour_integral x) (linepath a b)
 shows ((λt. f (Complex (Re a) t) ∗ i) has_integral x) {Im a..Im b}
proof -
 define path where path ≡ linepath a b
 define c d e g where c ≡ Re a and d ≡ Im a and e ≡ Im b and g ≡ e − d
 hence [simp]: a = Complex c d b = Complex c e by auto (subst Ha, auto)
```

have *hg*: *0* < *g* **unfolding** *g_def* **using** *Hb* **by** *auto*
have [*simp*]: *a* $*_R$ *z* = *a* ∗ *z* **for** *a* **and** *z* :: *complex* **by** (*rule complex_eqI*) *auto*
have ((λ*t*. *f* (*path t*) ∗ (*b* − *a*)) *has_integral x*) {*0*..*1*}
unfolding *path_def* **by** (*subst has_contour_integral_linepath* [*symmetric*]) (*intro Hf*)
moreover have *path t* = *Complex c* (*g* $*_R$ *t* + *d*) **for** *t*
unfolding *path_def linepath_def g_def*
by (*auto simp add*: *field_simps legacy_Complex_simps*)
moreover have *b* − *a* = *g* ∗ i
unfolding *g_def* **by** (*auto simp add*: *legacy_Complex_simps*)
ultimately have
((λ*t*. *f* (*Complex c* (*g* $*_R$ *t* + *d*)) ∗ (*g* ∗ i)) *has_integral g* ∗ *x* $/_R$ *g* ^ *DIM*(*real*))
(*cbox* ((*d* − *d*) $/_R$ *g*) ((*e* − *d*) $/_R$ *g*))
by (*subst* (*6*) *g_def*) (*auto simp add*: *field_simps*)
hence ((λ*t*. *f* (*Complex c t*) ∗ i ∗ *g*) *has_integral x* ∗ *g*) {*d*..*e*}
by (*subst* (*asm*) *has_integral_affinity_iff*)
(*auto simp add*: *field_simps hg*)
hence ((λ*t*. *f* (*Complex c t*) ∗ i ∗ *g* ∗ (*1* / *g*)) *has_integral x* ∗ *g* ∗ (*1* / *g*)) {*d*..*e*}
by (*rule has_integral_mult_left*)
thus *?thesis* **using** *hg* **by** *auto*
qed

1.4 Lemmas on asymptotics

lemma *eventually_at_top_linorderI′*:
fixes *c* :: ′*a* :: {*no_top*, *linorder*}
assumes *h*: ⋀*x*. *c* < *x* ⟹ *P x*
shows *eventually P at_top*
proof (*rule eventually_mono*)
show $\forall_F$ *x in at_top*. *c* < *x* **by** (*rule eventually_gt_at_top*)
from *h* **show** ⋀*x*. *c* < *x* ⟹ *P x* **.**
qed

lemma *eventually_le_imp_bigo*:
assumes $\forall_F$ *x in F*. ‖*f x*‖ ≤ *g x*
shows *f* ∈ *O*[*F*](*g*)
proof −
from *assms* **have** $\forall_F$ *x in F*. ‖*f x*‖ ≤ *1* ∗ ‖*g x*‖ **by** *eventually_elim auto*
thus *?thesis* **by** (*rule bigoI*)
qed

lemma *eventually_le_imp_bigo′*:
assumes $\forall_F$ *x in F*. ‖*f x*‖ ≤ *g x*
shows (λ*x*. ‖*f x*‖) ∈ *O*[*F*](*g*)
proof −
from *assms* **have** $\forall_F$ *x in F*. ‖‖*f x*‖‖ ≤ *1* ∗ ‖*g x*‖
by *eventually_elim auto*
thus *?thesis* **by** (*rule bigoI*)
qed

lemma *le_imp_bigo*:
assumes ⋀*x*. ‖*f x*‖ ≤ *g x*
shows *f* ∈ *O*[*F*](*g*)
by (*intro eventually_le_imp_bigo eventuallyI assms*)

lemma *le_imp_bigo′*:
assumes ⋀*x*. ‖*f x*‖ ≤ *g x*

```
  shows (λx. ‖f x‖) ∈ O[F](g)
  by (intro eventually_le_imp_bigo′ eventuallyI assms)

lemma exp_bigo:
  fixes f g :: real ⇒ real
  assumes ∀F x in at_top. f x ≤ g x
  shows (λx. exp (f x)) ∈ O(λx. exp (g x))
proof −
  from assms have ∀F x in at_top. exp (f x) ≤ exp (g x) by simp
  hence ∀F x in at_top. ‖exp (f x)‖ ≤ 1 ∗ ‖exp (g x)‖ by simp
  thus ?thesis by blast
qed

lemma ev_le_imp_exp_bigo:
  fixes f g :: real ⇒ real
  assumes hf: ∀F x in at_top. 0 < f x
    and hg: ∀F x in at_top. 0 < g x
    and le: ∀F x in at_top. ln (f x) ≤ ln (g x)
  shows f ∈ O(g)
proof −
  have ∀F x in at_top. exp (ln (f x)) ≤ exp (ln (g x))
    using le by simp
  hence ∀F x in at_top. ‖f x‖ ≤ 1 ∗ ‖g x‖
    using hf hg by eventually_elim auto
  thus ?thesis by (intro bigoI)
qed

lemma smallo_ln_diverge_1:
  fixes f :: real ⇒ real
  assumes f_ln: f ∈ o(ln)
  shows LIM x at_top. x ∗ exp (− f x) :> at_top
proof −
  have (λx. ln x − f x) ∼[at_top] (λx. ln x)
    using assms by (simp add: asymp_equiv_altdef)
  moreover have filterlim (λx. ln x :: real) at_top at_top
    by real_asymp
  ultimately have filterlim (λx. ln x − f x) at_top at_top
    using asymp_equiv_at_top_transfer asymp_equiv_sym by blast
  hence filterlim (λx. exp (ln x − f x)) at_top at_top
    by (rule filterlim_compose[OF exp_at_top])
  moreover have ∀F x in at_top. exp (ln x − f x) = x ∗ exp (− f x)
    using eventually_gt_at_top[of 0]
    by eventually_elim (auto simp: exp_diff exp_minus field_simps)
  ultimately show ?thesis
    using filterlim_cong by fast
qed

lemma ln_ln_asymp_pos: ∀F x :: real in at_top. 0 < ln (ln x) by real_asymp
lemma ln_asymp_pos: ∀F x :: real in at_top. 0 < ln x by real_asymp
lemma x_asymp_pos: ∀F x :: real in at_top. 0 < x by auto
```

1.5 Lemmas of *floor*, *ceil* and *nat_powr*

```
lemma nat_le_self: 0 ≤ x ⟹ nat (int x) ≤ x by auto
lemma floor_le: ⋀x :: real. ⌊x⌋ ≤ x by auto
lemma ceil_ge: ⋀x :: real. x ≤ ⌈x⌉ by auto
```

lemma *nat_lt_real_iff*:
  $(n :: nat) < (a :: real) = (n < nat \lceil a \rceil)$
proof −
  have $n < a = (of\_int\ n < a)$ **by** *auto*
  also have ... $= (n < \lceil a \rceil)$ **by** (*rule less_ceiling_iff* [*symmetric*])
  also have ... $= (n < nat \lceil a \rceil)$ **by** *auto*
  finally show *?thesis* **.**
qed

lemma *nat_le_real_iff*:
  $(n :: nat) \leq (a :: real) = (n < nat\ (\lfloor a \rfloor + 1))$
proof −
  have $n \leq a = (of\_int\ n \leq a)$ **by** *auto*
  also have ... $= (n \leq \lfloor a \rfloor)$ **by** (*rule le_floor_iff* [*symmetric*])
  also have ... $= (n < \lfloor a \rfloor + 1)$ **by** *auto*
  also have ... $= (n < nat\ (\lfloor a \rfloor + 1))$ **by** *auto*
  finally show *?thesis* **.**
qed

lemma *of_real_nat_power*: $n\ nat\_powr\ (of\_real\ x :: complex) = of\_real\ (n\ nat\_powr\ x)$ **for** $n\ x$
  by (*subst of_real_of_nat_eq* [*symmetric*])
    (*subst powr_of_real*, *auto*)

lemma *norm_nat_power*: $\|n\ nat\_powr\ (s :: complex)\| = n\ powr\ (Re\ s)$
  unfolding *powr_def* **by** *auto*

1.6 Elementary estimation of *exp* and *ln*

lemma *ln_when_ge_3*:
  $1 < ln\ x$ **if** $3 \leq x$ **for** $x :: real$
proof (*rule ccontr*)
  assume $\neg\ 1 < ln\ x$
  hence $exp\ (ln\ x) \leq exp\ 1$ **by** *auto*
  hence $x \leq exp\ 1$ **using** *that* **by** *auto*
  thus *False* **using** *e_less_272 that* **by** *auto*
qed

lemma *exp_lemma_1*:
  fixes $x :: real$
  assumes $1 \leq x$
  shows $1 + exp\ x \leq exp\ (2 * x)$
proof −
  let $?y = exp\ x$
  have $ln\ 2 \leq x$ **using** *assms ln_2_less_1* **by** *auto*
  hence $exp\ (ln\ 2) \leq ?y$ **by** (*subst exp_le_cancel_iff*)
  hence $(3\ /\ 2)^2 \leq (?y - 1\ /\ 2)^2$ **by** *auto*
  hence $0 \leq -\ 5\ /\ 4 + (?y - 1\ /\ 2)^2$ **by** (*simp add*: *power2_eq_square*)
  also have ... $= ?y^2 - ?y - 1$ **by** (*simp add*: *power2_eq_square field_simps*)
  finally show *?thesis* **by** (*simp add*: *exp_double*)
qed

lemma *ln_bound_1*:
  fixes $t :: real$
  assumes *Ht*: $0 \leq t$
  shows $ln\ (14 + 4 * t) \leq 4 * ln\ (t + 2)$

```
proof -
  have ln (14 + 4 * t) ≤ ln (14 / 2 * (t + 2)) using Ht by auto
  also have ... = ln 7 + ln (t + 2) using Ht by (subst ln_mult) auto
  also have ... ≤ 3 * ln (t + 2) + ln (t + 2) proof -
    have (14 :: real) ≤ 2 powr 4 by auto
    hence exp (ln (14 :: real)) ≤ exp (4 * ln 2)
      unfolding powr_def by (subst exp_ln) auto
    hence ln (14 :: real) ≤ 4 * ln 2 by (subst (asm) exp_le_cancel_iff)
    hence ln (14 / 2 :: real) ≤ 3 * ln 2 by (subst ln_div) auto
    also have ... ≤ 3 * ln (t + 2) using Ht by auto
    finally show ?thesis by auto
  qed
  also have ... = 4 * ln (t + 2) by auto
  finally show ?thesis by (auto simp add: field_simps)
qed
```

1.7 Miscellaneous lemmas

```
abbreviation fds_zeta_complex :: complex fds ≡ fds_zeta

lemma powr_mono_lt_1_cancel:
  fixes x a b :: real
  assumes Hx: 0 < x ∧ x < 1
  shows (x powr a ≤ x powr b) = (b ≤ a)
  by (smt (verit, best) Hx powr_less_mono')

abbreviation mangoldt_real :: _ ⇒ real ≡ mangoldt
abbreviation mangoldt_complex :: _ ⇒ complex ≡ mangoldt

lemma norm_fds_mangoldt_complex:
  ⋀n. ‖fds_nth (fds mangoldt_complex) n‖ = mangoldt_real n by (simp add: fds_nth_fds)

lemma suminf_norm_bound:
  fixes f :: nat ⇒ 'a :: banach
  assumes summable g
    and ⋀n. ‖f n‖ ≤ g n
  shows ‖suminf f‖ ≤ (∑n. g n)
proof -
  have *: summable (λn. ‖f n‖)
    by (rule summable_comparison_test' [where g = g])
       (use assms in auto)
  hence ‖suminf f‖ ≤ (∑n. ‖f n‖) by (rule summable_norm)
  also have (∑n. ‖f n‖) ≤ (∑n. g n)
    by (rule suminf_le) (use assms * in auto)
  finally show ?thesis .
qed

lemma C₁_gt_zero: 0 < C₁ unfolding PNT_const_C₁_def by auto

unbundle no pnt_syntax

end
theory Relation_of_PNTs
imports
  PNT_Remainder_Library
begin
```

```
unbundle pnt_syntax
unbundle prime_counting_syntax
```

2 Implication relation of many forms of prime number theorem

```
definition rem_est :: real ⇒ real ⇒ real ⇒ _ where
rem_est c m n ≡ O(λ x. x * exp (−c * ln x powr m * ln (ln x) powr n))

definition Li :: real ⇒ real where Li x ≡ integral {2..x} (λx. 1 / ln x)
definition PNT_1 where PNT_1 c m n ≡ ((λx. π x − Li x) ∈ rem_est c m n)
definition PNT_2 where PNT_2 c m n ≡ ((λx. ϑ x − x) ∈ rem_est c m n)
definition PNT_3 where PNT_3 c m n ≡ ((λx. ψ x − x) ∈ rem_est c m n)

lemma rem_est_compare_powr:
  fixes c m n :: real
  assumes h: 0 < m m < 1
  shows (λx. x powr (2 / 3)) ∈ rem_est c m n
  unfolding rem_est_def using assms
  by (cases c 0 :: real rule: linorder_cases; real_asymp)

lemma PNT_3_imp_PNT_2:
  fixes c m n :: real
  assumes h: 0 < m m < 1 and PNT_3 c m n
  shows PNT_2 c m n
proof −
  have 1: (λ x. ψ x − x) ∈ rem_est c m n
    using assms(3) unfolding PNT_3_def by auto
  have (λx. ψ x − ϑ x) ∈ O(λx. ln x * sqrt x) by (rule ψ_minus_ϑ_bigo)
  moreover have (λx. ln x * sqrt x) ∈ O(λx. x powr (2 / 3)) by real_asymp
  ultimately have 2: (λx. ψ x − ϑ x) ∈ rem_est c m n
    using rem_est_compare_powr [OF h, of c n] unfolding rem_est_def
    by (blast intro: landau_o.big.trans)
  have (λx. ψ x − x − (ψ x − ϑ x)) ∈ rem_est c m n
    using 1 2 unfolding rem_est_def by (rule sum_in_bigo)
  thus ?thesis unfolding PNT_2_def by simp
qed

definition r₁ where r₁ x ≡ π x − Li x for x
definition r₂ where r₂ x ≡ ϑ x − x for x

lemma pi_represent_by_theta:
  fixes x :: real
  assumes 2 ≤ x
  shows π x = ϑ x / (ln x) + integral {2..x} (λt. ϑ t / (t * (ln t)^2))
proof −
  note integral_unique [OF π_conv_ϑ_integral]
  with assms show ?thesis by auto
qed

lemma Li_integrate_by_part:
  fixes x :: real
  assumes 2 ≤ x
  shows
```

```
  (λx. 1 / (ln x)^2) integrable_on {2..x}
  Li x = x / (ln x) − 2 / (ln 2) + integral {2..x} (λt. 1 / (ln t)^2)
proof −
  have (λx. x ∗ (− 1 / (x ∗ (ln x)^2))) integrable_on {2..x}
    by (rule integrable_continuous_interval)
       ((rule continuous_intros)+, auto)
  hence (λx. − (if x = 0 then 0 else 1 / (ln x)^2)) integrable_on {2..x}
    by simp
  moreover have ((λt. 1 / ln t) has_vector_derivative −1 / (t ∗ (ln t)^2)) (at t)
    when Ht: 2 ≤ t for t
  proof −
    define a where a ≡ (0 ∗ ln t − 1 ∗ (1 / t))/(ln t ∗ ln t)
    have DERIV (λt. 1 / (ln t)) t :> a
    unfolding a_def
    proof (rule derivative_intros DERIV_ln_divide)+
      from Ht show 0 < t by linarith
      note ln_gt_zero and Ht thus ln t ≠ 0 by auto
    qed
    also have a = −1 / (t ∗ (ln t)^2)
      unfolding a_def by (simp add: power2_eq_square)
    finally have DERIV (λt. 1 / (ln t)) t :> −1 / (t ∗ (ln t)^2) by auto
    thus ?thesis
      by (subst has_real_derivative_iff_has_vector_derivative [symmetric])
  qed
  ultimately have ((λx. 1 ∗ (1 / ln x)) has_integral
    x ∗ (1 / ln x) − 2 ∗ (1 / ln 2) − integral {2..x} (λx. x ∗ (−1 / (x ∗ (ln x)^2))))
    {2..x}
    using ‹2 ≤ x› by (intro integration_by_part') auto
  note 3 = this [simplified]
  have ((λx. 1 / ln x) has_integral (x / ln x − 2 / ln 2 + integral {2..x} (λx. 1 / (ln x)^2))) {2..x}
  proof −
    define a where a t ≡ if t = 0 then 0 else 1 / (ln t)^2 for t :: real
    have ⋀t :: real. t ∈ {2..x} ⟹ a t = 1 / (ln t)^2
      unfolding a_def by auto
    hence 4: integral {2..x} a = integral {2..x} (λx. 1 / (ln x)^2) by (rule integral_cong)
    from 3 show ?thesis
      by (subst (asm) 4 [unfolded a_def])
  qed
  thus Li x = x / ln x − 2 / ln 2 + integral {2..x} (λt. 1 / (ln t)^2) unfolding Li_def by auto
  show (λx. 1 / (ln x)^2) integrable_on {2..x}
    by (rule integrable_continuous_interval)
       ((rule continuous_intros)+, auto)
qed

lemma ϑ_integrable:
  fixes x :: real
  assumes 2 ≤ x
  shows (λt. ϑ t / (t ∗ (ln t)^2)) integrable_on {2..x}
by (rule π_conv_ϑ_integral [THEN has_integral_integrable], rule assms)

lemma r_1_represent_by_r_2:
  fixes x :: real
  assumes Hx: 2 ≤ x
  shows (λt. r_2 t / (t ∗ (ln t)^2)) integrable_on {2..x} (is ?P)
    r_1 x = r_2 x / (ln x) + 2 / ln 2 + integral {2..x} (λt. r_2 t / (t ∗ (ln t)^2)) (is ?Q)
```

```
proof -
  have 0: ⋀t. t ∈ {2..x} ⟹ (ϑ t − t) / (t ∗ (ln t)^2) = ϑ t / (t ∗ (ln t)^2) − 1 / (ln t)^2
    by (subst diff_divide_distrib, auto)
  note integrables = ϑ_integrable Li_integrate_by_part(1)
  let ?D = integral {2..x} (λt. ϑ t / (t ∗ (ln t)^2)) −
    integral {2..x} (λt. 1 / (ln t)^2)
  have ((λt. ϑ t / (t ∗ (ln t)^2) − 1 / (ln t)^2) has_integral
    ?D) {2..x}
  unfolding r_2_def by
    (rule has_integral_diff)
    (rule integrables [THEN integrable_integral], rule Hx)+
  hence 0: ((λt. r_2 t / (t ∗ (ln t)^2)) has_integral
    ?D) {2..x}
  unfolding r_2_def by (subst has_integral_cong [OF 0])
  thus ?P by (rule has_integral_integrable)
  note 1 = 0 [THEN integral_unique]
  have 2: r_2 x / ln x = ϑ x / ln x − x / ln x
    unfolding r_2_def by (rule diff_divide_distrib)
  from pi_represent_by_theta and Li_integrate_by_part(2) and assms
  have π x − Li x = ϑ x / ln x
    + integral {2..x} (λt. ϑ t / (t ∗ (ln t)^2))
    − (x / ln x − 2 / ln 2 + integral {2..x} (λt. 1 / (ln t)^2))
    by auto
  also have ... = r_2 x / ln x + 2 / ln 2
    + integral {2..x} (λt. r_2 t / (t ∗ (ln t)^2))
    by (subst 2, subst 1) auto
  finally show ?Q unfolding r_1_def by auto
qed

lemma exp_integral_asymp:
  fixes f f' :: real ⇒ real
  assumes cf: continuous_on {a..} f
      and der: ⋀x. a < x ⟹ DERIV f x :> f' x
      and td: ((λx. x ∗ f' x) ⟶ 0) at_top
      and f_ln: f ∈ o(ln)
  shows (λx. integral {a..x} (λt. exp (−f t))) ∼[at_top] (λx. x ∗ exp(−f x))
proof (rule asymp_equivI', rule lhospital_at_top_at_top)
  have cont_exp: continuous_on {a..} (λt. exp (− f t))
    using cf by (intro continuous_intros)
  show ∀_F x in at_top. ((λx. integral {a..x} (λt. exp (− f t)))
    has_real_derivative exp (− f x)) (at x) (is eventually ?P ?F)
  proof (rule eventually_at_top_linorderI')
    fix x assume 1: a < x
    hence 2: a ≤ x by linarith
    have 3: (at x within {a..x+1}) = (at x)
      by (rule at_within_interior) (auto intro: 1)
    show ?P x
      by (subst 3 [symmetric], rule integral_has_real_derivative)
         (rule continuous_on_subset [OF cont_exp], auto intro: 2)
  qed
  have ∀_F x in at_top. ((λx. x ∗ exp (− f x))
    has_real_derivative 1 ∗ exp (− f x) + exp (− f x) ∗ (− f' x) ∗ x) (at x)
    (is eventually ?P ?F)
  proof (rule eventually_at_top_linorderI')
    fix x assume 1: a < x
```

```
    hence 2: (at x within {a<..}) = (at x) by (auto intro: at_within_open)
    show ?P x
      by (subst 2 [symmetric], intro derivative_intros)
        (subst 2, rule der, rule 1)
  qed
  moreover have
    1 ∗ exp (− f x) + exp (− f x) ∗ (− f′ x) ∗ x
    = exp (− f x) ∗ (1 − x ∗ f′ x) for x :: real
    by (simp add: field_simps)
  ultimately show ∀F x in at_top.
      ((λx. x ∗ exp (− f x))
    has_real_derivative exp (− f x) ∗ (1 − x ∗ f′ x)) (at x) by auto
  show LIM x at_top. x ∗ exp (− f x) :> at_top
    using f_ln by (rule smallo_ln_diverge_1)
  have ((λx. 1 / (1 − x ∗ f′ x)) ⟶ 1 / (1 − 0)) at_top
    by ((rule tendsto_intros)+, rule td, linarith)
  thus ((λx. exp (− f x) / (exp (− f x) ∗ (1 − x ∗ f′ x))) ⟶ 1) at_top by auto
  have ((λx. 1 − x ∗ f′ x) ⟶ 1 − 0) at_top
    by ((rule tendsto_intros)+, rule td)
  hence 0: ((λx. 1 − x ∗ f′ x) ⟶ 1) at_top by simp
  hence ∀F x in at_top. 0 < 1 − x ∗ f′ x
    by (rule order_tendstoD) linarith
  moreover have ∀F x in at_top. 0 < 1 − x ∗ f′ x ⟶ exp (− f x) ∗ (1 − x ∗ f′ x) ≠ 0 by auto
  ultimately show ∀F x in at_top. exp (− f x) ∗ (1 − x ∗ f′ x) ≠ 0
    by (rule eventually_rev_mp)
qed

lemma x_mul_exp_larger_than_const:
  fixes c :: real and g :: real ⇒ real
  assumes g_ln: g ∈ o(ln)
  shows (λx. c) ∈ O(λx. x ∗ exp(−g x))
proof −
  have LIM x at_top. x ∗ exp (− g x) :> at_top
    using g_ln by (rule smallo_ln_diverge_1)
  hence ∀F x in at_top. 1 ≤ x ∗ exp (− g x)
    using filterlim_at_top by fast
  hence ∀F x in at_top. ‖c‖ ∗ 1 ≤ ‖c‖ ∗ ‖x ∗ exp (− g x)‖
    by (rule eventually_rev_mp)
      (auto simp del: mult_1_right
           intro!: eventuallyI mult_left_mono)
  thus (λx. c :: real) ∈ O(λx. x ∗ exp (− g x)) by auto
qed

lemma integral_bigo_exp′:
  fixes a :: real and f g g′ :: real ⇒ real
  assumes f_bound: f ∈ O(λx. exp(−g x))
    and Hf:  ⋀x. a ≤ x ⟹ f integrable_on {a..x}
    and Hf′: ⋀x. a ≤ x ⟹ (λx. |f x|) integrable_on {a..x}
    and Hg:  continuous_on {a..} g
    and der: ⋀x. a < x ⟹ DERIV g x :> g′ x
    and td:  ((λx. x ∗ g′ x) ⟶ 0) at_top
    and g_ln: g ∈ o(ln)
  shows (λx. integral{a..x} f) ∈ O(λx. x ∗ exp(−g x))
proof −
  have ⋀y. continuous_on {a..y} g
```

```
    by (rule continuous_on_subset, rule Hg) auto
  hence ⋀y. (λx. exp(−g x)) integrable_on {a..y}
    by (intro integrable_continuous_interval)
       (rule continuous_intros)+
  hence ⋀y. (λx. |exp(−g x)|) integrable_on {a..y} by simp
  hence (λx. integral{a..x} f) ∈ O(λx. 1 + integral{a..x} (λx. |exp(−g x)|))
    using assms by (intro integral_bigo)
  hence (λx. integral{a..x} f) ∈ O(λx. 1 + integral{a..x} (λx. exp(−g x))) by simp
  also have (λx. 1 + integral{a..x} (λx. exp(−g x))) ∈ O(λx. x ∗ exp(−g x))
  proof (rule sum_in_bigo)
    show (λx. 1 :: real) ∈ O(λx. x ∗ exp (− g x))
      by (intro x_mul_exp_larger_than_const g_ln)
    show (λx. integral {a..x} (λx. exp (− g x))) ∈ O(λx. x ∗ exp (− g x))
      by (rule asymp_equiv_imp_bigo, rule exp_integral_asymp, auto intro: assms)
  qed
  finally show ?thesis .
qed

lemma integral_bigo_exp:
  fixes a b :: real and f g g′ :: real ⇒ real
  assumes le: a ≤ b
    and f_bound: f ∈ O(λx. exp(−g x))
    and Hf: ⋀x. a ≤ x ⟹ f integrable_on {a..x}
    and Hf′: ⋀x. b ≤ x ⟹ (λx. |f x|) integrable_on {b..x}
    and Hg: continuous_on {b..} g
    and der: ⋀x. b < x ⟹ DERIV g x :> g′ x
    and td: ((λx. x ∗ g′ x) ⟶ 0) at_top
    and g_ln:g ∈ o(ln)
  shows (λx. integral {a..x} f) ∈ O(λx. x ∗ exp(−g x))
proof −
  have (λx. integral {a..b} f) ∈ O(λx. x ∗ exp(−g x))
    by (intro x_mul_exp_larger_than_const g_ln)
  moreover have (λx. integral {b..x} f) ∈ O(λx. x ∗ exp(−g x))
    by (intro integral_bigo_exp′ [where ?g′ = g′]
             f_bound Hf Hf′ Hg der td g_ln)
       (use le Hf integrable_cut′ in auto)
  ultimately have (λx. integral {a..b} f + integral {b..x} f) ∈ O(λx. x ∗ exp(−g x))
    by (rule sum_in_bigo)
  moreover have integral {a..x} f = integral {a..b} f + integral {b..x} f when b ≤ x for x
    by (subst eq_commute, rule Henstock_Kurzweil_Integration.integral_combine)
       (insert le that, auto intro: Hf)
  hence ∀F x in at_top. integral {a..x} f = integral {a..b} f + integral {b..x} f
    by (rule eventually_at_top_linorderI)
  ultimately show ?thesis
    by (simp add: landau_o.big.in_cong)
qed

lemma integrate_r2_estimate:
  fixes c m n :: real
  assumes hm: 0 < m m < 1
    and h: r2 ∈ rem_est c m n
  shows (λx. integral {2..x} (λt. r2 t / (t ∗ (ln t)^2))) ∈ rem_est c m n
unfolding rem_est_def
proof (subst mult.assoc,
       subst minus_mult_left [symmetric],
```

```
    rule integral_bigo_exp)
show (2 :: real) ≤ 3 by auto
show (λx. c ∗ (ln x powr m ∗ ln (ln x) powr n)) ∈ o(ln)
  using hm by real_asymp
have ln x ≠ 1 when 3 ≤ x for x :: real
  using ln_when_ge_3 [of x] that by auto
thus continuous_on {3..} (λx. c ∗ (ln x powr m ∗ ln (ln x) powr n))
  by (intro continuous_intros) auto
show (λt. r2 t / (t ∗ (ln t)^2)) integrable_on {2..x}
  if 2 ≤ x for x using that by (rule r1_represent_by_r2(1))
define g where g x ≡
  c ∗ (m ∗ ln x powr (m − 1) ∗ (1 / x ∗ 1) ∗ ln (ln x) powr n
    + n ∗ ln (ln x) powr (n − 1) ∗ (1 / ln x ∗ (1 / x)) ∗ ln x powr m)
  for x
show ((λx. c ∗ (ln x powr m ∗ ln (ln x) powr n)) has_real_derivative g x) (at x)
  if 3 < x for x
proof −
  have ∗: at x within {3<..} = at x
    by (rule at_within_open) (auto intro: that)
  moreover have
    ((λx. c ∗ (ln x powr m ∗ ln (ln x) powr n)) has_real_derivative g x)
     (at x within {3<..})
  unfolding g_def using that
  by (intro derivative_intros DERIV_mult DERIV_cmult)
     (auto intro: ln_when_ge_3 DERIV_ln_divide simp add: ∗)
  ultimately show ?thesis by auto
qed
show ((λx. x ∗ g x) ⟶ 0) at_top
  unfolding g_def using hm by real_asymp
have nz: ∀F t :: real in at_top. t ∗ (ln t)^2 ≠ 0
proof (rule eventually_at_top_linorderI')
  fix x :: real assume 1 < x
  thus x ∗ (ln x)^2 ≠ 0 by auto
qed
define h where h x ≡ exp (− c ∗ ln x powr m ∗ ln (ln x) powr n) for x
have (λt. r2 t / (t ∗ (ln t)^2)) ∈ O(λx. (x ∗ h x) / (x ∗ (ln x)^2))
  by (rule landau_o.big.divide_right, rule nz)
     (unfold h_def, fold rem_est_def, rule h)
also have (λx. (x ∗ h x) / (x ∗ (ln x)^2)) ∈ O(λx. h x)
proof −
  have (λx :: real. 1 / (ln x)^2) ∈ O(λx. 1) by real_asymp
  hence (λx. h x ∗ (1 / (ln x)^2)) ∈ O(λx. h x ∗ 1)
    by (rule landau_o.big.mult_left)
  thus ?thesis
    by (auto simp add: field_simps intro!: landau_o.big.ev_eq_trans2)
qed
finally show (λt. r2 t / (t ∗ (ln t)^2))
  ∈ O(λx. exp (− (c ∗ (ln x powr m ∗ ln (ln x) powr n))))
  unfolding h_def by (simp add: algebra_simps)
have (λx. r2 x / (x ∗ (ln x)^2)) absolutely_integrable_on {2..x}
  if ∗:2 ≤ x for x
proof (rule set_integrableI_bounded)
  show {2..x} ∈ sets lebesgue by auto
  show emeasure lebesgue {2..x} < ∞ using ∗ by auto
  have (λt. r2 t / (t ∗ (ln t)^2) ∗ indicator {2..x} t) ∈ borel_measurable lebesgue
```

```
      using ∗ by (intro integrable_integral
        [THEN has_integral_implies_lebesgue_measurable_real])
        (rule r_1_represent_by_r_2(1))
    thus (λt. indicat_real {2..x} t ∗_R (r_2 t / (t ∗ (ln t)^2))) ∈ borel_measurable lebesgue
      by (simp add: mult_ac)
    let ?C = (ln 4 + 1) / (ln 2)^2 :: real
    show AE t∈{2..x} in lebesgue. ‖r_2 t / (t ∗ (ln t)^2)‖ ≤ ?C
    proof (rule AE_I2, safe)
      fix t assume t ∈ {2..x}
      hence h: 1 ≤ t 2 ≤ t by auto
      hence 0 ≤ ϑ t ∧ ϑ t < ln 4 ∗ t by (auto intro: ϑ_upper_bound)
      hence ∗:|ϑ t| ≤ ln 4 ∗ t by auto
      have 1 ≤ ln t / ln 2 using h by auto
      hence 1 ≤ (ln t / ln 2)^2 by auto
      also have ... = (ln t)^2 / (ln 2)^2 unfolding power2_eq_square by auto
      finally have 1 ≤ (ln t)^2 / (ln 2)^2 .
      hence |r_2 t| ≤ |ϑ t| + |t| unfolding r_2_def by auto
      also have ... ≤ ln 4 ∗ t + 1 ∗ t using h ∗ by auto
      also have ... = (ln 4 + 1) ∗ t by (simp add: algebra_simps)
      also have ... ≤ (ln 4 + 1) ∗ t ∗ ((ln t)^2 / (ln 2)^2)
        by (auto simp add: field_simps)
           (rule add_mono; rule rev_mp[OF h(2)], auto)
      finally have ∗:|r_2 t| ≤ ?C ∗ (t ∗ (ln t)^2) by auto
      thus ‖r_2 t / (t ∗ (ln t)^2)‖ ≤ ?C
        using h ∗ by (auto simp add: field_simps)
    qed
  qed
  hence ⋀x. 2 ≤ x ⟹ (λx. |r_2 x / (x ∗ (ln x)^2)|) integrable_on {2..x}
    by (fold real_norm_def)
       (rule absolutely_integrable_on_def [THEN iffD1, THEN conjunct2])
  thus ⋀x. 3 ≤ x ⟹ (λx. |r_2 x / (x ∗ (ln x)^2)|) integrable_on {3..x}
    using ‹2 ≤ 3› integrable_cut' by blast
qed

lemma r_2_div_ln_estimate:
  fixes c m n :: real
  assumes hm: 0 < m m < 1
    and h: r_2 ∈ rem_est c m n
  shows (λx. r_2 x / (ln x) + 2 / ln 2) ∈ rem_est c m n
proof -
  have (λx. r_2 x / ln x) ∈ O(r_2)
  proof (intro bigoI eventually_at_top_linorderI)
    fix x :: real assume 1:exp 1 ≤ x
    have 2:(0 :: real) < exp 1 by simp
    hence 3:0 < x using 1 by linarith
    have 4: 0 ≤ |r_2 x| by auto
    have (1 :: real) = ln (exp 1) by simp
    also have ... ≤ ln x using 1 2 3 by (subst ln_le_cancel_iff)
    finally have 1 ≤ ln x .
    thus ‖r_2 x / ln x‖ ≤ 1 ∗ ‖r_2 x‖
      by (auto simp add: field_simps, subst mult_le_cancel_right1, auto)
  qed
  with h have 1: (λx. r_2 x / ln x) ∈ rem_est c m n
    unfolding rem_est_def using landau_o.big_trans by blast
  moreover have (λx :: real. 2 / ln 2) ∈ O(λx. x powr (2 / 3))
```

by *real_asymp*
hence $(\lambda x :: real.\ 2\ /\ ln\ 2) \in rem\_est\ c\ m\ n$
using *rem_est_compare_powr* [*OF hm*, *of c n*]
unfolding *rem_est_def* **by** (*rule landau_o.big.trans*)
ultimately show *?thesis*
unfolding *rem_est_def* **by** (*rule sum_in_bigo*)
qed

lemma *PNT_2_imp_PNT_1*:
fixes *l* :: *real*
assumes *h*: $0 < m$ $m < 1$ **and** *PNT_2 c m n*
shows *PNT_1 c m n*
proof −
from *assms*(*3*) **have** h': $r_2 \in rem\_est\ c\ m\ n$
unfolding *PNT_2_def* r_2*_def* **by** *auto*
let $?a = \lambda x.\ r_2\ x\ /\ ln\ x + 2\ /\ ln\ 2$
let $?b = \lambda x.\ integral\ \{2..x\}\ (\lambda t.\ r_2\ t\ /\ (t * (ln\ t)^2))$
have *1*: $\forall_F\ x\ in\ at\_top.\ \pi\ x - Li\ x = ?a\ x + ?b\ x$
by (*rule eventually_at_top_linorderI*, *fold* r_1*_def*)
(*rule* r_1*_represent_by_*r_2(*2*), *blast*)
have *2*: $(\lambda x.\ ?a\ x + ?b\ x) \in rem\_est\ c\ m\ n$
by (*unfold rem_est_def*, (*rule sum_in_bigo*; *fold rem_est_def*))
(*intro* r_2*_div_ln_estimate integrate_*r_2*_estimate h h'*)+
from *landau_o.big.in_cong* [*OF 1*] **and** *2* **show** *?thesis*
unfolding *PNT_1_def rem_est_def* **by** *blast*
qed

theorem *PNT_3_imp_PNT_1*:
fixes *l* :: *real*
assumes *h* : $0 < m$ $m < 1$ **and** *PNT_3 c m n*
shows *PNT_1 c m n*
by (*intro PNT_2_imp_PNT_1 PNT_3_imp_PNT_2 assms*)

hide_const (**open**) r_1 r_2
unbundle *no prime_counting_syntax* **and** *no pnt_syntax*

end
theory *PNT_Complex_Analysis_Lemmas*
imports
PNT_Remainder_Library
begin
unbundle *pnt_syntax*

3 Some basic theorems in complex analysis

3.1 Introduction rules for holomorphic functions and analytic functions

lemma *holomorphic_on_shift* [*holomorphic_intros*]:
assumes *f holomorphic_on* $((\lambda z.\ s + z)$ ` $A)$
shows $(\lambda z.\ f\ (s + z))$ *holomorphic_on A*
proof −
have $(f \circ (\lambda z.\ s + z))$ *holomorphic_on A*
using *assms* **by** (*intro holomorphic_on_compose holomorphic_intros*)
thus *?thesis* **unfolding** *comp_def* **by** *auto*
qed

