Lambert Series

Manuel Eberl

April 18, 2024

Abstract

This entry provides a formalisation of *Lambert series*, i.e. series of the form $L(a_n, q) = \sum_{n=1}^{\infty} a_n q^n/(1-q^n)$ where a_n is a sequence of real or complex numbers. Proofs for all the basic properties are provided, such as

- the precise region in which $L(a_n, q)$ converges
- the functional equation $L(a_n, \frac{1}{q}) = -(\sum_{n=1}^{\infty} a_n) - L(a_n, q)$
- the power series expansion of $L(a_n, q)$ at $q = 0$
- the connection $L(a_n, q) = \sum_{k=1}^{\infty} f(q^k)$ for $f(z) = \sum_{n=1}^{\infty} a_n z^n$ that links a Lambert series to its "corresponding" power series
- connections to various number-theoretic functions, e.g. the divisor σ function via $\sum_{n=1}^{\infty} \sigma_\alpha(n) q^n = L(n^\alpha, q)$

The formalisation mainly follows the chapter on Lambert series in Konrad Knopp's classic textbook *Theory and Application of Infinite Series* [1] and includes all results presented therein.

Contents

1 Missing Library Material

theory *Lambert_Series_Library*
imports
"HOL-Complex_Analysis.Complex_Analysis"
"HOL-Library.Landau_Symbols"
"HOL-Real_Asymp.Real_Asymp"
begin

1.1 Miscellaneous

lemma *power_less_1_iff: "x ≥ 0 ⟹ (x :: real) ^ n < 1 ⟷ x < 1 ∧ n > 0"*
by *(metis not_gr_zero not_less_iff_gr_or_eq power_0 real_root_lt_1_iff real_root_pos2)*

lemma *fls_nth_sum: "fls_nth (∑x∈A. f x) n = (∑x∈A. fls_nth (f x) n)"*
by *(induction A rule: infinite_finite_induct) auto*

lemma *two_times_choose_two: "2 * (n choose 2) = n * (n - 1)"*
unfolding *choose_two* **by** *(simp add: algebra_simps)*

lemma *Nats_not_empty [simp]: "ℕ ≠ {}"*
using *Nats_1* **by** *blast*

1.2 Infinite sums

lemma *has_sum_iff: "(f has_sum S) A ⟷ f summable_on A ∧ infsum f A = S"*
using *infsumI summable_iff_has_sum_infsum* **by** *blast*

lemma *summable_on_reindex_bij_witness:*
assumes *"⋀a. a ∈ S ⟹ i (j a) = a"*
assumes *"⋀a. a ∈ S ⟹ j a ∈ T"*
assumes *"⋀b. b ∈ T ⟹ j (i b) = b"*
assumes *"⋀b. b ∈ T ⟹ i b ∈ S"*
assumes *"⋀a. a ∈ S ⟹ h (j a) = g a"*
shows *"g summable_on S ⟷ h summable_on T"*
using *has_sum_reindex_bij_witness[of S i j T h g, OF assms refl]*
by *(simp add: summable_on_def)*

lemma *has_sum_diff:*
fixes *f g :: "'a ⇒ 'b::{topological_ab_group_add}"*
assumes *‹(f has_sum a) A›*
assumes *‹(g has_sum b) A›*
shows *‹((λx. f x - g x) has_sum (a - b)) A›*
using *has_sum_add[of f A a "λx. -g x" "-b"] assms* **by** *(simp add: has_sum_uminus)*

lemma *summable_on_diff:*

```
  fixes f g :: "'a ⇒ 'b::{topological_ab_group_add}"
  assumes ‹f summable_on A›
  assumes ‹g summable_on A›
  shows ‹(λx. f x - g x) summable_on A›
  by (metis (full_types) assms summable_on_def has_sum_diff)

lemma infsum_diff:
  fixes f g :: "'a ⇒ 'b::{topological_ab_group_add, t2_space}"
  assumes ‹f summable_on A›
  assumes ‹g summable_on A›
  shows ‹infsum (λx. f x - g x) A = infsum f A - infsum g A›
proof -
  have ‹((λx. f x - g x) has_sum (infsum f A - infsum g A)) A›
    by (simp add: assms has_sum_diff)
  then show ?thesis
    using infsumI by blast
qed

lemma summable_norm_add:
  assumes "summable (λn. norm (f n))" "summable (λn. norm (g n))"
  shows   "summable (λn. norm (f n + g n))"
proof (rule summable_comparison_test)
  show "summable (λn. norm (f n) + norm (g n))"
    by (intro summable_add assms)
  show "∃N. ∀n≥N. norm (norm (f n + g n)) ≤ norm (f n) + norm (g n)"
    by (intro exI[of _ 0] allI impI) (auto simp: norm_triangle_ineq)
qed

lemma summable_norm_diff:
  assumes "summable (λn. norm (f n))" "summable (λn. norm (g n))"
  shows   "summable (λn. norm (f n - g n))"
  using summable_norm_add[of f "λn. -g n"] assms by simp

lemma sums_imp_has_prod_exp:
  fixes f :: "_ ⇒'a::{real_normed_field,banach}"
  assumes "f sums F"
  shows   "(λn. exp (f n)) has_prod exp F"
proof -
  have "(λn. exp (∑i≤n. f i)) ⟶ exp F"
    by (intro tendsto_intros) (use assms in ‹auto simp: sums_def' atLeast0AtMost›)
  also have "(λn. exp (∑i≤n. f i)) = (λn. ∏i≤n. exp (f i))"
    by (simp add: exp_sum)
  finally have "raw_has_prod (λn. exp (f n)) 0 (exp F)"
    unfolding raw_has_prod_def by auto
  thus ?thesis
    unfolding has_prod_def by blast
qed

lemma telescope_summable_iff:
```

```
  fixes f :: "nat ⇒ 'a::{real_normed_vector}"
  shows "summable (λn. f (Suc n) - f n) ⟷ convergent f"
proof
  assume "convergent f"
  thus "summable (λn. f (Suc n) - f n)"
    using telescope_summable[of f] by (auto simp: convergent_def)
next
  assume "summable (λn. f (Suc n) - f n)"
  hence "convergent (λn. ∑i<n. f (Suc i) - f i)"
    by (simp add: summable_iff_convergent)
  also have "(λn. ∑i<n. f (Suc i) - f i) = (λn. f n - f 0)"
    by (subst sum_lessThan_telescope) auto
  also have "convergent ... ⟷ convergent f"
    by (rule convergent_diff_const_right_iff)
  finally show "convergent f" .
qed

lemma telescope_summable_iff':
  fixes f :: "nat ⇒ 'a::{real_normed_vector}"
  shows "summable (λn. f n - f (Suc n)) ⟷ convergent f"
  using telescope_summable_iff[of "λn. -f n"] by (simp flip: convergent_minus_iff)

lemma norm_summable_mult_bounded:
  assumes "summable (λn. norm (f n))"
  assumes "g ∈ O(λ_. 1)"
  shows   "summable (λn. norm (f n * g n))"
proof -
  from assms(2) obtain C where C: "C > 0" "eventually (λn. norm (g n)
≤ C) at_top"
    by (auto elim!: landau_o.bigE)
  show ?thesis
  proof (rule summable_comparison_test_ev)
    show "summable (λn. norm (f n) * C)"
      by (subst mult.commute) (intro summable_mult assms)
    show "eventually (λn. norm (norm (f n * g n)) ≤ norm (f n) * C) at_top"
      using C(2) by eventually_elim (use C(1) in ‹auto intro!: mult_mono
simp: norm_mult›)
  qed
qed

lemma summable_powser_comparison_test_bigo:
  fixes f g :: "nat ⇒ 'a :: {real_normed_field, banach}"
  assumes "summable f" "g ∈ O(λn. f n * c ^ n)" "norm c < 1"
  shows   "summable (λn. norm (g n))"
proof (rule summable_comparison_test_bigo)
  have "summable (λn. norm (f n * c ^ n))"
    by (rule powser_insidea[of _ 1]) (use assms in auto)
  thus "summable (λn. norm (norm (f n * c ^ n)))"
    by simp
```

```
  show "(λn. norm (g n)) ∈ O(λn. norm (f n * c ^ n))"
    using assms(2) by simp
qed

lemma geometric_sums_gen:
  assumes "norm (x :: 'a :: real_normed_field) < 1"
  shows   "(λn. x ^ (n + k)) sums (x ^ k / (1 - x))"
proof -
  have "(λn. x ^ k * x ^ n) sums (x ^ k * (1 / (1 - x)))"
    by (intro sums_mult geometric_sums assms)
  thus ?thesis
    by (simp add: power_add mult_ac)
qed

lemma has_sum_geometric:
  fixes x :: "'a :: {real_normed_field, banach}"
  assumes "norm x < 1"
  shows   "((λn. x ^ n) has_sum (x ^ m / (1 - x))) {m..}"
proof -
  have "((λn. x ^ n) has_sum (1 / (1 - x))) UNIV"
    using assms
    by (intro norm_summable_imp_has_sum)
       (auto intro: geometric_sums summable_geometric simp: norm_power)
  hence "((λn. x ^ m * x ^ n) has_sum (x ^ m * (1 / (1 - x)))) UNIV"
    by (rule has_sum_cmult_right)
  also have "?this ⟷ ?thesis"
    by (rule has_sum_reindex_bij_witness[of _ "λn. n - m" "λn. n + m"])
(auto simp: power_add)
  finally show ?thesis .
qed

lemma n_powser_sums:
  fixes q :: "'a :: {real_normed_field,banach}"
  assumes q: "norm q < 1"
  shows   "(λn. of_nat n * q ^ n) sums (q / (1 - q) ^ 2)"
proof -
  have "(λn. q * (of_nat (Suc n) * q ^ n)) sums (q * (1 / (1 - q)^2))"
    using q by (intro sums_mult geometric_deriv_sums)
  also have "(λn. q * (of_nat (Suc n) * q ^ n)) = (λn. of_nat (Suc n)
* q ^ Suc n)"
    by (simp add: algebra_simps)
  finally have "(λn. of_nat n * q ^ n) sums (q * (1 / (1 - q)^2) + of_nat
0 * q ^ 0)"
    by (rule sums_Suc)
  thus "(λn. of_nat n * q ^ n) sums (q / (1 - q) ^ 2)"
    by simp
qed
```

1.3 Convergence radius

```
lemma tendsto_imp_conv_radius_eq:
  assumes "(λn. ereal (norm (f n) powr (1 / real n))) ⟶ c'" "c =
inverse c'"
  shows   "conv_radius f = c"
proof -
  have "(λn. ereal (root n (norm (f n)))) ⟶ c'"
  proof (rule Lim_transform_eventually)
    show "(λn. ereal (norm (f n) powr (1 / real n))) ⟶ c'"
      using assms by simp
    show "∀F x in sequentially. ereal (norm (f x) powr (1 / real x))
=
                                  ereal (root x (norm (f x)))"
      using eventually_gt_at_top[of 0]
    proof eventually_elim
      case (elim n)
      show ?case using elim
        by (cases "f n = 0") (simp_all add: root_powr_inverse)
    qed
  qed
  thus ?thesis
    unfolding conv_radius_def using assms by (simp add: limsup_root_limit)
qed

lemma conv_radius_powr_real: "conv_radius (λn. real n powr a) = 1"
proof (rule tendsto_imp_conv_radius_eq)
  have "(λn. ereal ((real n powr a) powr (1 / real n))) ⟶ ereal 1"
    by (rule tendsto_ereal) real_asymp
  thus "(λn. ereal (norm (real n powr a) powr (1 / real n))) ⟶ ereal
1"
    by simp
qed (simp_all add: one_ereal_def)

lemma conv_radius_one_over: "conv_radius (λn. 1 / of_nat n :: 'a :: {real_normed_field,
banach}) = 1"
proof (rule tendsto_imp_conv_radius_eq)
  have "(λn. ereal ((1 / n) powr (1 / real n))) ⟶ ereal 1"
    by (rule tendsto_ereal) real_asymp
  thus "(λn. ereal (norm (1 / of_nat n :: 'a) powr (1 / real n))) ⟶
ereal 1"
    by (simp add: norm_divide)
qed (simp_all add: one_ereal_def)

lemma conv_radius_mono:
  assumes "eventually (λn. norm (f n) ≥ norm (g n)) sequentially"
  shows   "conv_radius f ≤ conv_radius g"
  unfolding conv_radius_def
proof (rule ereal_inverse_antimono[OF _ Limsup_mono])
  have "limsup (λn. 0) ≤ limsup (λn. ereal (root n (norm (g n))))"
```

```
    by (rule Limsup_mono) (auto intro!: eventually_mono[OF eventually_gt_at_top[of
0]])
  thus "limsup (λn. ereal (root n (norm (g n)))) ≥ 0"
    by (simp add: Limsup_const)
next
  show "∀F x in sequentially. ereal (root x (norm (g x))) ≤ ereal (root
x (norm (f x)))"
    using assms eventually_gt_at_top[of 0] by eventually_elim auto
qed

lemma conv_radius_const [simp]:
  assumes "c ≠ 0"
  shows   "conv_radius (λ_. c) = 1"
proof (rule tendsto_imp_conv_radius_eq)
  show "(λn. ereal (norm c powr (1 / real n))) ⟶ ereal 1"
    by (rule tendsto_ereal) (use assms in real_asymp)
qed auto

lemma conv_radius_bigo_polynomial:
  assumes "f ∈ O(λn. of_nat n ^ k)"
  shows   "conv_radius f ≥ 1"
proof -
  from assms obtain C where ev: "C > 0" "eventually (λn. norm (f n) ≤
C * real n ^ k) at_top"
    by (elim landau_o.bigE) (auto simp: norm_power)
  have "(λx. (C * real x ^ k) powr (1 / real x)) ⟶ 1"
    using ev(1) by real_asymp
  hence "conv_radius (λn. C * real n ^ k) = inverse (ereal 1)" using ev(1)
    by (intro tendsto_imp_conv_radius_eq[OF _ refl] tendsto_ereal)
       (simp add: norm_mult norm_power abs_mult)
  moreover have "conv_radius (λn. C * real n ^ k) ≤ conv_radius f"
    by (intro conv_radius_mono eventually_mono[OF ev(2)]) auto
  ultimately show ?thesis
    by (simp add: one_ereal_def)
qed
```