```
lemma holomorphic_logderiv [holomorphic_intros]:
  assumes f holomorphic_on A open A ⋀z. z ∈ A ⟹ f z ≠ 0
  shows (λs. logderiv f s) holomorphic_on A
  using assms unfolding logderiv_def by (intro holomorphic_intros)

lemma holomorphic_glue_to_analytic:
  assumes o: open S open T
     and hf: f holomorphic_on S
     and hg: g holomorphic_on T
     and hI: ⋀z. z ∈ S ⟹ z ∈ T ⟹ f z = g z
     and hU: U ⊆ S ∪ T
  obtains h
   where h analytic_on U
         ⋀z. z ∈ S ⟹ h z = f z
         ⋀z. z ∈ T ⟹ h z = g z
proof -
  define h where h z ≡ if z ∈ S then f z else g z for z
  show ?thesis proof
    have h holomorphic_on S ∪ T
      unfolding h_def by (rule holomorphic_on_If_Un) (use assms in auto)
    thus h analytic_on U
      by (subst analytic_on_holomorphic) (use hU o in auto)
  next
    fix z assume *:z ∈ S
    show h z = f z unfolding h_def using * by auto
  next
    fix z assume *:z ∈ T
    show h z = g z unfolding h_def using * hI by auto
  qed
qed

lemma analytic_on_powr_right [analytic_intros]:
  assumes f analytic_on s
  shows (λz. w powr f z) analytic_on s
proof (cases w = 0)
  case False
  with assms show ?thesis
    unfolding analytic_on_def holomorphic_on_def field_differentiable_def
    by (metis (full_types) DERIV_chain' has_field_derivative_powr_right)
qed simp
```

3.2 Factorization of analytic function on compact region

```
definition not_zero_on (infixr ‹not'_zero'_on› 46)
  where f not_zero_on S ≡ ∃ z ∈ S. f z ≠ 0

lemma not_zero_on_obtain:
  assumes f not_zero_on S and S ⊆ T
  obtains t where f t ≠ 0 and t ∈ T
using assms unfolding not_zero_on_def by auto

lemma analytic_on_holomorphic_connected:
  assumes hf: f analytic_on S
    and con: connected A
    and ne: ξ ∈ A and AS: A ⊆ S
```

obtains T T' **where**
  f holomorphic_on T *f holomorphic_on* T'
  open T *open* T' $A \subseteq T$ $S \subseteq T'$ *connected* T
proof –
  obtain T'
  where *oT'*: *open* T' **and** *sT'*: $S \subseteq T'$
    and *holf'*: *f holomorphic_on* T'
    using *analytic_on_holomorphic hf* **by** *blast*
  define T **where** $T \equiv$ *connected_component_set* T' ξ
  have *TT'*: $T \subseteq T'$ **unfolding** *T_def* **by** (*rule connected_component_subset*)
  hence *holf*: *f holomorphic_on* T **using** *holf'* **by** *auto*
  have *opT*: *open* T **unfolding** *T_def* **using** *oT'* **by** (*rule open_connected_component*)
  have *conT*: *connected* T **unfolding** *T_def* **by** (*rule connected_connected_component*)
  have $A \subseteq T'$ **using** *AS sT'* **by** *blast*
  hence *AT*: $A \subseteq T$ **unfolding** *T_def* **using** *ne con* **by** (*intro connected_component_maximal*)
  show *?thesis* **using** *holf holf' opT oT' AT sT' conT that* **by** *blast*
qed

lemma *analytic_factor_zero*:
  assumes *hf*: *f analytic_on* S
    and *KS*: $K \subseteq S$ **and** *con*: *connected* K
    and *ξK*: $\xi \in K$ **and** *ξz*: $f\ \xi = 0$
    and *nz*: *f not_zero_on* K
  obtains g r n
    where $0 < n$ $0 < r$
        g analytic_on S *g not_zero_on* K
        $\bigwedge z.\ z \in S \Longrightarrow f\ z = (z - \xi)\hat{}n * g\ z$
        $\bigwedge z.\ z \in ball\ \xi\ r \Longrightarrow g\ z \neq 0$
proof –
  have *f analytic_on* S *connected* K
      $\xi \in K$ $K \subseteq S$ **using** *assms* **by** *auto*
  then obtain T T'
  where *holf*: *f holomorphic_on* T
    and *holf'*: *f holomorphic_on* T'
    and *opT*: *open* T **and** *oT'*: *open* T'
    and *KT*: $K \subseteq T$ **and** *ST'*: $S \subseteq T'$
    and *conT*: *connected* T
    by (*rule analytic_on_holomorphic_connected*)
  obtain η **where** *fη*: $f\ \eta \neq 0$ **and** *ηK*: $\eta \in K$
    using *nz* **by** (*rule not_zero_on_obtain*, *blast*)
  hence *ξT*: $\xi \in T$ **and** *ξT'*: $\xi \in T'$
    and *ηT*: $\eta \in T$ **using** *ξK ηK KT KS ST'* **by** *blast+*
  hence *nc*: $\neg$ *f constant_on* T **using** *fη ξz* **unfolding** *constant_on_def* **by** *fastforce*
  obtain g r n
  where *1*: $0 < n$ **and** *2*: $0 < r$
    and *bT*: $ball\ \xi\ r \subseteq T$
    and *hg*: *g holomorphic_on ball* ξ r
    and *fw*: $\bigwedge z.\ z \in ball\ \xi\ r \Longrightarrow f\ z = (z - \xi)\ \hat{}\ n * g\ z$
    and *gw*: $\bigwedge z.\ z \in ball\ \xi\ r \Longrightarrow g\ z \neq 0$
    by (*rule holomorphic_factor_zero_nonconstant*, (*rule holf opT conT ξT ξz nc*)+, *blast*)
  have *sT*: $S \subseteq T' - \{\xi\} \cup ball\ \xi\ r$ **using** *2 ST'* **by** *auto*
  have *hz*: $(\lambda z.\ f\ z\ /\ (z - \xi)\ \hat{}\ n)$ *holomorphic_on* $(T' - \{\xi\})$
    using *holf'* **by** ((*intro holomorphic_intros*)+, *auto*)
  obtain h
   where *3*: *h analytic_on* S

and *hf*: $\bigwedge z.\ z \in T' - \{\xi\} \Longrightarrow h\ z = f\ z\ /\ (z - \xi)\ \hat{}\ n$
and *hb*: $\bigwedge z.\ z \in ball\ \xi\ r \Longrightarrow h\ z = g\ z$
by (*rule holomorphic_glue_to_analytic*
[**where** $f = \lambda z.\ f\ z\ /\ (z - \xi)\ \hat{}\ n$ **and**
$g = g$ **and** $S = T' - \{\xi\}$ **and** $T = ball\ \xi\ r$ **and** $U = S$])
(*use oT′ 2 ST′ hg fw hz* **in** ‹*auto simp add*: *holomorphic_intros*›)
have $\xi \in ball\ \xi\ r$ **using** *2* **by** *auto*
hence $h\ \xi \neq 0$ **using** *hb gw 2* **by** *auto*
hence *4*: *h not_zero_on K* **unfolding** *not_zero_on_def* **using** *ξK* **by** *auto*
have *5*: $f\ z = (z - \xi)\hat{}n * h\ z$ **if** ∗: $z \in S$ **for** *z*
proof −
consider $z = \xi \mid z \in S - \{\xi\}$ **using** ∗ **by** *auto*
thus *?thesis* **proof** *cases*
assume ∗: $z = \xi$
show *?thesis* **using** *ξz 1* **by** (*subst* (*1 2*) ∗, *auto*)
next
assume ∗: $z \in S - \{\xi\}$
show *?thesis* **using** *hf ST′* ∗ **by** (*auto simp add*: *field_simps*)
qed
qed
have *6*: $\bigwedge w.\ w \in ball\ \xi\ r \Longrightarrow h\ w \neq 0$ **using** *hb gw* **by** *auto*
show *?thesis* **by** ((*standard*; *rule 1 2 3 4 5 6*), *blast+*)
qed

lemma *analytic_compact_finite_zeros*:
assumes *af*: *f analytic_on S*
and *KS*: $K \subseteq S$
and *con*: *connected K*
and *cm*: *compact K*
and *nz*: *f not_zero_on K*
shows *finite* $\{z \in K.\ f\ z = 0\}$
proof (*cases f constant_on K*)
assume ∗: *f constant_on K*
have $\bigwedge z.\ z \in K \Longrightarrow f\ z \neq 0$ **using** *nz* ∗ **unfolding** *not_zero_on_def constant_on_def* **by** *auto*
hence ∗∗: $\{z \in K.\ f\ z = 0\} = \{\}$ **by** *auto*
thus *?thesis* **by** (*subst* ∗∗, *auto*)
next
assume ∗: ¬ *f constant_on K*
obtain *ξ* **where** *ne*: $\xi \in K$ **using** *not_zero_on_obtain nz* **by** *blast*
obtain *T T′* **where** *opT*: *open T* **and** *conT*: *connected T*
and *ST*: $K \subseteq T$ **and** *holf*: *f holomorphic_on T*
and *f holomorphic_on T′*
by (*metis af KS con ne analytic_on_holomorphic_connected*)
have ¬ *f constant_on T* **using** *ST* ∗ **unfolding** *constant_on_def* **by** *blast*
thus *?thesis* **using** *holf opT conT cm ST* **by** (*intro holomorphic_compact_finite_zeros*)
qed

3.2.1 Auxiliary propositions for theorem *analytic_factorization*

definition *analytic_factor_p′* **where**
‹*analytic_factor_p′ f S K* ≡
$\exists\, g\ n.\ \exists\, \alpha :: nat \Rightarrow complex.$
g analytic_on S
$\wedge\ (\forall\, z \in K.\ g\ z \neq 0)$
$\wedge\ (\forall\, z \in S.\ f\ z = g\ z * (\prod k{<}n.\ z - \alpha\ k))$
$\wedge\ \alpha\ `\ \{..{<}n\} \subseteq K$›

```
definition analytic_factor_p where
 ‹analytic_factor_p F ≡
  ∀f S K. f analytic_on S
    ⟶ K ⊆ S
    ⟶ connected K
    ⟶ compact K
    ⟶ f not_zero_on K
    ⟶ {z ∈ K. f z = 0} = F
    ⟶ analytic_factor_p' f S K›

lemma analytic_factorization_E:
  shows analytic_factor_p {}
unfolding analytic_factor_p_def
proof (intro conjI allI impI)
  fix f S K
  assume af: f analytic_on S
     and KS: K ⊆ S
     and con: connected K
     and cm: compact K
     and nz: {z ∈ K. f z = 0} = {}
  show analytic_factor_p' f S K
  unfolding analytic_factor_p'_def
  proof (intro ballI conjI exI)
    show f analytic_on S ⋀z. z ∈ K ⟹ f z ≠ 0
         ⋀z. z ∈ S ⟹ f z = f z * (∏k<(0 :: nat). z − (λ_. 0) k)
      by (rule af, use nz in auto)
    show (λk :: nat. 0) ‘ {..<0} ⊆ K by auto
  qed
qed

lemma analytic_factorization_I:
  assumes ind: analytic_factor_p F
    and ξni: ξ ∉ F
  shows analytic_factor_p (insert ξ F)
unfolding analytic_factor_p_def
proof (intro allI impI)
  fix f S K
  assume af: f analytic_on S
    and KS: K ⊆ S
    and con: connected K
    and nz: f not_zero_on K
    and cm: compact K
    and zr: {z ∈ K. f z = 0} = insert ξ F
  show analytic_factor_p' f S K
  proof −
    have f analytic_on S K ⊆ S connected K
         ξ ∈ K f ξ = 0 f not_zero_on K
    using af KS con zr nz by auto
    then obtain h r k
    where 0 < k and 0 < r and ah: h analytic_on S
      and nh: h not_zero_on K
      and f_z: ⋀z. z ∈ S ⟹ f z = (z − ξ) ^ k * h z
      and ball: ⋀z. z ∈ ball ξ r ⟹ h z ≠ 0
    by (rule analytic_factor_zero) blast
```

```
      hence hξ: h ξ ≠ 0 using ball by auto
      hence ⋀z. z ∈ K ⟹ h z = 0 ⟷ f z = 0 ∧ z ≠ ξ by (subst f_z) (use KS in auto)
      hence {z ∈ K. h z = 0} = {z ∈ K. f z = 0} − {ξ} by auto
      also have ... = F by (subst zr, intro Diff_insert_absorb ξni)
      finally have {z ∈ K. h z = 0} = F .
      hence analytic_factor_p' h S K
        using ind ah KS con cm nh
        unfolding analytic_factor_p_def by auto
      then obtain g n and α :: nat ⇒ complex
      where ag: g analytic_on S and
        ng: ⋀z. z ∈ K ⟹ g z ≠ 0 and
        h_z: ⋀z. z ∈ S ⟹ h z = g z * (∏k<n. z − α k) and
        Imα: α ` {..<n} ⊆ K
      unfolding analytic_factor_p'_def by fastforce
      define β where β j ≡ if j < n then α j else ξ for j
      show ?thesis
      unfolding analytic_factor_p'_def
      proof (intro ballI conjI exI)
        show g analytic_on S ⋀z. z ∈ K ⟹ g z ≠ 0
          by (rule ag, rule ng)
      next
        fix z assume *: z ∈ S
        show f z = g z * (∏j<n+k. z − β j)
        proof −
          have (∏j<n. z − β j) = (∏j<n. z − α j)
              (∏j=n..<n+k. z − β j) = (z − ξ) ^ k
          unfolding β_def by auto
          moreover have (∏j<n+k. z − β j) = (∏j<n. z − β j) * (∏j=n..<n+k. z − β j)
          by (metis Metric_Arith.nnf_simps(8) atLeast0LessThan
             not_add_less1 prod.atLeastLessThan_concat zero_order(1))
          ultimately have (∏j<n+k. z − β j) = (z − ξ) ^ k * (∏j<n. z − α j) by auto
          moreover have f z = g z * ((z − ξ) ^ k * (∏j<n. z − α j))
          by (subst f_z; (subst h_z)?, use * in auto)
          ultimately show ?thesis by auto
        qed
      next
        show β ` {..<n + k} ⊆ K unfolding β_def using Imα ‹ξ ∈ K› by auto
      qed
    qed
  qed
```

A nontrivial analytic function on connected compact region can be factorized as a everywhere-non-zero function and linear terms $z - s_0$ for all zeros s_0. Note that the connected assumption of K may be removed, but we remain it just for simplicity of proof.

```
theorem analytic_factorization:
  assumes af: f analytic_on S
    and KS: K ⊆ S
    and con: connected K
    and compact K
    and f not_zero_on K
  obtains g n and α :: nat ⇒ complex where
    g analytic_on S
    ⋀z. z ∈ K ⟹ g z ≠ 0
    ⋀z. z ∈ S ⟹ f z = g z * (∏k<n. (z − α k))
    α ` {..<n} ⊆ K
proof −
```

```
  have ‹finite {z ∈ K. f z = 0}› using assms by (rule analytic_compact_finite_zeros)
  moreover have ‹finite F ⟹ analytic_factor_p F› for F
    by (induct rule: finite_induct; rule analytic_factorization_E analytic_factorization_I)
  ultimately have analytic_factor_p {z ∈ K. f z = 0} by auto
  hence analytic_factor_p' f S K unfolding analytic_factor_p_def using assms by auto
  thus ?thesis unfolding analytic_factor_p'_def using assms that by metis
qed
```

3.3 Schwarz theorem in complex analysis

```
lemma Schwarz_Lemma1:
  fixes f :: complex ⇒ complex
    and ξ :: complex
  assumes f holomorphic_on ball 0 1
    and f 0 = 0
    and ⋀z. ‖z‖ < 1 ⟹ ‖f z‖ ≤ 1
    and ‖ξ‖ < 1
  shows ‖f ξ‖ ≤ ‖ξ‖
proof (cases f constant_on ball 0 1)
  assume f constant_on ball 0 1
  thus ?thesis unfolding constant_on_def
    using assms by auto
next
  assume nc: ¬f constant_on ball 0 1
  have ⋀z. ‖z‖ < 1 ⟹ ‖f z‖ < 1
  proof -
    fix z :: complex assume *: ‖z‖ < 1
    have ‖f z‖ ≠ 1
    proof
      assume ‖f z‖ = 1
      hence ⋀w. w ∈ ball 0 1 ⟹ ‖f w‖ ≤ ‖f z‖
        using assms(3) by auto
      hence f constant_on ball 0 1
        by (intro maximum_modulus_principle [where U = ball 0 1 and ξ = z])
          (use * assms(1) in auto)
      thus False using nc by blast
    qed
    with assms(3) [OF *] show ‖f z‖ < 1 by auto
  qed
  thus ‖f ξ‖ ≤ ‖ξ‖ by (intro Schwarz_Lemma(1), use assms in auto)
qed

theorem Schwarz_Lemma2:
  fixes f :: complex ⇒ complex
    and ξ :: complex
  assumes holf: f holomorphic_on ball 0 R
    and hR: 0 < R and nz: f 0 = 0
    and bn: ⋀z. ‖z‖ < R ⟹ ‖f z‖ ≤ 1
    and ξR: ‖ξ‖ < R
  shows ‖f ξ‖ ≤ ‖ξ‖ / R
proof -
  define φ where φ z ≡ f (R * z) for z :: complex
  have ‖ξ / R‖ < 1 using ξR hR by (subst nonzero_norm_divide, auto)
  moreover have f holomorphic_on (*) (R :: complex) ‘ ball 0 1
    by (rule holomorphic_on_subset, rule holf)
      (use hR in ‹auto simp: norm_mult›)
```

```
  hence (f ∘ (λz. R ∗ z)) holomorphic_on ball 0 1
    by (auto intro: holomorphic_on_compose)
  moreover have φ 0 = 0 unfolding φ_def using nz by auto
  moreover have ⋀z. ‖z‖ < 1 ⟹ ‖φ z‖ ≤ 1
  proof −
    fix z :: complex assume ∗: ‖z‖ < 1
    have ‖R∗z‖ < R using hR ∗ by (fold scaleR_conv_of_real) auto
    thus ‖φ z‖ ≤ 1 unfolding φ_def using bn by auto
  qed
  ultimately have ‖φ (ξ / R)‖ ≤ ‖ξ / R‖
    unfolding comp_def by (fold φ_def, intro Schwarz_Lemma1)
  thus ?thesis unfolding φ_def using hR by (subst (asm) nonzero_norm_divide, auto)
qed
```

3.4 Borel-Carathedory theorem

Borel-Carathedory theorem, from book *Theorem 5.5*, *The Theory of Functions*, *E. C. Titchmarsh*

```
lemma Borel_Caratheodory1:
  assumes hr: 0 < R 0 < r r < R
    and f0: f 0 = 0
    and hf: ⋀z. ‖z‖ < R ⟹ Re (f z) ≤ A
    and holf: f holomorphic_on (ball 0 R)
    and zr: ‖z‖ ≤ r
  shows ‖f z‖ ≤ 2∗r/(R−r) ∗ A
proof −
  have A_ge_0: A ≥ 0
  using f0 hf by (metis hr(1) norm_zero zero_complex.simps(1))
  then consider A = 0 | A > 0 by linarith
  thus ‖f z‖ ≤ 2 ∗ r/(R−r) ∗ A
  proof (cases)
    assume ∗: A = 0
    have 1: ⋀w. w ∈ ball 0 R ⟹ ‖exp (f w)‖ ≤ ‖exp (f 0)‖ using hf f0 ∗ by auto
    have 2: exp ∘ f constant_on (ball 0 R)
      by (rule maximum_modulus_principle [where f = exp ∘ f and U = ball 0 R])
          (use 1 hr(1) in ‹auto intro: holomorphic_on_compose holf holomorphic_on_exp›)
    have f constant_on (ball 0 R)
    proof (rule classical)
      assume ∗: ¬ f constant_on ball 0 R
      have open (f ‘ (ball 0 R))
        by (rule open_mapping_thm [where S = ball 0 R], use holf ∗ in auto)
      then obtain e where e > 0 and cball 0 e ⊆ f ‘ (ball 0 R)
        by (metis hr(1) f0 centre_in_ball imageI open_contains_cball)
      then obtain w
        where hw: w ∈ ball 0 R f w = e
        by (metis abs_of_nonneg imageE less_eq_real_def mem_cball_0 norm_of_real subset_eq)
      have exp e = exp (f w)
        using hw(2) by (fold exp_of_real) auto
      also have ... = exp (f 0)
        using hw(1) 2 hr(1) unfolding constant_on_def comp_def by auto
      also have ... = exp (0 :: real) by (subst f0) auto
      finally have e = 0 by auto
      with ‹e > 0› show ?thesis by blast
    qed
    hence f z = 0 using f0 hr zr unfolding constant_on_def by auto
    hence ‖f z‖ = 0 by auto
```

```
  also have ... ≤ 2 * r/(R−r) * A using hr ‹A ≥ 0› by auto
  finally show ?thesis .
next
  assume A_gt_0: A > 0
  define φ where φ z ≡ (f z)/(2*A − f z) for z :: complex
  have φ_bound: ‖φ z‖ ≤ 1 if *: ‖z‖ < R for z
  proof −
    define u v where u ≡ Re (f z) and v ≡ Im (f z)
    hence u ≤ A unfolding u_def using hf * by blast
    hence u^2 ≤ (2*A−u)^2 using A_ge_0 by (simp add: sqrt_ge_absD)
    hence u^2 + v^2 ≤ (2*A−u)^2 + (−v)^2 by auto
    moreover have 2*A − f z = Complex (2*A−u) (−v) by (simp add: complex_eq_iff u_def v_def)
    hence ‖f z‖^2 = u^2 + v^2
          ‖2*A − f z‖^2 = (2*A−u)^2 + (−v)^2
    unfolding u_def v_def using cmod_power2 complex.sel by presburger+
    ultimately have ‖f z‖^2 ≤ ‖2*A − f z‖^2 by auto
    hence ‖f z‖ ≤ ‖2*A − f z‖ by auto
    thus ?thesis unfolding φ_def by (subst norm_divide) (simp add: divide_le_eq_1)
  qed
  moreover have nz: ⋀z :: complex. z ∈ ball 0 R ⟹ 2*A − f z ≠ 0
  proof
    fix z :: complex
    assume *: z ∈ ball 0 R
      and eq: 2*A − f z = 0
    hence Re (f z) ≤ A using hf by auto
    moreover have Re (f z) = 2*A
      by (metis eq Re_complex_of_real right_minus_eq)
    ultimately show False using A_gt_0 by auto
  qed
  ultimately have φ holomorphic_on ball 0 R
    unfolding φ_def comp_def by (intro holomorphic_intros holf)
  moreover have φ 0 = 0 unfolding φ_def using f0 by auto
  ultimately have *: ‖φ z‖ ≤ ‖z‖ / R
    using hr(1) φ_bound zr hr Schwarz_Lemma2 by auto
  also have ... < 1 using zr hr by auto
  finally have hφ: ‖φ z‖ ≤ r / R ‖φ z‖ < 1 1 + φ z ≠ 0
  proof (safe)
    show ‖φ z‖ ≤ r / R using * zr hr(1)
      by (metis divide_le_cancel dual_order.trans nle_le)
  next
    assume 1 + φ z = 0
    hence φ z = −1 using add_eq_0_iff by blast
    thus ‖φ z‖ < 1 ⟹ False by auto
  qed
  have 2*A − f z ≠ 0 using nz hr(3) zr by auto
  hence f z = 2*A*φ z / (1 + φ z)
    using hφ(3) unfolding φ_def by (auto simp add: field_simps)
  hence ‖f z‖ = 2*A*‖φ z‖ / ‖1 + φ z‖
    by (auto simp add: norm_divide norm_mult A_ge_0)
  also have ... ≤ 2*A*(‖φ z‖ / (1 − ‖φ z‖))
  proof −
    have ‖1 + φ z‖ ≥ 1 − ‖φ z‖
      by (metis norm_diff_ineq norm_one)
    thus ?thesis
      by (simp, rule divide_left_mono, use A_ge_0 in auto)
```

```
        (intro mult_pos_pos, use hφ(2) in auto)
    qed
    also have ... ≤ 2∗A∗((r/R) / (1 − r/R))
    proof −
      have ∗: a / (1 − a) ≤ b / (1 − b)
        if a < 1 b < 1 a ≤ b for a b :: real
      using that by (auto simp add: field_simps)
      have ‖φ z‖ / (1 − ‖φ z‖) ≤ (r/R) / (1 − r/R)
        by (rule ∗; (intro hφ)?) (use hr in auto)
      thus ?thesis by (rule mult_left_mono, use A_ge_0 in auto)
    qed
    also have ... = 2∗r/(R−r) ∗ A using hr(1) by (auto simp add: field_simps)
    finally show ?thesis .
  qed
qed

lemma Borel_Caratheodory2:
  assumes hr: 0 < R 0 < r r < R
    and hf: ⋀z. ‖z‖ < R ⟹ Re (f z − f 0) ≤ A
    and holf: f holomorphic_on (ball 0 R)
    and zr: ‖z‖ ≤ r
  shows ‖f z − f 0‖ ≤ 2∗r/(R−r) ∗ A
proof −
  define g where g z ≡ f z − f 0 for z
  show ?thesis
    by (fold g_def, rule Borel_Caratheodory1)
       (unfold g_def, insert assms, auto intro: holomorphic_intros)
qed

theorem Borel_Caratheodory3:
  assumes hr: 0 < R 0 < r r < R
    and hf: ⋀w. w ∈ ball s R ⟹ Re (f w − f s) ≤ A
    and holf: f holomorphic_on (ball s R)
    and zr: z ∈ ball s r
  shows ‖f z − f s‖ ≤ 2∗r/(R−r) ∗ A
proof −
  define g where g w ≡ f (s + w) for w
  have ⋀w. ‖w‖ < R ⟹ Re (f (s + w) − f s) ≤ A
    by (intro hf) (auto simp add: dist_complex_def)
  hence ‖g (z − s) − g 0‖ ≤ 2∗r/(R−r) ∗ A
    by (intro Borel_Caratheodory2, unfold g_def, insert assms)
       (auto intro: holomorphic_intros simp add: dist_complex_def norm_minus_commute)
  thus ?thesis unfolding g_def by auto
qed
```

3.5 Lemma 3.9

These lemmas is referred to the following material: Theorem 3.9, *The Theory of the Riemann Zeta−Function, E. C. Titchmarsh, D. R. Heath−Brown.*

```
lemma lemma_3_9_beta1:
  fixes f M r s₀
  assumes zl: 0 < r 0 ≤ M
    and hf: f holomorphic_on ball 0 r
    and ne: ⋀z. z ∈ ball 0 r ⟹ f z ≠ 0
    and bn: ⋀z. z ∈ ball 0 r ⟹ ‖f z / f 0‖ ≤ exp M
```

```
  shows ‖logderiv f 0‖ ≤ 4 * M / r
    and ∀s∈cball 0 (r / 4). ‖logderiv f s‖ ≤ 8 * M / r
proof (goal_cases)
  obtain g
  where holg: g holomorphic_on ball 0 r
    and exp_g: ⋀x. x ∈ ball 0 r ⟹ exp (g x) = f x
    by (rule holomorphic_logarithm_exists [of ball 0 r f 0])
       (use zl(1) ne hf in auto)
  have f0: exp (g 0) = f 0 using exp_g zl(1) by auto
  have Re (g z − g 0) ≤ M if *: ‖z‖ < r for z
  proof −
    have exp (Re (g z − g 0)) = ‖exp (g z − g 0)‖
      by (rule norm_exp_eq_Re [symmetric])
    also have ... = ‖f z / f 0‖
      by (subst exp_diff, subst f0, subst exp_g)
         (use * in auto)
    also have ... ≤ exp M by (rule bn) (use * in auto)
    finally show ?thesis by auto
  qed
  hence ‖g z − g 0‖ ≤ 2 * (r / 2) / (r − r / 2) * M
    if *: ‖z‖ ≤ r / 2 for z
    by (intro Borel_Caratheodory2 [where f = g])
       (use zl(1) holg * in auto)
  also have ... = 2 * M using zl(1) by auto
  finally have hg: ⋀z. ‖z‖ ≤ r / 2 ⟹ ‖g z − g 0‖ ≤ 2 * M .
  have result: ‖logderiv f s‖ ≤ 2 * M / r′
    when cball s r′ ⊆ cball 0 (r / 2) 0 < r′ ‖s‖ < r / 2 for s r′
  proof −
    have contain: ⋀z. ‖s − z‖ ≤ r′ ⟹ ‖z‖ ≤ r / 2
      using that by (auto simp add: cball_def subset_eq dist_complex_def)
    have contain′: ‖z‖ < r when ‖s − z‖ ≤ r′ for z
      using zl(1) contain [of z] that by auto
    have s_in_ball: s ∈ ball 0 r using that(3) zl(1) by auto
    have deriv f s = deriv (λx. exp (g x)) s
      by (rule deriv_cong_ev, subst eventually_nhds)
         (rule exI [where x = ball 0 (r / 2)], use exp_g zl(1) that(3) in auto)
    also have ... = exp (g s) * deriv g s
      by (intro DERIV_fun_exp [THEN DERIV_imp_deriv] field_differentiable_derivI)
         (meson holg open_ball s_in_ball holomorphic_on_imp_differentiable_at)
    finally have df: logderiv f s = deriv g s
    proof −
      assume deriv f s = exp (g s) * deriv g s
      moreover have f s ≠ 0 by (intro ne s_in_ball)
      ultimately show ?thesis
        unfolding logderiv_def using exp_g [OF s_in_ball] by auto
    qed
    have ⋀z. ‖s − z‖ = r′ ⟹ ‖g z − g 0‖ ≤ 2 * M
      using contain by (intro hg) auto
    moreover have (λz. g z − g 0) holomorphic_on cball s r′
      by (rule holomorphic_on_subset [where s=ball 0 r], insert holg)
         (auto intro: holomorphic_intros contain′ simp add: dist_complex_def)
    moreover hence continuous_on (cball s r′) (λz. g z − g 0)
      by (rule holomorphic_on_imp_continuous_on)
    ultimately have ‖(deriv ^^ 1) (λz. g z − g 0) s‖ ≤ fact 1 * (2 * M) / r′ ^ 1
      using that(2) by (intro Cauchy_inequality) auto
```

```
    also have ... = 2 * M / r' by auto
    also have deriv g s = deriv (λz. g z − g 0) s
      by (subst deriv_diff, auto)
         (rule holomorphic_on_imp_differentiable_at, use holg s_in_ball in auto)
    hence ‖deriv g s‖ = ‖(deriv ^^ 1) (λz. g z − g 0) s‖
      by (auto simp add: derivative_intros)
    ultimately show ?thesis by (subst df) auto
  qed
  case 1 show ?case using result [of 0 r / 2] zl(1) by auto
  case 2 show ?case proof safe
    fix s :: complex assume hs: s ∈ cball 0 (r / 4)
    hence z ∈ cball s (r / 4) ⟹ ‖z‖ ≤ r / 2 for z
      using norm_triangle_sub [of z s]
      by (auto simp add: dist_complex_def norm_minus_commute)
    hence ‖logderiv f s‖ ≤ 2 * M / (r / 4)
      by (intro result) (use zl(1) hs in auto)
    also have ... = 8 * M / r by auto
    finally show ‖logderiv f s‖ ≤ 8 * M / r .
  qed
qed

lemma lemma_3_9_beta1':
  fixes f M r s₀
  assumes zl: 0 < r 0 ≤ M
    and hf: f holomorphic_on ball s r
    and ne: ⋀z. z ∈ ball s r ⟹ f z ≠ 0
    and bn: ⋀z. z ∈ ball s r ⟹ ‖f z / f s‖ ≤ exp M
    and hs: z ∈ cball s (r / 4)
  shows ‖logderiv f z‖ ≤ 8 * M / r
proof −
  define g where g z ≡ f (s + z) for z
  have ∀ z ∈ cball 0 (r / 4). ‖logderiv g z‖ ≤ 8 * M / r
    by (intro lemma_3_9_beta1 assms, unfold g_def)
       (auto simp add: dist_complex_def intro!: assms holomorphic_on_shift)
  note bspec [OF this, of z − s]
  moreover have f field_differentiable at z
    by (rule holomorphic_on_imp_differentiable_at [where ?s = ball s r])
       (insert hs zl(1), auto intro: hf simp add: dist_complex_def)
  ultimately show ?thesis unfolding g_def using hs
    by (auto simp add: dist_complex_def logderiv_shift)
qed

lemma lemma_3_9_beta2:
  fixes f M r
  assumes zl: 0 < r 0 ≤ M
    and af: f analytic_on cball 0 r
    and f0: f 0 ≠ 0
    and rz: ⋀z. z ∈ cball 0 r ⟹ Re z > 0 ⟹ f z ≠ 0
    and bn: ⋀z. z ∈ cball 0 r ⟹ ‖f z / f 0‖ ≤ exp M
    and hg: Γ ⊆ {z ∈ cball 0 (r / 2). f z = 0}
  shows − Re (logderiv f 0) ≤ 8 * M / r + Re (∑ z∈Γ. 1 / z)
proof −
  have nz': f not_zero_on cball 0 (r / 2)
    unfolding not_zero_on_def using f0 zl(1) by auto
  hence fin_zeros: finite {z ∈ cball 0 (r / 2). f z = 0}
```

```
  by (intro analytic_compact_finite_zeros [where S = cball 0 r])
     (use af zl in auto)
obtain g n and α :: nat ⇒ complex
where ag: g analytic_on cball 0 r
  and ng: ⋀z. z ∈ cball 0 (r / 2) ⟹ g z ≠ 0
  and fac: ⋀z. z ∈ cball 0 r ⟹ f z = g z * (∏k<n. (z − α k))
  and Imα: α ‘ {..<n} ⊆ cball 0 (r / 2)
  by (rule analytic_factorization [
    where K = cball 0 (r / 2)
     and S = cball 0 r and f = f])
     (use zl(1) af nz' in auto)
have g0: ‖g 0‖ ≠ 0 using ng zl(1) by auto
hence g holomorphic_on cball 0 r
      (λz. g z / g 0) holomorphic_on cball 0 r
  using ag by (auto simp add: analytic_intros intro: analytic_imp_holomorphic)
hence holg:
    g holomorphic_on ball 0 r
    (λz. g z / g 0) holomorphic_on ball 0 r
    continuous_on (cball 0 r) (λz. g z / g 0)
  by (auto intro!: holomorphic_on_imp_continuous_on
                   holomorphic_on_subset [where t = ball 0 r])
have nz_α: ⋀k. k < n ⟹ α k ≠ 0 using zl(1) f0 fac by auto
have ‖g z / g 0‖ ≤ exp M if *: z ∈ sphere 0 r for z
proof −
  let ?p = ‖(∏k<n. (z − α k)) / (∏k<n. (0 − α k))‖
  have 1: ‖f z / f 0‖ ≤ exp M using bn * by auto
  have 2: ‖f z / f 0‖ = ‖g z / g 0‖ * ?p
    by (subst norm_mult [symmetric], subst (1 2) fac)
       (use that zl(1) in auto)
  have ?p = (∏k<n. (‖z − α k‖ / ‖0 − α k‖))
    by (auto simp add: prod_norm [symmetric] norm_divide prod_dividef)
  also have ‖z − α k‖ ≥ ‖0 − α k‖ if k < n for k
  proof (rule ccontr)
    assume **: ¬ ‖z − α k‖ ≥ ‖0 − α k‖
    have r = ‖z‖ using * by auto
    also have ... ≤ ‖0 − α k‖ + ‖z − α k‖ by (simp add: norm_triangle_sub)
    also have ... < 2 * ‖α k‖ using ** by auto
    also have α k ∈ cball 0 (r / 2) using Imα that by blast
    hence 2 * ‖α k‖ ≤ r by auto
    finally show False by linarith
  qed
  hence ⋀k. k < n ⟹ ‖z − α k‖ / ‖0 − α k‖ ≥ 1
    using nz_α by (subst le_divide_eq_1_pos) auto
  hence (∏k<n. (‖z − α k‖ / ‖0 − α k‖)) ≥ 1 by (rule prod_ge_1) simp
  finally have 3: ?p ≥ 1 .
  have rule1: b = a * c ⟹ a ≥ 0 ⟹ c ≥ 1 ⟹ a ≤ b for a b c :: real
    by (metis landau_omega.R_mult_left_mono more_arith_simps(6))
  have ‖g z / g 0‖ ≤ ‖f z / f 0‖
    by (rule rule1) (rule 2 3 norm_ge_zero)+
  thus ?thesis using 1 by linarith
qed
hence ⋀z. z ∈ cball 0 r ⟹ ‖g z / g 0‖ ≤ exp M
  using holg
  by (auto intro: maximum_modulus_frontier
     [where f = λz. g z / g 0 and S = cball 0 r])
```

```
hence bn′: ⋀z. z ∈ cball 0 (r / 2) ⟹ ‖g z / g 0‖ ≤ exp M using zl(1) by auto
have ag′: g analytic_on cball 0 (r / 2)
  by (rule analytic_on_subset [where S = cball 0 r])
     (use ag zl(1) in auto)
have ‖logderiv g 0‖ ≤ 4 ∗ M / (r / 2)
  by (rule lemma_3_9_beta1(1) [where f = g])
     (use zl ng bn′ holg in auto)
also have ... = 8 ∗ M / r by auto
finally have bn_g: ‖logderiv g 0‖ ≤ 8 ∗ M / r unfolding logderiv_def by auto
let ?P = λw. ∏k<n. (w − α k)
let ?S′ = ∑k<n. logderiv (λz. z − α k) 0
let ?S = ∑k<n. − (1 / α k)
have g field_differentiable at 0 using holg zl(1)
  by (auto intro!: holomorphic_on_imp_differentiable_at)
hence ld_g: g log_differentiable 0 unfolding log_differentiable_def using g0 by auto
have logderiv ?P 0 = ?S′ and ld_P: ?P log_differentiable 0
  by (auto intro!: logderiv_linear nz_α logderiv_prod)
note this(1)
also have ?S′ = ?S
  by (rule sum.cong)
     (use nz_α in auto cong: logderiv_linear(1))
finally have cd_P: logderiv ?P 0 = ?S .
have logderiv f 0 = logderiv (λz. g z ∗ ?P z) 0
  by (rule logderiv_cong_ev, subst eventually_nhds)
     (intro exI [where x = ball 0 r], use fac zl(1) in auto)
also have ... = logderiv g 0 + logderiv ?P 0
  by (subst logderiv_mult) (use ld_g ld_P in auto)
also have ... = logderiv g 0 + ?S using cd_P by auto
finally have Re (logderiv f 0) = Re (logderiv g 0) + Re ?S by simp
moreover have − Re (∑z∈Γ. 1 / z) ≤ Re ?S
proof −
  have − Re (∑z∈Γ. 1 / z) = (∑z∈Γ. Re (− (1 / z))) by (auto simp add: sum_negf)
  also have ... ≤ (∑k<n. Re (− (1 / α k)))
  proof (rule sum_le_included)
    show ∀z∈Γ. ∃k∈{..<n}. α k = z ∧ Re (− (1 / z)) ≤ Re (− (1 / α k))
         (is Ball _ ?P)
    proof
      fix z assume hz: z ∈ Γ
      have ∃k∈{..<n}. α k = z
      proof (rule ccontr)
        assume ne_α: ¬ (∃k∈{..<n}. α k = z)
        have z_in: z ∈ cball 0 (r / 2) z ∈ cball 0 r using hg hz zl(1) by auto
        hence g z ≠ 0 using ng by auto
        moreover have (∏k<n. (z − α k)) ≠ 0 using ne_α hz by auto
        ultimately have f z ≠ 0 using fac z_in by auto
        moreover have f z = 0 using hz hg by auto
        ultimately show False by auto
      qed
      thus ?P z by auto
    qed
    show ∀k∈{..<n}. 0 ≤ Re (− (1 / α k)) (is Ball _ ?P)
    proof
      fix k assume ∗: k∈{..<n}
      have 1: α k ∈ cball 0 r using Imα zl(1) ∗ by auto
      hence (∏j<n. (α k − α j)) = 0
```

```
        by (subst prod_zero_iff) (use ∗ in auto)
      with 1 have f (α k) = 0 by (subst fac) auto
      hence Re (α k) ≤ 0 using 1 rz f0 by fastforce
      hence Re (1 ∗ cnj (α k)) ≤ 0 by auto
      thus ?P k using Re_complex_div_le_0 by auto
    qed
    show finite {..<n} by auto
    have Γ ⊆ {z ∈ cball 0 (r / 2). f z = 0} using hg by auto
    thus finite Γ using fin_zeros by (rule finite_subset)
  qed
  also have ... = Re ?S by auto
  finally show ?thesis .
 qed
 ultimately have − Re (logderiv f 0) − Re (∑ z∈Γ. 1 / z) ≤ Re (− logderiv g 0) by auto
 also have ... ≤ ‖− logderiv g 0‖ by (rule complex_Re_le_cmod)
 also have ... ≤ 8 ∗ M / r by simp (rule bn_g)
 finally show ?thesis by auto
qed

theorem lemma_3_9_beta3:
 fixes f M r and s :: complex
 assumes zl: 0 < r 0 ≤ M
  and af: f analytic_on cball s r
  and f0: f s ≠ 0
  and rz: ⋀z. z ∈ cball s r ⟹ Re z > Re s ⟹ f z ≠ 0
  and bn: ⋀z. z ∈ cball s r ⟹ ‖f z / f s‖ ≤ exp M
  and hg: Γ ⊆ {z ∈ cball s (r / 2). f z = 0}
 shows − Re (logderiv f s) ≤ 8 ∗ M / r + Re (∑ z∈Γ. 1 / (z − s))
proof −
 define g where g ≡ f ∘ (λz. s + z)
 define Δ where Δ ≡ (λz. z − s) ‘ Γ
 hence 1: g analytic_on cball 0 r
  unfolding g_def using af
  by (intro analytic_on_compose) (auto simp add: analytic_intros)
 moreover have g 0 ≠ 0 unfolding g_def using f0 by auto
 moreover have (Re z > 0 ⟶ g z ≠ 0) ∧ ‖g z / g 0‖ ≤ exp M
  if hz: z ∈ cball 0 r for z
 proof (intro impI conjI)
  assume hz': 0 < Re z
  thus g z ≠ 0 unfolding g_def comp_def
   using hz by (intro rz) (auto simp add: dist_complex_def)
 next
  show ‖g z / g 0‖ ≤ exp M
   unfolding g_def comp_def using hz
   by (auto simp add: dist_complex_def intro!: bn)
 qed
 moreover have Δ ⊆ {z ∈ cball 0 (r / 2). g z = 0}
 proof safe
  fix z assume z ∈ Δ
  hence s + z ∈ Γ unfolding Δ_def by auto
  thus g z = 0 z ∈ cball 0 (r / 2)
   unfolding g_def comp_def using hg by (auto simp add: dist_complex_def)
 qed
 ultimately have − Re (logderiv g 0) ≤ 8 ∗ M / r +  Re (∑ z∈Δ. 1 / z)
  by (intro lemma_3_9_beta2) (use zl in auto)
```

```
  also have ... = 8 * M / r +  Re (∑ z∈Γ. 1 / (z − s))
    unfolding Δ_def by (subst sum.reindex) (unfold inj_on_def, auto)
  finally show ?thesis
    unfolding g_def comp_def using zl(1)
    by (subst (asm) logderiv_shift)
       (auto intro: analytic_on_imp_differentiable_at [OF af])
qed

unbundle no pnt_syntax

end
theory Zeta_Zerofree
imports
  PNT_Complex_Analysis_Lemmas
begin
unbundle pnt_syntax
```

4 Zero-free region of zeta function

```
lemma cos_inequality_1:
  fixes x :: real
  shows 3 + 4 * cos x + cos (2 * x) ≥ 0
proof −
  have cos (2 * x) = (cos x)^2 − (sin x)^2
    by (rule cos_double)
  also have ... = (cos x)^2 − (1 − (cos x)^2)
    unfolding sin_squared_eq ..
  also have ... = 2 * (cos x)^2 − 1 by auto
  finally have 1: cos (2 * x) = 2 * (cos x)^2 − 1 .
  have 0 ≤ 2 * (1 + cos x)^2 by auto
  also have ... = 3 + 4 * cos x + (2 * (cos x)^2 − 1)
    by (simp add: field_simps power2_eq_square)
  finally show ?thesis unfolding 1.
qed

lemma multiplicative_fds_zeta:
  completely_multiplicative_function (fds_nth fds_zeta_complex)
  by standard auto

lemma fds_mangoldt_eq:
  fds mangoldt_complex = −(fds_deriv fds_zeta / fds_zeta)
proof −
  have fds_nth fds_zeta_complex 1 ≠ 0 by auto
  hence fds_nth (fds_deriv fds_zeta_complex / fds_zeta) n = −fds_nth fds_zeta n * mangoldt n for n
    using multiplicative_fds_zeta
    by (intro fds_nth_logderiv_completely_multiplicative)
  thus ?thesis by (intro fds_eqI, auto)
qed

lemma abs_conv_abscissa_log_deriv:
  abs_conv_abscissa (fds_deriv fds_zeta_complex / fds_zeta) ≤ 1
  by (rule abs_conv_abscissa_completely_multiplicative_log_deriv
      [OF multiplicative_fds_zeta, unfolded abs_conv_abscissa_zeta], auto)

lemma abs_conv_abscissa_mangoldt:
```

```
  abs_conv_abscissa (fds mangoldt_complex) ≤ 1
  using abs_conv_abscissa_log_deriv
  by (subst fds_mangoldt_eq, subst abs_conv_abscissa_minus)

lemma
  assumes s: Re s > 1
  shows eval_fds_mangoldt: eval_fds (fds mangoldt) s = − deriv zeta s / zeta s
    and abs_conv_mangoldt: fds_abs_converges (fds mangoldt) s
proof −
  from abs_conv_abscissa_log_deriv
  have 1: abs_conv_abscissa (fds_deriv fds_zeta_complex / fds_zeta) < ereal (s · 1)
    using s by (intro le_ereal_less, auto simp: one_ereal_def)
  have 2: abs_conv_abscissa fds_zeta_complex < ereal (s · 1)
    using s by (subst abs_conv_abscissa_zeta, auto)
  hence 3: fds_abs_converges (fds_deriv fds_zeta_complex / fds_zeta) s
    by (intro fds_abs_converges) (rule 1)
  have eval_fds (fds mangoldt) s = eval_fds (−(fds_deriv fds_zeta_complex / fds_zeta)) s
    using fds_mangoldt_eq by auto
  also have ... = −eval_fds (fds_deriv fds_zeta_complex / fds_zeta) s
    by (intro eval_fds_uminus fds_abs_converges_imp_converges 3)
  also have ... = −(eval_fds (fds_deriv fds_zeta_complex) s / eval_fds fds_zeta s)
    using s by (subst eval_fds_log_deriv; ((intro 1 2)?, (auto intro!: eval_fds_zeta_nonzero)?))
  also have ... = − deriv zeta s / zeta s
    using s by (subst eval_fds_zeta, blast, subst eval_fds_deriv_zeta, auto)
  finally show eval_fds (fds mangoldt) s = − deriv zeta s / zeta s .
  show fds_abs_converges (fds mangoldt) s
    by (subst fds_mangoldt_eq) (intro fds_abs_converges_uminus 3)
qed

lemma sums_mangoldt:
  fixes s :: complex
  assumes s: Re s > 1
  shows ((λn. mangoldt n / n nat_powr s) has_sum − deriv zeta s / zeta s) {1..}
proof −
  let ?f = (λn. mangoldt n / n nat_powr s)
  have 1: fds_abs_converges (fds mangoldt) s
    by (intro abs_conv_mangoldt s)
  hence 2: fds_converges (fds mangoldt) s
    by (rule fds_abs_converges_imp_converges)
  hence summable (λn. ‖fds_nth (fds mangoldt) n / nat_power n s‖)
    by (fold fds_abs_converges_def, intro 1)
  moreover have (λn. fds_nth (fds mangoldt) n / nat_power n s) sums (− deriv zeta s / zeta s)
    by (subst eval_fds_mangoldt(1) [symmetric], intro s, fold fds_converges_iff, intro 2)
   ultimately have ((λn. fds_nth (fds mangoldt) n / n nat_powr s) has_sum − deriv zeta s / zeta s)
UNIV
    by (fold nat_power_complex_def, rule norm_summable_imp_has_sum)
  moreover have [simp]: (if n = 0 then 0 else mangoldt n) = mangoldt n for n by auto
  ultimately have (?f has_sum − deriv zeta s / zeta s) UNIV by (auto simp add: fds_nth_fds)
  hence 3: (?f has_sum − deriv zeta s / zeta s) UNIV by auto
  have sum ?f {0} = 0 by auto
  moreover have (?f has_sum sum ?f {0}) {0}
    by (rule has_sum_finite, auto)
  ultimately have (?f has_sum 0) {0} by auto
  hence (?f has_sum − deriv zeta s / zeta s − 0) (UNIV − {0})
    by (intro has_sum_Diff 3, auto)
```

```
  moreover have UNIV − {0 :: nat} = {1..} by auto
  ultimately show (?f has_sum − deriv zeta s / zeta s) {1..} by auto
qed

lemma sums_Re_logderiv_zeta:
  fixes σ t :: real
  assumes s: σ > 1
  shows ((λn. mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (t ∗ ln n))
    has_sum Re (− deriv zeta (Complex σ t) / zeta (Complex σ t))) {1..}
proof −
  have ((λx. Re (mangoldt_complex x / x nat_powr Complex σ t))
    has_sum Re (− deriv zeta (Complex σ t) / zeta (Complex σ t))) {1..}
    using s by (intro has_sum_Re sums_mangoldt) auto
  moreover have Re (mangoldt n / n nat_powr (Complex σ t))
    = mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (t ∗ ln n) if ∗: 1 ≤ n for n
  proof −
    let ?n = n :: complex
    have 1 / n nat_powr (Complex σ t) = n nat_powr (Complex (−σ) (−t))
      by (fold powr_minus_divide, auto simp add: legacy_Complex_simps)
    also have . . . = exp (Complex (−σ ∗ ln n) (−t ∗ ln n))
      unfolding powr_def by (auto simp add: field_simps legacy_Complex_simps, use ∗ in linarith)
    finally have Re (1 / n nat_powr (Complex σ t)) = Re . . . by auto
    also have . . . = n nat_powr (−σ) ∗ cos (t ∗ ln n)
      by (unfold powr_def, subst Re_exp, use ∗ in auto)
    finally have 1: mangoldt_real n ∗ Re (1 / n nat_powr (Complex σ t))
      = mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (t ∗ ln n) by auto
    have rule_1: Re (w ∗ z) = Re w ∗ Re z if ∗: Im w = 0 for z w :: complex using ∗ by auto
    have Re (mangoldt n ∗ (1 / n nat_powr (Complex σ t)))
      = mangoldt_real n ∗ Re (1 / n nat_powr (Complex σ t))
      by (subst rule_1, auto)
    with 1 show ?thesis by auto
  qed
  ultimately show ((λn. mangoldt_real n ∗ n nat_powr (− σ) ∗ cos (t ∗ ln (real n)))
    has_sum Re (− deriv zeta (Complex σ t) / zeta (Complex σ t))) {1..}
    by (subst has_sum_cong) auto
qed

lemma logderiv_zeta_ineq:
  fixes σ t :: real
  assumes s: σ > 1
  shows 3 ∗ Re (logderiv zeta (Complex σ 0)) + 4 ∗ Re (logderiv zeta (Complex σ t))
    + Re (logderiv zeta (Complex σ (2∗t))) ≤ 0 (is ?x ≤ 0)
proof −
  have [simp]: Re (−z) = − Re z for z by auto
  have ((λn.
      3 ∗ (mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (0 ∗ ln n))
    + 4 ∗ (mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (t ∗ ln n))
    + 1 ∗ (mangoldt_real n ∗ n nat_powr (−σ) ∗ cos (2∗t ∗ ln n))
    ) has_sum
      3 ∗ Re (− deriv zeta (Complex σ 0) / zeta (Complex σ 0))
    + 4 ∗ Re (− deriv zeta (Complex σ t) / zeta (Complex σ t))
    + 1 ∗ Re (− deriv zeta (Complex σ (2∗t)) / zeta (Complex σ (2∗t)))
    ) {1..}
  by (intro has_sum_add has_sum_cmult_right sums_Re_logderiv_zeta s)
  hence ∗: ((λn. mangoldt_real n ∗ n nat_powr (−σ)
```

```
    * (3 + 4 * cos (t * ln n) + cos (2 * (t * ln n)))
    ) has_sum −?x) {1..}
  unfolding logderiv_def by (auto simp add: field_simps)
  have −?x ≥ 0
  by (rule has_sum_nonneg, rule *,
     intro mult_nonneg_nonneg,
     auto intro: mangoldt_nonneg cos_inequality_1)
  thus ?x ≤ 0 by linarith
qed

lemma sums_zeta_real:
  fixes r :: real
  assumes 1 < r
  shows (∑n. (n₊) powr −r) = Re (zeta r)
proof −
  have (∑n. (n₊) powr −r) = (∑n. Re (n₊ powr (−r :: complex)))
    by (subst of_real_nat_power) auto
  also have ... = (∑n. Re (n₊ powr − (r :: complex))) by auto
  also have ... = Re (∑n. n₊ powr − (r :: complex))
    by (intro Re_suminf [symmetric] summable_zeta)
       (use assms in auto)
  also have ... = Re (zeta r)
    using Re_complex_of_real zeta_conv_suminf assms by presburger
  finally show ?thesis .
qed

lemma inverse_zeta_bound':
  assumes 1 < Re s
  shows ‖inverse (zeta s)‖ ≤ Re (zeta (Re s))
proof −
  write moebius_mu (‹μ›)
  let ?f = λn :: nat. μ (n₊) / (n₊) powr s
  let ?g = λn :: nat. (n₊) powr − Re s
  have ‖μ n :: complex‖ ≤ 1 for n by (auto simp add: power_neg_one_If moebius_mu_def)
  hence 1: ‖?f n‖ ≤ ?g n for n
    by (auto simp add: powr_minus norm_divide norm_powr_real_powr field_simps)
  have inverse (zeta s) = (∑n. ?f n)
    by (intro sums_unique inverse_zeta_sums assms)
  hence ‖inverse (zeta s)‖ = ‖∑n. ?f n‖ by auto
  also have ... ≤ (∑n. ?g n) by (intro suminf_norm_bound summable_zeta_real assms 1)
  finally show ?thesis using sums_zeta_real assms by auto
qed

lemma zeta_bound':
  assumes 1 < Re s
  shows ‖zeta s‖ ≤ Re (zeta (Re s))
proof −
  let ?f = λn :: nat. (n₊) powr − s
  let ?g = λn :: nat. (n₊) powr − Re s
  have zeta s = (∑n. ?f n) by (intro sums_unique sums_zeta assms)
  hence ‖zeta s‖ = ‖∑n. ?f n‖ by auto
  also have ... ≤ (∑n. ?g n)
    by (intro suminf_norm_bound summable_zeta_real assms)
       (subst norm_nat_power, auto)
  also have ... = Re (zeta (Re s)) by (subst sums_zeta_real) (use assms in auto)
```

```
  finally show ?thesis .
qed

lemma zeta_bound_trivial':
  assumes 1 / 2 ≤ Re s ∧ Re s ≤ 2
    and |Im s| ≥ 1 / 11
  shows ‖zeta s‖ ≤ 12 + 2 * |Im s|
proof -
  have ‖pre_zeta 1 s‖ ≤ ‖s‖ / Re s
    by (rule pre_zeta_1_bound) (use assms in auto)
  also have ... ≤ (|Re s| + |Im s|) / Re s
  proof -
    have ‖s‖ ≤ |Re s| + |Im s| using cmod_le by auto
    thus ?thesis using assms by (auto intro: divide_right_mono)
  qed
  also have ... = 1 + |Im s| / Re s
    using assms by (simp add: field_simps)
  also have ... ≤ 1 + |Im s| / (1 / 2)
    using assms by (intro add_left_mono divide_left_mono) auto
  finally have 1: ‖pre_zeta 1 s‖ ≤ 1 + 2 * |Im s| by auto
  have ‖1 / (s - 1)‖ = 1 / ‖s - 1‖ by (subst norm_divide) auto
  also have ... ≤ 11 proof -
    have 1 / 11 ≤ |Im s| by (rule assms(2))
    also have ... = |Im (s - 1)| by auto
    also have ... ≤ ‖s - 1‖ by (rule abs_Im_le_cmod)
    finally show ?thesis by (intro mult_imp_div_pos_le) auto
  qed
  finally have 2: ‖1 / (s - 1)‖ ≤ 11 by auto
  have zeta s = pre_zeta 1 s + 1 / (s - 1) by (intro zeta_pole_eq) (use assms in auto)
  moreover have ‖...‖ ≤ ‖pre_zeta 1 s‖ + ‖1 / (s - 1)‖ by (rule norm_triangle_ineq)
  ultimately have ‖zeta s‖ ≤ ... by auto
  also have ... ≤ 12 + 2 * |Im s| using 1 2 by auto
  finally show ?thesis .
qed

lemma zeta_bound_gt_1:
  assumes 1 < Re s
  shows ‖zeta s‖ ≤ Re s / (Re s - 1)
proof -
  have ‖zeta s‖ ≤ Re (zeta (Re s)) by (intro zeta_bound' assms)
  also have ... ≤ ‖zeta (Re s)‖ by (rule complex_Re_le_cmod)
  also have ... = ‖pre_zeta 1 (Re s) + 1 / (Re s - 1)‖
    by (subst zeta_pole_eq) (use assms in auto)
  also have ... ≤ ‖pre_zeta 1 (Re s)‖ + ‖1 / (Re s - 1) :: complex‖
    by (rule norm_triangle_ineq)
  also have ... ≤ 1 + 1 / (Re s - 1)
  proof -
    have ‖pre_zeta 1 (Re s)‖ ≤ ‖Re s :: complex‖ / Re (Re s)
      by (rule pre_zeta_1_bound) (use assms in auto)
    also have ... = 1 using assms by auto
    moreover have ‖1 / (Re s - 1) :: complex‖ = 1 / (Re s - 1)
      by (subst norm_of_real) (use assms in auto)
    ultimately show ?thesis by auto
  qed
  also have ... = Re s / (Re s - 1)
```

```
    using assms by (auto simp add: field_simps)
  finally show ?thesis .
qed

lemma zeta_bound_trivial:
  assumes 1 / 2 ≤ Re s and |Im s| ≥ 1 / 11
  shows ‖zeta s‖ ≤ 12 + 2 * |Im s|
proof (cases Re s ≤ 2)
  assume Re s ≤ 2
  thus ?thesis by (intro zeta_bound_trivial') (use assms in auto)
next
  assume ¬ Re s ≤ 2
  hence *: Re s > 1 Re s > 2 by auto
  hence ‖zeta s‖ ≤ Re s / (Re s − 1) by (intro zeta_bound_gt_1)
  also have ... ≤ 2 using * by (auto simp add: field_simps)
  also have ... ≤ 12 + 2 * |Im s| by auto
  finally show ?thesis .
qed

lemma zeta_nonzero_small_imag':
  assumes |Im s| ≤ 13 / 22 and Re s ≥ 1 / 2 and Re s < 1
  shows zeta s ≠ 0
proof −
  have ‖pre_zeta 1 s‖ ≤ (1 + ‖s‖ / Re s) / 2 * 1 powr − Re s
    by (rule pre_zeta_bound) (use assms(2) in auto)
  also have ... ≤ 129 / 100 proof −
    have ‖s‖ / Re s ≤ 79 / 50
    proof (rule ccontr)
      assume ¬ ‖s‖ / Re s ≤ 79 / 50
      hence sqrt (6241 / 2500) < ‖s‖ / Re s by (simp add: real_sqrt_divide)
      also have ... = ‖s‖ / sqrt ((Re s)^2) using assms(2) by simp
      also have ... = sqrt (1 + (Im s / Re s)^2)
        unfolding cmod_def using assms(2)
        by (auto simp add: real_sqrt_divide [symmetric] field_simps
                simp del: real_sqrt_abs)
      finally have 1: 6241 / 2500 < 1 + (Im s / Re s)^2 by auto
      have |Im s / Re s| ≤ |6 / 5| using assms by (auto simp add: field_simps abs_le_square_iff)
      hence (Im s / Re s)^2 ≤ (6 / 5)^2 by (subst (asm) abs_le_square_iff)
      hence 2: 1 + (Im s / Re s)^2 ≤ 61 / 25 unfolding power2_eq_square by auto
      from 1 2 show False by auto
    qed
    hence (1 + ‖s‖ / Re s) / 2 ≤ (129 / 50) / 2 by (subst divide_right_mono) auto
    also have ... = 129 / 100 by auto
    finally show ?thesis by auto
  qed
  finally have 1: ‖pre_zeta 1 s‖ ≤ 129 / 100 .
  have ‖s − 1‖ < 100 / 129 proof −
    from assms have (Re (s − 1))^2 ≤ (1 / 2)^2 by (simp add: abs_le_square_iff [symmetric])
     moreover have (Im (s − 1))^2 ≤ (13 / 22)^2 using assms(1) by (simp add: abs_le_square_iff
[symmetric])
    ultimately have (Re (s − 1))^2 + (Im (s − 1))^2 ≤ 145 / 242 by (auto simp add: power2_eq_square)
    hence sqrt ((Re (s − 1))^2 + (Im (s − 1))^2) ≤ sqrt (145 / 242) by (rule real_sqrt_le_mono)
    also have ... < sqrt ((100 / 129)^2) by (subst real_sqrt_less_iff) (simp add: power2_eq_square)
    finally show ?thesis unfolding cmod_def by auto
  qed
```

```
  moreover have ‖s − 1‖ ≠ 0 using assms(3) by auto
  ultimately have 2: ‖1 / (s − 1)‖ > 129 / 100 by (auto simp add: field_simps norm_divide)
  from 1 2 have 0 < ‖1 / (s − 1)‖ − ‖pre_zeta 1 s‖ by auto
  also have ... ≤ ‖pre_zeta 1 s + 1 / (s − 1)‖ by (subst add.commute) (rule norm_diff_ineq)
  also from assms(3) have s ≠ 1 by auto
  hence ‖pre_zeta 1 s + 1 / (s − 1)‖ = ‖zeta s‖ using zeta_pole_eq by auto
  finally show ?thesis by auto
qed

lemma zeta_nonzero_small_imag:
  assumes |Im s| ≤ 13 / 22 and Re s > 0 and s ≠ 1
  shows zeta s ≠ 0
proof −
  consider Re s ≤ 1 / 2 | 1 / 2 ≤ Re s ∧ Re s < 1 | Re s ≥ 1 by fastforce
  thus ?thesis proof cases
    case 1 hence zeta (1 − s) ≠ 0 using assms by (intro zeta_nonzero_small_imag') auto
    moreover case 1
    ultimately show ?thesis using assms(2) zeta_zero_reflect_iff by auto
  next
    case 2 thus ?thesis using assms(1) by (intro zeta_nonzero_small_imag') auto
  next
    case 3 thus ?thesis using zeta_Re_ge_1_nonzero assms(3) by auto
  qed
qed

lemma inverse_zeta_bound:
  assumes 1 < Re s
  shows ‖inverse (zeta s)‖ ≤ Re s / (Re s − 1)
proof −
  have ‖inverse (zeta s)‖ ≤ Re (zeta (Re s)) by (intro inverse_zeta_bound' assms)
  also have ... ≤ ‖zeta (Re s)‖ by (rule complex_Re_le_cmod)
  also have ... ≤ Re (Re s) / (Re (Re s) − 1)
    by (intro zeta_bound_gt_1) (use assms in auto)
  also have ... = Re s / (Re s − 1) by auto
  finally show ?thesis .
qed

lemma deriv_zeta_bound:
  fixes s :: complex
  assumes Hr: 0 < r and Hs: s ≠ 1
    and hB: ⋀w. ‖s − w‖ = r ⟹ ‖pre_zeta 1 w‖ ≤ B
  shows ‖deriv zeta s‖ ≤ B / r + 1 / ‖s − 1‖^2
proof −
  have ‖deriv zeta s‖ = ‖deriv (pre_zeta 1) s − 1 / (s − 1)^2‖
  proof −
    let ?A = UNIV − {1 :: complex}
    let ?f = λs. pre_zeta 1 s + 1 / (s − 1)
    let ?v = deriv (pre_zeta 1) s + (0 ∗ (s − 1) − 1 ∗ (1 − 0)) / (s − 1)^2
    let ?v' = deriv (pre_zeta 1) s − 1 / (s − 1 :: complex)^2
    have ∀ z∈?A. zeta z = pre_zeta 1 z + 1 / (z − 1)
      by (auto intro: zeta_pole_eq)
    hence ∀_F z in nhds s. zeta z = pre_zeta 1 z + 1 / (z − 1)
      using Hs by (subst eventually_nhds, intro exI [where x = ?A]) auto
    hence DERIV zeta s :> ?v' = DERIV ?f s :> ?v'
      by (intro DERIV_cong_ev) auto
```

```
    moreover have DERIV ?f s :> ?v
      unfolding power2_eq_square
      by (intro derivative_intros field_differentiable_derivI holomorphic_pre_zeta
          holomorphic_on_imp_differentiable_at [where s = ?A])
        (use Hs in auto)
    moreover have ?v = ?v' by (auto simp add: field_simps)
    ultimately have DERIV zeta s :> ?v' by auto
    moreover have DERIV zeta s :> deriv zeta s
      by (intro field_differentiable_derivI field_differentiable_at_zeta)
        (use Hs in auto)
    ultimately have ?v' = deriv zeta s by (rule DERIV_unique)
    thus ?thesis by auto
  qed
  also have ... ≤ ‖deriv (pre_zeta 1) s‖ + ‖1 / (s − 1)^2‖ by (rule norm_triangle_ineq4)
  also have ... ≤ B / r + 1 / ‖s − 1‖^2
  proof −
    have ‖(deriv ^^ 1) (pre_zeta 1) s‖ ≤ fact 1 ∗ B / r ^ 1
      by (intro Cauchy_inequality holomorphic_pre_zeta continuous_on_pre_zeta assms) auto
    thus ?thesis by (auto simp add: norm_divide norm_power)
  qed
  finally show ?thesis .
qed

lemma zeta_lower_bound:
  assumes 0 < Re s s ≠ 1
  shows 1 / ‖s − 1‖ − ‖s‖ / Re s ≤ ‖zeta s‖
proof −
  have ‖pre_zeta 1 s‖ ≤ ‖s‖ / Re s by (intro pre_zeta_1_bound assms)
  hence 1 / ‖s − 1‖ − ‖s‖ / Re s ≤ ‖1 / (s − 1)‖ − ‖pre_zeta 1 s‖
    using assms by (auto simp add: norm_divide)
  also have ... ≤ ‖pre_zeta 1 s + 1 / (s − 1)‖
    by (subst add.commute) (rule norm_diff_ineq)
  also have ... = ‖zeta s‖ using assms by (subst zeta_pole_eq) auto
  finally show ?thesis .
qed

lemma logderiv_zeta_bound:
  fixes σ :: real
  assumes 1 < σ σ ≤ 23 / 20
  shows ‖logderiv zeta σ‖ ≤ 5 / 4 ∗ (1 / (σ − 1))
proof −
  have ‖pre_zeta 1 s‖ ≤ sqrt 2 if ∗: ‖σ − s‖ = 1 / sqrt 2 for s :: complex
  proof −
    have 1: 0 < Re s proof −
      have 1 − Re s ≤ Re (σ − s) using assms(1) by auto
      also have Re (σ − s) ≤ ‖σ − s‖ by (rule complex_Re_le_cmod)
      also have ... = 1 / sqrt 2 by (rule ∗)
      finally have 1 − 1 / sqrt 2 ≤ Re s by auto
      moreover have 0 < 1 − 1 / sqrt 2 by auto
      ultimately show ?thesis by linarith
    qed
    hence ‖pre_zeta 1 s‖ ≤ ‖s‖ / Re s by (rule pre_zeta_1_bound)
    also have ... ≤ sqrt 2 proof −
      define x y where x ≡ Re s and y ≡ Im s
      have sqrt ((σ − x)^2 + y^2) = 1 / sqrt 2
```

```
      using ∗ unfolding cmod_def x_def y_def by auto
    also have . . . = sqrt (1 / 2) by (auto simp add: field_simps real_sqrt_mult [symmetric])
    finally have 2: x^2 + y^2 − 2∗σ∗x + σ^2 = 1 / 2 by (auto simp add: field_simps power2_eq_square)
    have y^2 ≤ x^2 proof (rule ccontr)
      assume ¬ y^2 ≤ x^2
      hence x^2 < y^2 by auto
      with 2 have 2∗x^2 − 2∗σ∗x + σ^2 < 1 / 2 by auto
      hence 2 ∗ (x − σ / 2)^2 < (1 − σ^2) / 2 by (auto simp add: field_simps power2_eq_square)
      also have . . . < 0 using ‹1 < σ› by auto
      finally show False by auto
    qed
    moreover have x ≠ 0 unfolding x_def using 1 by auto
    ultimately have sqrt ((x^2 + y^2) / x^2) ≤ sqrt 2 by (auto simp add: field_simps)
    with 1 show ?thesis unfolding cmod_def x_def y_def by (auto simp add: real_sqrt_divide)
  qed
  finally show ?thesis .
qed
hence ‖deriv zeta σ‖ ≤ sqrt 2 / (1 / sqrt 2) + 1 / ‖(σ :: complex) − 1‖^2
  by (intro deriv_zeta_bound) (use assms(1) in auto)
also have . . . ≤ 2 + 1 / (σ − 1)^2
  by (subst in_Reals_norm) (use assms(1) in auto)
also have . . . = (2 ∗ σ^2 − 4 ∗ σ + 3) / (σ − 1)^2
proof −
  have σ ∗ σ − 2 ∗ σ + 1 = (σ − 1) ∗ (σ − 1) by (auto simp add: field_simps)
  also have . . . ≠ 0 using assms(1) by auto
  finally show ?thesis by (auto simp add: power2_eq_square field_simps)
qed
finally have 1: ‖deriv zeta σ‖ ≤ (2 ∗ σ^2 − 4 ∗ σ + 3) / (σ − 1)^2 .
have (2 − σ) / (σ − 1) = 1 / ‖(σ :: complex) − 1‖ − ‖σ :: complex‖ / Re σ
  using assms(1) by (auto simp add: field_simps in_Reals_norm)
also have . . . ≤ ‖zeta σ‖ by (rule zeta_lower_bound) (use assms(1) in auto)
finally have 2: (2 − σ) / (σ − 1) ≤ ‖zeta σ‖ .
have 4 ∗ (2 ∗ σ^2 − 4 ∗ σ + 3) − 5 ∗ (2 − σ) = 8 ∗ (σ − 11 / 16)^2 − 57 / 32
  by (auto simp add: field_simps power2_eq_square)
also have . . . ≤ 0 proof −
  have 0 ≤ σ − 11 / 16 using assms(1) by auto
  moreover have σ − 11 / 16 ≤ 37 / 80 using assms(2) by auto
  ultimately have (σ − 11 / 16)^2 ≤ (37 / 80)^2 by auto
  thus ?thesis by (auto simp add: power2_eq_square)
qed
finally have 4 ∗ (2 ∗ σ^2 − 4 ∗ σ + 3) − 5 ∗ (2 − σ) ≤ 0 .
moreover have 0 < 2 − σ using assms(2) by auto
ultimately have 3: (2 ∗ σ^2 − 4 ∗ σ + 3) / (2 − σ) ≤ 5 / 4 by (subst pos_divide_le_eq) auto
moreover have 0 ≤ 2 ∗ σ^2 − 4 ∗ σ + 3 proof −
  have 0 ≤ 2 ∗ (σ − 1)^2 + 1 by auto
  also have . . . = 2 ∗ σ^2 − 4 ∗ σ + 3 by (auto simp add: field_simps power2_eq_square)
  finally show ?thesis .
qed
moreover have 0 < (2 − σ) / (σ − 1) using assms by auto
ultimately have ‖logderiv zeta σ‖ ≤ ((2 ∗ σ^2 − 4 ∗ σ + 3) / (σ − 1)^2) / ((2 − σ) / (σ − 1))
  unfolding logderiv_def using 1 2 by (subst norm_divide) (rule frac_le, auto)
also have . . . = (2 ∗ σ^2 − 4 ∗ σ + 3) / (2 − σ) ∗ (1 / (σ − 1))
  by (simp add: power2_eq_square)
also have . . . ≤ 5 / 4 ∗ (1 / (σ − 1))
  using 3 by (rule mult_right_mono) (use assms(1) in auto)
```