1.4 Limits

```
lemma oscillation_imp_not_tendsto:
  assumes "eventually (λn. f (g n) ∈ A) sequentially" "filterlim g F
sequentially"
  assumes "eventually (λn. f (h n) ∈ B) sequentially" "filterlim h F
sequentially"
  assumes "closed A" "closed B" "A ∩ B = {}"
  shows   "¬filterlim f (nhds c) F"
proof
  assume *: "filterlim f (nhds c) F"
  have "filterlim (λn. f (g n)) (nhds c) sequentially"
    using * assms(2) by (rule filterlim_compose)
```

```
  with assms(1,5) have "c ∈ A"
    by (metis Lim_in_closed_set sequentially_bot)
  have "filterlim (λn. f (h n)) (nhds c) sequentially"
    using * assms(4) by (rule filterlim_compose)
  with assms(3,6) have "c ∈ B"
    by (metis Lim_in_closed_set sequentially_bot)
  with ‹c ∈ A› and ‹A ∩ B = {}› show False
    by blast
qed

lemma oscillation_imp_not_convergent:
  assumes "frequently (λn. f n ∈ A) sequentially"
  assumes "frequently (λn. f n ∈ B) sequentially"
  assumes "closed A" "closed B" "A ∩ B = {}"
  shows   "¬convergent f"
proof -
  obtain g :: "nat ⇒ nat" where g: "strict_mono g" "⋀n. f (g n) ∈ A"
    using assms(1) infinite_enumerate
    unfolding cofinite_eq_sequentially [symmetric] INFM_iff_infinite
    by blast
  obtain h :: "nat ⇒ nat" where h: "strict_mono h" "⋀n. f (h n) ∈ B"
    using assms(2) infinite_enumerate
    unfolding cofinite_eq_sequentially [symmetric] INFM_iff_infinite
    by blast
  have "¬f ⟶⟶ L" for L
  proof (rule oscillation_imp_not_tendsto)
    show "∀⇩F n in sequentially. f (g n) ∈ A" "∀⇩F n in sequentially.
f (h n) ∈ B"
      using g h by auto
  qed (use g h assms in ‹auto intro: filterlim_subseq›)
  thus ?thesis
    unfolding convergent_def by blast
qed

lemma seq_bigo_1_iff:
  "g ∈ O(λ_::nat. 1) ⟷ bounded (range g)"
proof
  assume "g ∈ O(λ_. 1)"
  then obtain C where"eventually (λn. norm (g n) ≤ C) at_top"
    by (elim landau_o.bigE) auto
  then obtain N where N: "⋀n. n ≥ N ⟹ norm (g n) ≤ C"
    by (auto simp: eventually_at_top_linorder)
  hence "norm (g n) ≤ Max (insert C (norm ` g ` {..<N}))" for n
    by (cases "n < N") (auto simp: Max_ge_iff)
  thus "bounded (range g)"
    by (auto simp: bounded_iff)
next
  assume "bounded (range g)"
  then obtain C where "norm (g n) ≤ C" for n
```

```
    by (auto simp: bounded_iff)
  thus "g ∈ O(λ_. 1)"
    by (intro bigoI[where c = C]) auto
qed

lemma incseq_convergent':
  assumes "incseq (g :: nat ⇒ real)" "g ∈ O(λ_. 1)"
  shows   "convergent g"
proof -
  from assms(2) have "bounded (range g)"
    by (simp add: seq_bigo_1_iff)
  then obtain C where C: "|g n| ≤ C" for n
    unfolding bounded_iff by auto
  show ?thesis
  proof (rule incseq_convergent)
    show "incseq g"
      by fact
  next
    have "g i ≤ C" for i :: nat
      using C[of i] by auto
    thus "∀i. g i ≤ C"
      by blast
  qed (auto simp: convergent_def)
qed

lemma decseq_convergent':
  assumes "decseq (g :: nat ⇒ real)" "g ∈ O(λ_. 1)"
  shows   "convergent g"
  using incseq_convergent'[of "λn. -g n "] assms
  by (simp flip: convergent_minus_iff add: decseq_eq_incseq)

lemma filterlim_of_int_iff:
  fixes c :: "'a :: real_normed_algebra_1"
  assumes "F ≠ bot"
  shows "filterlim (λx. of_int (f x)) (nhds c) F ⟷
           (∃c'. c = of_int c' ∧ eventually (λx. f x = c') F)"
proof
  assume "∃c'. c = of_int c' ∧ eventually (λx. f x = c') F"
  then obtain c' where c': "c = of_int c'" "eventually (λx. f x = c')
F"
    by blast
  from c'(2) have "eventually (λx. of_int (f x) = c) F"
    by eventually_elim (auto simp: c'(1))
  thus "filterlim (λx. of_int (f x)) (nhds c) F"
    by (rule tendsto_eventually)
next
  assume *: "filterlim (λx. of_int (f x)) (nhds c) F"
  show "(∃c'. c = of_int c' ∧ eventually (λx. f x = c') F)"
  proof (cases "c ∈ ℤ")
```

```
    case False
    hence "setdist {c} ℤ > 0"
      by (subst setdist_gt_0_compact_closed) auto
    with * have "eventually (λx. dist (of_int (f x)) c < setdist {c}
ℤ) F"
      unfolding tendsto_iff by blast
    then obtain x where "dist (of_int (f x)) c < setdist {c} ℤ"
      using ‹F ≠ bot› eventually_happens by blast
    moreover have "dist c (of_int (f x)) ≥ setdist {c} ℤ"
      by (rule setdist_le_dist) auto
    ultimately show ?thesis
      by (simp add: dist_commute)
  next
    case True
    then obtain c' where c: "c = of_int c'"
      by (elim Ints_cases)
    have "eventually (λx. dist (of_int (f x)) c < 1) F"
      using * unfolding tendsto_iff by auto
    hence "eventually (λx. f x = c') F"
      by eventually_elim (auto simp: c dist_of_int)
    with c show ?thesis
      by auto
  qed
qed

lemma filterlim_of_nat_iff:
  fixes c :: "'a :: real_normed_algebra_1"
  assumes "F ≠ bot"
  shows "filterlim (λx. of_nat (f x)) (nhds c) F ⟷
          (∃c'. c = of_nat c' ∧ eventually (λx. f x = c') F)"
proof
  assume "∃c'. c = of_nat c' ∧ eventually (λx. f x = c') F"
  then obtain c' where c': "c = of_nat c'" "eventually (λx. f x = c')
F"
    by blast
  from c'(2) have "eventually (λx. of_nat (f x) = c) F"
    by eventually_elim (auto simp: c'(1))
  thus "filterlim (λx. of_nat (f x)) (nhds c) F"
    by (rule tendsto_eventually)
next
  assume *: "filterlim (λx. of_nat (f x)) (nhds c) F"
  show "(∃c'. c = of_nat c' ∧ eventually (λx. f x = c') F)"
  proof (cases "c ∈ ℕ")
    case False
    hence "setdist {c} ℕ > 0"
      by (subst setdist_gt_0_compact_closed) auto
    with * have "eventually (λx. dist (of_nat (f x)) c < setdist {c}
ℕ) F"
      unfolding tendsto_iff by blast
```

```
    then obtain x where "dist (of_nat (f x)) c < setdist {c} ℕ"
      using ‹F ≠ bot› eventually_happens by blast
    moreover have "dist c (of_nat (f x)) ≥ setdist {c} ℕ"
      by (rule setdist_le_dist) auto
    ultimately show ?thesis
      by (simp add: dist_commute)
  next
    case True
    then obtain c' where c: "c = of_nat c'"
      by (elim Nats_cases)
    have "eventually (λx. dist (of_nat (f x)) c < 1) F"
      using * unfolding tendsto_iff by auto
    hence "eventually (λx. f x = c') F"
      by eventually_elim (auto simp: c dist_of_nat)
    with c show ?thesis
      by auto
  qed
qed

lemma uniform_limit_compose:
  assumes "uniform_limit B (λx y. f x y) (λy. f' y) F" "⋀y. y ∈ A ⟹
g y ∈ B"
  shows   "uniform_limit A (λx y. f x (g y)) (λy. f' (g y)) F"
proof -
  have "uniform_limit (g ` A) (λx y. f x y) (λy. f' y) F"
    using assms(1) by (rule uniform_limit_on_subset) (use assms(2) in
blast)
  thus "uniform_limit A (λx y. f x (g y)) (λy. f' (g y)) F"
    unfolding uniform_limit_iff by auto
qed

lemma uniform_limit_const':
  assumes "filterlim f (nhds c) F"
  shows   "uniform_limit A (λx y. f x) (λy. c) F"
proof -
  have "∀⇩F n in F. ∀x∈A. dist (f n) c < ε" if ε: "ε > 0" for ε :: real
  proof -
    from assms and ε have "∀⇩F n in F. dist (f n) c < ε"
      unfolding tendsto_iff by blast
    thus ?thesis
      by eventually_elim auto
  qed
  thus ?thesis
    unfolding uniform_limit_iff by blast
qed

lemma uniform_limit_singleton_iff [simp]:
  "uniform_limit {x} f g F ⟷ filterlim (λy. f y x) (nhds (g x)) F"
  by (simp add: uniform_limit_iff tendsto_iff)
```

end

2 Some Facts About Number-Theoretic Functions

theory *Number_Theoretic_Functions_Extras*
imports
"Dirichlet_Series.Dirichlet_Series_Analysis"
"Dirichlet_Series.Divisor_Count"
Lambert_Series_Library
begin

lemma (**in** *nat_power_field*) *nat_power_minus:*
"a ≠ 0 ∨ n ≠ 0 ⟹ nat_power n (-a) = inverse (nat_power n a)"
using *nat_power_diff[of n 0 a]* **by** *(cases "n = 0") (simp_all add: field_simps)*

lemma *divisor_sigma_minus:*
fixes *a :: "'a :: {nat_power_field, field_char_0}"*
shows *"divisor_sigma (-a) n = divisor_sigma a n / nat_power n a"*
proof *(cases "n = 0")*
case *n: False*
have *"divisor_sigma (-a :: 'a) n = (∑ d | d dvd n. nat_power d (-a))"*
by *(simp add: divisor_sigma_def)*
also have *"... = (∑ d | d dvd n. nat_power d a / nat_power n a)"*
using *n* **by** *(intro sum.reindex_bij_witness[of _ "λd. n div d" "λd. n div d"])*
(auto elim!: dvdE simp: field_simps nat_power_minus nat_power_mult_distrib)
also have *"... = divisor_sigma a n / nat_power n a"*
by *(simp add: sum_divide_distrib divisor_sigma_def)*
finally show *?thesis* **.**
qed *auto*

lemma *norm_moebius_mu:*
"norm (moebius_mu n ::'a :: {real_normed_algebra_1, comm_ring_1}) = ind squarefree n"
by *(subst of_int_moebius_mu [symmetric], subst norm_of_int) (auto simp: abs_moebius_mu)*