```
  finally show ?thesis .
qed

lemma Re_logderiv_zeta_bound:
  fixes σ :: real
  assumes 1 < σ σ ≤ 23 / 20
  shows Re (logderiv zeta σ) ≥ − 5 / 4 ∗ (1 / (σ − 1))
proof −
  have − Re (logderiv zeta σ) = Re (− logderiv zeta σ) by auto
  also have Re (− logderiv zeta σ) ≤ ‖− logderiv zeta σ‖ by (rule complex_Re_le_cmod)
  also have . . . = ‖logderiv zeta σ‖ by auto
  also have . . . ≤ 5 / 4 ∗ (1 / (σ − 1)) by (intro logderiv_zeta_bound assms)
  finally show ?thesis by auto
qed

locale zeta_bound_param =
  fixes ϑ φ :: real ⇒ real
  assumes zeta_bn′: ⋀z. 1 − ϑ (Im z) ≤ Re z ⟹ Im z ≥ 1 / 11 ⟹ ‖zeta z‖ ≤ exp (φ (Im z))
    and ϑ_pos: ⋀t. 0 < ϑ t ∧ ϑ t ≤ 1 / 2
    and φ_pos: ⋀t. 1 ≤ φ t
    and inv_ϑ: ⋀t. φ t / ϑ t ≤ 1 / 960 ∗ exp (φ t)
    and moϑ: antimono ϑ and moφ: mono φ
begin
  definition region ≡ {z. 1 − ϑ (Im z) ≤ Re z ∧ Im z ≥ 1 / 11}
  lemma zeta_bn: ⋀z. z ∈ region ⟹ ‖zeta z‖ ≤ exp (φ (Im z))
    using zeta_bn′ unfolding region_def by auto
  lemma ϑ_pos′: ⋀t. 0 < ϑ t ∧ ϑ t ≤ 1
    using ϑ_pos by (smt (verit) exp_ge_add_one_self exp_half_le2)
  lemma φ_pos′: ⋀t. 0 < φ t using φ_pos by (smt (verit, ccfv_SIG))
end

locale zeta_bound_param_1 = zeta_bound_param +
  fixes γ :: real
  assumes γ_cnd: γ ≥ 13 / 22
begin
  definition r where r ≡ ϑ (2 ∗ γ + 1)
end

locale zeta_bound_param_2 = zeta_bound_param_1 +
  fixes σ δ :: real
  assumes σ_cnd: σ ≥ 1 + exp (− φ(2 ∗ γ + 1))
     and δ_cnd: δ = γ ∨ δ = 2 ∗ γ
begin
  definition s where s ≡ Complex σ δ
end

context zeta_bound_param_2 begin
declare dist_complex_def [simp] norm_minus_commute [simp]
declare legacy_Complex_simps [simp]

lemma cball_lm:
  assumes z ∈ cball s r
  shows r ≤ 1 |Re z − σ| ≤ r |Im z − δ| ≤ r
       1 / 11 ≤ Im z Im z ≤ 2 ∗ γ + r
proof −
```

```
  have |Re (z − s)| ≤ ‖z − s‖ |Im (z − s)| ≤ ‖z − s‖
    by (rule abs_Re_le_cmod) (rule abs_Im_le_cmod)
  moreover have ‖z − s‖ ≤ r using assms by auto
  ultimately show 1: |Re z − σ| ≤ r |Im z − δ| ≤ r unfolding s_def by auto
  moreover have 3: r ≤ 1 / 2 unfolding r_def using ϑ_pos by auto
  ultimately have 2: |Re z − σ| ≤ 1 / 2 |Im z − δ| ≤ 1 / 2 by auto
  moreover have δ ≤ 2 ∗ γ using δ_cnd γ_cnd by auto
  ultimately show Im z ≤ 2 ∗ γ + r using 1 by auto
  have 1 / 11 ≤ δ − 1 / 2 using δ_cnd γ_cnd by auto
  also have ... ≤ Im z using 2 by (auto simp del: Num.le_divide_eq_numeral1)
  finally show 1 / 11 ≤ Im z .
  from 3 show r ≤ 1 by auto
qed

lemma cball_in_region:
  shows cball s r ⊆ region
proof
  fix z :: complex
  assume hz: z ∈ cball s r
  note lm = cball_lm [OF hz]
  hence 1 − ϑ (Im z) ≤ 1 − ϑ (2 ∗ γ + ϑ (2 ∗ γ + 1))
    unfolding r_def using moϑ lm by (auto intro: antimonoD)
  also have ... ≤ 1 + exp (−φ (2 ∗ γ + 1)) − ϑ (2 ∗ γ + 1)
  proof −
    have 2 ∗ γ + ϑ (2 ∗ γ + 1) ≤ 2 ∗ γ + 1
      unfolding r_def using ϑ_pos' by auto
    hence ϑ (2 ∗ γ + 1) − ϑ (2 ∗ γ + ϑ (2 ∗ γ + 1)) ≤ 0
      using moϑ by (auto intro: antimonoD)
    also have 0 ≤ exp (−φ (2 ∗ γ + 1)) by auto
    finally show ?thesis by auto
  qed
  also have ... ≤ σ − r using σ_cnd unfolding r_def s_def by auto
  also have ... ≤ Re z using lm by auto
  finally have 1 − ϑ (Im z) ≤ Re z .
  thus z ∈ region unfolding region_def using lm by auto
qed

lemma Re_s_gt_1:
  shows 1 < Re s
proof −
  have ∗: exp (− φ (2 ∗ γ + 1)) > 0 by auto
  show ?thesis using σ_cnd s_def by auto (use ∗ in linarith)
qed

lemma zeta_analytic_on_region:
  shows zeta analytic_on region
  by (rule analytic_zeta) (unfold region_def, auto)

lemma zeta_div_bound:
  assumes z ∈ cball s r
  shows ‖zeta z / zeta s‖ ≤ exp (3 ∗ φ (2 ∗ γ + 1))
proof −
  let ?φ = φ (2 ∗ γ + 1)
  have ‖zeta z‖ ≤ exp (φ (Im z)) using cball_in_region zeta_bn assms by auto
  also have ... ≤ exp (?φ)
```

```
  proof -
    have Im z ≤ 2 * γ + 1 using cball_lm [OF assms] by auto
    thus ?thesis by auto (rule monoD [OF moφ])
  qed
  also have ‖inverse (zeta s)‖ ≤ exp (2 * ?φ)
  proof -
    have ‖inverse (zeta s)‖ ≤ Re s / (Re s - 1)
      by (intro inverse_zeta_bound Re_s_gt_1)
    also have ... = 1 + 1 / (Re s - 1)
      using Re_s_gt_1 by (auto simp add: field_simps)
    also have ... ≤ 1 + exp (?φ)
    proof -
      have Re s - 1 ≥ exp (-?φ) using s_def σ_cnd by auto
      hence 1 / (Re s - 1) ≤ 1 / exp (-?φ)
        using Re_s_gt_1 by (auto intro: divide_left_mono)
      thus ?thesis by (auto simp add: exp_minus field_simps)
    qed
    also have ... ≤ exp (2 * ?φ) by (intro exp_lemma_1 less_imp_le φ_pos)
    finally show ?thesis .
  qed
  ultimately have ‖zeta z * inverse (zeta s)‖ ≤ exp (?φ) * exp (2 * ?φ)
    by (subst norm_mult, intro mult_mono') auto
  also have ... = exp (3 * ?φ) by (subst exp_add [symmetric]) auto
  finally show ?thesis by (auto simp add: divide_inverse)
qed

lemma logderiv_zeta_bound:
  shows Re (logderiv zeta s) ≥ - 24 * φ (2 * γ + 1) / r
    and ⋀β. σ - r / 2 ≤ β ⟹ zeta (Complex β δ) = 0 ⟹
      Re (logderiv zeta s) ≥ - 24 * φ (2 * γ + 1) / r + 1 / (σ - β)
proof -
  have 1: 0 < r unfolding r_def using ϑ_pos' by auto
  have 2: 0 ≤ 3 * φ (2 * γ + 1) using φ_pos' by (auto simp add: less_imp_le)
  have 3: zeta s ≠ 0 ⋀z. Re s < Re z ⟹ zeta z ≠ 0
    using Re_s_gt_1 by (auto intro!: zeta_Re_gt_1_nonzero)
  have 4: zeta analytic_on cball s r
    by (rule analytic_on_subset;
      rule cball_in_region zeta_analytic_on_region)
  have 5: z ∈ cball s r ⟹ ‖zeta z / zeta s‖ ≤ exp (3 * φ (2 * γ + 1))
    for z by (rule zeta_div_bound)
  have 6: {} ⊆ {z ∈ cball s (r / 2). zeta z = 0} by auto
  have 7: {Complex β δ} ⊆ {z ∈ cball s (r / 2). zeta z = 0}
    if σ - r / 2 ≤ β zeta (Complex β δ) = 0 for β
  proof -
    have β ≤ σ
      using zeta_Re_gt_1_nonzero [of Complex β δ] Re_s_gt_1 that(2)
      unfolding s_def by fastforce
    thus ?thesis using that unfolding s_def by auto
  qed
  have - Re (logderiv zeta s) ≤ 8 * (3 * φ (2 * γ + 1)) / r + Re (∑z∈{}. 1 / (z - s))
    by (intro lemma_3_9_beta3 1 2 3 4 5 6)
  thus Re (logderiv zeta s) ≥ - 24 * φ (2 * γ + 1) / r by auto
  show Re (logderiv zeta s) ≥ - 24 * φ (2 * γ + 1) / r + 1 / (σ - β)
    if *: σ - r / 2 ≤ β zeta (Complex β δ) = 0 for β
  proof -
```

```
    have bs: β ≠ σ using ∗(2) 3(1) unfolding s_def by auto
    hence bs': 1 / (β − σ) = − 1 / (σ − β) by (auto simp add: field_simps)
    have inv_r: 1 / (Complex r 0) = Complex (1 / r) 0 if r ≠ 0 for r
      using that by (auto simp add: field_simps)
    have − Re (logderiv zeta s) ≤ 8 ∗ (3 ∗ φ (2 ∗ γ + 1)) / r + Re (∑ z∈{Complex β δ}. 1 / (z − s))
      by (intro lemma_3_9_beta3 1 2 3 4 5 7 ∗)
    thus ?thesis unfolding s_def
      by (auto simp add: field_simps)
        (subst (asm) inv_r, use bs bs' in auto)
  qed
qed
end

context zeta_bound_param_1 begin
lemma zeta_nonzero_region':
  assumes 1 + 1 / 960 ∗ (r / φ (2 ∗ γ + 1)) − r / 2 ≤ β
    and zeta (Complex β γ) = 0
  shows 1 − β ≥ 1 / 29760 ∗ (r / φ (2 ∗ γ + 1))
proof −
  let ?φ = φ (2 ∗ γ + 1) and ?ϑ = ϑ (2 ∗ γ + 1)
  define σ where σ ≡ 1 + 1 / 960 ∗ (r / φ (2 ∗ γ + 1))
  define a where a ≡ − 5 / 4 ∗ (1 / (σ − 1))
  define b where b ≡ − 24 ∗ φ (2 ∗ γ + 1) / r + 1 / (σ − β)
  define c where c ≡ − 24 ∗ φ (2 ∗ γ + 1) / r
  have 1 + exp (− ?φ) ≤ σ
  proof −
    have 960 ∗ exp (− ?φ) = 1 / (1 / 960 ∗ exp ?φ)
      by (auto simp add: exp_add [symmetric] field_simps)
    also have ... ≤ 1 / (?φ / ?ϑ) proof −
      have ?φ / ?ϑ ≤ 1 / 960 ∗ exp ?φ by (rule inv_ϑ)
      thus ?thesis by (intro divide_left_mono) (use ϑ_pos φ_pos' in auto)
    qed
    also have ... = r / ?φ unfolding r_def by auto
    finally show ?thesis unfolding σ_def by auto
  qed
  note ∗ = this γ_cnd
  interpret z: zeta_bound_param_2 ϑ φ γ σ γ by (standard, use ∗ in auto)
  interpret z': zeta_bound_param_2 ϑ φ γ σ 2 ∗ γ by (standard, use ∗ in auto)
  have r ≤ 1 unfolding r_def using ϑ_pos' [of 2 ∗ γ + 1] by auto
  moreover have 1 ≤ φ (2 ∗ γ + 1) using φ_pos by auto
  ultimately have r / φ (2 ∗ γ + 1) ≤ 1 by auto
  moreover have 0 < r 0 < φ (2 ∗ γ + 1) unfolding r_def using ϑ_pos' φ_pos' by auto
  hence 0 < r / φ (2 ∗ γ + 1) by auto
  ultimately have 1: 1 < σ σ ≤ 23 / 20 unfolding σ_def by auto
  hence Re (logderiv zeta σ) ≥ a unfolding a_def by (intro Re_logderiv_zeta_bound)
  hence Re (logderiv zeta (Complex σ 0)) ≥ a by auto
  moreover have Re (logderiv zeta z.s) ≥ b unfolding b_def
    by (rule z.logderiv_zeta_bound) (use assms r_def σ_def in auto)
  hence Re (logderiv zeta (Complex σ γ)) ≥ b unfolding z.s_def by auto
  moreover have Re (logderiv zeta z'.s) ≥ c unfolding c_def by (rule z'.logderiv_zeta_bound)
  hence Re (logderiv zeta (Complex σ (2 ∗ γ))) ≥ c unfolding z'.s_def by auto
  ultimately have 3 ∗ a + 4 ∗ b + c
    ≤ 3 ∗ Re (logderiv zeta (Complex σ 0)) + 4 ∗ Re (logderiv zeta (Complex σ γ))
    + Re (logderiv zeta (Complex σ (2 ∗ γ))) by auto
  also have ... ≤ 0 by (rule logderiv_zeta_ineq, rule 1)
```

```
  finally have 3 * a + 4 * b + c ≤ 0 .
  hence 4 / (σ − β) ≤ 15 / 4 * (1 / (σ − 1)) + 120 * φ (2 * γ + 1) / r
    unfolding a_def b_def c_def by auto
  also have ... = 3720 * φ (2 * γ + 1) / r unfolding σ_def by auto
  finally have 2: inverse (σ − β) ≤ 930 * φ (2 * γ + 1) / r by (auto simp add: inverse_eq_divide)
  have 3: σ − β ≥ 1 / 930 * (r / φ (2 * γ + 1))
  proof −
    have 1 / 930 * (r / φ (2 * γ + 1)) = 1 / (930 * (φ (2 * γ + 1) / r))
      by (auto simp add: field_simps)
    also have ... ≤ σ − β proof −
      have β ≤ 1 using assms(2) zeta_Re_gt_1_nonzero [of Complex β γ] by fastforce
      also have 1 < σ by (rule 1)
      finally have β < σ .
      thus ?thesis using 2 by (auto intro: inverse_le_imp_le)
    qed
    finally show ?thesis .
  qed
  show ?thesis proof −
    let ?x = r / φ (2 * γ + 1)
    have 1 / 29760 * ?x = 1 / 930 * ?x − 1 / 960 * ?x by auto
    also have ... ≤ (σ − β) − (σ − 1) using 3 by (subst (2) σ_def) auto
    also have ... = 1 − β by auto
    finally show ?thesis .
  qed
qed

lemma zeta_nonzero_region:
  assumes zeta (Complex β γ) = 0
  shows 1 − β ≥ 1 / 29760 * (r / φ (2 * γ + 1))
proof (cases 1 + 1 / 960 * (r / φ (2 * γ + 1)) − r / 2 ≤ β)
  case True
  thus ?thesis using assms by (rule zeta_nonzero_region')
next
  case False
  let ?x = r / φ (2 * γ + 1)
  assume 1: ¬ 1 + 1 / 960 * ?x − r / 2 ≤ β
  have 0 < r using ϑ_pos' unfolding r_def by auto
  hence 1 / 930 * ?x ≤ r / 2
    using φ_pos [of 2 * γ + 1] by (auto intro!: mult_imp_div_pos_le)
  hence 1 / 29760 * ?x ≤ r / 2 − 1 / 960 * ?x by auto
  also have ... ≤ 1 − β using 1 by auto
  finally show ?thesis .
qed
end

context zeta_bound_param begin
theorem zeta_nonzero_region:
  assumes zeta (Complex β γ) = 0 and Complex β γ ≠ 1
  shows 1 − β ≥ 1 / 29760 * (ϑ (2 * |γ| + 1) / φ (2 * |γ| + 1))
proof (cases |γ| ≥ 13 / 22)
  case True
  assume 1: 13 / 22 ≤ |γ|
  have 2: zeta (Complex β |γ|) = 0
  proof (cases γ ≥ 0)
    case True thus ?thesis using assms by auto
```

```
  next
    case False thus ?thesis by (auto simp add: complex_cnj [symmetric] intro: assms)
  qed
  interpret z: zeta_bound_param_1 ϑ φ ‹|γ|› by standard (use 1 in auto)
  show ?thesis by (intro z.zeta_nonzero_region [unfolded z.r_def] 2)
next
  case False
  hence 1: |γ| ≤ 13 / 22 by auto
  show ?thesis
  proof (cases 0 < β, rule ccontr)
    case True thus False using zeta_nonzero_small_imag [of Complex β γ] assms 1 by auto
  next
    have 0 < ϑ (2 ∗ |γ| + 1) ϑ (2 ∗ |γ| + 1) ≤ 1 1 ≤ φ (2 ∗ |γ| + 1)
      using ϑ_pos' φ_pos by auto
    hence 1 / 29760 ∗ (ϑ (2 ∗ |γ| + 1) / φ (2 ∗ |γ| + 1)) ≤ 1 by auto
    also case False hence 1 ≤ 1 − β by auto
    finally show ?thesis .
  qed
qed
end

lemma zeta_bound_param_nonneg:
  fixes ϑ φ :: real ⇒ real
  assumes zeta_bn': ⋀z. 1 − ϑ (Im z) ≤ Re z ⟹ Im z ≥ 1 / 11 ⟹ ‖zeta z‖ ≤ exp (φ (Im z))
    and ϑ_pos: ⋀t. 0 ≤ t ⟹ 0 < ϑ t ∧ ϑ t ≤ 1 / 2
    and φ_pos: ⋀t. 0 ≤ t ⟹ 1 ≤ φ t
    and inv_ϑ: ⋀t. 0 ≤ t ⟹ φ t / ϑ t ≤ 1 / 960 ∗ exp (φ t)
    and moϑ: ⋀x y. 0 ≤ x ⟹ x ≤ y ⟹ ϑ y ≤ ϑ x
    and moφ: ⋀x y. 0 ≤ x ⟹ x ≤ y ⟹ φ x ≤ φ y
  shows zeta_bound_param (λt. ϑ (max 0 t)) (λt. φ (max 0 t))
  by standard (insert assms, auto simp add: antimono_def mono_def)

interpretation classical_zeta_bound:
  zeta_bound_param λt. 1 / 2 λt. 4 ∗ ln (12 + 2 ∗ max 0 t)
proof −
  define ϑ :: real ⇒ real where ϑ ≡ λt. 1 / 2
  define φ :: real ⇒ real where φ ≡ λt. 4 ∗ ln (12 + 2 ∗ t)
  have zeta_bound_param (λt. ϑ (max 0 t)) (λt. φ (max 0 t))
  proof (rule zeta_bound_param_nonneg)
    fix z assume ∗: 1 − ϑ (Im z) ≤ Re z Im z ≥ 1 / 11
    have ‖zeta z‖ ≤ 12 + 2 ∗ |Im z|
      using ∗ unfolding ϑ_def by (intro zeta_bound_trivial) auto
    also have ... = exp (ln (12 + 2 ∗ Im z)) using ∗(2) by auto
    also have ... ≤ exp (φ (Im z)) proof −
      have 0 ≤ ln (12 + 2 ∗ Im z) using ∗(2) by auto
      thus ?thesis unfolding φ_def by auto
    qed
    finally show ‖zeta z‖ ≤ exp (φ (Im z)) .
  next
    fix t :: real assume ∗: 0 ≤ t
    have φ t / ϑ t = 8 ∗ ln (12 + 2 ∗ t) unfolding φ_def ϑ_def by auto
    also have ... ≤ 8 ∗ (5 / 2 + t)
    proof −
      have ln (12 + 2 ∗ t) = ln (12 ∗ (1 + t / 6)) by auto
      also have ... = ln 12 + ln (1 + t / 6)
```

```
      unfolding ln_mult using * by simp
    also have ... ≤ 5 / 2 + t / 6
    proof (rule add_mono)
      have (144 :: real) < (271 / 100) ^ 5
        by (simp add: power_numeral_reduce)
      also have 271 / 100 < exp (1 :: real)
        using e_approx_32 by (simp add: abs_if split: if_split_asm)
      hence (271 / 100) ^ 5 < exp (1 :: real) ^ 5
        by (rule power_strict_mono) auto
      also have ... = exp ((5 :: nat) * (1 :: real))
        by (rule exp_of_nat_mult [symmetric])
      also have ... = exp (5 :: real)
        by auto
      finally have exp (ln (12 :: real) * (2 :: nat)) ≤ exp 5
        by (subst exp_of_nat2_mult) auto
      thus ln (12 :: real) ≤ 5 / 2
        by auto
      show ln (1 + t / 6) ≤ t / 6
        by (intro ln_add_one_self_le_self) (use * in auto)
    qed
    finally show ?thesis using * by auto
  qed
  also have ... ≤ 1 / 960 * exp (φ t)
  proof -
    have 8 * (5 / 2 + t) - 1 / 960 * (12 + 2 * t) ^ 4
        = -(1 / 60 * t ^ 4 + 2 / 5 * t ^ 3 + 18 / 5 * t ^ 2 + 32 / 5 * t + 8 / 5)
      by (simp add: power_numeral_reduce field_simps)
    also have ... ≤ 0 using *
      by (subst neg_le_0_iff_le) (auto intro: add_nonneg_nonneg)
    moreover have exp (φ t) = (12 + 2 * t) ^ 4
    proof -
      have exp (φ t) = (12 + 2 * t) powr (real 4) unfolding φ_def powr_def using * by auto
      also have ... = (12 + 2 * t) ^ 4 by (rule powr_realpow) (use * in auto)
      finally show ?thesis .
    qed
    ultimately show ?thesis by auto
  qed
  finally show φ t / ϑ t ≤ 1 / 960 * exp (φ t) .
next
  fix t :: real assume *: 0 ≤ t
  have (1 :: real) ≤ 4 * 1 by auto
  also have ... ≤ 4 * ln 12
  proof -
    have exp (1 :: real) ≤ 3 by (rule exp_le)
    also have ... ≤ exp (ln 12) by auto
    finally have (1 :: real) ≤ ln 12 using exp_le_cancel_iff by blast
    thus ?thesis by auto
  qed
  also have ... ≤ 4 * ln (12 + 2 * t) using * by auto
  finally show 1 ≤ φ t unfolding φ_def .
next
  show ⋀t. 0 < ϑ t ∧ ϑ t ≤ 1 / 2
       ⋀x y. 0 ≤ x ⟹ x ≤ y ⟹ ϑ y ≤ ϑ x
       ⋀x y. 0 ≤ x ⟹ x ≤ y ⟹ φ x ≤ φ y
    unfolding ϑ_def φ_def by auto
```

```
  qed
  thus zeta_bound_param (λt. 1 / 2) (λt. 4 ∗ ln (12 + 2 ∗ max 0 t))
    unfolding ϑ_def φ_def by auto
qed

theorem zeta_nonzero_region:
  assumes zeta (Complex β γ) = 0 and Complex β γ ≠ 1
  shows 1 − β ≥ C_1 / ln (|γ| + 2)
proof −
  have 1 / 952320 ∗ (1 / ln (|γ| + 2))
      ≤ 1 / 29760 ∗ (1 / 2 / (4 ∗ ln (12 + 2 ∗ max 0 (2 ∗ |γ| + 1)))) (is ?x ≤ ?y)
  proof −
    have ln (14 + 4 ∗ |γ|) ≤ 4 ∗ ln (|γ| + 2) by (rule ln_bound_1) auto
    hence 1 / 238080 / (4 ∗ ln (|γ| + 2)) ≤ 1 / 238080 / (ln (14 + 4 ∗ |γ|))
      by (intro divide_left_mono) auto
    also have . . . = ?y by auto
    finally show ?thesis by auto
  qed
  also have . . . ≤ 1 − β by (intro classical_zeta_bound.zeta_nonzero_region assms)
  finally show ?thesis unfolding PNT_const_C_1_def by auto
qed

unbundle no pnt_syntax

end
theory PNT_Subsummable
imports
  PNT_Remainder_Library
begin
unbundle pnt_syntax

definition has_subsum where has_subsum f S x ≡ (λn. if n ∈ S then f n else 0) sums x
definition subsum where subsum f S ≡ ∑n. if n ∈ S then f n else 0
definition subsummable (infix ‹subsummable› 50)
  where f subsummable S ≡ summable (λn. if n ∈ S then f n else 0)

syntax _subsum :: pttrn ⇒ nat set ⇒ 'a ⇒ 'a
  (‹(2∑ ‘_ ∈ (_)./ _)› [0, 0, 10] 10)
syntax_consts _subsum == subsum
translations
  ∑ ‘ x∈S. t => CONST subsum (λx. t) S

syntax _subsum_prop :: pttrn ⇒ bool ⇒ 'a ⇒ 'a
  (‹(2∑ ‘_ | (_)./ _)› [0, 0, 10] 10)
syntax_consts _subsum_prop == subsum
translations
  ∑ ‘ x|P. t => CONST subsum (λx. t) {x. P}

syntax _subsum_ge :: pttrn ⇒ nat ⇒ 'a ⇒ 'a
  (‹(2∑ ‘_ ≥ _./ _)› [0, 0, 10] 10)
syntax_consts _subsum_ge == subsum
translations
  ∑ ‘ x≥n. t => CONST subsum (λx. t) {n..}

lemma has_subsum_finite:
```

finite F $\Longrightarrow$ *has_subsum f F* (*sum f F*)
unfolding *has_subsum_def* **by** (*rule sums_If_finite_set*)

lemma *has_subsum_If_finite_set*:
assumes *finite F*
shows *has_subsum* (λ*n*. *if n* $\in$ *F then f n else 0*) *A* (*sum f* (*F* $\cap$ *A*))
proof −
have *F* $\cap$ *A* = {*x*. *x* $\in$ *A* $\wedge$ *x* $\in$ *F*} **by** *auto*
thus *?thesis* **unfolding** *has_subsum_def* **using** *assms*
by (*auto simp add*: *if_if_eq_conj intro*!: *sums_If_finite*)
qed

lemma *has_subsum_If_finite*:
assumes *finite* {*n* $\in$ *A*. *p n*}
shows *has_subsum* (λ*n*. *if p n then f n else 0*) *A* (*sum f* {*n* $\in$ *A*. *p n*})
unfolding *has_subsum_def* **using** *assms*
by (*auto simp add*: *if_if_eq_conj intro*!: *sums_If_finite*)

lemma *has_subsum_univ*:
f sums v $\Longrightarrow$ *has_subsum f UNIV v*
unfolding *has_subsum_def* **by** *auto*

lemma *subsumI*:
fixes *f* :: *nat* $\Rightarrow$ *'a* :: {*t2_space*, *comm_monoid_add*}
shows *has_subsum f A x* $\Longrightarrow$ *x* = *subsum f A*
unfolding *has_subsum_def subsum_def* **by** (*intro sums_unique*)

lemma *has_subsum_summable*:
has_subsum f A x $\Longrightarrow$ *f subsummable A*
unfolding *has_subsum_def subsummable_def* **by** (*rule sums_summable*)

lemma *subsummable_sums*:
fixes *f* :: *nat* $\Rightarrow$ *'a* :: {*comm_monoid_add*, *t2_space*}
shows *f subsummable S* $\Longrightarrow$ *has_subsum f S* (*subsum f S*)
unfolding *subsummable_def has_subsum_def subsum_def* **by** (*intro summable_sums*)

lemma *has_subsum_diff_finite*:
fixes *S* :: *'a* :: {*topological_ab_group_add*, *t2_space*}
assumes *finite F has_subsum f A S F* $\subseteq$ *A*
shows *has_subsum f* (*A* − *F*) (*S* − *sum f F*)
proof −
define *p* **where** *p n* $\equiv$ *if n* $\in$ *F then 0 else* (*if n* $\in$ *A then f n else 0*) **for** *n*
define *q* **where** *q n* $\equiv$ *if n* $\in$ *A* − *F then f n else 0* **for** *n*
have *F* $\cap$ *A* = *F* **using** *assms*(*3*) **by** *auto*
hence *p sums* (*S* − *sum f F*)
using *assms* **unfolding** *p_def has_subsum_def*
by (*auto intro*: *sums_If_finite_set'* [**where** *?S* = *S*]
simp: *sum_negf sum.inter_restrict* [*symmetric*])
moreover have *p* = *q* **unfolding** *p_def q_def* **by** *auto*
finally show *?thesis* **unfolding** *q_def has_subsum_def* **by** *auto*
qed

lemma *subsum_split*:
fixes *f* :: *nat* $\Rightarrow$ *'a* :: {*topological_ab_group_add*, *t2_space*}
assumes *f subsummable A finite F F* $\subseteq$ *A*

shows *subsum f A* = *sum f F* + *subsum f* (*A* − *F*)
proof −
  from *assms*(*1*) **have** *has_subsum f A* (*subsum f A*) **by** (*intro subsummable_sums*)
  hence *has_subsum f* (*A* − *F*) (*subsum f A* − *sum f F*)
    using *assms* **by** (*intro has_subsum_diff_finite*)
  hence *subsum f A* − *sum f F* = *subsum f* (*A* − *F*) **by** (*rule subsumI*)
  thus *?thesis* **by** (*auto simp add*: *algebra_simps*)
qed

lemma *has_subsum_zero* [*simp*]: *has_subsum* (λ*n*. *0*) *A 0* **unfolding** *has_subsum_def* **by** *auto*
lemma *zero_subsummable* [*simp*]: (λ*n*. *0*) *subsummable A* **unfolding** *subsummable_def* **by** *auto*
lemma *zero_subsum* [*simp*]: (∑ '*n*∈*A*. *0* :: ′*a* :: {*comm_monoid_add*, *t2_space*}) = *0* **unfolding** *subsum_def* **by** *auto*

lemma *has_subsum_minus*:
  fixes *f* :: *nat* ⇒ ′*a* :: *real_normed_vector*
  assumes *has_subsum f A a has_subsum g A b*
  shows *has_subsum* (λ*n*. *f n* − *g n*) *A* (*a* − *b*)
proof −
  define *p* **where** *p n* = (*if n* ∈ *A then f n else 0*) **for** *n*
  define *q* **where** *q n* = (*if n* ∈ *A then g n else 0*) **for** *n*
  have (λ*n*. *p n* − *q n*) *sums* (*a* − *b*)
    using *assms* **unfolding** *p_def q_def has_subsum_def* **by** (*intro sums_diff*)
  moreover have (*if n* ∈ *A then f n* − *g n else 0*) = *p n* − *q n* **for** *n*
    unfolding *p_def q_def* **by** *auto*
  ultimately show *?thesis* **unfolding** *has_subsum_def* **by** *auto*
qed

lemma *subsum_minus*:
  assumes *f subsummable A g subsummable A*
  shows *subsum f A* − *subsum g A* = (∑ '*n*∈*A*. *f n* − *g n* :: ′*a* :: *real_normed_vector*)
  by (*intro subsumI has_subsum_minus subsummable_sums assms*)

lemma *subsummable_minus*:
  assumes *f subsummable A g subsummable A*
  shows (λ*n*. *f n* − *g n* :: ′*a* :: *real_normed_vector*) *subsummable A*
  by (*auto intro*: *has_subsum_summable has_subsum_minus subsummable_sums assms*)

lemma *has_subsum_uminus*:
  assumes *has_subsum f A a*
  shows *has_subsum* (λ*n*. − *f n* :: ′*a* :: *real_normed_vector*) *A* (− *a*)
proof −
  have *has_subsum* (λ*n*. *0* − *f n*) *A* (*0* − *a*)
    by (*intro has_subsum_minus*) (*use assms* **in** *auto*)
  thus *?thesis* **by** *auto*
qed

lemma *subsum_uminus*:
  f subsummable A ⟹ − *subsum f A* = (∑ '*n*∈*A*. − *f n* :: ′*a* :: *real_normed_vector*)
  by (*intro subsumI has_subsum_uminus subsummable_sums*)

lemma *subsummable_uminus*:
  f subsummable A ⟹ (λ*n*. − *f n* :: ′*a* :: *real_normed_vector*) *subsummable A*
  by (*auto intro*: *has_subsum_summable has_subsum_uminus subsummable_sums*)

lemma *has_subsum_add*:
  fixes *f* :: *nat* ⇒ *'a* :: *real_normed_vector*
  assumes *has_subsum f A a has_subsum g A b*
  shows *has_subsum* (λ*n*. *f n* + *g n*) *A* (*a* + *b*)
proof –
  have *has_subsum* (λ*n*. *f n* − − *g n*) *A* (*a* − − *b*)
    by (*intro has_subsum_minus has_subsum_uminus assms*)
  thus *?thesis* **by** *auto*
qed

lemma *subsum_add*:
  assumes *f subsummable A g subsummable A*
  shows *subsum f A* + *subsum g A* = (∑ ‘*n*∈*A*. *f n* + *g n* :: *'a* :: *real_normed_vector*)
  by (*intro subsumI has_subsum_add subsummable_sums assms*)

lemma *subsummable_add*:
  assumes *f subsummable A g subsummable A*
  shows (λ*n*. *f n* + *g n* :: *'a* :: *real_normed_vector*) *subsummable A*
  by (*auto intro*: *has_subsum_summable has_subsum_add subsummable_sums assms*)

lemma *subsum_cong*:
  (⋀*x*. *x* ∈ *A* ⟹ *f x* = *g x*) ⟹ *subsum f A* = *subsum g A*
  unfolding *subsum_def* **by** (*intro suminf_cong*) *auto*

lemma *subsummable_cong*:
  fixes *f* :: *nat* ⇒ *'a* :: *real_normed_vector*
  shows (⋀*x*. *x* ∈ *A* ⟹ *f x* = *g x*) ⟹ (*f subsummable A*) = (*g subsummable A*)
  unfolding *subsummable_def* **by** (*intro summable_cong*) *auto*

lemma *subsum_norm_bound*:
  fixes *f* :: *nat* ⇒ *'a* :: *banach*
  assumes *g subsummable A* ⋀*n*. *n* ∈ *A* ⟹ ‖*f n*‖ ≤ *g n*
  shows ‖*subsum f A*‖ ≤ *subsum g A*
  using *assms* **unfolding** *subsummable_def subsum_def*
  by (*intro suminf_norm_bound*) *auto*

lemma *eval_fds_subsum*:
  fixes *f* :: *'a* :: {*nat_power*, *banach*, *real_normed_field*} *fds*
  assumes *fds_converges f s*
  shows *has_subsum* (λ*n*. *fds_nth f n* / *nat_power n s*) {*1*..} (*eval_fds f s*)
proof –
  let *?f* = λ*n*. *fds_nth f n* / *nat_power n s*
  let *?v* = *eval_fds f s*
  have *has_subsum* (λ*n*. *?f n*) *UNIV ?v*
    by (*intro has_subsum_univ fds_converges_iff* [*THEN iffD1*] *assms*)
  hence *has_subsum ?f* (*UNIV* − {*0*}) (*?v* − *sum ?f* {*0*})
    by (*intro has_subsum_diff_finite*) *auto*
  moreover have *UNIV* − {*0* :: *nat*} = {*1*..} **by** *auto*
  ultimately show *?thesis* **by** *auto*
qed

lemma *fds_abs_subsummable*:
  fixes *f* :: *'a* :: {*nat_power*, *banach*, *real_normed_field*} *fds*
  assumes *fds_abs_converges f s*
  shows (λ*n*. ‖*fds_nth f n* / *nat_power n s*‖) *subsummable* {*1*..}

```
proof -
  have summable (λn. ‖fds_nth f n / nat_power n s‖)
    by (subst fds_abs_converges_def [symmetric]) (rule assms)
  moreover have ‖fds_nth f n / nat_power n s‖ = 0 when ¬ 1 ≤ n for n
  proof -
    have n = 0 using that by auto
    thus ?thesis by auto
  qed
  hence (λn. if 1 ≤ n then ‖fds_nth f n / nat_power n s‖ else 0)
       = (λn. ‖fds_nth f n / nat_power n s‖) by auto
  ultimately show ?thesis unfolding subsummable_def by auto
qed

lemma subsum_mult2:
  fixes f :: nat ⇒ 'a :: real_normed_algebra
  shows f subsummable A ⟹ (∑ `x∈A. f x * c) = subsum f A * c
unfolding subsum_def subsummable_def
  by (subst suminf_mult2) (auto intro: suminf_cong)

lemma subsummable_mult2:
  fixes f :: nat ⇒ 'a :: real_normed_algebra
  assumes f subsummable A
  shows (λx. f x * c) subsummable A
proof -
  have summable (λn. (if n ∈ A then f n else 0) * c) (is ?P)
    using assms unfolding subsummable_def by (intro summable_mult2)
  moreover have ?P = ?thesis
    unfolding subsummable_def by (rule summable_cong) auto
  ultimately show ?thesis by auto
qed

lemma subsum_ge_limit:
  lim (λN. ∑ n = m..N. f n) = (∑ `n ≥ m. f n)
proof -
  define g where g n ≡ if n ∈ {m..} then f n else 0 for n
  have (∑ n. g n) = lim (λN. ∑ n<N. g n) by (rule suminf_eq_lim)
  also have ... = lim (λN. ∑ n<N + 1. g n)
    unfolding lim_def using LIMSEQ_ignore_initial_segment LIMSEQ_offset
    by (intro The_cong iffI) blast
  also have ... = lim (λN. ∑ n = m..N. f n)
  proof -
    have {x. x < N + 1 ∧ m ≤ x} = {m..N} for N by auto
    thus ?thesis unfolding g_def by (subst sum.inter_filter [symmetric]) auto
  qed
  finally show ?thesis unfolding subsum_def g_def by auto
qed

lemma has_subsum_ge_limit:
  fixes f :: nat ⇒ 'a :: {t2_space, comm_monoid_add, topological_space}
  assumes ((λN. ∑ n = m..N. f n) ⟶ l) at_top
  shows has_subsum f {m..} l
proof -
  define g where g n ≡ if n ∈ {m..} then f n else 0 for n
  have ((λN. ∑ n<N + 1. g n) ⟶ l) at_top
  proof -
```

```
    have {x. x < N + 1 ∧ m ≤ x} = {m..N} for N by auto
    with assms show ?thesis
      unfolding g_def by (subst sum.inter_filter [symmetric]) auto
  qed
  hence ((λN. ∑n<N. g n) ⟶ l) at_top by (rule LIMSEQ_offset)
  thus ?thesis unfolding has_subsum_def sums_def g_def by auto
qed

lemma eval_fds_complex:
  fixes f :: complex fds
  assumes fds_converges f s
  shows has_subsum (λn. fds_nth f n / n nat_powr s) {1..} (eval_fds f s)
proof −
  have has_subsum (λn. fds_nth f n / nat_power n s) {1..} (eval_fds f s)
    by (intro eval_fds_subsum assms)
  thus ?thesis unfolding nat_power_complex_def .
qed

lemma eval_fds_complex_subsum:
  fixes f :: complex fds
  assumes fds_converges f s
  shows eval_fds f s = (∑‘n ≥ 1. fds_nth f n / n nat_powr s)
        (λn. fds_nth f n / n nat_powr s) subsummable {1..}
proof (goal_cases)
  case 1 show ?case by (intro subsumI eval_fds_complex assms)
  case 2 show ?case by (intro has_subsum_summable) (rule eval_fds_complex assms)+
qed

lemma has_sum_imp_has_subsum:
  fixes x :: 'a :: {comm_monoid_add, t2_space}
  assumes (f has_sum x) A
  shows has_subsum f A x
proof −
  have (∀F x in at_top. sum f ({..<x} ∩ A) ∈ S)
    when open S x ∈ S for S
  proof −
    have ∀S. open S ⟶ x ∈ S ⟶ (∀F x in finite_subsets_at_top A. sum f x ∈ S)
      using assms unfolding has_sum_def tendsto_def .
    hence ∀F x in finite_subsets_at_top A. sum f x ∈ S using that by auto
    then obtain X where hX: finite X X ⊆ A
      and hY: ⋀Y. finite Y ⟹ X ⊆ Y ⟹ Y ⊆ A ⟹ sum f Y ∈ S
      unfolding eventually_finite_subsets_at_top by metis
    define n where n ≡ Max X + 1
    show ?thesis
    proof (subst eventually_sequentially, standard, safe)
      fix m assume Hm: n ≤ m
      moreover have x ∈ X ⟹ x < n for x
        unfolding n_def using Max_ge [OF hX(1), of x] by auto
      ultimately show sum f ({..<m} ∩ A) ∈ S
        using hX(2) by (intro hY, auto) (metis order.strict_trans2)
    qed
  qed
  thus ?thesis unfolding has_subsum_def sums_def tendsto_def
    by (simp add: sum.inter_restrict [symmetric])
qed
```

```
unbundle no pnt_syntax

end
theory Perron_Formula
imports
  PNT_Remainder_Library
  PNT_Subsummable
begin
unbundle pnt_syntax
```

5 Perron's formula

This version of Perron's theorem is referenced to: *Perron′s Formula and the Prime Number Theorem for Automorphic L−Functions*, *Jianya Liu*, *Y. Ye*

A contour integral estimation lemma that will be used both in proof of Perron's formula and the prime number theorem.