lemma *conv_radius_nat_power: "conv_radius (λn. nat_power n a :: 'a :: {nat_power_normed_field, banach}) = 1"*
proof *(rule tendsto_imp_conv_radius_eq)*
show *"(λn. ereal (norm (nat_power n a :: 'a) powr (1 / real n))) ⟶ ereal 1"*
proof *(rule Lim_transform_eventually)*
show *"(λn. ereal ((real n powr (a · 1)) powr (1 / real n))) ⟶ ereal 1"*
by *(rule tendsto_ereal) real_asymp*

```
    show "eventually (λn. (real n powr (a · 1)) powr (1 / real n) =
            ereal (norm (nat_power n a :: 'a) powr (1 / real n))) at_top"
      using eventually_gt_at_top[of 0] by eventually_elim (simp add: norm_nat_power)
  qed
qed (simp_all add: one_ereal_def)

lemma not_convergent_liouville_lambda:
  "¬convergent (liouville_lambda :: nat ⇒ 'a :: {real_normed_algebra,
comm_ring_1, semiring_char_0})"
proof -
  have "¬(liouville_lambda :: nat ⇒ 'a) ⟶ c" for c
  proof (rule oscillation_imp_not_tendsto)
    show "eventually (λn. liouville_lambda (2 ^ (2 * n)) ∈ {1 :: 'a})
sequentially"
      by auto
    show "filterlim (λn. 2 ^ (2 * n) :: nat) at_top sequentially"
      by real_asymp
    show "eventually (λn. liouville_lambda (2 ^ (2 * n + 1)) ∈ {-1 ::
'a}) sequentially"
      by (subst liouville_lambda.power) auto
    show "filterlim (λn. 2 ^ (2 * n + 1) :: nat) at_top sequentially"
      by real_asymp
  qed auto
  thus ?thesis
    by (auto simp: convergent_def)
qed

lemma conv_radius_liouville_lambda:
  "conv_radius (liouville_lambda :: nat ⇒ 'a :: {real_normed_field, banach})
= 1"
proof -
  have "¬summable (liouville_lambda :: nat ⇒ 'a)"
    using not_convergent_liouville_lambda[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  hence "conv_radius (liouville_lambda :: nat ⇒ 'a) ≤ norm (1 :: 'a)"
    by (intro conv_radius_leI') auto
  moreover have "conv_radius (liouville_lambda :: nat ⇒ 'a) ≥ 1"
  proof (rule conv_radius_bigo_polynomial)
    show "(liouville_lambda :: nat ⇒ 'a) ∈ O(λn. of_nat n ^ 0)"
      by (intro bigoI[of _ 1] eventually_mono[OF eventually_gt_at_top[of
0]])
         (auto simp: liouville_lambda_def norm_power)
  qed
  ultimately show ?thesis
    by (intro antisym) (auto simp: one_ereal_def)
qed

lemma not_convergent_mangoldt: "¬convergent (mangoldt :: nat ⇒ 'a ::
{real_normed_algebra_1})"
```

```
proof -
  have *: "¬primepow (6 * n :: nat)" for n
    by (rule not_primepowI[of 2 3]) auto
  have "¬(mangoldt :: nat ⇒ 'a) ⟶ c" for c
  proof (rule oscillation_imp_not_tendsto)
    show "eventually (λn. mangoldt (2 ^ n) ∈ {of_real (ln 2) :: 'a})
sequentially"
      by (auto simp: mangoldt_primepow)
    show "filterlim (λn. 2 ^ n :: nat) at_top sequentially"
      by real_asymp
    show "eventually (λn. mangoldt (6 * n) ∈ {0 :: 'a}) sequentially"
      using * by (auto simp: mangoldt_def)
    show "filterlim (λn. 6 * n :: nat) at_top sequentially"
      by real_asymp
  qed auto
  thus ?thesis
    by (auto simp: convergent_def)
qed

lemma conv_radius_mangoldt:
  "conv_radius (mangoldt :: nat ⇒ 'a :: {real_normed_field, banach})
= 1"
proof -
  have "¬summable (mangoldt :: nat ⇒ 'a)"
    using not_convergent_mangoldt[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  hence "conv_radius (mangoldt :: nat ⇒ 'a) ≤ norm (1 :: 'a)"
    by (intro conv_radius_leI') auto
  moreover have "conv_radius (mangoldt :: nat ⇒ 'a) ≥ 1"
  proof (rule conv_radius_bigo_polynomial)
    have "(mangoldt :: nat ⇒ 'a) ∈ O(λn. of_real (ln (real n)))"
      by (intro bigoI[of _ 1] eventually_mono[OF eventually_gt_at_top[of
0]])
         (auto simp: mangoldt_le)
    also have "(λn. of_real (ln (real n))) ∈ O(λn. of_nat n :: 'a)"
      by (subst landau_o.big.norm_iff [symmetric], unfold norm_of_real
norm_of_nat) real_asymp
    finally show "(mangoldt :: nat ⇒ 'a) ∈ O(λn. of_nat n ^ 1)"
      by simp
  qed
  ultimately show ?thesis
    by (intro antisym) (auto simp: one_ereal_def)
qed

lemma not_convergent_moebius_mu: "¬convergent (moebius_mu :: nat ⇒ 'a
:: real_normed_field)"
proof (rule oscillation_imp_not_convergent)
  have "infinite {p. prime (p :: nat)}"
    by (rule primes_infinite)
```

```
  hence "frequently (prime :: nat ⇒ bool) cofinite"
    by (simp add: Inf_many_def)
  hence "frequently (λn. moebius_mu n = (-1 :: 'a)) cofinite"
    by (rule frequently_elim1) (simp add: moebius_mu.prime)
  thus "frequently (λn. moebius_mu n ∈ {(-1 :: 'a)}) sequentially"
    using cofinite_eq_sequentially by fastforce
next
  have "infinite (range (λn. 4 * n :: nat))"
    by (subst finite_image_iff) (auto simp: inj_on_def)
  moreover {
    have "¬squarefree (2 ^ 2 :: nat)"
      by (subst squarefree_power_iff) auto
    also have "2 ^ 2 = (4 :: nat)"
      by simp
    finally have "range (λn. 4 * n :: nat) ⊆ {n::nat. ¬squarefree n}"
      by (auto dest: squarefree_multD)
  }
  ultimately have "frequently (λn::nat. ¬squarefree n) cofinite"
    unfolding INFM_iff_infinite using finite_subset by blast
  thus "∃F n in sequentially. moebius_mu n ∈ {0}"
    unfolding cofinite_eq_sequentially by (rule frequently_elim1) auto
qed auto

lemma conv_radius_moebius_mu:
  "conv_radius (moebius_mu :: nat ⇒ 'a :: {real_normed_field, banach})
= 1"
proof -
  have "¬summable (moebius_mu :: nat ⇒ 'a)"
    using not_convergent_moebius_mu[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  hence "conv_radius (moebius_mu :: nat ⇒ 'a) ≤ norm (1 :: 'a)"
    by (intro conv_radius_leI') auto
  moreover have "conv_radius (moebius_mu :: nat ⇒ 'a) ≥ conv_radius
(λ_. 1 :: 'a)"
    by (intro conv_radius_mono always_eventually) (auto simp: norm_moebius_mu
ind_def)
  ultimately show ?thesis
    by (intro antisym) (auto simp: one_ereal_def)
qed

lemma not_convergent_totient:
  "¬convergent (λn. of_nat (totient n) :: 'a :: {real_normed_field, banach})"
proof
  assume "convergent (λn. of_nat (totient n) :: 'a)"
  then obtain L where L: "eventually (λn. totient n = L) at_top"
    by (auto simp: convergent_def filterlim_of_nat_iff)
  then obtain N where N: "⋀n. n ≥ N ⟹ totient n = L"
    unfolding eventually_at_top_linorder by blast
  obtain p q where "prime p" "p > N" "prime q" "q > p"
```

```
    using bigger_prime by blast
  with N[of p] N[of q] show False
    by (auto simp: totient_prime)
qed

lemma conv_radius_totient:
  "conv_radius (λn. of_nat (totient n) :: 'a :: {real_normed_field, banach})
= 1"
proof -
  have "¬summable (λn. of_nat (totient n) :: 'a)"
    using not_convergent_totient[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  hence "conv_radius (λn. of_nat (totient n) :: 'a) ≤ norm (1 :: 'a)"
    by (intro conv_radius_leI') auto
  moreover have "conv_radius (λn. of_nat (totient n) :: 'a) ≥ 1"
  proof (rule conv_radius_bigo_polynomial)
    show "(λn. of_nat (totient n)) ∈ O(λn. of_nat n ^ 1)"
      by (intro bigoI[of _ 1] always_eventually) (auto simp: totient_le)
  qed
  ultimately show ?thesis
    by (intro antisym) (auto simp: one_ereal_def)
qed

end
```