```
lemma perron_aux_3':
  fixes f :: complex ⇒ complex and a b B T :: real
  assumes Ha: 0 < a and Hb: 0 < b and hT: 0 < T
    and Hf: ⋀t. t ∈ {−T..T} ⟹ ‖f (Complex b t)‖ ≤ B
    and Hf': (λs. f s ∗ a powr s / s) contour_integrable_on (linepath (Complex b (−T)) (Complex b T))
  shows ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral (linepath (Complex b (−T)) (Complex b T)) (λs. f s ∗ a powr
s / s)‖
      ≤ B ∗ a powr b ∗ ln (1 + T / b)
proof −
  define path where path ≡ linepath (Complex b (−T)) (Complex b T)
  define t' where t' t ≡ Complex (Re (Complex b (−T))) t for t
  define g where g t ≡ f (Complex b t) ∗ a powr (Complex b t) / Complex b t ∗ i for t
  have ‖f (Complex b 0)‖ ≤ B using hT by (auto intro: Hf [of 0])
  hence hB: 0 ≤ B using hT by (smt (verit) norm_ge_zero)
  have ((λt. f (t' t) ∗ a powr (t' t) / (t' t) ∗ i)
        has_integral contour_integral path (λs. f s ∗ a powr s / s)) {Im (Complex b (−T))..Im (Complex b
T)}
    unfolding t'_def using hT
    by (intro integral_linepath_same_Re, unfold path_def)
       (auto intro: has_contour_integral_integral Hf')
  hence h_int: (g has_integral contour_integral path (λs. f s ∗ a powr s / s)) {−T..T}
    unfolding g_def t'_def by auto
  hence int: g integrable_on {−T..T} by (rule has_integral_integrable)
  have contour_integral path (λs. f s ∗ a powr s / s) = integral {−T..T} g
    using h_int by (rule integral_unique [symmetric])
  also have ‖...‖ ≤ integral {−T..T} (λt. 2 ∗ B ∗ a powr b / (b + |t|))
  proof (rule integral_norm_bound_integral, goal_cases)
    case 1 from int show ?case .
    case 2 show ?case
      by (intro integrable_continuous_interval continuous_intros)
         (use Hb in auto)
  next
    fix t assume ∗: t ∈ {−T..T}
    have (b + |t|)^2 − 4 ∗ (b^2 + t^2) = − 3 ∗ (b − |t|)^2 + − 4 ∗ b ∗ |t|
      by (simp add: field_simps power2_eq_square)
    also have ... ≤ 0 using Hb by (intro add_nonpos_nonpos) auto
    finally have (b + |t|)^2 − 4 ∗ (b^2 + t^2) ≤ 0 .
```

```
    hence b + |t| ≤ 2 * ‖Complex b t‖
      unfolding cmod_def by (auto intro: power2_le_imp_le)
    hence a powr b / ‖Complex b t‖ ≤ a powr b / ((b + |t|) / 2)
      using Hb by (intro divide_left_mono) (auto intro!: mult_pos_pos)
    hence a powr b / ‖Complex b t‖ * ‖f (Complex b t)‖ ≤ a powr b / ((b + |t|) / 2) * B
      by (insert Hf [OF *], rule mult_mono) (use Hb in auto)
    thus ‖g t‖ ≤ 2 * B * a powr b / (b + |t|)
      unfolding g_def
      by (auto simp add: norm_mult norm_divide)
         (subst norm_powr_real_powr, insert Ha, auto simp add: mult_ac)
  qed
  also have ... = 2 * B * a powr b * integral {-T..T} (λt. 1 / (b + |t|))
    by (subst divide_inverse, subst integral_mult_right) (simp add: inverse_eq_divide)
  also have ... = 4 * B * a powr b * integral {0..T} (λt. 1 / (b + |t|))
  proof -
    let ?f = λt. 1 / (b + |t|)
    have integral {-T..0} ?f + integral {0..T} ?f = integral {-T..T} ?f
      by (intro Henstock_Kurzweil_Integration.integral_combine
              integrable_continuous_interval continuous_intros)
         (use Hb hT in auto)
    moreover have integral {-T..-0} (λt. ?f (-t)) = integral {0..T} ?f
      by (rule Henstock_Kurzweil_Integration.integral_reflect_real)
    hence integral {-T..0} ?f = integral {0..T} ?f by auto
    ultimately show ?thesis by auto
  qed
  also have ... = 4 * B * a powr b * ln (1 + T / b)
  proof -
    have ((λt. 1 / (b + |t|)) has_integral (ln (b + T) - ln (b + 0))) {0..T}
    proof (rule fundamental_theorem_of_calculus, goal_cases)
      case 1 show ?case using hT by auto
    next
      fix x assume *: x ∈ {0..T}
      have ((λx. ln (b + x)) has_real_derivative 1 / (b + x) * (0 + 1)) (at x within {0..T})
        by (intro derivative_intros) (use Hb * in auto)
      thus ((λx. ln (b + x)) has_vector_derivative 1 / (b + |x|)) (at x within {0..T})
        using * by (subst has_real_derivative_iff_has_vector_derivative [symmetric]) auto
    qed
    moreover have ln (b + T) - ln (b + 0) = ln (1 + T / b)
      using Hb hT by (simp add: ln_div field_simps)
    ultimately show ?thesis by auto
  qed
  finally have ‖1 / (2 * pi * i) * contour_integral path (λs. f s * a powr s / s)‖
    ≤ 1 / (2*pi) * 4 * B * a powr b * ln (1 + T / b)
    by (simp add: norm_divide norm_mult field_simps)
  also have ... ≤ 1 * B * a powr b * ln (1 + T / b)
  proof -
    have 1 / (2*pi) * 4 ≤ 1 using pi_gt3 by auto
    thus ?thesis by (intro mult_right_mono) (use hT Hb hB in auto)
  qed
  finally show ?thesis unfolding path_def by auto
qed

locale perron_locale =
  fixes b B H T x :: real and f :: complex fds
  assumes Hb: 0 < b and hT: b ≤ T
```

```
    and Hb': abs_conv_abscissa f < b
    and hH: 1 < H and hH': b + 1 ≤ H and Hx: 0 < x
    and hB: (∑`n ≥ 1. ‖fds_nth f n‖ / n nat_powr b) ≤ B
begin
definition r where r a ≡
  if a ≠ 1 then min (1 / (2 * T * |ln a|)) (2 + ln (T / b))
  else (2 + ln (T / b))
definition path where path ≡ linepath (Complex b (−T)) (Complex b T)
definition img_path where img_path ≡ path_image path
definition σ_a where σ_a ≡ abs_conv_abscissa f
definition region where region = {n :: nat. x − x / H ≤ n ∧ n ≤ x + x / H}
definition F where F (a :: real) ≡
  1 / (2 * pi * i) * contour_integral path (λs. a powr s / s) − (if 1 ≤ a then 1 else 0)
definition F' where F' (n :: nat) ≡ F (x / n)

lemma hT': 0 < T using Hb hT by auto
lemma cond: 0 < b b ≤ T 0 < T using Hb hT hT' by auto

lemma perron_integrable:
  assumes (0 :: real) < a
  shows (λs. a powr s / s) contour_integrable_on (linepath (Complex b (−T)) (Complex b T))
using cond assms
by (intro contour_integrable_continuous_linepath continuous_intros)
   (auto simp add: closed_segment_def legacy_Complex_simps field_simps)

lemma perron_aux_1':
  fixes U :: real
  assumes hU: 0 < U and Ha: 1 < a
  shows ‖F a‖ ≤ 1 / pi * a powr b / (T * |ln a|) + a powr −U * T / (pi * U)
proof −
  define f where f ≡ λs :: complex. a powr s / s
  note assms' = cond assms this
  define P_1 where P_1 ≡ linepath (Complex (−U) (−T)) (Complex b (−T))
  define P_2 where P_2 ≡ linepath (Complex b (−T)) (Complex b T)
  define P_3 where P_3 ≡ linepath (Complex b T) (Complex (−U) T)
  define P_4 where P_4 ≡ linepath (Complex (−U) T) (Complex (−U) (−T))
  define P where P ≡ P_1 +++ P_2 +++ P_3 +++ P_4
  define I_1 I_2 I_3 I_4 where
    I_1 ≡ contour_integral P_1 f and I_2 ≡ contour_integral P_2 f and
    I_3 ≡ contour_integral P_3 f and I_4 ≡ contour_integral P_4 f
  define rpath where rpath ≡ rectpath (Complex (−U) (−T)) (Complex b T)
  note P_defs = P_def P_1_def P_2_def P_3_def P_4_def
  note I_defs = I_1_def I_2_def I_3_def I_4_def
  have 1: ⋀A B x. A ⊆ B ⟹ x ∉ A ⟹ A ⊆ B − {x} by auto
  have path_image (rectpath (Complex (− U) (− T)) (Complex b T))
        ⊆ cbox (Complex (− U) (− T)) (Complex b T) − {0}
    using assms'
    by (intro 1 path_image_rectpath_subset_cbox)
       (auto simp add: path_image_rectpath)
  moreover have 0 ∈ box (Complex (− U) (− T)) (Complex b T)
    using assms' by (simp add: mem_box Basis_complex_def)
  ultimately have
    ((λs. a powr s / (s − 0)) has_contour_integral
      2 * pi * i * winding_number rpath 0 * a powr (0 :: complex)) rpath
    winding_number rpath 0 = 1
```

unfolding *rpath_def*
by (*intro Cauchy_integral_formula_convex_simple*
[**where** $S = cbox\ (Complex\ (-U)\ (-T))\ (Complex\ b\ T)$])
(*auto intro*!: *assms′ holomorphic_on_powr_right winding_number_rectpath*
simp add: *mem_box Basis_complex_def*)
hence (f *has_contour_integral* $2 * pi * i$) *rpath* **unfolding** *f_def* **using** *Ha* **by** *auto*
hence *2*: (f *has_contour_integral* $2 * pi * i$) P
unfolding *rpath_def P_defs rectpath_def Let_def* **by** *simp*
hence f *contour_integrable_on* P **by** (*intro has_contour_integral_integrable*) (*use 2* **in** *auto*)
hence *3*: f *contour_integrable_on* P_1 f *contour_integrable_on* P_2
f *contour_integrable_on* P_3 f *contour_integrable_on* P_4 **unfolding** *P_defs* **by** *auto*
from *2* **have** $I_1 + I_2 + I_3 + I_4 = 2 * pi * i$ **unfolding** *P_defs I_defs* **by** (*rule has_chain_integral_chain_integral4*
hence $I_2 - 2 * pi * i = -(I_1 + I_3 + I_4)$ **by** (*simp add*: *field_simps*)
hence $\|I_2 - 2 * pi * i\| = \|-(I_1 + I_3 + I_4)\|$ **by** *auto*
also have $\ldots = \|I_1 + I_3 + I_4\|$ **by** (*rule norm_minus_cancel*)
also have $\ldots \le \|I_1 + I_3\| + \|I_4\|$ **by** (*rule norm_triangle_ineq*)
also have $\ldots \le \|I_1\| + \|I_3\| + \|I_4\|$ **using** *norm_triangle_ineq* **by** *auto*
finally have $*$: $\|I_2 - 2 * pi * i\| \le \|I_1\| + \|I_3\| + \|I_4\|$ **.**
have *I_2_val*: $\|I_2 / (2 * pi * i) - 1\| \le 1 / (2*pi) * (\|I_1\| + \|I_3\| + \|I_4\|)$
proof −
have $I_2 - 2 * pi * i = (I_2 / (2 * pi * i) - 1) * (2 * pi * i)$ **by** (*auto simp add*: *field_simps*)
hence $\|I_2 - 2 * pi * i\| = \|(I_2 / (2 * pi * i) - 1) * (2 * pi * i)\|$ **by** *auto*
also have $\ldots = \|I_2 / (2 * pi * i) - 1\| * (2*pi)$ **by** (*auto simp add*: *norm_mult*)
finally have $\|I_2 / (2 * pi * i) - 1\| = 1 / (2*pi) * \|I_2 - 2 * pi * i\|$ **by** *auto*
also have $\ldots \le 1 / (2*pi) * (\|I_1\| + \|I_3\| + \|I_4\|)$
using $*$ **by** (*subst mult_le_cancel_left_pos*) *auto*
finally show *?thesis* **.**
qed
define Q **where** $Q\ t \equiv linepath\ (Complex\ (-U)\ t)\ (Complex\ b\ t)$ **for** t
define g **where** $g\ t \equiv contour\_integral\ (Q\ t)\ f$ **for** t
have *Q_1*: (f *has_contour_integral* I_1) $(Q\ (-T))$
using *3*(*1*) **unfolding** *P_1_def I_1_def Q_def*
by (*rule has_contour_integral_integral*)
have *Q_2*: (f *has_contour_integral* $-I_3$) $(Q\ T)$
using *3*(*3*) **unfolding** *P_3_def I_3_def Q_def*
by (*subst contour_integral_reversepath* [*symmetric*],
auto intro!: *has_contour_integral_integral*)
(*subst contour_integrable_reversepath_eq* [*symmetric*], *auto*)
have *subst_I_1_I_3*: $I_1 = g\ (-\ T)$ $I_3 = -\ g\ T$
using *Q_1 Q_2* **unfolding** *g_def* **by** (*auto simp add*: *contour_integral_unique*)
have *g_bound*: $\|g\ t\| \le a\ powr\ b / (T * |ln\ a|)$
when *Ht*: $|t| = T$ **for** t
proof −
have (f *has_contour_integral* $g\ t$) $(Q\ t)$ **proof** −
consider $t = T \mid t = -T$ **using** *Ht* **by** *fastforce*
hence f *contour_integrable_on* $Q\ t$ **using** *Q_1 Q_2* **by** (*metis has_contour_integral_integrable*)
thus *?thesis* **unfolding** *g_def* **by** (*rule has_contour_integral_integral*)
qed
hence $((\lambda x.\ a\ powr\ (x + Im\ (Complex\ (-U)\ t) * i) / (x + Im\ (Complex\ (-U)\ t) * i))$ *has_integral* $(g\ t))$
$\{Re\ (Complex\ (-U)\ t) .. Re\ (Complex\ b\ t)\}$
unfolding *Q_def f_def*
by (*subst has_contour_integral_linepath_same_Im_iff* [*symmetric*])
(*use hU Hb* **in** *auto*)
hence $*$: $((\lambda x.\ a\ powr\ (x + t * i) / (x + t * i))$ *has_integral* $g\ t)$ $\{-U..b\}$ **by** *auto*

hence ‖g t‖ = ‖integral {−U..b} (λx. a powr (x + t ∗ i) / (x + t ∗ i))‖ **by** (auto simp add: integral_unique)
also have ... ≤ integral {−U..b} (λx. a powr x / T)
proof (rule integral_norm_bound_integral)
show (λx. a powr (x + t ∗ i) / (x + t ∗ i)) integrable_on {−U..b} **using** ∗ **by** auto
have (λx. a powr x / (of_real T)) integrable_on {−U..b}
by (intro iffD2 [OF integrable_on_cdivide_iff] powr_integrable) (use hU Ha Hb hT′ **in** auto)
thus (λx. a powr x / T) integrable_on {−U..b} **by** auto
next
fix x **assume** x ∈ {−U..b}
have ‖a powr (x + t ∗ i)‖ = Re a powr Re (x + t ∗ i) **by** (rule norm_powr_real_powr) (use Ha **in** auto)
also have ... = a powr x **by** auto
finally have ∗: ‖a powr (x + t ∗ i)‖ = a powr x **.**
have T = |Im (x + t ∗ i)| **using** Ht **by** auto
also have ... ≤ ‖x + t ∗ i‖ **by** (rule abs_Im_le_cmod)
finally have T ≤ ‖x + t ∗ i‖ **.**
with ∗ **show** ‖a powr (x + t ∗ i) / (x + t ∗ i)‖ ≤ a powr x / T
by (subst norm_divide) (rule frac_le, use assms′ **in** auto)
qed
also have ... = integral {−U..b} (λx. a powr x) / T **by** auto
also have ... ≤ a powr b / (T ∗ |ln a|)
proof −
have integral {−U..b} (λx. a powr x) ≤ a powr b / |ln a|
by (rule powr_integral_bound_gt_1) (use hU Ha Hb **in** auto)
thus ?thesis **using** hT′ **by** (auto simp add: field_simps)
qed
finally show ?thesis **.**
qed
have $‖I_4‖$ ≤ a powr −U / U ∗ ‖Complex (− U) (− T) − Complex (− U) T‖
proof −
have f contour_integrable_on P_4 **by** (rule 3)
moreover have 0 ≤ a powr − U / U **using** hU **by** auto
moreover have ‖f z‖ ≤ a powr − U / U
when ∗: z ∈ closed_segment (Complex (− U) T) (Complex (− U) (− T)) **for** z
proof −
from ∗ **have** Re_z: Re z = − U
unfolding closed_segment_def
by (auto simp add: legacy_Complex_simps field_simps)
hence U = |Re z| **using** hU **by** auto
also have ... ≤ ‖z‖ **by** (rule abs_Re_le_cmod)
finally have zmod: U ≤ ‖z‖ **.**
have ‖f z‖ = ‖a powr z‖ / ‖z‖ **unfolding** f_def **by** (rule norm_divide)
also have ... ≤ a powr − U / U
by (subst norm_powr_real_powr, use Ha **in** auto)
(rule frac_le, use hU Re_z zmod **in** auto)
finally show ?thesis **.**
qed
ultimately show ?thesis **unfolding** I_4_def P_4_def **by** (rule contour_integral_bound_linepath)
qed
also have ... = a powr −U / U ∗ (2 ∗ T)
proof −
have sqrt ((2 ∗ T)^2) = |2 ∗ T| **by** (rule real_sqrt_abs)
thus ?thesis **using** hT′ **by** (auto simp add: field_simps legacy_Complex_simps)
qed

finally have I_4_bound: $\|I_4\| \le a\ powr\ -U\ /\ U * (2 * T)$ **.**
have $\|I_2\ /\ (2 * pi * \mathrm{i}) - 1\| \le 1\ /\ (2*pi) * (\|g\ (-\ T)\| + \|-\ g\ T\| + \|I_4\|)$
using $I_2\_val\ subst\_I_1\_I_3$ **by** *auto*
also have $\ldots \le 1\ /\ (2*pi) * (2 * a\ powr\ b\ /\ (T * |ln\ a|) + a\ powr\ -U\ /\ U * (2*T))$
proof −
have $\|g\ T\| \le a\ powr\ b\ /\ (T * |ln\ a|)$
$\|g\ (-\ T)\| \le a\ powr\ b\ /\ (T * |ln\ a|)$
using hT' **by** (*auto intro*: *g_bound*)
hence $\|g\ (-\ T)\| + \|-\ g\ T\| + \|I_4\| \le 2 * a\ powr\ b\ /\ (T * |ln\ a|) + a\ powr\ -U\ /\ U * (2*T)$
using I_4_bound **by** *auto*
thus *?thesis* **by** (*auto simp add*: *field_simps*)
qed
also have $\ldots = 1\ /\ pi * a\ powr\ b\ /\ (T * |ln\ a|) + a\ powr\ -U * T\ /\ (pi * U)$
using hT' **by** (*auto simp add*: *field_simps*)
finally show *?thesis*
using *Ha* **unfolding** $I_2\_def\ P_2\_def\ f\_def\ F\_def\ path\_def$ **by** *auto*
qed

lemma *perron_aux_1*:
assumes *Ha*: $1 < a$
shows $\|F\ a\| \le 1\ /\ pi * a\ powr\ b\ /\ (T * |ln\ a|)$ (**is** $\_ \le ?x$)
proof −
let $?y = \lambda U :: real.\ a\ powr\ -U * T\ /\ (pi * U)$
have $((\lambda U :: real.\ ?x) \longrightarrow ?x)\ at\_top$ **by** *auto*
moreover have $((\lambda U.\ ?y\ U) \longrightarrow 0)\ at\_top$ **using** *Ha* **by** *real_asymp*
ultimately have $((\lambda U.\ ?x + ?y\ U) \longrightarrow ?x + 0)\ at\_top$ **by** (*rule tendsto_add*)
hence $((\lambda U.\ ?x + ?y\ U) \longrightarrow ?x)\ at\_top$ **by** *auto*
moreover have $\|F\ a\| \le ?x + ?y\ U$ **when** *hU*: $0 < U$ **for** *U*
by (*subst perron_aux_1′* [*OF hU Ha*], *standard*)
hence $\forall_F\ U\ in\ at\_top.\ \|F\ a\| \le ?x + ?y\ U$
by (*rule eventually_at_top_linorderI′*)
ultimately show *?thesis*
by (*intro tendsto_lowerbound*) *auto*
qed

lemma *perron_aux_2′*:
fixes $U :: real$
assumes *hU*: $0 < U\ b < U$ **and** *Ha*: $0 < a \wedge a < 1$
shows $\|F\ a\| \le 1\ /\ pi * a\ powr\ b\ /\ (T * |ln\ a|) + a\ powr\ U * T\ /\ (pi * U)$
proof −
define *f* **where** $f \equiv \lambda s :: complex.\ a\ powr\ s\ /\ s$
note $assms' = cond\ assms\ hU$
define P_1 **where** $P_1 \equiv linepath\ (Complex\ b\ (-T))\ (Complex\ U\ (-T))$
define P_2 **where** $P_2 \equiv linepath\ (Complex\ U\ (-T))\ (Complex\ U\ T)$
define P_3 **where** $P_3 \equiv linepath\ (Complex\ U\ T)\ (Complex\ b\ T)$
define P_4 **where** $P_4 \equiv linepath\ (Complex\ b\ T)\ (Complex\ b\ (-T))$
define *P* **where** $P \equiv P_1 +\!\!+\!\!+ P_2 +\!\!+\!\!+ P_3 +\!\!+\!\!+ P_4$
define $I_1\ I_2\ I_3\ I_4$ **where**
$I_1 \equiv contour\_integral\ P_1\ f$ **and** $I_2 \equiv contour\_integral\ P_2\ f$ **and**
$I_3 \equiv contour\_integral\ P_3\ f$ **and** $I_4 \equiv contour\_integral\ P_4\ f$
define *rpath* **where** $rpath \equiv rectpath\ (Complex\ b\ (-\ T))\ (Complex\ U\ T)$
note $P\_defs = P\_def\ P_1\_def\ P_2\_def\ P_3\_def\ P_4\_def$
note $I\_defs = I_1\_def\ I_2\_def\ I_3\_def\ I_4\_def$
have $path\_image\ (rectpath\ (Complex\ b\ (-\ T))\ (Complex\ U\ T)) \subseteq cbox\ (Complex\ b\ (-\ T))\ (Complex\ U\ T)$

```
    by (intro path_image_rectpath_subset_cbox) (use assms' in auto)
  moreover have 0 ∉ cbox (Complex b (− T)) (Complex U T)
    using Hb unfolding cbox_def by (auto simp add: Basis_complex_def)
  ultimately have ((λs. a powr s / (s − 0)) has_contour_integral 0) rpath
    unfolding rpath_def
    by (intro Cauchy_theorem_convex_simple
            [where S = cbox (Complex b (− T)) (Complex U T)])
      (auto intro!: holomorphic_on_powr_right holomorphic_on_divide)
  hence (f has_contour_integral 0) rpath unfolding f_def using Ha by auto
  hence 1: (f has_contour_integral 0) P unfolding rpath_def P_defs rectpath_def Let_def by simp
  hence f contour_integrable_on P by (intro has_contour_integral_integrable) (use 1 in auto)
  hence 2: f contour_integrable_on P1 f contour_integrable_on P2
        f contour_integrable_on P3 f contour_integrable_on P4 unfolding P_defs by auto
  from 1 have I1 + I2 + I3 + I4 = 0 unfolding P_defs I_defs by (rule has_chain_integral_chain_integral4)
  hence I4 = − (I1 + I2 + I3) by (metis neg_eq_iff_add_eq_0)
  hence ‖I4‖ = ‖− (I1 + I2 + I3)‖ by auto
  also have ... = ‖I1 + I2 + I3‖ by (rule norm_minus_cancel)
  also have ... ≤ ‖I1 + I2‖ + ‖I3‖ by (rule norm_triangle_ineq)
  also have ... ≤ ‖I1‖ + ‖I2‖ + ‖I3‖ using norm_triangle_ineq by auto
  finally have ‖I4‖ ≤ ‖I1‖ + ‖I2‖ + ‖I3‖ .
  hence I4_val: ‖I4 / (2 ∗ pi ∗ i)‖ ≤ 1 / (2∗pi) ∗ (‖I1‖ + ‖I2‖ + ‖I3‖)
    by (auto simp add: norm_divide norm_mult field_simps)
  define Q where Q t ≡ linepath (Complex b t) (Complex U t) for t
  define g where g t ≡ contour_integral (Q t) f for t
  have Q_1: (f has_contour_integral I1) (Q (−T))
    using 2(1) unfolding P1_def I1_def Q_def
    by (rule has_contour_integral_integral)
  have Q_2: (f has_contour_integral −I3) (Q T)
    using 2(3) unfolding P3_def I3_def Q_def
    by (subst contour_integral_reversepath [symmetric],
        auto intro!: has_contour_integral_integral)
      (subst contour_integrable_reversepath_eq [symmetric], auto)
  have subst_I1_I3: I1 = g (− T) I3 = − g T
    using Q_1 Q_2 unfolding g_def by (auto simp add: contour_integral_unique)
  have g_bound: ‖g t‖ ≤ a powr b / (T ∗ |ln a|)
    when Ht: |t| = T for t
  proof −
    have (f has_contour_integral g t) (Q t) proof −
      consider t = T | t = −T using Ht by fastforce
      hence f contour_integrable_on Q t using Q_1 Q_2 by (metis has_contour_integral_integrable)
      thus ?thesis unfolding g_def by (rule has_contour_integral_integral)
    qed
    hence ((λx. a powr (x + Im (Complex b t) ∗ i) / (x + Im (Complex b t) ∗ i)) has_integral (g t))
          {Re (Complex b t) .. Re (Complex U t)}
      unfolding Q_def f_def
      by (subst has_contour_integral_linepath_same_Im_iff [symmetric])
        (use assms' in auto)
    hence ∗: ((λx. a powr (x + t ∗ i) / (x + t ∗ i)) has_integral g t) {b..U} by auto
    hence ‖g t‖ = ‖integral {b..U} (λx. a powr (x + t ∗ i) / (x + t ∗ i))‖ by (auto simp add: integral_unique)
    also have ... ≤ integral {b..U} (λx. a powr x / T)
    proof (rule integral_norm_bound_integral)
      show (λx. a powr (x + t ∗ i) / (x + t ∗ i)) integrable_on {b..U} using ∗ by auto
      have (λx. a powr x / (of_real T)) integrable_on {b..U}
        by (intro iffD2 [OF integrable_on_cdivide_iff] powr_integrable) (use assms' in auto)
      thus (λx. a powr x / T) integrable_on {b..U} by auto
```

next
fix x **assume** $x \in \{b..U\}$
have $\|a\ powr\ (x + t * \mathrm{i})\| = Re\ a\ powr\ Re\ (x + t * \mathrm{i})$ **by** (*rule norm_powr_real_powr*) (*use Ha* **in** *auto*)
also have ... $= a\ powr\ x$ **by** *auto*
finally have *1*: $\|a\ powr\ (x + t * \mathrm{i})\| = a\ powr\ x$ **.**
have $T = |Im\ (x + t * \mathrm{i})|$ **using** *Ht* **by** *auto*
also have ... $\le \|x + t * \mathrm{i}\|$ **by** (*rule abs_Im_le_cmod*)
finally have *2*: $T \le \|x + t * \mathrm{i}\|$ **.**
from *1 2* **show** $\|a\ powr\ (x + t * \mathrm{i}) / (x + t * \mathrm{i})\| \le a\ powr\ x / T$
by (*subst norm_divide*) (*rule frac_le, use hT′* **in** *auto*)
qed
also have ... $= integral\ \{b..U\}\ (\lambda x.\ a\ powr\ x) / T$ **by** *auto*
also have ... $\le a\ powr\ b / (T * |ln\ a|)$
proof −
have $integral\ \{b..U\}\ (\lambda x.\ a\ powr\ x) \le a\ powr\ b / |ln\ a|$
by (*rule powr_integral_bound_lt_1*) (*use assms′* **in** *auto*)
thus *?thesis* **using** *hT′* **by** (*auto simp add*: *field_simps*)
qed
finally show *?thesis* **.**
qed
have $\|I_2\| \le a\ powr\ U / U * \|Complex\ U\ T - Complex\ U\ (- T)\|$
proof −
have *f contour_integrable_on* P_2 **by** (*rule 2*)
moreover have $0 \le a\ powr\ U / U$ **using** *hU* **by** *auto*
moreover have $\|f\ z\| \le a\ powr\ U / U$
when ∗: $z \in closed\_segment\ (Complex\ U\ (- T))\ (Complex\ U\ T)$ **for** z
proof −
from ∗ **have** *Re_z*: $Re\ z = U$
unfolding *closed_segment_def*
by (*auto simp add*: *legacy_Complex_simps field_simps*)
hence $U = |Re\ z|$ **using** *hU* **by** *auto*
also have ... $\le \|z\|$ **by** (*rule abs_Re_le_cmod*)
finally have *zmod*: $U \le \|z\|$ **.**
have $\|f\ z\| = \|a\ powr\ z\| / \|z\|$ **unfolding** *f_def* **by** (*rule norm_divide*)
also have ... $\le a\ powr\ U / U$
by (*subst norm_powr_real_powr, use Ha* **in** *auto*)
(*rule frac_le, use hU Re_z zmod* **in** *auto*)
finally show *?thesis* **.**
qed
ultimately show *?thesis* **unfolding** I_2*_def* P_2*_def* **by** (*rule contour_integral_bound_linepath*)
qed
also have ... $\le a\ powr\ U / U * (2 * T)$
proof −
have $sqrt\ ((2 * T)^2) = |2 * T|$ **by** (*rule real_sqrt_abs*)
thus *?thesis* **using** *hT′* **by** (*simp add*: *field_simps legacy_Complex_simps*)
qed
finally have I_2*_bound*: $\|I_2\| \le a\ powr\ U / U * (2 * T)$ **.**
have $\|I_4 / (2 * pi * \mathrm{i})\| \le 1 / (2*pi) * (\|g\ (- T)\| + \|I_2\| + \|- g\ T\|)$
using I_4*_val subst_*I_1*_*I_3 **by** *auto*
also have ... $\le 1 / (2*pi) * (2 * a\ powr\ b / (T * |ln\ a|) + a\ powr\ U / U * (2*T))$
proof −
have $\|g\ T\| \le a\ powr\ b / (T * |ln\ a|)$
$\|g\ (- T)\| \le a\ powr\ b / (T * |ln\ a|)$
using *hT′* **by** (*auto intro*: *g_bound*)

hence ‖g (− T)‖ + ‖− g T‖ + ‖I_2‖ ≤ 2 ∗ a powr b / (T ∗ |ln a|) + a powr U / U ∗ (2∗T)
using I_2_bound **by** auto
thus ?thesis **by** (auto simp add: field_simps)
qed
also have ... = 1 / pi ∗ a powr b / (T ∗ |ln a|) + a powr U ∗ T / (pi ∗ U)
using hT′ **by** (auto simp add: field_simps)
finally have ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral (reversepath P_4) f‖
≤ 1 / pi ∗ a powr b / (T ∗ |ln a|) + a powr U ∗ T / (pi ∗ U)
unfolding I_4_def P_4_def **by** (subst contour_integral_reversepath) auto
thus ?thesis **using** Ha **unfolding** I_4_def P_4_def f_def F_def path_def **by** auto
qed

lemma perron_aux_2:
assumes Ha: 0 < a ∧ a < 1
shows ‖F a‖ ≤ 1 / pi ∗ a powr b / (T ∗ |ln a|) (**is** _ ≤ ?x)
proof −
let ?y = λU :: real. a powr U ∗ T / (pi ∗ U)
have ((λU :: real. ?x) ⟶ ?x) at_top **by** auto
moreover have ((λU. ?y U) ⟶ 0) at_top **using** Ha **by** real_asymp
ultimately have ((λU. ?x + ?y U) ⟶ ?x + 0) at_top **by** (rule tendsto_add)
hence ((λU. ?x + ?y U) ⟶ ?x) at_top **by** auto
moreover have ‖F a‖ ≤ ?x + ?y U **when** hU: 0 < U b < U **for** U
by (subst perron_aux_2′ [OF hU Ha], standard)
hence ∀$_F$ U in at_top. ‖F a‖ ≤ ?x + ?y U
by (rule eventually_at_top_linorderI′) (use Hb **in** auto)
ultimately show ?thesis
by (intro tendsto_lowerbound) auto
qed

lemma perron_aux_3:
assumes Ha: 0 < a
shows ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral path (λs. a powr s / s)‖ ≤ a powr b ∗ ln (1 + T / b)
proof −
have ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral (linepath (Complex b (−T)) (Complex b T)) (λs. 1 ∗ a powr s / s)‖
≤ 1 ∗ a powr b ∗ ln (1 + T / b)
by (rule perron_aux_3′) (auto intro: Ha cond perron_integrable)
thus ?thesis **unfolding** path_def **by** auto
qed

lemma perron_aux′:
assumes Ha: 0 < a
shows ‖F a‖ ≤ a powr b ∗ r a
proof −
note assms′ = assms cond
define P **where** P ≡ 1 / (2 ∗ pi ∗ i) ∗ contour_integral path (λs. a powr s / s)
have lm_1: 1 + ln (1 + T / b) ≤ 2 + ln (T / b)
proof −
have 1 ≤ T / b **using** hT Hb **by** auto
hence 1 + T / b ≤ 2 ∗ (T / b) **by** auto
hence ln (1 + T / b) ≤ ln 2 + ln (T / b)
by (subst ln_mult_pos [symmetric]) auto
thus ?thesis **using** ln_2_less_1 **by** auto
qed
have ∗: ‖F a‖ ≤ a powr b ∗ (2 + ln (T / b))

```
  proof (cases 1 ≤ a)
    assume Ha': 1 ≤ a
    have ‖P − 1‖ ≤ ‖P‖ + 1 by (simp add: norm_triangle_le_diff)
    also have ... ≤ a powr b ∗ ln (1 + T / b) + 1
    proof −
      have ‖P‖ ≤ a powr b ∗ ln (1 + T / b)
        unfolding P_def by (intro perron_aux_3 assms')
      thus ?thesis by auto
    qed
    also have ... ≤ a powr b ∗ (2 + ln (T / b))
    proof −
      have 1 = a powr 0 using Ha' by auto
      also have a powr 0 ≤ a powr b using Ha' Hb by (intro powr_mono) auto
      finally have a powr b ∗ ln (1 + T / b) + 1 ≤ a powr b ∗ (1 + ln (1 + T / b))
        by (auto simp add: algebra_simps)
      also have ... ≤ a powr b ∗ (2 + ln (T / b)) using Ha' lm_1 by auto
      finally show ?thesis .
    qed
    finally show ?thesis using Ha' unfolding F_def P_def by auto
  next
    assume Ha': ¬ 1 ≤ a
    hence ‖P‖ ≤ a powr b ∗ ln (1 + T / b)
      unfolding P_def by (intro perron_aux_3 assms')
    also have ... ≤ a powr b ∗ (2 + ln (T / b))
      by (rule mult_left_mono) (use lm_1 in auto)
    finally show ?thesis using Ha' unfolding F_def P_def by auto
  qed
  consider 0 < a ∧ a ≠ 1 | a = 1 using Ha by linarith
  thus ?thesis proof cases
    define c where c = 1 / 2 ∗ a powr b / (T ∗ |ln a|)
    assume Ha': 0 < a ∧ a ≠ 1
    hence (0 < a ∧ a < 1) ∨ a > 1 by auto
    hence ‖F a‖ ≤ 1 / pi ∗ a powr b / (T ∗ |ln a|)
      using perron_aux_1 perron_aux_2 by auto
    also have ... ≤ c unfolding c_def
      using Ha' hT' pi_gt3 by (auto simp add: field_simps)
    finally have ‖F a‖ ≤ c .
    hence ‖F a‖ ≤ min c (a powr b ∗ (2 + ln (T / b))) using ∗ by auto
    also have ... = a powr b ∗ r a
      unfolding r_def c_def using Ha' by auto (subst min_mult_distrib_left, auto)
    finally show ?thesis using Ha' unfolding P_def by auto
  next
    assume Ha': a = 1
    with ∗ show ?thesis unfolding r_def by auto
  qed
qed

lemma r_bound:
  assumes Hn: 1 ≤ n
  shows r (x / n) ≤ H / T + (if n ∈ region then 2 + ln (T / b) else 0)
proof (cases n ∈ region)
  assume ∗: n ∉ region
  then consider n < x − x / H | x + x / H < n unfolding region_def by auto
  hence 1 / |ln (x / n)| ≤ 2 ∗ H
  proof cases
```

```
  have hH': 1 / (1 − 1 / H) > 1 using hH by auto
  case 1 hence x / n > x / (x − x / H)
    using Hx hH Hn by (intro divide_strict_left_mono) auto
  also have x / (x − x / H) = 1 / (1 − 1 / H)
    using Hx hH by (auto simp add: field_simps)
  finally have xn: x / n > 1 / (1 − 1 / H) .
  moreover have xn': x / n > 1 using xn hH' by linarith
  ultimately have |ln (x / n)| > ln (1 / (1 − 1 / H))
    using hH Hx Hn by auto
  hence 1 / |ln (x / n)| < 1 / ln (1 / (1 − 1 / H))
    using xn' hH' by (intro divide_strict_left_mono mult_pos_pos ln_gt_zero) auto
  also have ... ≤ H proof −
    have ln (1 − 1 / H) ≤ − (1 / H)
      using hH by (intro ln_one_minus_pos_upper_bound) auto
    hence −1 / ln (1 − 1 / H) ≤ −1 / (− (1 / H))
      using hH by (intro divide_left_mono_neg) (auto intro: divide_neg_pos)
    also have ... = H by auto
    finally show ?thesis
      by (subst (2) inverse_eq_divide [symmetric])
         (subst ln_inverse, use hH in auto)
  qed
  finally show ?thesis using hH by auto
next
  case 2 hence x / n < x / (x + x / H)
    using Hx hH Hn by (auto intro!: divide_strict_left_mono mult_pos_pos add_pos_pos)
  also have ... = 1 / (1 + 1 / H)
  proof −
    have 0 < x + x * H using Hx hH by (auto intro: add_pos_pos)
    thus ?thesis using Hx hH by (auto simp add: field_simps)
  qed
  finally have xn: x / n < 1 / (1 + 1 / H) .
  also have hH': ... < 1 using hH by (auto simp add: field_simps)
  finally have xn': 0 < x / n ∧ x / n < 1 using Hx Hn by auto
  have 1 / |ln (x / n)| = −1 / ln (x / n)
    using xn' by (auto simp add: field_simps)
  also have ... ≤ 2 * H proof −
    have ln (x / n) < ln (1 / (1 + 1 / H))
      using xn xn' by (subst ln_less_cancel_iff) (blast, linarith)
    also have ... = − ln (1 + 1 / H)
      by (simp add: divide_inverse ln_inverse)
    also have ... ≤ − 1 / (2 * H)
    proof −
      have − ln (1 + 1 / H) = ln (inverse (1 + 1 / H))
        by (simp add: ln_inverse)
      also have ... = ln (1 − 1 / (H + 1))
        using hH by (auto simp: field_simps)
      also have ... ≤ − (1 / (H + 1))
        using hH by (auto intro: ln_one_minus_pos_upper_bound)
      also have ... ≤ − 1 / (2 * H)
        using hH by (auto simp: field_simps)
      finally show ?thesis .
    qed
    finally have −1 / ln (x / n) ≤ −1 / (−1 / (2 * H))
      by (intro divide_left_mono_neg) (insert xn' hH, auto simp add: field_simps)
    thus ?thesis by auto
```

```
    qed
    finally show ?thesis .
  qed
  hence (1 / |ln (x / n)|) / (2 ∗ T) ≤ (2 ∗ H) / (2 ∗ T)
    using hT′ by (intro divide_right_mono) auto
  hence 1 / (2 ∗ T ∗ |ln (x / n)|) ≤ H / T
    by (simp add: field_simps)
  moreover have x / n ≠ 1 using ∗ hH unfolding region_def by auto
  ultimately show ?thesis unfolding r_def using ∗ by auto
next
  assume ∗: n ∈ region
  moreover have 2 + ln (T / b) ≤ H / T + (2 + ln (T / b))
    using hH hT′ by auto
  ultimately show ?thesis unfolding r_def by auto
qed

lemma perron_aux:
  assumes Hn: 0 < n
  shows ‖F′ n‖ ≤ 1 / n nat_powr b ∗ (x powr b ∗ H / T)
    + (if n ∈ region then 3 ∗ (2 + ln (T / b)) else 0) (is ?P ≤ ?Q)
proof −
  have ‖F (x / n)‖ ≤ (x / n) powr b ∗ r (x / n)
    by (rule perron_aux′) (use Hx Hn in auto)
  also have ... ≤ (x / n) powr b ∗ (H / T + (if n ∈ region then 2 + ln (T / b) else 0))
    by (intro mult_left_mono r_bound) (use Hn in auto)
  also have ... ≤ ?Q
  proof −
    have ∗: (x / n) powr b ∗ (H / T) = 1 / n nat_powr b ∗ (x powr b ∗ H / T)
      using Hx Hn by (subst powr_divide) (auto simp add: field_simps)
    moreover have (x / n) powr b ∗ (H / T + (2 + ln (T / b)))
      ≤ 1 / n nat_powr b ∗ (x powr b ∗ H / T) + 3 ∗ (2 + ln (T / b))
      when Hn′: n ∈ region
    proof −
      have (x / n) powr b ≤ 3
      proof −
        have x − x / H ≤ n using Hn′ unfolding region_def by auto
        moreover have x / H < x / 1 using hH Hx by (intro divide_strict_left_mono) auto
        ultimately have x / n ≤ x / (x − x / H)
          using Hx hH Hn by (intro divide_left_mono mult_pos_pos) auto
        also have ... = 1 + 1 / (H − 1)
          using Hx hH by (auto simp add: field_simps)
        finally have (x / n) powr b ≤ (1 + 1 / (H − 1)) powr b
          using Hx Hn Hb by (intro powr_mono2) auto
        also have ... ≤ exp (b / (H − 1))
        proof −
          have ln (1 + 1 / (H − 1)) ≤ 1 / (H − 1)
            using hH by (intro ln_add_one_self_le_self) auto
          hence b ∗ ln (1 + 1 / (H − 1)) ≤ b ∗ (1 / (H − 1))
            using Hb by (intro mult_left_mono) auto
          thus ?thesis unfolding powr_def by auto
        qed
        also have ... ≤ exp 1 using Hb hH′ by auto
        also have ... ≤ 3 by (rule exp_le)
        finally show ?thesis .
      qed
```

```
      moreover have 0 ≤ ln (T / b) using hT Hb by (auto intro!: ln_ge_zero)
      ultimately show ?thesis using hT
        by (subst ring_distribs, subst ∗, subst add_le_cancel_left)
           (intro mult_right_mono, auto intro!: add_nonneg_nonneg)
    qed
    ultimately show ?thesis by auto
  qed
  finally show ?thesis unfolding F′_def .
qed

definition a where a n ≡ fds_nth f n

lemma finite_region: finite region
  unfolding region_def by (subst nat_le_real_iff) auto

lemma zero_notin_region: 0 ∉ region
  unfolding region_def using hH Hx by (auto simp add: field_simps)

lemma path_image_conv:
  assumes s ∈ img_path
  shows conv_abscissa f < s · 1
proof −
  from assms have Re s = b
    unfolding img_path_def path_def
    by (auto simp add: closed_segment_def legacy_Complex_simps field_simps)
  thus ?thesis using Hb′ conv_le_abs_conv_abscissa [of f] by auto
qed

lemma converge_on_path:
  assumes s ∈ img_path
  shows fds_converges f s
  by (intro fds_converges path_image_conv assms)

lemma summable_on_path:
  assumes s ∈ img_path
  shows (λn. a n / n nat_powr s) subsummable {1..}
  unfolding a_def by (intro eval_fds_complex_subsum(2) converge_on_path assms)

lemma zero_notin_path:
  shows 0 ∉ closed_segment (Complex b (− T)) (Complex b T)
  using Hb unfolding img_path_def path_def
  by (auto simp add: closed_segment_def legacy_Complex_simps field_simps)

lemma perron_bound:
  ‖∑ ‘n ≥ 1. a n ∗ F′ n‖ ≤ x powr b ∗ H ∗ B / T
    + 3 ∗ (2 + ln (T / b)) ∗ (∑ n∈region. ‖a n‖)
proof −
  define M where M ≡ 3 ∗ (2 + ln (T / b))
  have sum_1: (λn. ‖a n / n nat_powr (b :: complex)‖) subsummable {1..}
    unfolding a_def
    by (fold nat_power_complex_def)
       (fastforce intro: Hb′ fds_abs_subsummable fds_abs_converges)
  hence sum_2: (λn. ‖a n‖ ∗ 1 / n nat_powr b) subsummable {1..}
  proof −
    have ‖a n / n nat_powr (b :: complex)‖ = ‖a n‖ ∗ 1 / n nat_powr b for n
```

```
      by (auto simp add: norm_divide field_simps norm_powr_real_powr')
    thus ?thesis using sum_1 by auto
  qed
  hence sum_3: (λn. ‖a n‖ * 1 / n nat_powr b * (x powr b * H / T)) subsummable {1..}
    by (rule subsummable_mult2)
  moreover have sum_4: (λn. if n ∈ region then M * ‖a n‖ else 0) subsummable {1..}
    by (intro has_subsum_summable, rule has_subsum_If_finite)
       (insert finite_region, auto)
  moreover have ‖a n * F' n‖
    ≤ ‖a n‖ * 1 / n nat_powr b * (x powr b * H / T)
    + (if n ∈ region then M * ‖a n‖ else 0) (is ?x' ≤ ?x)
    when n ∈ {1..} for n
  proof −
    have ‖a n * F' n‖ ≤ ‖a n‖ *
      (1 / n nat_powr b * (x powr b * H / T) + (if n ∈ region then M else 0))
      unfolding M_def
      by (subst norm_mult)
         (intro mult_left_mono perron_aux, use that in auto)
    also have ... = ?x by (simp add: field_simps)
    finally show ?thesis .
  qed
  ultimately have ‖∑ `n ≥ 1. a n * F' n‖
    ≤ (∑ `n ≥ 1. ‖a n‖ * 1 / n nat_powr b * (x powr b * H / T)
    + (if n ∈ region then M * ‖a n‖ else 0))
    by (intro subsum_norm_bound subsummable_add)
  also have ... ≤ x powr b * H * B / T + M * (∑n∈region. ‖a n‖)
  proof −
    have (∑ `n ≥ 1. (if n ∈ region then M * ‖a n‖ else 0))
       = (∑n ∈ region ∩ {1..}. M * ‖a n‖)
      by (intro subsumI [symmetric] has_subsum_If_finite_set finite_region)
    also have ... = M * (∑n∈region. ‖a n‖)
    proof −
      have region ∩ {1..} = region
        using zero_notin_region zero_less_iff_neq_zero by (auto intro: Suc_leI)
      thus ?thesis by (subst sum_distrib_left) (use zero_notin_region in auto)
    qed
    also have
      (∑ `n ≥ 1. ‖a n‖ * 1 / n nat_powr b * (x powr b * H / T))
      ≤ x powr b * H * B / T
      by (subst subsum_mult2, rule sum_2, insert hB hH hT', fold a_def)
         (auto simp add: field_simps, subst (1) mult.commute, auto intro: mult_right_mono)
    ultimately show ?thesis
      by (subst subsum_add [symmetric]) ((rule sum_3 sum_4)+, auto)
  qed
  finally show ?thesis unfolding M_def .
qed

lemma perron:
  (λs. eval_fds f s * x powr s / s) contour_integrable_on path
  ‖sum_upto a x − 1 / (2 * pi * i) * contour_integral path (λs. eval_fds f s * x powr s / s)‖
    ≤ x powr b * H * B / T + 3 * (2 + ln (T / b)) * (∑n∈region. ‖a n‖)
proof (goal_cases)
  define g where g s ≡ eval_fds f s * x powr s / s for s :: complex
  define h where h s n ≡ a n / n nat_powr s * (x powr s / s) for s :: complex and n :: nat
  define G where G n ≡ contour_integral path (λs. (x / n) powr s / s) for n :: nat
```

```
define H where H n ≡ 1 / (2 ∗ pi ∗ i) ∗ G n for n :: nat
have h_integrable: (λs. h s n) contour_integrable_on path when 0 < n for n
  using Hb Hx unfolding path_def h_def
  by (intro contour_integrable_continuous_linepath continuous_intros)
     (use that zero_notin_path in auto)
have contour_integral path g = contour_integral path (λs. ∑‘n ≥ 1. h s n)
proof (rule contour_integral_eq, fold img_path_def)
  fix s assume ∗: s ∈ img_path
  hence g s = (∑‘n ≥ 1. a n / n nat_powr s) ∗ (x powr s / s)
    unfolding g_def a_def
    by (subst eval_fds_complex_subsum) (auto intro!: converge_on_path)
  also have . . . = (∑‘n ≥ 1. a n / n nat_powr s ∗ (x powr s / s))
    by (intro subsum_mult2 [symmetric] summable) (intro summable_on_path ∗)
  finally show g s = (∑‘n ≥ 1. h s n) unfolding h_def .
qed
also have
  sum_1: (λn. contour_integral path (λs. h s n)) subsummable {1..}
  and . . . = (∑‘n ≥ 1. contour_integral path (λs. h s n))
proof (goal_cases)
  have ((λN. contour_integral path (λs. sum (h s) {1..N}))
    ⟶ contour_integral path (λs. subsum (h s) {1..})) at_top
  proof (rule contour_integral_uniform_limit)
    show valid_path path unfolding path_def by auto
    show sequentially ≠ bot by auto
  next
    fix t :: real
    show ‖vector_derivative path (at t)‖ ≤ sqrt (4 ∗ T²)
      unfolding path_def by (auto simp add: legacy_Complex_simps)
  next
    from path_image_conv
    have ∗: uniformly_convergent_on img_path (λN s. ∑n≤N. fds_nth f n / nat_power n s)
      by (intro uniformly_convergent_eval_fds) (unfold path_def img_path_def, auto)
    have ∗: uniformly_convergent_on img_path (λN s. ∑n = 1..N. a n / n nat_powr s)
    proof −
      have (∑n≤N. fds_nth f n / nat_power n s) = (∑n = 1..N. a n / n nat_powr s) for N s
      proof −
        have (∑n≤N. fds_nth f n / nat_power n s) = (∑n≤N. a n / n nat_powr s)
          unfolding a_def nat_power_complex_def by auto
        also have . . . = (∑n∈{..N} − {0}. a n / n nat_powr s)
          by (subst sum_diff1) auto
        also have . . . = (∑n = 1..N. a n / n nat_powr s)
        proof −
          have {..N} − {0} = {1..N} by auto
          thus ?thesis by auto
        qed
        finally show ?thesis by auto
      qed
      thus ?thesis using ∗ by auto
    qed
    hence uniform_limit img_path
      (λN s. ∑n = 1..N. a n / n nat_powr s)
      (λs. ∑‘n ≥ 1. a n / n nat_powr s) at_top
    proof −
      have uniform_limit img_path
        (λN s. ∑n = 1..N. a n / n nat_powr s)
```

(λs. *lim* (λN. ∑n = 1..N. a n / n nat_powr s)) at_top
using ∗ **by** (subst (asm) uniformly_convergent_uniform_limit_iff)
moreover have *lim* (λN. ∑n = 1..N. a n / n nat_powr s) = (∑ 'n ≥ 1. a n / n nat_powr s) **for** s
by (rule subsum_ge_limit)
ultimately show ?thesis **by** auto
qed
moreover have *bounded* ((λs. subsum (λn. a n / n nat_powr s) {1..}) ' img_path) (**is** bounded ?A)
proof −
have *bounded* (eval_fds f ' img_path)
by (intro compact_imp_bounded compact_continuous_image continuous_on_eval_fds)
(use path_image_conv img_path_def path_def **in** auto)
moreover have ... = ?A
unfolding a_def **by** (intro image_cong refl eval_fds_complex_subsum(1) converge_on_path)
ultimately show ?thesis **by** auto
qed
moreover have 0 ∉ closed_segment (Complex b (− T)) (Complex b T)
using Hb **by** (auto simp: closed_segment_def legacy_Complex_simps algebra_simps)
hence *bounded* ((λs. x powr s / s) ' img_path)
unfolding img_path_def path_def **using** Hx Hb
by (intro compact_imp_bounded compact_continuous_image continuous_intros) auto
ultimately have uniform_limit img_path
(λN s. (∑n = 1..N. a n / n nat_powr s) ∗ (x powr s / s))
(λs. (∑ 'n ≥ 1. a n / n nat_powr s) ∗ (x powr s / s)) at_top (**is** ?P)
by (intro uniform_lim_mult uniform_limit_const)
moreover have ?P = uniform_limit (path_image path)
(λN s. sum (h s) {1..N}) (λs. subsum (h s) {1..}) at_top (**is** ?P = ?Q)
unfolding h_def
by (fold img_path_def, rule uniform_limit_cong′, subst sum_distrib_right [symmetric], rule refl)
(subst subsum_mult2, intro summable_on_path, auto)
ultimately show ?Q **by** blast
next
from h_integrable
show ∀$_F$ N in at_top. (λs. sum (h s) {1..N}) contour_integrable_on path
unfolding h_def **by** (intro eventuallyI contour_integrable_sum) auto
qed
hence ∗: has_subsum (λn. contour_integral path (λs. h s n)) {1..} (contour_integral path (λs. subsum (h s) {1..}))
using h_integrable **by** (subst (asm) contour_integral_sum) (auto intro: has_subsum_ge_limit)
case 1 **from** ∗ **show** ?case **unfolding** h_def **by** (intro has_subsum_summable)
case 2 **from** ∗ **show** ?case **unfolding** h_def **by** (rule subsumI)
qed
note this(2) **also have**
sum_2: (λn. a n ∗ G n) subsummable {1..}
and ... = (∑ 'n ≥ 1. a n ∗ G n)
proof (goal_cases)
have ∗: a n ∗ G n = contour_integral path (λs. h s n) **when** Hn: n ∈ {1..} **for** n :: nat
proof −
have (λs. (x / n) powr s / s) contour_integrable_on path
unfolding path_def **by** (rule perron_integrable) (use Hn Hx hT **in** auto)
moreover have contour_integral path (λs. h s n) = contour_integral path (λs. a n ∗ ((x / n) powr s / s))
proof (intro contour_integral_cong refl)
fix s :: complex
have (x / n) powr s ∗ n powr s = ((x / n :: complex) ∗ n) powr s

```
        by (rule powr_times_real [symmetric]) (use Hn Hx in auto)
      also have ... = x powr s using Hn by auto
      finally have (x / n) powr s = x powr s / n powr s using Hn by (intro eq_divide_imp) auto
      thus h s n = a n * ((x / n) powr s / s) unfolding h_def by (auto simp add: field_simps)
    qed
    ultimately show ?thesis unfolding G_def by (subst (asm) contour_integral_lmul) auto
  qed
  case 1 show ?case by (subst subsummable_cong) (use * sum_1 in auto)
  case 2 show ?case by (intro subsum_cong * [symmetric])
qed
note this(2) finally have
  1 / (2 * pi * i) * contour_integral path g = (∑`n ≥ 1. a n * G n) * (1 / (2 * pi * i)) by auto
also have
  sum_3: (λn. a n * G n * (1 / (2 * pi * i))) subsummable {1..}
  and ... = (∑`n ≥ 1. a n * G n * (1 / (2 * pi * i)))
  by (intro subsummable_mult2 subsum_mult2 [symmetric] sum_2)+
note this(2) also have
  sum_4: (λn. a n * H n) subsummable {1..}
  and ... = (∑`n ≥ 1. a n * H n)
  unfolding H_def using sum_3 by auto
note this(2) also have
  ... − (∑`n ≥ 1. if n ≤ x then a n else 0)
   = (∑`n ≥ 1. a n * H n − (if n ≤ x then a n else 0))
  using sum_4
  by (rule subsum_minus(1), unfold subsummable_def)
     (auto simp add: if_if_eq_conj nat_le_real_iff)
moreover have (∑`n ≥ 1. if n ≤ x then a n else 0) = sum_upto a x
proof −
  have (∑`n ≥ 1. if n ≤ x then a n else 0) = (∑n :: nat|n ∈ {1..} ∧ n ≤ x. a n)
    by (intro subsumI [symmetric] has_subsum_If_finite) (auto simp add: nat_le_real_iff)
  also have ... = sum_upto a x
  proof −
    have {n :: nat. n ∈ {1..} ∧ n ≤ x} = {n. 0 < n ∧ n ≤ x} by auto
    thus ?thesis unfolding sum_upto_def by auto
  qed
  finally show ?thesis .
qed
moreover have (∑`n ≥ 1. a n * H n − (if n ≤ x then a n else 0)) = (∑`n ≥ 1. a n * F′ n)
  unfolding F_def F′_def G_def H_def by (rule subsum_cong) (auto simp add: algebra_simps)
ultimately have result: ‖sum_upto a x − 1 / (2 * pi * i) * contour_integral path g‖ = ‖∑`n ≥ 1. a
n * F′ n‖
  by (subst norm_minus_commute) auto
case 1 show ?case
proof −
  have closed_segment (Complex b (− T)) (Complex b T) ⊆ {s. conv_abscissa f < ereal (s · 1)}
    using path_image_conv unfolding img_path_def path_def by auto
  thus ?thesis unfolding path_def
    by (intro contour_integrable_continuous_linepath continuous_intros)
       (use Hx zero_notin_path in auto)
qed
case 2 show ?case using perron_bound result unfolding g_def by linarith
qed
end

theorem perron_formula:
```

fixes b B H T x :: *real* **and** f :: *complex fds*
assumes *Hb*: $0 < b$ **and** *hT*: $b \leq T$
and *Hb'*: *abs_conv_abscissa f* $< b$
and *hH*: $1 < H$ **and** *hH'*: $b + 1 \leq H$ **and** *Hx*: $0 < x$
and *hB*: $(\sum`n \geq 1.\ \|fds\_nth\ f\ n\|\ /\ n\ nat\_powr\ b) \leq B$
shows ($\lambda s.$ *eval_fds f s* $*$ *x powr s* / *s*) *contour_integrable_on* (*linepath* (*Complex b* ($-T$)) (*Complex b T*))
$\|sum\_upto\ (fds\_nth\ f)\ x - 1\ /\ (2 * pi * \mathrm{i}) *$
contour_integral (*linepath* (*Complex b* ($-T$)) (*Complex b T*)) ($\lambda s.$ *eval_fds f s* $*$ *x powr s* / *s*)$\|$
$\leq x\ powr\ b * H * B\ /\ T + 3 * (2 + ln\ (T\ /\ b)) * (\sum n \mid x - x\ /\ H \leq n \wedge n \leq x + x\ /\ H.\ \|fds\_nth\ f\ n\|)$
proof (*goal_cases*)
interpret *z*: *perron_locale* **using** *assms* **unfolding** *perron_locale_def* **by** *auto*
case *1* **show** *?case* **using** *z.perron*(*1*) **unfolding** *z.path_def* **.**
case *2* **show** *?case* **using** *z.perron*(*2*) **unfolding** *z.path_def z.region_def z.a_def* **.**
qed

theorem *perron_asymp*:
fixes b x :: *real*
assumes *b*: $b > 0$ *ereal b* $>$ *abs_conv_abscissa f*
assumes *x*: $0 < x$ $x \notin \mathbb{N}$
defines $L \equiv (\lambda T.$ *linepath* (*Complex b* ($-T$)) (*Complex b T*))
shows (($\lambda T.$ *contour_integral* (L T) ($\lambda s.$ *eval_fds f s* $*$ *of_real x powr s* / *s*))
$\longrightarrow$ $2 * pi * \mathrm{i} *$ *sum_upto* ($\lambda n.$ *fds_nth f n*) *x*) *at_top*
proof $-$
define *R* **where** $R = (\lambda H.\ \{n.\ x - x\ /\ H \leq real\ n \wedge real\ n \leq x + x\ /\ H\})$
have *R_altdef*: $R\ H = \{n.\ dist\ (of\_nat\ n)\ x \leq x\ /\ H\}$ **for** *H*
unfolding *R_def* **by** (*intro Collect_cong*) (*auto simp*: *dist_norm*)
obtain *H* **where** *H*: $H > 1$ $H \geq b + 1$ $R\ H = (if\ x \in \mathbb{N}\ then\ \{nat\ \lfloor x \rfloor\}\ else\ \{\})$
proof (*cases* $x \in \mathbb{N}$)
case *True* **thus** *?thesis* **using** *x* **by** *auto*
next
case *False*
define *d* **where** $d = setdist\ \{x\}\ \mathbb{N}$
have $0 \in (\mathbb{N} :: real\ set)$ **by** *auto*
hence $(\mathbb{N} :: real\ set) \neq \{\}$ **by** *blast*
hence $d > 0$
unfolding *d_def* **using** *False* **by** (*subst setdist_gt_0_compact_closed*) *auto*
define *H* **where** $H = Max\ \{2,\ b + 1,\ 2 * x\ /\ d\}$
have *H*: $H \geq 2$ $H \geq b + 1$ $H \geq 2 * x\ /\ d$
unfolding *H_def* **by** (*rule Max.coboundedI*; *simp*)+

show *?thesis*
proof (*rule that*[*of H*])
have $n \notin R\ H$ **for** n :: *nat*
proof $-$
have $x\ /\ H \leq x\ /\ (2 * x\ /\ d)$
using *H x* ‹$d > 0$›
by (*intro divide_left_mono*) (*auto intro*!: *mult_pos_pos*)
also have $\ldots < d$
using *x* ‹$d > 0$› **by** *simp*
also have $d \leq dist\ (of\_nat\ n)\ x$
unfolding *d_def* **by** (*subst dist_commute*, *rule setdist_le_dist*) *auto*
finally show $n \notin R\ H$
by (*auto simp*: *R_altdef*)

```
      qed
      thus R H = (if x ∈ ℕ then {nat ⌊x⌋} else {})
        using False by auto
    qed (use H in auto)
  qed

  define g where g = (λs. eval_fds f s ∗ of_real x powr s / s)
  define I where I = (λT. contour_integral (L T) g)
  define c where c = 2 ∗ pi ∗ i
  define A where A = sum_upto (fds_nth f)
  define B where B = subsum (λn. norm (fds_nth f n) / n nat_powr b) {0+..}
  define X where X = (if x ∈ ℤ then {nat ⌊x⌋} else {})

  have norm_le: norm (A x − I T / c) ≤ x powr b ∗ H ∗ B / T if T: T ≥ b for T
  proof −
    interpret perron_locale b B H T x f
      by standard (use b T x H(1,2) in ‹auto simp: B_def›)
    from perron
    have norm (A x − I T / c) ≤ x powr b ∗ H ∗ B / T
        + 3 ∗ (∑ n∈R H. norm (fds_nth f n)) ∗ (2 + ln (T / b))
      by (simp add: I_def A_def g_def a_def local.path_def L_def c_def R_def
                    region_def algebra_simps)
    also have (∑ n∈R H. norm (fds_nth f n)) = 0
      using x H by auto
    finally show norm (A x − I T / c) ≤ x powr b ∗ H ∗ B / T
      by simp
  qed
  have eventually (λT. norm (A x − I T / c) ≤ x powr b ∗ H ∗ B / T) at_top
    using eventually_ge_at_top[of b] by eventually_elim (use norm_le in auto)
  moreover have ((λT. x powr b ∗ H ∗ B / T) ⟶ 0) at_top
    by real_asymp
  ultimately have lim: ((λT. A x − I T / c) ⟶ 0) at_top
    using Lim_null_comparison by fast
  have ((λT. −c ∗ (A x − I T / c) + c ∗ A x) ⟶ −c ∗ 0 + c ∗ A x) at_top
    by (rule tendsto_intros lim)+
  also have (λT. −c ∗ (A x − I T / c) + c ∗ A x) = I
    by (simp add: algebra_simps c_def)
  finally show ?thesis
    by (simp add: c_def A_def I_def g_def)
qed

unbundle no pnt_syntax

end
theory PNT_with_Remainder
imports
  Relation_of_PNTs
  Zeta_Zerofree
  Perron_Formula
begin
unbundle pnt_syntax
```

6 Estimation of the order of $\frac{\zeta'(s)}{\zeta(s)}$

```
notation primes_psi (‹ψ›)
```

```
lemma zeta_div_bound':
  assumes 1 + exp (− 4 ∗ ln (14 + 4 ∗ t)) ≤ σ
    and 13 / 22 ≤ t
    and z ∈ cball (Complex σ t) (1 / 2)
  shows ‖zeta z / zeta (Complex σ t)‖ ≤ exp (12 ∗ ln (14 + 4 ∗ t))
proof −
  interpret z: zeta_bound_param_2
    λt. 1 / 2 λt. 4 ∗ ln (12 + 2 ∗ max 0 t) t σ t
    unfolding zeta_bound_param_1_def zeta_bound_param_2_def
              zeta_bound_param_1_axioms_def zeta_bound_param_2_axioms_def
    using assms by (auto intro: classical_zeta_bound.zeta_bound_param_axioms)
  show ?thesis using z.zeta_div_bound assms(2) assms(3)
    unfolding z.s_def z.r_def by auto
qed

lemma zeta_div_bound:
  assumes 1 + exp (− 4 ∗ ln (14 + 4 ∗ |t|)) ≤ σ
    and 13 / 22 ≤ |t|
    and z ∈ cball (Complex σ t) (1 / 2)
  shows ‖zeta z / zeta (Complex σ t)‖ ≤ exp (12 ∗ ln (14 + 4 ∗ |t|))
proof (cases 0 ≤ t)
  case True with assms(2) have 13 / 22 ≤ t by auto
  thus ?thesis using assms by (auto intro: zeta_div_bound')
next
  case False with assms(2) have Ht: t ≤ − 13 / 22 by auto
  moreover have 1: Complex σ (− t) = cnj (Complex σ t) by (auto simp add: legacy_Complex_simps)
  ultimately have ‖zeta (cnj z) / zeta (Complex σ (− t))‖ ≤ exp (12 ∗ ln (14 + 4 ∗ (− t)))
    using assms(1) assms(3)
    by (intro zeta_div_bound', auto simp add: dist_complex_def)
       (subst complex_cnj_diff [symmetric], subst complex_mod_cnj)
  thus ?thesis using Ht by (subst (asm) 1) (simp add: norm_divide)
qed

definition C₂ where C₂ ≡ 319979520 :: real

lemma C₂_gt_zero: 0 < C₂ unfolding C₂_def by auto

lemma logderiv_zeta_order_estimate':
∀F t in (abs going_to at_top).
  ∀σ. 1 − 1 / 7 ∗ C₁ / ln (|t| + 3) ≤ σ
  ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C₂ ∗ (ln (|t| + 3))²
proof −
  define F where F :: real filter ≡ abs going_to at_top
  define r where r t ≡ C₁ / ln (|t| + 3) for t :: real
  define s where s σ t ≡ Complex (σ + 2 / 7 ∗ r t) t for σ t
  have r_nonneg: 0 ≤ r t for t unfolding PNT_const_C₁_def r_def by auto
  have ‖logderiv zeta (Complex σ t)‖ ≤ C₂ ∗ (ln (|t| + 3))²
    when h: 1 − 1 / 7 ∗ r t ≤ σ
           exp (− 4 ∗ ln (14 + 4 ∗ |t|)) ≤ 1 / 7 ∗ r t
           8 / 7 ∗ r t ≤ |t|
           8 / 7 ∗ r t ≤ 1 / 2
           13 / 22 ≤ |t| for σ t
  proof −
    have ‖logderiv zeta (Complex σ t)‖ ≤ 8 ∗ (12 ∗ ln (14 + 4 ∗ |t|)) / (8 / 7 ∗ r t)
```

```
proof (rule lemma_3_9_beta1' [where ?s = s σ t], goal_cases)
  case 1 show ?case unfolding PNT_const_C₁_def r_def by auto
  case 2 show ?case by auto
  have notin_ball: 1 ∉ ball (s σ t) (8 / 7 ∗ r t)
  proof −
    note h(3)
    also have |t| = |Im (Complex (σ + 2 / 7 ∗ r t) t − 1)| by auto
    also have ... ≤ ‖Complex (σ + 2 / 7 ∗ r t) t − 1‖ by (rule abs_Im_le_cmod)
    finally show ?thesis
      unfolding s_def by (auto simp add: dist_complex_def)
  qed
  case 3 show ?case by (intro holomorphic_zeta notin_ball)
  case 6 show ?case
    using r_nonneg unfolding s_def
    by (auto simp add: dist_complex_def legacy_Complex_simps)
  fix z assume Hz: z ∈ ball (s σ t) (8 / 7 ∗ r t)
  show zeta z ≠ 0
  proof (rule ccontr)
    assume ¬ zeta z ≠ 0
    hence zero: zeta (Complex (Re z) (Im z)) = 0 by auto
    have r t ≤ C₁ / ln (|Im z| + 2)
    proof −
      have ‖s σ t − z‖ < 1
        using Hz h(4) by (auto simp add: dist_complex_def)
      hence |t − Im z| < 1
        using abs_Im_le_cmod [of s σ t − z]
        unfolding s_def by (auto simp add: legacy_Complex_simps)
      hence |Im z| < |t| + 1 by auto
      thus ?thesis unfolding r_def
        by (intro divide_left_mono mult_pos_pos)
           (subst ln_le_cancel_iff, use C₁_gt_zero in auto)
    qed
    also have ... ≤ 1 − Re z
      using notin_ball Hz by (intro zeta_nonzero_region zero) auto
    also have ... < 1 − Re (s σ t) + 8 / 7 ∗ r t
    proof −
      have Re (s σ t − z) ≤ |Re (s σ t − z)| by auto
      also have ... < 8 / 7 ∗ r t
        using Hz abs_Re_le_cmod [of s σ t − z]
        by (auto simp add: dist_complex_def)
      ultimately show ?thesis by auto
    qed
    also have ... = 1 − σ + 6 / 7 ∗ r t unfolding s_def by auto
    also have ... ≤ r t using h(1) by auto
    finally show False by auto
  qed
  from Hz have z ∈ cball (s σ t) (1 / 2)
    using h(4) by auto
  thus ‖zeta z / zeta (s σ t)‖ ≤ exp (12 ∗ ln (14 + 4 ∗ |t|))
    using h(1) h(2) unfolding s_def
    by (intro zeta_div_bound h(5)) auto
qed
also have ... = 84 / r t ∗ ln (14 + 4 ∗ |t|)
  by (auto simp add: field_simps)
also have ... ≤ 336 / C₁ ∗ ln (|t| + 2) ∗ ln (|t| + 3)
```