3 Some Abel-Style Summation Tests

```
theory Summation_Tests_More
  imports "HOL-Analysis.Analysis" "HOL-Library.Landau_Symbols" Lambert_Series_Library
begin
```

The following five summation tests are taking from Chapter 10 of Knopp's textbook [**?**]. He introduces a strong variant of Abel's summation test and then deduces from it four summation test named after Abel, Dirichlet, du Bois-Reymond, and Dedekind.

```
lemma abel_partial_summation:
  fixes f g :: "nat ⇒ 'a :: comm_ring_1"
  defines "F ≡ (λn. ∑k≤n. f k)"
  shows   "(∑r=n+1..n+k. f r * g r) =
             (∑r=n+1..n+k. F r * (g r - g (Suc r))) -
             F n * g (Suc n) + F (n + k) * g (n + k + 1)"
  by (induction k) (auto simp: F_def algebra_simps)

theorem abel_summation_test_strong:
  fixes f g :: "nat ⇒ 'a :: {real_normed_field, banach}"
  defines "F ≡ (λn. ∑k≤n. f k)"
  assumes "summable (λr. F r * (g r - g (Suc r)))"
  assumes "convergent (λr. F r * g (Suc r))"
```

```
  shows   "summable (λr. f r * g r)"
  unfolding summable_iff_convergent'
proof (rule Cauchy_convergent)
  show "Cauchy (λn. ∑r≤n. f r * g r)"
    unfolding Cauchy_def
  proof safe
    fix ε :: real assume ε: "ε > 0"
    let ?A = "λn. (∑r≤n. F r * (g r - g (Suc r)))"
    from assms(2) have "Cauchy (λn. ∑r≤n. F r * (g r - g (Suc r)))"
      unfolding summable_iff_convergent' using convergent_Cauchy by blast
    moreover have "ε / 2 > 0"
      using ε by auto
    ultimately obtain M1 where M1: "dist (?A m) (?A n) < ε / 2" if "m
≥ M1" "n ≥ M1" for m n
      unfolding Cauchy_def by fast
    have M1': "norm (∑r=m..n. F r * (g r - g (Suc r))) < ε / 2" if "M1
< m" "m ≤ Suc n" for m n
    proof -
      have "dist (?A n) (?A (m - 1)) < ε / 2"
        by (rule M1) (use that in auto)
      also have "dist (?A n) (?A (m - 1)) = norm (∑r∈{..n}-{..m - 1}.
F r * (g r - g (Suc r)))"
        unfolding dist_norm using that by (subst Groups_Big.sum_diff)
auto
      also have "{..n} - {..m - 1} = {m..n}"
        using that by auto
      finally show ?thesis .
    qed

    from ε have "ε / 4 > 0"
      by auto
    from assms(3) obtain c where "(λr. F r * g (Suc r)) ⟶ c"
      by (auto simp: convergent_def)
    with ‹ε / 4 > 0› have "eventually (λr. dist (F r * g (Suc r)) c <
ε / 4) sequentially"
      using tendstoD by blast
    then obtain M2 where M2: "dist (F r * g (Suc r)) c < ε / 4" if "r
≥ M2" for r
      unfolding eventually_at_top_linorder by blast

    show "∃M. ∀m≥M. ∀n≥M. dist (∑r≤m. f r * g r) (∑r≤n. f r * g
r) < ε"
    proof (rule exI[of _ "max M1 M2"], safe)
      fix m n :: nat assume "m ≥ max M1 M2" "n ≥ max M1 M2"
      thus "dist (∑r≤m. f r * g r) (∑r≤n. f r * g r) < ε"
      proof (induction m n rule: linorder_wlog)
        case (le m n)
        define k where "k = n - m"
        from le have n_eq: "n = m + k"
```

```
      by (auto simp: k_def)
    have "dist (∑r≤m. f r * g r) (∑r≤n. f r * g r) =
          norm ((∑r≤n. f r * g r) - (∑r≤m. f r * g r))"
      by (simp add: dist_norm norm_minus_commute)
    also have "(∑r≤n. f r * g r) - (∑r≤m. f r * g r) = (∑r∈{..n}-{..m}.
f r * g r)"
      using le by (subst Groups_Big.sum_diff) auto
    also have "{..n} - {..m} = {m+1..m+k}"
      by (auto simp: n_eq)
    also have "(∑r=m+1..m+k. f r * g r) =
                 (∑r=m+1..m+k. F r * (g r - g (Suc r))) -
                  F m * g (Suc m) + F (m + k) * g (Suc (m + k))"
      unfolding F_def by (subst abel_partial_summation) simp_all
    also have "norm ... ≤ norm (∑r=m+1..m+k. F r * (g r - g (Suc
r))) +
                    dist (F m * g (Suc m)) c + dist (F (m + k) * g (Suc
(m + k))) c"
      by norm
    also have "... < ε / 2 + ε / 4 + ε / 4"
      using le by (intro add_strict_mono M1' M2) auto
    also have "... = ε"
      by simp
    finally show "dist (∑r≤m. f r * g r) (∑r≤n. f r * g r) < ε"
.
  qed (simp add: dist_commute max.commute)
 qed
qed
qed

corollary abel_summation_test:
  fixes f g :: "nat ⇒ real"
  assumes "summable f"
  assumes "incseq g" "g ∈ O(λ_. 1)"
  shows   "summable (λr. f r * g r)"
proof (rule abel_summation_test_strong)
  have "convergent g"
    by (rule incseq_convergent') fact+
  thus "convergent (λn. (∑k≤n. f k) * g (Suc n))" using assms(1)
    by (intro convergent_mult) (simp_all add: convergent_Suc_iff summable_iff_convergent')
  show "summable (λn. (∑k≤n. f k) * (g n - g (Suc n)))"
  proof (subst mult.commute, rule summable_norm_cancel, rule norm_summable_mult_bounded)
    have "summable (λn. g (Suc n) - g n)"
      using ‹convergent g› by (simp add: telescope_summable_iff)
    also have "(λn. g (Suc n) - g n) = (λn. norm (g n - g (Suc n)))"
      using ‹incseq g› by (auto simp: incseq_def fun_eq_iff)
    finally show "summable (λn. norm (g n - g (Suc n)))" .
  next
    have "bounded (range (λn. ∑k<n. f k))"
      by (rule summable_imp_sums_bounded) fact
```

```
    hence "(λn. ∑k<n. f k) ∈ O(λ_. 1)"
      by (simp add: seq_bigo_1_iff)
    hence "(λn. ∑k<Suc n. f k) ∈ O(λ_. 1)"
      by (rule landau_o.big.compose) (rule filterlim_Suc)
    also have "(λn. {..<Suc n}) = (λn. {..n})"
      by auto
    finally show "(λn. ∑k≤n. f k) ∈ O(λ_. 1)" .
  qed
qed

corollary dirichlet_summation_test:
  fixes f g :: "nat ⇒ real"
  assumes "(λn. ∑r≤n. f r) ∈ O(λ_. 1)"
  assumes "decseq g" "g ∈ o(λ_. 1)"
  shows   "summable (λr. f r * g r)"
proof (rule abel_summation_test_strong)
  have "(λx. g (Suc x)) ∈ o(λx. 1)"
    using assms(3) by (rule landau_o.small.compose) (rule filterlim_Suc)
  have "(λn. (∑r≤n. f r) * g (Suc n)) ∈ o(λ_. 1 * 1)"
    by (rule landau_o.big_small_mult) fact+
  thus "convergent (λr. sum f {..r} * g (Suc r))"
    by (auto dest!: smalloD_tendsto simp: convergent_def)
next
  have "g ⟶ 0"
    using assms(3) by (auto dest!: smalloD_tendsto simp: convergent_def)
  hence "convergent g"
    by (auto simp: convergent_def)
  show "summable (λn. (∑k≤n. f k) * (g n - g (Suc n)))"
  proof (subst mult.commute, rule summable_norm_cancel, rule norm_summable_mult_bounded)
    have "summable (λn. g n - g (Suc n))"
      using ‹convergent g› by (simp add: telescope_summable_iff')
    also have "(λn. g n - g (Suc n)) = (λn. norm (g n - g (Suc n)))"
      using ‹decseq g› by (auto simp: decseq_Suc_iff fun_eq_iff)
    finally show "summable (λn. norm (g n - g (Suc n)))" .
  qed fact
qed

corollary dubois_reymond_summation_test:
  fixes f g :: "nat ⇒ 'a :: {real_normed_field, banach}"
  assumes "summable f"
  assumes "summable (λr. norm (g r - g (Suc r)))"
  shows   "summable (λr. f r * g r)"
proof (rule abel_summation_test_strong)
  have "summable (λr. g r - g (Suc r))"
    using assms(2) by (rule summable_norm_cancel)
  hence "convergent g"
    by (subst (asm) telescope_summable_iff')
  show "convergent (λr. sum f {..r} * g (Suc r))"
    using assms(1) ‹convergent g›
```

```
    by (intro convergent_mult) (auto simp: convergent_Suc_iff summable_iff_convergent')

  show "summable (λn. (∑k≤n. f k) * (g n - g (Suc n)))"
  proof (subst mult.commute, rule summable_norm_cancel, rule norm_summable_mult_bounded)
    have "bounded (range (λn. ∑k<n. f k))"
      by (rule summable_imp_sums_bounded) fact
    hence "(λn. ∑k<n. f k) ∈ O(λ_. 1)"
      by (simp add: seq_bigo_1_iff)
    hence "(λn. ∑k<Suc n. f k) ∈ O(λ_. 1)"
      by (rule landau_o.big.compose) (rule filterlim_Suc)
    also have "(λn. {..<Suc n}) = (λn. {..n})"
      by auto
    finally show "(λn. ∑k≤n. f k) ∈ O(λ_. 1)" .
  qed fact
qed

corollary dedekind_summation_test:
  fixes f g :: "nat ⇒ 'a :: {real_normed_field, banach}"
  assumes "(λn. ∑k≤n. f k) ∈ O(λ_. 1)"
  assumes "summable (λr. norm (g r - g (Suc r)))"
  assumes "g ∈ o(λ_. 1)"
  shows   "summable (λr. f r * g r)"
proof (rule abel_summation_test_strong)
  have "(λx. g (Suc x)) ∈ o(λx. 1)"
    using assms(3) by (rule landau_o.small.compose) (rule filterlim_Suc)
  have "(λn. (∑r≤n. f r) * g (Suc n)) ∈ o(λ_. 1 * 1)"
    by (rule landau_o.big_small_mult) fact+
  thus "convergent (λr. sum f {..r} * g (Suc r))"
    by (auto dest!: smalloD_tendsto simp: convergent_def)
  show "summable (λn. (∑k≤n. f k) * (g n - g (Suc n)))"
    by (subst mult.commute, rule summable_norm_cancel, rule norm_summable_mult_bounded)
fact+
qed

end
```

4 Lambert Series

```
theory Lambert_Series
  imports
  "HOL-Complex_Analysis.Complex_Analysis"
  "HOL-Real_Asymp.Real_Asymp"
  "Dirichlet_Series.Dirichlet_Series_Analysis"
  "Dirichlet_Series.Divisor_Count"
  Polylog.Polylog
  Lambert_Series_Library
  Number_Theoretic_Functions_Extras
  Summation_Tests_More
begin
```