```
    proof -
      have 84 / r t * ln (14 + 4 * |t|) ≤ 84 / r t * (4 * ln (|t| + 2))
        using r_nonneg by (intro mult_left_mono mult_right_mono ln_bound_1) auto
      thus ?thesis unfolding r_def by (simp add: mult_ac)
    qed
    also have ... ≤ 336 / C1 * (ln (|t| + 3))^2
      unfolding power2_eq_square
      by (simp add: mult_ac, intro divide_right_mono mult_right_mono)
         (subst ln_le_cancel_iff, use C1_gt_zero in auto)
    also have ... = C2 * (ln (|t| + 3))^2
      unfolding PNT_const_C1_def C2_def by auto
    finally show ?thesis .
  qed
  hence
    ∀F t in F.
        exp (- 4 * ln (14 + 4 * |t|)) ≤ 1 / 7 * r t
    ⟶ 8 / 7 * r t ≤ |t|
    ⟶ 8 / 7 * r t ≤ 1 / 2
    ⟶ 13 / 22 ≤ |t|
    ⟶ (∀σ. 1 - 1 / 7 * r t ≤ σ
      ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C2 * (ln (|t| + 3))^2)
    by (blast intro: eventuallyI)
  moreover have ∀F t in F. exp (- 4 * ln (14 + 4 * |t|)) ≤ 1 / 7 * r t
    unfolding F_def r_def PNT_const_C1_def
    by (rule eventually_going_toI) real_asymp
  moreover have ∀F t in F. 8 / 7 * r t ≤ |t|
    unfolding F_def r_def PNT_const_C1_def
    by (rule eventually_going_toI) real_asymp
  moreover have ∀F t in F. 8 / 7 * r t ≤ 1 / 2
    unfolding F_def r_def PNT_const_C1_def
    by (rule eventually_going_toI) real_asymp
  moreover have ∀F t in F. 13 / 22 ≤ |t|
    unfolding F_def by (rule eventually_going_toI) real_asymp
  ultimately have
    ∀F t in F. (∀σ. 1 - 1 / 7 * r t ≤ σ
      ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C2 * (ln (|t| + 3))^2)
    by eventually_elim blast
  thus ?thesis unfolding F_def r_def by auto
qed

definition C3 where
C3 ≡ SOME T. 0 < T ∧
  (∀t. T ≤ |t| ⟶
    (∀σ. 1 - 1 / 7 * C1 / ln (|t| + 3) ≤ σ
     ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C2 * (ln (|t| + 3))^2))

lemma C3_prop:
  0 < C3 ∧
  (∀t. C3 ≤ |t| ⟶
    (∀σ. 1 - 1 / 7 * C1 / ln (|t| + 3) ≤ σ
    ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C2 * (ln (|t| + 3))^2))
proof -
  obtain T' where hT:
  ⋀t. T' ≤ |t| ⟹
    (∀σ. 1 - 1 / 7 * C1 / ln (|t| + 3) ≤ σ
```

```
      ⟶ ‖logderiv zeta (Complex σ t)‖ ≤ C_2 * (ln (|t| + 3))^2)
    using logderiv_zeta_order_estimate'
      [unfolded going_to_def, THEN rev_iffD1,
      OF eventually_filtercomap_at_top_linorder] by blast
  define T where T ≡ max 1 T'
  show ?thesis unfolding C_3_def
    by (rule someI [of _ T]) (unfold T_def, use hT in auto)
qed

lemma C_3_gt_zero: 0 < C_3 using C_3_prop by blast

lemma logderiv_zeta_order_estimate:
  assumes 1 - 1 / 7 * C_1 / ln (|t| + 3) ≤ σ C_3 ≤ |t|
  shows ‖logderiv zeta (Complex σ t)‖ ≤ C_2 * (ln (|t| + 3))^2
  using assms C_3_prop by blast

definition zeta_zerofree_region
  where zeta_zerofree_region ≡ {s. s ≠ 1 ∧ 1 - C_1 / ln (|Im s| + 2) < Re s}
definition logderiv_zeta_region
  where logderiv_zeta_region ≡ {s. C_3 ≤ |Im s| ∧ 1 - 1 / 7 * C_1 / ln (|Im s| + 3) ≤ Re s}
definition zeta_strip_region
  where zeta_strip_region σ T ≡ {s. s ≠ 1 ∧ σ ≤ Re s ∧ |Im s| ≤ T}
definition zeta_strip_region'
  where zeta_strip_region' σ T ≡ {s. s ≠ 1 ∧ σ ≤ Re s ∧ C_3 ≤ |Im s| ∧ |Im s| ≤ T}

lemma strip_in_zerofree_region:
  assumes 1 - C_1 / ln (T + 2) < σ
  shows zeta_strip_region σ T ⊆ zeta_zerofree_region
proof
  fix s assume Hs: s ∈ zeta_strip_region σ T
  hence Hs': s ≠ 1 σ ≤ Re s |Im s| ≤ T unfolding zeta_strip_region_def by auto
  from this(3) have C_1 / ln (T + 2) ≤ C_1 / ln (|Im s| + 2)
    using C_1_gt_zero by (intro divide_left_mono mult_pos_pos) auto
  thus s ∈ zeta_zerofree_region using Hs' assms unfolding zeta_zerofree_region_def by auto
qed

lemma strip_in_logderiv_zeta_region:
  assumes 1 - 1 / 7 * C_1 / ln (T + 3) ≤ σ
  shows zeta_strip_region' σ T ⊆ logderiv_zeta_region
proof
  fix s assume Hs: s ∈ zeta_strip_region' σ T
  hence Hs': s ≠ 1 σ ≤ Re s C_3 ≤ |Im s| |Im s| ≤ T unfolding zeta_strip_region'_def by auto
  from this(4) have C_1 / (7 * ln (T + 3)) ≤ C_1 / (7 * ln (|Im s| + 3))
    using C_1_gt_zero by (intro divide_left_mono mult_pos_pos) auto
  thus s ∈ logderiv_zeta_region using Hs' assms unfolding logderiv_zeta_region_def by auto
qed

lemma strip_condition_imp:
  assumes 0 ≤ T 1 - 1 / 7 * C_1 / ln (T + 3) ≤ σ
  shows 1 - C_1 / ln (T + 2) < σ
proof -
  have ln (T + 2) ≤ 7 * ln (T + 2) using assms(1) by auto
  also have ... < 7 * ln (T + 3) using assms(1) by auto
  finally have C_1 / (7 * ln (T + 3)) < C_1 / ln (T + 2)
    using C_1_gt_zero assms(1) by (intro divide_strict_left_mono mult_pos_pos) auto
```

```
  thus ?thesis using assms(2) by auto
qed

lemma zeta_zerofree_region:
  assumes s ∈ zeta_zerofree_region
  shows zeta s ≠ 0
  using zeta_nonzero_region [of Re s Im s] assms
  unfolding zeta_zerofree_region_def by auto

lemma logderiv_zeta_region_estimate:
  assumes s ∈ logderiv_zeta_region
  shows ‖logderiv zeta s‖ ≤ C_2 * (ln (|Im s| + 3))^2
  using logderiv_zeta_order_estimate [of Im s Re s] assms
  unfolding logderiv_zeta_region_def by auto

definition C_4 :: real where C_4 ≡ 1 / 6666241

lemma C_4_prop:
  ∀_F x in at_top. C_4 / ln x ≤ C_1 / (7 * ln (x + 3))
  unfolding PNT_const_C_1_def C_4_def by real_asymp

lemma C_4_gt_zero: 0 < C_4 unfolding C_4_def by auto

definition C_5_prop where
C_5_prop C_5 ≡
  0 < C_5 ∧ (∀_F x in at_top. (∀t. |t| ≤ x
    ⟶ ‖logderiv zeta (Complex (1 − C_4 / ln x) t)‖ ≤ C_5 * (ln x)^2))

lemma logderiv_zeta_bound_vertical':
  ∃C_5. C_5_prop C_5
proof −
  define K where K ≡ cbox (Complex 0 (−C_3)) (Complex 2 C_3)
  define Γ where Γ ≡ {s ∈ K. zeta' s = 0}
  have zeta' not_zero_on K
    unfolding not_zero_on_def K_def using C_3_gt_zero
    by (intro bexI [where x = 2])
       (auto simp add: zeta_eq_zero_iff_zeta' zeta_2 in_cbox_complex_iff)
  hence fin: finite Γ
    unfolding Γ_def K_def
    by (auto intro!: convex_connected analytic_compact_finite_zeros zeta'_analytic)
  define α where α ≡ if Γ = {} then 0 else (1 + Max (Re ` Γ)) / 2
  define K' where K' ≡ cbox (Complex α (−C_3)) (Complex 1 C_3)
  have Hα: α ∈ {0..<1}
  proof (cases Γ = {})
    case True thus ?thesis unfolding α_def by auto
  next
    case False hence hΓ: Γ ≠ {} .
    moreover have Re a < 1 if Ha: a ∈ Γ for a
    proof (rule ccontr)
      assume ¬ Re a < 1 hence 1 ≤ Re a by auto
      hence zeta' a ≠ 0 by (subst zeta'_eq_zero_iff) (use zeta_Re_ge_1_nonzero in auto)
      thus False using Ha unfolding Γ_def by auto
    qed
    moreover have ∃a∈Γ. 0 ≤ Re a
    proof −
```

from $h\Gamma$ **have** $\exists\, a.\ a \in \Gamma$ **by** *auto*
moreover have $\bigwedge a.\ a \in \Gamma \Longrightarrow 0 \le Re\ a$
unfolding Γ_*def* *K*_*def* **by** (*auto simp add*: *in*_*cbox*_*complex*_*iff*)
ultimately show *?thesis* **by** *auto*
qed
ultimately have $0 \le Max\ (Re\ `\ \Gamma)\ Max\ (Re\ `\ \Gamma) < 1$
using *fin* **by** (*auto simp add*: *Max*_*ge*_*iff*)
thus *?thesis* **unfolding** α_*def* **using** $h\Gamma$ **by** *auto*
qed
have *nonzero*: $zeta'\ z \neq 0$ **when** $z \in K'$ **for** *z*
proof (*rule ccontr*)
assume $\neg\ zeta'\ z \neq 0$
moreover have $K' \subseteq K$ **unfolding** K'_*def* *K*_*def*
by (*rule subset*_*box*_*imp*) (*insert* $H\alpha$, *simp add*: *Basis*_*complex*_*def*)
ultimately have *Hz*: $z \in \Gamma$ **unfolding** Γ_*def* **using** *that* **by** *auto*
hence $Re\ z \le Max\ (Re\ `\ \Gamma)$ **using** *fin* **by** (*intro Max*_*ge*) *auto*
also have $\ldots < \alpha$
proof −
from *Hz* **have** $\Gamma \neq \{\}$ **by** *auto*
thus *?thesis* **using** $H\alpha$ **unfolding** α_*def* **by** *auto*
qed
finally have $Re\ z < \alpha$ **.**
moreover from ‹$z \in K'$› **have** $\alpha \le Re\ z$
unfolding K'_*def* **by** (*simp add*: *in*_*cbox*_*complex*_*iff*)
ultimately show *False* **by** *auto*
qed
hence *logderiv* $zeta'$ *analytic*_*on* K' **by** (*intro analytic*_*intros*)
moreover have *compact* K' **unfolding** K'_*def* **by** *auto*
ultimately have *bounded* ((*logderiv* $zeta'$) ` K')
by (*intro analytic*_*imp*_*holomorphic holomorphic*_*on*_*imp*_*continuous*_*on*
*compact*_*imp*_*bounded compact*_*continuous*_*image*)
from *this* [*THEN rev*_*iffD1*, *OF bounded*_*pos*]
obtain *M* **where**
hM: $\bigwedge s.\ s \in K' \Longrightarrow \|logderiv\ zeta'\ s\| \le M$ **by** *auto*
have $(\lambda t.\ C_2 * (ln\ (t + 3))^2) \in O(\lambda x.\ (ln\ x)^2)$ **using** C_2_*gt*_*zero* **by** *real*_*asymp*
then obtain γ **where**
$H\gamma$: $\forall_F\ x\ in\ at\_top.\ \|C_2 * (ln\ (x + 3))^2\| \le \gamma * \|(ln\ x)^2\|$
unfolding *bigo*_*def* **by** *auto*
define C_5 **where** $C_5 \equiv max\ 1\ \gamma$
have C_5_*gt*_*zero*: $0 < C_5$ **unfolding** C_5_*def* **by** *auto*
have $\forall_F\ x\ in\ at\_top.\ \gamma * (ln\ x)^2 \le C_5 * (ln\ x)^2$
by (*intro eventuallyI mult*_*right*_*mono*) (*unfold* C_5_*def*, *auto*)
with $H\gamma$ **have** hC_5: $\forall_F\ x\ in\ at\_top.\ C_2 * (ln\ (x + 3))^2 \le C_5 * (ln\ x)^2$
by *eventually*_*elim* (*use* C_2_*gt*_*zero* **in** *auto*)
have $\|logderiv\ zeta\ (Complex\ (1 - C_4\ /\ ln\ x)\ t)\| \le C_5 * (ln\ x)^2$
when *h*: $C_3 \le |t|\ |t| \le x\ 1 < x$
$C_4\ /\ ln\ x \le C_1\ /\ (7 * ln\ (x + 3))$
$C_2 * (ln\ (x + 3))^2 \le C_5 * (ln\ x)^2$ **for** *x t*
proof −
have $Re\ (Complex\ (1 - C_4\ /\ ln\ x)\ t) \neq Re\ 1$ **using** C_4_*gt*_*zero* *h*(*3*) **by** *auto*
hence $Complex\ (1 - C_4\ /\ ln\ x)\ t \neq 1$ **by** *metis*
hence $Complex\ (1 - C_4\ /\ ln\ x)\ t \in zeta\_strip\_region'\ (1 - C_4\ /\ ln\ x)\ x$
unfolding *zeta*_*strip*_*region*$'$_*def* **using** *h*(*1*) *h*(*2*) **by** *auto*
moreover hence $1 - 1\ /\ 7 * C_1\ /\ ln\ (x + 3) \le 1 - C_4\ /\ ln\ x$ **using** *h*(*4*) **by** *auto*
ultimately have $\|logderiv\ zeta\ (Complex\ (1 - C_4\ /\ ln\ x)\ t)\| \le C_2 * (ln\ (|Im\ (Complex\ (1 - C_4\ /$

ln x) *t*)| + *3*))^2
    using *strip_in_logderiv_zeta_region* [**where** *?σ* = *1* − C_4 / *ln x* **and** *?T* = *x*]
    by (*intro logderiv_zeta_region_estimate*) *auto*
  also have . . . ≤ C_2 ∗ (*ln* (*x* + *3*))^2
    by (*intro mult_left_mono*, *subst power2_le_iff_abs_le*)
      (*use* C_2*_gt_zero h*(*2*) *h*(*3*) **in** *auto*)
  also have . . . ≤ C_5 ∗ (*ln x*)^2 **by** (*rule h*(*5*))
  finally show *?thesis* **.**
qed
hence ∀$_F$ *x in at_top*. ∀ *t*. C_3 ≤ |*t*| ⟶ |*t*| ≤ *x*
  ⟶ *1* < *x* ⟶ C_4 / *ln x* ≤ C_1 / (*7* ∗ *ln* (*x* + *3*))
  ⟶ C_2 ∗ (*ln* (*x* + *3*))^2 ≤ C_5 ∗ (*ln x*)^2
  ⟶ ‖*logderiv zeta* (*Complex* (*1* − C_4 / *ln x*) *t*)‖ ≤ C_5 ∗ (*ln x*)^2
  by (*intro eventuallyI*) *blast*
moreover have ∀$_F$ *x in at_top*. (*1* :: *real*) < *x* **by** *auto*
ultimately have *1*: ∀$_F$ *x in at_top*. ∀ *t*. C_3 ≤ |*t*| ⟶ |*t*| ≤ *x*
  ⟶ ‖*logderiv zeta* (*Complex* (*1* − C_4 / *ln x*) *t*)‖ ≤ C_5 ∗ (*ln x*)^2
  using C_4*_prop* hC_5 **by** *eventually_elim blast*
define *f* **where** *f x* ≡ *1* − C_4 / *ln x* **for** *x*
define *g* **where** *g x t* ≡ *Complex* (*f x*) *t* **for** *x t*
let *?P* = λ*x t*. ‖*logderiv zeta* (*g x t*)‖ ≤ *M* + *ln x* / C_4
have α < *1* **using** *H*α **by** *auto*
hence ∀$_F$ *x in at_top*. α ≤ *f x* **unfolding** *f_def* **using** C_4*_gt_zero* **by** *real_asymp*
moreover have *f_lt_1*: ∀$_F$ *x in at_top*. *f x* < *1* **unfolding** *f_def* **using** C_4*_gt_zero* **by** *real_asymp*
ultimately have ∀$_F$ *x in at_top*. ∀ *t*. |*t*| ≤ C_3 ⟶ *g x t* ∈ *K′* − {*1*}
 unfolding *g_def K′_def* **by** *eventually_elim* (*auto simp add*: *in_cbox_complex_iff legacy_Complex_simps*)
moreover have ‖*logderiv zeta* (*g x t*)‖ ≤ *M* + *1* / (*1* − *f x*)
  when *h*: *g x t* ∈ *K′* − {*1*} *f x* < *1* **for** *x t*
proof −
  from *h*(*1*) **have** *ne_1*: *g x t* ≠ *1* **by** *auto*
  hence ‖*logderiv zeta* (*g x t*)‖ = ‖*logderiv zeta′* (*g x t*) − *1* / (*g x t* − *1*)‖
    using *h*(*1*) *nonzero*
    by (*subst logderiv_zeta_eq_zeta′*)
      (*auto simp add*: *zeta_eq_zero_iff_zeta′* [*symmetric*])
  also have . . . ≤ ‖*logderiv zeta′* (*g x t*)‖ + ‖*1* / (*g x t* − *1*)‖ **by** (*rule norm_triangle_ineq4*)
  also have . . . ≤ *M* + *1* / (*1* − *f x*)
  proof −
    have ‖*logderiv zeta′* (*g x t*)‖ ≤ *M* **using** *that* **by** (*auto intro*: *hM*)
    moreover have |*Re* (*g x t* − *1*)| ≤ ‖*g x t* − *1*‖ **by** (*rule abs_Re_le_cmod*)
    hence ‖*1* / (*g x t* − *1*)‖ ≤ *1* / (*1* − *f x*)
      using *ne_1 h*(*2*)
      by (*auto simp add*: *norm_divide g_def*
          intro!: *divide_left_mono mult_pos_pos*)
    ultimately show *?thesis* **by** *auto*
  qed
  finally show *?thesis* **.**
qed
hence ∀$_F$ *x in at_top*. ∀ *t*. *f x* < *1*
  ⟶ *g x t* ∈ *K′* − {*1*}
  ⟶ ‖*logderiv zeta* (*g x t*)‖ ≤ *M* + *1* / (*1* − *f x*) **by** *auto*
ultimately have ∀$_F$ *x in at_top*. ∀ *t*. |*t*| ≤ C_3 ⟶ ‖*logderiv zeta* (*g x t*)‖ ≤ *M* + *1* / (*1* − *f x*)
  using *f_lt_1* **by** *eventually_elim blast*
hence ∀$_F$ *x in at_top*. ∀ *t*. |*t*| ≤ C_3 ⟶ ‖*logderiv zeta* (*g x t*)‖ ≤ *M* + *ln x* / C_4 **unfolding** *f_def* **by** *auto*
moreover have ∀$_F$ *x in at_top*. *M* + *ln x* / C_4 ≤ C_5 ∗ (*ln x*)^2 **using** C_4*_gt_zero* C_5*_gt_zero* **by**

real_asymp
  ultimately have *2*: $\forall_F$ *x in at_top*. $\forall$ *t*. $|t| \le C_3 \longrightarrow$ ‖*logderiv zeta* (*g x t*)‖ $\le C_5 * (ln\ x)^2$ **by** *eventually_elim auto*
  show *?thesis*
  proof (*unfold* C_5*_prop_def*, *intro exI conjI*)
    show *0* $< C_5$ **by** (*rule* C_5*_gt_zero*)+
    have $\forall_F$ *x in at_top*. $\forall$ *t*. $C_3 \le |t| \lor |t| \le C_3$
      by (*rule eventuallyI*) *auto*
    with *1 2* **show** $\forall_F$ *x in at_top*. $\forall$ *t*. $|t| \le x \longrightarrow$ ‖*logderiv zeta* (*Complex* (*1* − C_4 / *ln x*) *t*)‖ $\le C_5 * (ln\ x)^2$
      unfolding *f_def g_def* **by** *eventually_elim blast*
  qed
qed

definition C_5 **where** $C_5 \equiv$ *SOME* C_5. C_5*_prop* C_5

lemma
  C_5*_gt_zero*: *0* $< C_5$ (**is** *?prop_1*) **and**
  logderiv_zeta_bound_vertical:
    $\forall_F$ *x in at_top*. $\forall$ *t*. $|t| \le x$
      $\longrightarrow$ ‖*logderiv zeta* (*Complex* (*1* − C_4 / *ln x*) *t*)‖ $\le C_5 * (ln\ x)^2$ (**is** *?prop_2*)
proof −
  have C_5*_prop* C_5 **unfolding** C_5*_def*
    by (*rule someI_ex*) (*rule logderiv_zeta_bound_vertical′*)
  thus *?prop_1 ?prop_2* **unfolding** C_5*_prop_def* **by** *auto*
qed

7 Deducing prime number theorem using Perron's formula

locale *prime_number_theorem* =
  fixes *c* ε :: *real*
  assumes *Hc*: *0* < *c* **and** *Hc′*: $c * c < 2 * C_4$ **and** *H*ε: *0* $< \varepsilon$ $2 * \varepsilon < c$
begin
notation *primes_psi* (‹ψ›)
definition *H* **where** *H x* ≡ *exp* (*c* / *2* ∗ (*ln x*) *powr* (*1* / *2*)) **for** *x* :: *real*
definition *T* **where** *T x* ≡ *exp* (*c* ∗ (*ln x*) *powr* (*1* / *2*)) **for** *x* :: *real*
definition *a* **where** *a x* ≡ *1* − C_4 / (*c* ∗ (*ln x*) *powr* (*1* / *2*)) **for** *x* :: *real*
definition *b* **where** *b x* ≡ *1* + *1* / (*ln x*) **for** *x* :: *real*
definition *B* **where** *B x* ≡ *5* / *4* ∗ *ln x* **for** *x* :: *real*
definition *f* **where** *f x s* ≡ *x powr s* / *s* ∗ *logderiv zeta s* **for** *x* :: *real* **and** *s* :: *complex*
definition *R* **where** *R x* ≡
  x powr (*b x*) ∗ *H x* ∗ *B x* / *T x* + *3* ∗ (*2* + *ln* (*T x* / *b x*))
  ∗ ($\sum$ *n* | *x* − *x* / *H x* ≤ *n* ∧ *n* ≤ *x* + *x* / *H x*. ‖*fds_nth* (*fds mangoldt_complex*) *n*‖) **for** *x* :: *real*
definition *Rc′* **where** *Rc′* ≡ *O*(λ*x*. *x* ∗ *exp* (− (*c* / *2* − ε) ∗ *ln x powr* (*1* / *2*)))
definition *Rc* **where** *Rc* ≡ *O*(λ*x*. *x* ∗ *exp* (− (*c* / *2* − *2* ∗ ε) ∗ *ln x powr* (*1* / *2*)))
definition z_1 **where** z_1 *x* ≡ *Complex* (*a x*) (− *T x*) **for** *x*
definition z_2 **where** z_2 *x* ≡ *Complex* (*b x*) (− *T x*) **for** *x*
definition z_3 **where** z_3 *x* ≡ *Complex* (*b x*) (*T x*) **for** *x*
definition z_4 **where** z_4 *x* ≡ *Complex* (*a x*) (*T x*) **for** *x*
definition *rect* **where** *rect x* ≡ *cbox* (z_1 *x*) (z_3 *x*) **for** *x*
definition *rect′* **where** *rect′ x* ≡ *rect x* − {*1*} **for** *x*
definition P_t **where** P_t *x t* ≡ *linepath* (*Complex* (*a x*) *t*) (*Complex* (*b x*) *t*) **for** *x t*
definition P_1 **where** P_1 *x* ≡ *linepath* (z_1 *x*) (z_4 *x*) **for** *x*
definition P_2 **where** P_2 *x* ≡ *linepath* (z_2 *x*) (z_3 *x*) **for** *x*
definition P_3 **where** P_3 *x* ≡ P_t *x* (− *T x*) **for** *x*

```
definition P₄ where P₄ x ≡ Pₜ x (T x) for x
definition Pᵣ where Pᵣ x ≡ rectpath (z₁ x) (z₃ x) for x

lemma Rc_eq_rem_est:
  Rc = rem_est (c / 2 − 2 ∗ ε) (1 / 2) 0
proof −
  have ∗: ∀F x :: real in at_top. 0 < ln (ln x) by real_asymp
  show ?thesis unfolding Rc_def rem_est_def
    by (rule landau_o.big.cong) (use ∗ in eventually_elim, auto)
qed

lemma residue_f:
  residue (f x) 1 = − x
proof −
  define A where A ≡ box (Complex 0 (− 1 / 2)) (Complex 2 (1 / 2))
  have hA: 0 ∉ A 1 ∈ A open A
    unfolding A_def by (auto simp add: mem_box Basis_complex_def)
  have zeta' s ≠ 0 when s ∈ A for s
  proof −
    have s ≠ 1 ⟹ zeta s ≠ 0
      using that unfolding A_def
      by (intro zeta_nonzero_small_imag)
        (auto simp add: mem_box Basis_complex_def)
    thus ?thesis by (subst zeta'_eq_zero_iff) auto
  qed
  hence h: (λs. x powr s / s ∗ logderiv zeta' s) holomorphic_on A
    by (intro holomorphic_intros) (use hA in auto)
  have h': (λs. x powr s / (s ∗ (s − 1))) holomorphic_on A − {1}
    by (auto intro!: holomorphic_intros) (use hA in auto)
  have s_ne_1: ∀F s :: complex in at 1. s ≠ 1
    by (subst eventually_at_filter) auto
  moreover have ∀F s in at 1. zeta s ≠ 0
    by (intro non_zero_neighbour_pole is_pole_zeta)
  ultimately have ∀F s in at 1. logderiv zeta s = logderiv zeta' s − 1 / (s − 1)
    by eventually_elim (rule logderiv_zeta_eq_zeta')
  moreover have
    f x s = x powr s / s ∗ logderiv zeta' s − x powr s / s / (s − 1)
    when logderiv zeta s = logderiv zeta' s − 1 / (s − 1) s ≠ 0 s ≠ 1 for s :: complex
    unfolding f_def by (subst that(1)) (insert that, auto simp add: field_simps)
  hence ∀F s :: complex in at 1. s ≠ 0 ⟶ s ≠ 1
    ⟶ logderiv zeta s = logderiv zeta' s − 1 / (s − 1)
    ⟶ f x s = x powr s / s ∗ logderiv zeta' s − x powr s / s / (s − 1)
    by (intro eventuallyI) blast
  moreover have ∀F s :: complex in at 1. s ≠ 0
    by (subst eventually_at_topological)
       (intro exI [of _ UNIV − {0}], auto)
  ultimately have ∀F s :: complex in at 1. f x s = x powr s / s ∗ logderiv zeta' s − x powr s / s / (s − 1)
    using s_ne_1 by eventually_elim blast
  hence residue (f x) 1 = residue (λs. x powr s / s ∗ logderiv zeta' s − x powr s / s / (s − 1)) 1
    by (intro residue_cong refl)
  also have ... = residue (λs. x powr s / s ∗ logderiv zeta' s) 1 − residue (λs. x powr s / s / (s − 1)) 1
    by (subst residue_diff [where ?s = A]) (use h h' hA in auto)
  also have ... = − x
  proof −
    have residue (λs. x powr s / s ∗ logderiv zeta' s) 1 = 0
```

    by (*rule residue_holo* [**where** *?s* = *A*]) (*use hA h* **in** *auto*)
  moreover have *residue* (λs. *x powr s* / *s* / (*s* − *1*)) *1* = (*x* :: *complex*) *powr 1* / *1*
    by (*rule residue_simple* [**where** *?s* = *A*]) (*use hA* **in** ‹*auto intro*!: *holomorphic_intros*›)
  ultimately show *?thesis* **by** *auto*
 qed
 finally show *?thesis* **.**
qed

lemma *rect_in_strip*:
 rect x − {*1*} ⊆ *zeta_strip_region* (*a x*) (*T x*)
 unfolding *rect_def zeta_strip_region_def* z_1*_def* z_3*_def*
 by (*auto simp add*: *in_cbox_complex_iff*)

lemma *rect_in_strip'*:
 {*s* ∈ *rect x*. C_3 ≤ |*Im s*|} ⊆ *zeta_strip_region'* (*a x*) (*T x*)
 unfolding *rect_def zeta_strip_region'_def* z_1*_def* z_3*_def*
 using C_3*_gt_zero* **by** (*auto simp add*: *in_cbox_complex_iff*)

lemma
 rect'_in_zerofree: $\forall_F$ *x in at_top*. *rect' x* ⊆ *zeta_zerofree_region* **and**
 rect_in_logderiv_zeta: $\forall_F$ *x in at_top*. {*s* ∈ *rect x*. C_3 ≤ |*Im s*|} ⊆ *logderiv_zeta_region*
proof (*goal_cases*)
 case *1* **have**
  $\forall_F$ *x in at_top*. C_4 / *ln x* ≤ C_1 / (*7* ∗ *ln* (*x* + *3*)) **by** (*rule* C_4*_prop*)
 moreover have *LIM x at_top*. *exp* (*c* ∗ (*ln x*) *powr* (*1* / *2*)) :> *at_top* **using** *Hc* **by** *real_asymp*
 ultimately have *h*:
  $\forall_F$ *x in at_top*. C_4 / *ln* (*exp* (*c* ∗ (*ln x*) *powr* (*1* / *2*)))
  ≤ C_1 / (*7* ∗ *ln* (*exp* (*c* ∗ (*ln x*) *powr* (*1* / *2*)) + *3*)) (**is** *eventually ?P _*)
  by (*rule eventually_compose_filterlim*)
 moreover have
  ?P x ⟹ *zeta_strip_region* (*a x*) (*T x*) ⊆ *zeta_zerofree_region*
  (**is** _ ⟹ *?Q*) **for** *x* **unfolding** *T_def a_def*
  by (*intro strip_in_zerofree_region strip_condition_imp*) *auto*
 hence $\forall_F$ *x in at_top*. *?P x* ⟶ *?Q x* **by** (*intro eventuallyI*) *blast*
 ultimately show *?case* **unfolding** *rect'_def* **by** *eventually_elim* (*use rect_in_strip* **in** *auto*)
 case *2* **from** *h* **have**
  ?P x ⟹ *zeta_strip_region'* (*a x*) (*T x*) ⊆ *logderiv_zeta_region*
  (**is** _ ⟹ *?Q*) **for** *x* **unfolding** *T_def a_def*
  by (*intro strip_in_logderiv_zeta_region*) *auto*
 hence $\forall_F$ *x in at_top*. *?P x* ⟶ *?Q x* **by** (*intro eventuallyI*) *blast*
 thus *?case* **using** *h* **by** *eventually_elim* (*use rect_in_strip'* **in** *auto*)
qed

lemma *zeta_nonzero_in_rect*:
 $\forall_F$ *x in at_top*. ∀ *s*. *s* ∈ *rect' x* ⟶ *zeta s* ≠ *0*
 using *rect'_in_zerofree* **by** *eventually_elim* (*use zeta_zerofree_region* **in** *auto*)

lemma *zero_notin_rect*: $\forall_F$ *x in at_top*. *0* ∉ *rect' x*
proof −
 have $\forall_F$ *x in at_top*. C_4 / (*c* ∗ (*ln x*) *powr* (*1* / *2*)) < *1*
  using *Hc* **by** *real_asymp*
 thus *?thesis*
  unfolding *rect'_def rect_def* z_1*_def* z_4*_def T_def a_def*
  by *eventually_elim* (*simp add*: *in_cbox_complex_iff*)
qed

lemma *f_analytic*:
  $\forall_F$ *x in at_top. f x analytic_on rect′ x*
  using *zeta_nonzero_in_rect zero_notin_rect* **unfolding** *f_def*
  by *eventually_elim* (*intro analytic_intros*, *auto simp*: *rect′_def*)

lemma *path_image_in_rect_1*:
  assumes $0 \le T\ x \wedge a\ x \le b\ x$
  shows *path_image* $(P_1\ x) \subseteq$ *rect x* $\wedge$ *path_image* $(P_2\ x) \subseteq$ *rect x*
  unfolding P_1*_def* P_2*_def rect_def* z_1*_def* z_2*_def* z_3*_def* z_4*_def*
  by (*simp*, *intro conjI closed_segment_subset*)
    (*insert assms*, *auto simp add*: *in_cbox_complex_iff*)

lemma *path_image_in_rect_2*:
  assumes $0 \le T\ x \wedge a\ x \le b\ x \wedge t \in \{-T\ x..T\ x\}$
  shows *path_image* $(P_t\ x\ t) \subseteq$ *rect x*
  unfolding P_t*_def rect_def* z_1*_def* z_3*_def*
  by (*simp*, *intro conjI closed_segment_subset*)
    (*insert assms*, *auto simp add*: *in_cbox_complex_iff*)

definition *path_in_rect′* **where**
path_in_rect′ x $\equiv$
  path_image $(P_1\ x) \subseteq$ *rect′ x* $\wedge$ *path_image* $(P_2\ x) \subseteq$ *rect′ x* $\wedge$
  path_image $(P_3\ x) \subseteq$ *rect′ x* $\wedge$ *path_image* $(P_4\ x) \subseteq$ *rect′ x*

lemma *path_image_in_rect′*:
  assumes $0 < T\ x \wedge a\ x < 1 \wedge 1 < b\ x$
  shows *path_in_rect′ x*
proof −
  have *path_image* $(P_1\ x) \subseteq$ *rect x* $\wedge$ *path_image* $(P_2\ x) \subseteq$ *rect x*
    by (*rule path_image_in_rect_1*) (*use assms* **in** *auto*)
  moreover have *path_image* $(P_3\ x) \subseteq$ *rect x path_image* $(P_4\ x) \subseteq$ *rect x*
    unfolding P_3*_def* P_4*_def*
    by (*intro path_image_in_rect_2*, (*use assms* **in** *auto*)[*1*])+
  moreover have
    $1 \notin$ *path_image* $(P_1\ x) \wedge 1 \notin$ *path_image* $(P_2\ x) \wedge$
     $1 \notin$ *path_image* $(P_3\ x) \wedge 1 \notin$ *path_image* $(P_4\ x)$
    unfolding P_1*_def* P_2*_def* P_3*_def* P_4*_def* P_t*_def* z_1*_def* z_2*_def* z_3*_def* z_4*_def* **using** *assms*
    by (*auto simp add*: *closed_segment_def legacy_Complex_simps field_simps*)
  ultimately show *?thesis* **unfolding** *path_in_rect′_def rect′_def* **by** *blast*
qed

lemma *asymp_1*:
  $\forall_F$ *x in at_top.* $0 < T\ x \wedge a\ x < 1 \wedge 1 < b\ x$
  unfolding *T_def a_def b_def*
  by (*intro eventually_conj*, *insert Hc* C_4*_gt_zero*) (*real_asymp*)+

lemma *f_continuous_on*:
  $\forall_F$ *x in at_top.* $\forall$ *A*$\subseteq$*rect′ x. continuous_on A* (*f x*)
  using *f_analytic*
  by (*eventually_elim*, *safe*)
    (*intro holomorphic_on_imp_continuous_on analytic_imp_holomorphic*,
     elim analytic_on_subset)

lemma *contour_integrability*:

```
  ∀_F x in at_top.
    f x contour_integrable_on P_1 x ∧ f x contour_integrable_on P_2 x ∧
    f x contour_integrable_on P_3 x ∧ f x contour_integrable_on P_4 x
proof -
  have ∀_F x in at_top. path_in_rect' x
    using asymp_1 by eventually_elim (rule path_image_in_rect')
  thus ?thesis using f_continuous_on
    unfolding P_1_def P_2_def P_3_def P_4_def P_t_def path_in_rect'_def
    by eventually_elim
       (intro conjI contour_integrable_continuous_linepath,
        fold z_1_def z_2_def z_3_def z_4_def, auto)
qed

lemma contour_integral_rectpath':
  assumes f x analytic_on (rect' x) 0 < T x ∧ a x < 1 ∧ 1 < b x
  shows contour_integral (P_r x) (f x) = - 2 * pi * i * x
proof -
  define z where z ≡ (1 + b x) / 2
  have Hz: z ∈ box (z_1 x) (z_3 x)
    unfolding z_1_def z_3_def z_def using assms(2)
    by (auto simp add: mem_box Basis_complex_def)
  have Hz': z ≠ 1 unfolding z_def using assms(2) by auto
  have connected (rect' x)
  proof -
    have box_nonempty: box (z_1 x) (z_3 x) ≠ {} using Hz by auto
    hence aff_dim (closure (box (z_1 x) (z_3 x))) = 2
      by (subst closure_aff_dim, subst aff_dim_open) auto
    thus ?thesis
      unfolding rect'_def using box_nonempty
      by (subst (asm) closure_box)
         (auto intro: connected_punctured_convex simp add: rect_def)
  qed
  moreover have Hz'': z ∈ rect' x
    unfolding rect'_def rect_def using box_subset_cbox Hz Hz' by auto
  ultimately obtain T where hT:
    f x holomorphic_on T open T rect' x ⊆ T connected T
    using analytic_on_holomorphic_connected assms(1) by (metis dual_order.refl)
  define U where U ≡ T ∪ box (z_1 x) (z_3 x)
  have one_in_box: 1 ∈ box (z_1 x) (z_3 x)
    unfolding z_1_def z_3_def z_def using assms(2) by (auto simp add: mem_box Basis_complex_def)
  have contour_integral (P_r x) (f x) = 2 * pi * i *
    (∑ s∈{1}. winding_number (P_r x) s * residue (f x) s)
  proof (rule Residue_theorem)
    show finite {1} valid_path (P_r x) pathfinish (P_r x) = pathstart (P_r x)
      unfolding P_r_def by auto
    show open U unfolding U_def using hT(2) by auto
    show connected U unfolding U_def
      by (intro connected_Un hT(4) convex_connected)
         (use Hz Hz'' hT(3) in auto)
    have f x holomorphic_on box (z_1 x) (z_3 x) - {1}
      by (rule holomorphic_on_subset, rule analytic_imp_holomorphic, rule assms(1))
         (unfold rect'_def rect_def, use box_subset_cbox in auto)
    hence f x holomorphic_on ((T - {1}) ∪ (box (z_1 x) (z_3 x) - {1}))
      by (intro holomorphic_on_Un) (use hT(1) hT(2) in auto)
    moreover have ... = U - {1} unfolding U_def by auto
```

```
    ultimately show f x holomorphic_on U − {1} by auto
    have Hz: Re (z₁ x) ≤ Re (z₃ x) Im (z₁ x) ≤ Im (z₃ x)
      unfolding z₁_def z₃_def using assms(2) by auto
    have path_image (P_r x) = rect x − box (z₁ x) (z₃ x)
      unfolding rect_def P_r_def
      by (intro path_image_rectpath_cbox_minus_box Hz)
    thus path_image (P_r x) ⊆ U − {1}
      using one_in_box hT(3) U_def unfolding rect'_def by auto
    have hU': rect x ⊆ U
      using hT(3) one_in_box unfolding U_def rect'_def by auto
    show ∀z. z ∉ U ⟶ winding_number (P_r x) z = 0
      using Hz P_r_def hU' rect_def winding_number_rectpath_outside by fastforce
  qed
  also have ... = − 2 * pi * i * x unfolding P_r_def
    by (simp add: residue_f, subst winding_number_rectpath, auto intro: one_in_box)
  finally show ?thesis .
qed

lemma contour_integral_rectpath:
  ∀_F x in at_top. contour_integral (P_r x) (f x) = − 2 * pi * i * x
  using f_analytic asymp_1 by eventually_elim (rule contour_integral_rectpath')

lemma valid_paths:
  valid_path (P₁ x) valid_path (P₂ x) valid_path (P₃ x) valid_path (P₄ x)
  unfolding P₁_def P₂_def P₃_def P₄_def P_t_def by auto

lemma integral_rectpath_split:
  assumes f x contour_integrable_on P₁ x ∧ f x contour_integrable_on P₂ x ∧
          f x contour_integrable_on P₃ x ∧ f x contour_integrable_on P₄ x
  shows contour_integral (P₃ x) (f x) + contour_integral (P₂ x) (f x)
       − contour_integral (P₄ x) (f x) − contour_integral (P₁ x) (f x) = contour_integral (P_r x) (f x)
proof −
  define Q₁ where Q₁ ≡ linepath (z₃ x) (z₄ x)
  define Q₂ where Q₂ ≡ linepath (z₄ x) (z₁ x)
  have Q_eq: Q₁ = reversepath (P₄ x) Q₂ = reversepath (P₁ x)
    unfolding Q₁_def Q₂_def P₁_def P₄_def P_t_def by (fold z₃_def z₄_def) auto
  hence contour_integral Q₁ (f x) = − contour_integral (P₄ x) (f x)
        contour_integral Q₂ (f x) = − contour_integral (P₁ x) (f x)
    by (auto intro: contour_integral_reversepath valid_paths)
  moreover have contour_integral (P₃ x +++ P₂ x +++ Q₁ +++ Q₂) (f x)
       = contour_integral (P₃ x) (f x) + contour_integral (P₂ x) (f x)
       + contour_integral Q₁ (f x) + contour_integral Q₂ (f x)
  proof −
    have 1: pathfinish (P₂ x) = pathstart (Q₁ +++ Q₂) pathfinish Q₁ = pathstart Q₂
      unfolding P₂_def Q₁_def Q₂_def by auto
    have 2: valid_path Q₁ valid_path Q₂ unfolding Q₁_def Q₂_def by auto
    have 3: f x contour_integrable_on P₁ x f x contour_integrable_on P₂ x
            f x contour_integrable_on P₃ x f x contour_integrable_on P₄ x
            f x contour_integrable_on Q₁ f x contour_integrable_on Q₂
      using assms by (auto simp add: Q_eq intro: contour_integrable_reversepath valid_paths)
    show ?thesis by (subst contour_integral_join |
      auto intro: valid_paths valid_path_join contour_integrable_joinI 1 2 3)+
  qed
  ultimately show ?thesis
    unfolding P_r_def z₁_def z₃_def rectpath_def
```

by (*simp add*: *Let_def*, *fold* P_t*_def* P_3*_def* z_1*_def* z_2*_def* z_3*_def* z_4*_def*)
(*fold* P_2*_def* Q_1*_def* Q_2*_def*, *auto*)
qed

lemma P_2*_eq*:
$\forall_F$ *x in at_top. contour_integral* (P_2 *x*) (*f x*) + *2* ∗ *pi* ∗ i ∗ *x*
= *contour_integral* (P_1 *x*) (*f x*) − *contour_integral* (P_3 *x*) (*f x*) + *contour_integral* (P_4 *x*) (*f x*)
proof −
have $\forall_F$ *x in at_top. contour_integral* (P_3 *x*) (*f x*) + *contour_integral* (P_2 *x*) (*f x*)
− *contour_integral* (P_4 *x*) (*f x*) − *contour_integral* (P_1 *x*) (*f x*) = − *2* ∗ *pi* ∗ i ∗ *x*
using *contour_integrability contour_integral_rectpath asymp_1 f_analytic*
by *eventually_elim* (*metis integral_rectpath_split*)
thus *?thesis* **by** (*auto simp add*: *field_simps*)
qed

lemma *estimation_*P_1:
(λ*x*. ‖*contour_integral* (P_1 *x*) (*f x*)‖) ∈ *Rc*
proof −
define *r* **where** *r x* ≡
C_5 ∗ (*c* ∗ (*ln x*) *powr* (*1* / *2*))^2 ∗ *x powr a x* ∗ *ln* (*1* + *T x* / *a x*) **for** *x*
note *logderiv_zeta_bound_vertical*
moreover have *LIM x at_top. T x* :> *at_top*
unfolding *T_def* **using** *Hc* **by** *real_asymp*
ultimately have $\forall_F$ *x in at_top.* ∀ *t*. |*t*| ≤ *T x*
⟶ ‖*logderiv zeta* (*Complex* (*1* − C_4 / *ln* (*T x*)) *t*)‖ ≤ C_5 ∗ (*ln* (*T x*))^2
unfolding *a_def* **by** (*rule eventually_compose_filterlim*)
hence $\forall_F$ *x in at_top.* ∀ *t*. |*t*| ≤ *T x*
⟶ ‖*logderiv zeta* (*Complex* (*a x*) *t*)‖ ≤ C_5 ∗ (*c* ∗ (*ln x*) *powr* (*1* / *2*))^2
unfolding *a_def T_def* **by** *auto*
moreover have $\forall_F$ *x in at_top.* (*f x*) *contour_integrable_on* (P_1 *x*)
using *contour_integrability* **by** *eventually_elim auto*
hence $\forall_F$ *x in at_top.* (λ*s*. *logderiv zeta s* ∗ *x powr s* / *s*) *contour_integrable_on* (P_1 *x*)
unfolding *f_def* **by** *eventually_elim* (*auto simp add*: *field_simps*)
moreover have $\forall_F$ *x* :: *real in at_top. 0* < *x* **by** *auto*
moreover have $\forall_F$ *x in at_top. 0* < *a x* **unfolding** *a_def* **using** *Hc* **by** *real_asymp*
ultimately have $\forall_F$ *x in at_top.*
‖*1* / (*2* ∗ *pi* ∗ i) ∗ *contour_integral* (P_1 *x*) (λ*s*. *logderiv zeta s* ∗ *x powr s* / *s*)‖ ≤ *r x*
unfolding *r_def* P_1*_def* z_1*_def* z_4*_def* **using** *asymp_1*
by *eventually_elim* (*rule perron_aux_3′*, *auto*)
hence $\forall_F$ *x in at_top.* ‖*1* / (*2* ∗ *pi* ∗ i) ∗ *contour_integral* (P_1 *x*) (*f x*)‖ ≤ *r x*
unfolding *f_def* **by** *eventually_elim* (*auto simp add*: *mult_ac*)
hence (λ*x*. ‖*1* / (*2* ∗ *pi* ∗ i) ∗ *contour_integral* (P_1 *x*) (*f x*)‖) ∈ *O*(*r*)
unfolding *f_def* **by** (*rule eventually_le_imp_bigo′*)
moreover have *r* ∈ *Rc*
proof −
define r_1 **where** r_1 *x* ≡ C_5 ∗ c^2 ∗ *ln x* ∗ *ln* (*1* + *T x* / *a x*) **for** *x*
define r_2 **where** r_2 *x* ≡ *exp* (*a x* ∗ *ln x*) **for** *x*
have r_1 ∈ *O*(λ*x*. (*ln x*)^2)
unfolding r_1*_def T_def a_def* **using** *Hc* C_5*_gt_zero* **by** *real_asymp*
moreover have r_2 ∈ *Rc′*
proof −
have *1*: ‖r_2 *x*‖ ≤ *x* ∗ *exp* (− (*c* / *2* − ε) ∗ (*ln x*) *powr* (*1* / *2*))
when *h*: *0* < *x 0* < *ln x* **for** *x*
proof −
have *a x* ∗ *ln x* = *ln x* + − C_4 / *c* ∗ (*ln x*) *powr* (*1* / *2*)

```
        unfolding a_def using h(2) Hc
        by (auto simp add: field_simps powr_add [symmetric] frac_eq_eq)
      hence r_2 x = exp (. . . ) unfolding r_2_def by blast
      also have . . . = x ∗ exp (− C_4 / c ∗ (ln x) powr (1 / 2))
        by (subst exp_add) (use h(1) in auto)
      also have . . . ≤ x ∗ exp (− (c / 2 − ε) ∗ (ln x) powr (1 / 2))
        by (intro mult_left_mono, subst exp_le_cancel_iff, intro mult_right_mono)
           (use Hc Hc' Hε C_4_gt_zero h in ‹auto simp: field_simps intro: add_increasing2›)
      finally show ?thesis unfolding r_2_def by auto
    qed
    have ∀_F x in at_top. ‖r_2 x‖ ≤ x ∗ exp (− (c / 2 − ε) ∗ (ln x) powr (1 / 2))
      using ln_asymp_pos x_asymp_pos by eventually_elim (rule 1)
    thus ?thesis unfolding Rc'_def by (rule eventually_le_imp_bigo)
  qed
  ultimately have (λx. r_1 x ∗ r_2 x)
    ∈ O(λx. (ln x)^2 ∗ (x ∗ exp (− (c / 2 − ε) ∗ (ln x) powr (1 / 2))))
    unfolding Rc'_def by (rule landau_o.big.mult)
  moreover have (λx. (ln x)^2 ∗ (x ∗ exp (− (c / 2 − ε) ∗ (ln x) powr (1 / 2)))) ∈ Rc
    unfolding Rc_def using Hc Hε
    by (real_asymp simp add: field_simps)
  ultimately have (λx. r_1 x ∗ r_2 x) ∈ Rc
    unfolding Rc_def by (rule landau_o.big_trans)
  moreover have ∀_F x in at_top. r x = r_1 x ∗ r_2 x
    using ln_ln_asymp_pos ln_asymp_pos x_asymp_pos
    unfolding r_def r_1_def r_2_def a_def powr_def power2_eq_square
    by (eventually_elim) (simp add: field_simps exp_add [symmetric])
  ultimately show ?thesis unfolding Rc_def
    using landau_o.big.ev_eq_trans2 by auto
 qed
 ultimately have (λx. ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_1 x) (f x)‖) ∈ Rc
   unfolding Rc_def by (rule landau_o.big_trans)
 thus ?thesis unfolding Rc_def by (simp add: norm_divide)
qed

lemma estimation_P_t':
 assumes h:
   1 < x ∧ max 1 C_3 ≤ T x a x < 1 ∧ 1 < b x
   {s ∈ rect x. C_3 ≤ |Im s|} ⊆ logderiv_zeta_region
   f x contour_integrable_on P_3 x ∧ f x contour_integrable_on P_4 x
   and Ht: |t| = T x
 shows ‖contour_integral (P_t x t) (f x)‖ ≤ C_2 ∗ exp 1 ∗ x / T x ∗ (ln (T x + 3))^2 ∗ (b x − a x)
proof −
 consider t = T x | t = − T x using Ht by fastforce
 hence f x contour_integrable_on P_t x t
   using Ht h(4) unfolding P_t_def P_3_def P_4_def by cases auto
 moreover have ‖f x s‖ ≤ exp 1 ∗ x / T x ∗ (C_2 ∗ (ln (T x + 3))^2)
   when s ∈ closed_segment (Complex (a x) t) (Complex (b x) t) for s
 proof −
   have Hs: s ∈ path_image (P_t x t) using that unfolding P_t_def by auto
   have path_image (P_t x t) ⊆ rect x
     by (rule path_image_in_rect_2) (use h(2) Ht in auto)
   moreover have Hs': Re s ≤ b x Im s = t
   proof −
     have u ≤ 1 =⇒ (1 − u) ∗ a x ≤ (1 − u) ∗ b x for u
       using h(2) by (intro mult_left_mono) auto
```

```
      thus Re s ≤ b x Im s = t
        using that h(2) unfolding closed_segment_def
        by (auto simp add: legacy_Complex_simps field_simps)
    qed
    hence C_3 ≤ |Im s| using h(1) Ht by auto
    ultimately have s ∈ logderiv_zeta_region using Hs h(3) by auto
    hence ‖logderiv zeta s‖ ≤ C_2 ∗ (ln (|Im s| + 3))^2
      by (rule logderiv_zeta_region_estimate)
    also have ... = C_2 ∗ (ln (T x + 3))^2 using Hs'(2) Ht by auto
    also have ‖x powr s / s‖ ≤ exp 1 ∗ x / T x
    proof −
      have ‖x powr s‖ = Re x powr Re s using h(1) by (intro norm_powr_real_powr) auto
      also have ... = x powr Re s by auto
      also have ... ≤ x powr b x by (intro powr_mono Hs') (use h(1) in auto)
      also have ... = exp 1 ∗ x
        using h(1) unfolding powr_def b_def by (auto simp add: field_simps exp_add)
      finally have ‖x powr s‖ ≤ exp 1 ∗ x .
      moreover have T x ≤ ‖s‖ using abs_Im_le_cmod [of s] Hs'(2) h(1) Ht by auto
      hence 1: ‖x powr s‖ / ‖s‖ ≤ ‖x powr s‖ / T x
        using h(1) by (intro divide_left_mono mult_pos_pos) auto
      ultimately have ... ≤ exp 1 ∗ x / T x
        by (intro divide_right_mono) (use h(1) in auto)
      thus ?thesis using 1 by (subst norm_divide) linarith
    qed
    ultimately show ?thesis unfolding f_def
      by (subst norm_mult, intro mult_mono, auto)
         (metis norm_ge_zero order.trans)
  qed
  ultimately have ‖contour_integral (P_t x t) (f x)‖
    ≤ exp 1 ∗ x / T x ∗ (C_2 ∗ (ln (T x + 3))^2) ∗ ‖Complex (b x) t − Complex (a x) t‖
    unfolding P_t_def
    by (intro contour_integral_bound_linepath)
       (use C_2_gt_zero h(1) in auto)
  also have ... = C_2 ∗ exp 1 ∗ x / T x ∗ (ln (T x + 3))^2 ∗ (b x − a x)
    using h(2) by (simp add: legacy_Complex_simps)
  finally show ?thesis .
qed

lemma estimation_P_t:
  (λx. ‖contour_integral (P_3 x) (f x)‖) ∈ Rc ∧
   (λx. ‖contour_integral (P_4 x) (f x)‖) ∈ Rc
proof −
  define r where r x ≡ C_2 ∗ exp 1 ∗ x / T x ∗ (ln (T x + 3))^2 ∗ (b x − a x) for x
  define p where p x ≡ ‖contour_integral (P_3 x) (f x)‖ ≤ r x ∧ ‖contour_integral (P_4 x) (f x)‖ ≤ r x
for x
  have ∀_F x in at_top. 1 < x ∧ max 1 C_3 ≤ T x
    unfolding T_def by (rule eventually_conj) (simp, use Hc in real_asymp)
  hence ∀_F x in at_top. ∀ t. |t| = T x ⟶ ‖contour_integral (P_t x t) (f x)‖ ≤ r x (is eventually ?P _)
    unfolding r_def using asymp_1 rect_in_logderiv_zeta contour_integrability
    by eventually_elim (use estimation_P_t' in blast)
  moreover have ⋀x. ?P x ⟹ 0 < T x ⟹ p x
    unfolding p_def P_3_def P_4_def by auto
  hence ∀_F x in at_top. ?P x ⟶ 0 < T x ⟶ p x
    by (intro eventuallyI) blast
  ultimately have ∀_F x in at_top. p x using asymp_1 by eventually_elim blast
```

hence $\forall_F$ *x in at_top*.
‖‖‖*contour_integral* (P_3 *x*) (*f x*)‖‖‖ ≤ *1* ∗ ‖*r x*‖ ∧
‖‖‖*contour_integral* (P_4 *x*) (*f x*)‖‖‖ ≤ *1* ∗ ‖*r x*‖
unfolding *p_def* **by** *eventually_elim auto*
hence (λ*x*. ‖*contour_integral* (P_3 *x*) (*f x*)‖) ∈ *O*(*r*) ∧ (λ*x*. ‖*contour_integral* (P_4 *x*) (*f x*)‖) ∈ *O*(*r*)
by (*subst* (*asm*) *eventually_conj_iff*, *blast*)+
moreover have *r* ∈ *Rc*
unfolding *r_def Rc_def a_def b_def T_def* **using** *Hc Hε*
by (*real_asymp simp add*: *field_simps*)
ultimately show *?thesis*
unfolding *Rc_def* **using** *landau_o.big_trans* **by** *blast*
qed

lemma *Re_path_*P_2:
⋀*z*. *z* ∈ *path_image* (P_2 *x*) ⟹ *Re z* = *b x*
unfolding P_2*_def* z_2*_def* z_3*_def*
by (*auto simp add*: *closed_segment_def legacy_Complex_simps field_simps*)

lemma *estimation_*P_2:
(λ*x*. ‖*1* / (*2* ∗ *pi* ∗ i) ∗ *contour_integral* (P_2 *x*) (*f x*) + *x*‖) ∈ *Rc*
proof −
define *r* **where** *r x* ≡ ‖*contour_integral* (P_1 *x*) (*f x*)‖ +
‖*contour_integral* (P_3 *x*) (*f x*)‖ + ‖*contour_integral* (P_4 *x*) (*f x*)‖ **for** *x*
have [*simp*]: ‖*a* − *b* + *c*‖ ≤ ‖*a*‖ + ‖*b*‖ + ‖*c*‖ **for** *a b c* :: *complex*
using *adhoc_norm_triangle norm_triangle_ineq4* **by** *blast*
have $\forall_F$ *x in at_top*. ‖*contour_integral* (P_2 *x*) (*f x*) + *2* ∗ *pi* ∗ i ∗ *x*‖ ≤ *r x*
unfolding *r_def* **using** P_2*_eq* **by** *eventually_elim auto*
hence (λ*x*. ‖*contour_integral* (P_2 *x*) (*f x*) + *2* ∗ *pi* ∗ i ∗ *x*‖) ∈ *O*(*r*)
by (*rule eventually_le_imp_bigo*′)
moreover have *r* ∈ *Rc*
using *estimation_*P_1 *estimation_*P_t
unfolding *r_def Rc_def* **by** (*intro sum_in_bigo*) *auto*
ultimately have (λ*x*. ‖*contour_integral* (P_2 *x*) (*f x*) + *2* ∗ *pi* ∗ i ∗ *x*‖) ∈ *Rc*
unfolding *Rc_def* **by** (*rule landau_o.big_trans*)
hence (λ*x*. ‖*1* / (*2* ∗ *pi* ∗ i) ∗ (*contour_integral* (P_2 *x*) (*f x*) + *2* ∗ *pi* ∗ i ∗ *x*)‖) ∈ *Rc*
unfolding *Rc_def* **by** (*auto simp add*: *norm_mult norm_divide*)
thus *?thesis* **by** (*auto simp add*: *algebra_simps*)
qed

lemma *estimation_R*:
R ∈ *Rc*
proof −
define Γ **where** Γ *x* ≡ {*n* :: *nat*. *x* − *x* / *H x* ≤ *n* ∧ *n* ≤ *x* + *x* / *H x*} **for** *x*
have *1*: (λ*x*. *x powr b x* ∗ *H x* ∗ *B x* / *T x*) ∈ *Rc*
unfolding *b_def H_def B_def T_def Rc_def* **using** *Hc Hε*
by (*real_asymp simp add*: *field_simps*)
have ‖∑*n*∈Γ *x*. ‖*fds_nth* (*fds mangoldt_complex*) *n*‖‖ ≤ (*2* ∗ *x* / *H x* + *1*) ∗ *ln* (*x* + *x* / *H x*)
when *h*: *0* < *x* − *x* / *H x 0* < *x* / *H x 0* ≤ *ln* (*x* + *x* / *H x*) **for** *x*
proof −
have ‖∑*n*∈Γ *x*. ‖*fds_nth* (*fds mangoldt_complex*) *n*‖‖ = (∑*n*∈Γ *x*. ‖*fds_nth* (*fds mangoldt_complex*) *n*‖)
by *simp* (*subst abs_of_nonneg*, *auto intro*: *sum_nonneg*)
also have ... = *sum mangoldt_real* (Γ *x*)
by (*subst norm_fds_mangoldt_complex*) (*rule refl*)
also have ... ≤ *card* (Γ *x*) ∗ *ln* (*x* + *x* / *H x*)

proof (*rule sum_bounded_above*)
fix *n* **assume** $n \in \Gamma\ x$
hence *Hn*: $0 < n$ $n \le x + x / H\ x$ **unfolding** Γ*_def* **using** *h* **by** *auto*
hence *mangoldt_real n* $\le$ *ln n* **by** (*intro mangoldt_le*)
also have ... $\le$ *ln* $(x + x / H\ x)$ **using** *Hn* **by** *auto*
finally show *mangoldt_real n* $\le$ *ln* $(x + x / H\ x)$ **.**
qed
also have ... $\le (2 * x / H\ x + 1) * ln\ (x + x / H\ x)$
proof −
have Γ*_eq*: $\Gamma\ x = \{nat\ \lceil x - x / H\ x\rceil..<nat\ (\lfloor x + x / H\ x\rfloor + 1)\}$
unfolding Γ*_def* **by** (*subst nat_le_real_iff*) (*subst nat_ceiling_le_eq* [*symmetric*], *auto*)
moreover have $nat\ (\lfloor x + x / H\ x\rfloor + 1) = \lfloor x + x / H\ x\rfloor + 1$ **using** *h*(*1*) *h*(*2*) **by** *auto*
moreover have $nat\ \lceil x - x / H\ x\rceil = \lceil x - x / H\ x\rceil$ **using** *h*(*1*) **by** *auto*
moreover have $\lfloor x + x / H\ x\rfloor \le x + x / H\ x$ **by** (*rule floor_le*)
moreover have $\lceil x - x / H\ x\rceil \ge x - x / H\ x$ **by** (*rule ceil_ge*)
ultimately have $(nat\ (\lfloor x + x / H\ x\rfloor + 1) :: real) - nat\ \lceil x - x / H\ x\rceil \le 2 * x / H\ x + 1$ **by** *linarith*
hence $card\ (\Gamma\ x) \le 2 * x / H\ x + 1$ **using** *h*(*2*) **by** (*subst* Γ*_eq*) (*auto simp add*: *of_nat_diff_real*)
thus *?thesis* **using** *h*(*3*) **by** (*rule mult_right_mono*)
qed
finally show *?thesis* **.**
qed
hence $\forall_F\ x$ *in at_top*.
$0 < x - x / H\ x \longrightarrow 0 < x / H\ x \longrightarrow 0 \le ln\ (x + x / H\ x)$
$\longrightarrow \|\sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|\| \le (2 * x / H\ x + 1) * ln\ (x + x / H\ x)$
by (*intro eventuallyI*) *blast*
moreover have $\forall_F\ x$ *in at_top*. $0 < x - x / H\ x$ **unfolding** *H_def* **using** *Hc H*ε **by** *real_asymp*
moreover have $\forall_F\ x$ *in at_top*. $0 < x / H\ x$ **unfolding** *H_def* **using** *Hc H*ε **by** *real_asymp*
moreover have $\forall_F\ x$ *in at_top*. $0 \le ln\ (x + x / H\ x)$ **unfolding** *H_def* **using** *Hc H*ε **by** *real_asymp*
ultimately have $\forall_F\ x$ *in at_top*. $\|\sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|\| \le (2 * x / H\ x + 1) * ln\ (x + x / H\ x)$
by *eventually_elim blast*
hence $(\lambda x.\ \sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|) \in O(\lambda x.\ (2 * x / H\ x + 1) * ln\ (x + x / H\ x))$
by (*rule eventually_le_imp_bigo*)
moreover have $(\lambda x.\ (2 * x / H\ x + 1) * ln\ (x + x / H\ x)) \in Rc'$
unfolding *Rc′_def H_def* **using** *Hc H*ε
by (*real_asymp simp add*: *field_simps*)
ultimately have $(\lambda x.\ \sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|) \in Rc'$
unfolding *Rc′_def* **by** (*rule landau_o.big_trans*)
hence $(\lambda x.\ 3 * (2 + ln\ (T\ x / b\ x)) * (\sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|))$
$\in O(\lambda x.\ 3 * (2 + ln\ (T\ x / b\ x)) * (x * exp\ (-\ (c / 2 - \varepsilon) * (ln\ x)\ powr\ (1 / 2))))$
unfolding *Rc′_def* **by** (*intro landau_o.big.mult_left*) *auto*
moreover have $(\lambda x.\ 3 * (2 + ln\ (T\ x / b\ x)) * (x * exp\ (-\ (c / 2 - \varepsilon) * (ln\ x)\ powr\ (1 / 2)))) \in Rc$
unfolding *Rc_def T_def b_def* **using** *Hc H*ε **by** (*real_asymp simp add*: *field_simps*)
ultimately have *2*: $(\lambda x.\ 3 * (2 + ln\ (T\ x / b\ x)) * (\sum n{\in}\Gamma\ x.\ \|fds\_nth\ (fds\ mangoldt\_complex)\ n\|)) \in Rc$
unfolding *Rc_def* **by** (*rule landau_o.big_trans*)
from *1 2* **show** *?thesis* **unfolding** *Rc_def R_def* Γ*_def* **by** (*rule sum_in_bigo*)
qed

lemma *perron_psi*:
$\forall_F\ x$ *in at_top*. $\|\psi\ x + 1 / (2 * pi * \mathrm{i}) * contour\_integral\ (P_2\ x)\ (f\ x)\| \le R\ x$
proof −
have *Hb*: $\forall_F\ x$ *in at_top*. $1 < b\ x$ **unfolding** *b_def* **by** *real_asymp*

hence ∀$_F$ *x in at_top. 0 < b x* **by** *eventually_elim auto*
moreover have ∀$_F$ *x in at_top. b x ≤ T x* **unfolding** *b_def T_def* **using** *Hc* **by** *real_asymp*
moreover have ∀$_F$ *x in at_top. abs_conv_abscissa (fds mangoldt_complex) < ereal (b x)*
proof −
  have *abs_conv_abscissa (fds mangoldt_complex) ≤ 1* **by** (*rule abs_conv_abscissa_mangoldt*)
  hence ∀$_F$ *x in at_top. 1 < b x ⟶ abs_conv_abscissa (fds mangoldt_complex) < ereal (b x)*
    by (*auto intro*: *eventuallyI*
        simp add: *le_ereal_less one_ereal_def*)
  thus *?thesis* **using** *Hb* **by** (*rule eventually_mp*)
qed
moreover have ∀$_F$ *x in at_top. 1 < H x* **unfolding** *H_def* **using** *Hc* **by** *real_asymp*
moreover have ∀$_F$ *x in at_top. b x + 1 ≤ H x* **unfolding** *b_def H_def* **using** *Hc* **by** *real_asymp*
moreover have ∀$_F$ *x :: real in at_top. 0 < x* **by** *auto*
moreover have ∀$_F$ *x in at_top.*
  (∑ '*n≥1. ‖fds_nth (fds mangoldt_complex) n‖ / n nat_powr b x*) ≤ *B x*
  (**is** *eventually ?P ?F*)
proof −
  have *?P x* **when** *Hb*: *1 < b x ∧ b x ≤ 23 / 20* **for** *x*
  proof −
    have (∑ '*n≥1. ‖fds_nth (fds mangoldt_complex) n‖ / n nat_powr (b x)*)
      = (∑ '*n≥1. mangoldt_real n / n nat_powr (b x)*)
     by (*subst norm_fds_mangoldt_complex*) (*rule refl*)
    also have ... = − *Re* (*logderiv zeta* (*b x*))
    proof −
      have ((*λn. mangoldt_real n ∗ n nat_powr* (−*b x*) ∗ *cos* (*0 ∗ ln* (*real n*)))
        has_sum Re (− *deriv zeta* (*Complex* (*b x*) *0*) / *zeta* (*Complex* (*b x*) *0*))) {*1*..}
       by (*intro sums_Re_logderiv_zeta*) (*use Hb* **in** *auto*)
      moreover have *Complex* (*b x*) *0* = *b x* **by** (*rule complex_eqI*) *auto*
      moreover have *Re* (− *deriv zeta* (*b x*) / *zeta* (*b x*)) = − *Re* (*logderiv zeta* (*b x*))
       unfolding *logderiv_def* **by** *auto*
      ultimately have ((*λn. mangoldt_real n ∗ n nat_powr* (−*b x*)) *has_sum*
          − *Re* (*logderiv zeta* (*b x*))) {*1*..} **by** *auto*
      hence − *Re* (*logderiv zeta* (*b x*)) = (∑ '*n≥1. mangoldt_real n ∗ n nat_powr* (−*b x*))
       by (*intro has_sum_imp_has_subsum subsumI*)
      also have ... = (∑ '*n≥1. mangoldt_real n / n nat_powr* (*b x*))
       by (*intro subsum_cong*) (*auto simp add*: *powr_minus_divide*)
      finally show *?thesis* **by** *auto*
    qed
    also have ... ≤ |*Re* (*logderiv zeta* (*b x*))| **by** *auto*
    also have ... ≤ ‖*logderiv zeta* (*b x*)‖ **by** (*rule abs_Re_le_cmod*)
    also have ... ≤ *5 / 4* ∗ (*1* / (*b x* − *1*))
      by (*rule logderiv_zeta_bound*) (*use Hb* **in** *auto*)
    also have ... = *B x* **unfolding** *b_def B_def* **by** *auto*
    finally show *?thesis* **.**
  qed
  hence ∀$_F$ *x in at_top. 1 < b x ∧ b x ≤ 23 / 20 ⟶ ?P x* **by** *auto*
  moreover have ∀$_F$ *x in at_top. b x ≤ 23 / 20* **unfolding** *b_def* **by** *real_asymp*
  ultimately show *?thesis* **using** *Hb* **by** *eventually_elim auto*
qed
ultimately have ∀$_F$ *x in at_top.*
  ‖*sum_upto* (*fds_nth* (*fds mangoldt_complex*)) *x* − *1* / (*2 ∗ pi ∗* i)
    ∗ *contour_integral* (P_2 *x*) (*λs. eval_fds* (*fds mangoldt_complex*) *s ∗ x powr s / s*)‖ ≤ *R x*
  unfolding *R_def* P_2*_def* z_2*_def* z_3*_def*
  by *eventually_elim* (*rule perron_formula*(*2*))
moreover have ∀$_F$ *x in at_top. sum_upto* (*fds_nth* (*fds mangoldt_complex*)) *x* = *ψ x* **for** *x* :: *real*

```
    unfolding primes_psi_def sum_upto_def by auto
  moreover have
     contour_integral (P_2 x) (λs. eval_fds (fds mangoldt_complex) s ∗ x powr s / s)
    = contour_integral (P_2 x) (λs. − (x powr s / s ∗ logderiv zeta s))
    when 1 < b x for x
  proof (rule contour_integral_eq, goal_cases)
    case (1 s)
    hence Re s = b x by (rule Re_path_P_2)
    hence eval_fds (fds mangoldt_complex) s = − deriv zeta s / zeta s
      by (intro eval_fds_mangoldt) (use that in auto)
    thus ?case unfolding logderiv_def by (auto simp add: field_simps)
  qed
  hence ∀_F x in at_top. 1 < b x ⟶
      contour_integral (P_2 x) (λs. eval_fds (fds mangoldt_complex) s ∗ x powr s / s)
    = contour_integral (P_2 x) (λs. − (x powr s / s ∗ logderiv zeta s))
    using Hb by (intro eventuallyI) blast
  ultimately have ∀_F x in at_top.
    ‖ψ x − 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (λs. − (x powr s / s ∗ logderiv zeta s))‖ ≤ R x
    using Hb by eventually_elim auto
  thus ?thesis unfolding f_def
    by eventually_elim (auto simp add: contour_integral_neg)
qed

lemma estimation_perron_psi:
  (λx. ‖ψ x + 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x)‖) ∈ Rc
proof −
  have (λx. ‖ψ x + 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x)‖) ∈ O(R)
    by (intro eventually_le_imp_bigo' perron_psi)
  moreover have R ∈ Rc by (rule estimation_R)
  ultimately show ?thesis unfolding Rc_def by (rule landau_o.big_trans)
qed

theorem prime_number_theorem:
  PNT_3 (c / 2 − 2 ∗ ε) (1 / 2) 0
proof −
  define r where r x ≡
      ‖ψ x + 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x)‖
    + ‖1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x) + x‖ for x
  have ‖ψ x − x‖ ≤ r x for x
  proof −
    have ‖ψ x − x‖ = ‖(ψ x :: complex) − x‖
      by (fold dist_complex_def, simp add: dist_real_def)
    also have … ≤ ‖ψ x − − 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x)‖
      + ‖x − − 1 / (2 ∗ pi ∗ i) ∗ contour_integral (P_2 x) (f x)‖
      by (fold dist_complex_def, rule dist_triangle2)
    finally show ?thesis unfolding r_def by (simp add: add_ac)
  qed
  hence (λx. ψ x − x) ∈ O(r) by (rule le_imp_bigo)
  moreover have r ∈ Rc
    unfolding r_def Rc_def
    by (intro sum_in_bigo, fold Rc_def)
       (rule estimation_perron_psi, rule estimation_P_2)
  ultimately show ?thesis unfolding PNT_3_def
    by (subst Rc_eq_rem_est [symmetric], unfold Rc_def)
       (rule landau_o.big_trans)
```

```
qed

no_notation primes_psi (‹ψ›)
end

unbundle prime_counting_syntax

theorem prime_number_theorem:
  shows (λx. π x − Li x) ∈ O(λx. x ∗ exp (− 1 / 3653 ∗ (ln x) powr (1 / 2)))
proof −
  define c :: real where c ≡ 1 / 1826
  define ε :: real where ε ≡ 1 / 26681512
  interpret z: prime_number_theorem c ε
    unfolding c_def ε_def by standard (auto simp: C4_def)
  have PNT_3 (c / 2 − 2 ∗ ε) (1 / 2) 0 by (rule z.prime_number_theorem)
  hence PNT_1 (c / 2 − 2 ∗ ε) (1 / 2) 0 by (auto intro: PNT_3_imp_PNT_1)
  thus (λx. π x − Li x) ∈ O(λx. x ∗ exp (− 1 / 3653 ∗ (ln x) powr (1 / 2)))
    unfolding PNT_1_def rem_est_def c_def ε_def
    by (rule landau_o.big.ev_eq_trans1, use ln_ln_asymp_pos in eventually_elim)
      (auto intro: eventually_at_top_linorderI [of 1] simp: powr_half_sqrt)
qed

hide_const (open) C3 C4 C5
unbundle no prime_counting_syntax and no pnt_syntax

end
```