4.1 Definition

Given any sequence $a(n)$ for $n \geq 1$, the corresponding *Lambert series* is defined as

$$L(a,q) = \sum_{n=1}^{\infty} a(n) \frac{q^n}{1-q^n} \ .$$

```
definition lambert :: "(nat ⇒ 'a :: {real_normed_field, banach}) ⇒ 'a
⇒ 'a" where
  "lambert a q =
     (let f = (λn. a (Suc n) * q ^ (Suc n) / (1 - q ^ (Suc n))) in
      if  summable f then ∑n. f n else 0)"

lemma lambert_eqI:
  assumes "(λn. a (Suc n) * q ^ (Suc n) / (1 - q ^ (Suc n))) sums x"
  shows   "lambert a q = x"
  using assms unfolding lambert_def Let_def sums_iff by simp

lemma lambert_cong [cong]:
  "(⋀n. n > 0 ⟹ a n = a' n) ⟹ q = q' ⟹ lambert a q = lambert a'
q'"
  by (simp add: lambert_def)

lemma lambert_0 [simp]: "lambert a 0 = 0"
  by (simp add: lambert_def)

lemma lambert_0' [simp]: "lambert (λ_. 0) q = 0"
  by (simp add: lambert_def)

lemma lambert_cmult: "lambert (λn. c * a n) q = c * lambert a q"
proof (cases "c = 0")
  case False
  define f where "f = (λn. a (Suc n) * q ^ (Suc n) / (1 - q ^ (Suc n)))"
  show ?thesis
  proof (cases "summable f")
    case True
    hence "(λn. c * (a (Suc n) * q ^ (Suc n) / (1 - q ^ (Suc n)))) sums
(c * (∑n. f n))"
      unfolding mult.assoc by (intro sums_mult) (auto simp: f_def)
    thus ?thesis using True
      by (intro lambert_eqI) (auto simp: lambert_def f_def algebra_simps)
  next
    case False
    hence "¬summable (λn. c * f n)"
      using ‹c ≠ 0› by simp
    with False show ?thesis
      by (simp add: lambert_def f_def algebra_simps)
  qed
qed auto
```

```
lemma lambert_cmult': "lambert (λn. a n * c) q = lambert a q * c"
  using lambert_cmult[of c a q] by (simp add: mult_ac)

lemma lambert_uminus: "lambert (λn. -a n) q = -lambert a q"
  using lambert_cmult[of "-1" a q] by simp
```

We will later see that if $\sum_{n=1}^{\infty} a(n)$ exists then the Lambert series converges everywhere except on the unit circle; otherwise it has the same convergence radius as a (and that radius then has to be < 1).

```
definition lambert_conv_radius :: "(nat ⇒ 'a :: {banach, real_normed_field})
⇒ ereal"
  where "lambert_conv_radius a = (if summable a then ∞ else conv_radius
a)"

lemma lambert_conv_radius_gt_1_iff: "lambert_conv_radius a > 1 ⟷ summable
a"
proof
  assume *: "lambert_conv_radius a > 1"
  {
    assume "¬summable a"
    hence "conv_radius a > 1"
      using * by (auto simp: lambert_conv_radius_def)
    hence "summable (λn. a n * 1 ^ n)"
      by (intro summable_in_conv_radius) (auto simp: one_ereal_def)
    with ‹¬summable a› have False
      by simp
  }
  thus "summable a"
    by blast
qed (auto simp: lambert_conv_radius_def)
```

4.2 Uniform convergence, continuity, holomorphicity

We will now show some (uniform) convergence results for $L(a, q)$, which will then give us the holomorphicity and continuity of $L(a, q)$. We will also show some absolute summability results.

```
context
  fixes a :: "nat ⇒ 'a :: {real_normed_field, banach}"
  fixes f :: "nat ⇒ 'a ⇒ 'a" and A :: "'a"
  defines "f ≡ λk q. a k * q ^ k / (1 - q ^ k)"
  defines "A ≡ (∑n. a (Suc n))"
begin
```

Let $a(n)$ have convergence radius r. In discs of radius $\min(1, r)$, the Lambert series for $a(n)$ converges uniformly. This is a simple application of Weierstraß's M test.

```
lemma uniform_limit_lambert1_aux:
```

```
  fixes r :: real
  assumes "0 < r" "r < min 1 (conv_radius a)"
  shows   "uniform_limit (ball 0 r) (λn q. (∑k<n. f (Suc k) q)) (λq.
∑k. f (Suc k) q) sequentially"
proof -
  from assms have r: "r > 0" "r < 1" "r < conv_radius a"
    by auto
  show "uniform_limit (ball 0 r) (λn q. (∑k<n. f (Suc k) q)) (λq. ∑k.
f (Suc k) q) sequentially"
  proof (rule Weierstrass_m_test_ev)
    have "eventually (λk. 1 - r ^ k ≥ 1 / 2) at_top"
      using r by real_asymp
    hence "eventually (λk. ∀q∈ball 0 r. norm (f k q) ≤ 2 * norm (a k)
* r ^ k) at_top"
      using eventually_gt_at_top[of 0]
    proof eventually_elim
      case k: (elim k)
      show "∀q∈ball 0 r. norm (f k q) ≤ 2 * norm (a k) * r ^ k"
      proof
        fix q :: 'a assume q: "q ∈ ball 0 r"
        have "norm (f k q) = norm (a k) * norm q ^ k / norm (1 - q ^ k)"
          by (simp add: f_def norm_mult norm_divide norm_power)
        also {
          have "1 / 2 ≤ 1 - r ^ k"
            using k by simp
          also have "... ≤ norm (1 :: 'a) - norm (q ^ k)"
            using q by (auto simp: norm_power intro!: power_mono)
          also have "... ≤ norm (1 - q ^ k)"
            by norm
          finally have "norm (1 - q ^ k) ≥ 1 / 2" .
        }
        hence "norm (a k) * norm q ^ k / norm (1 - q ^ k) ≤
               norm (a k) * r ^ k / (1 / 2)"
          using q r k
          by (intro mult_mono power_mono frac_le)
             (auto intro!: mult_pos_pos simp: power_less_1_iff norm_power
dest!: power_eq_1_iff)
        finally show "norm (f k q) ≤ 2 * norm (a k) * r ^ k"
          by simp
      qed
    qed
    thus "∀F k in sequentially. ∀q∈ball 0 r. norm (f (Suc k) q) ≤ 2
* norm (a (Suc k)) * r ^ Suc k"
      by (rule eventually_compose_filterlim[OF _ filterlim_Suc])
  next
    have "summable (λk. 2 * (norm (a (Suc k) * of_real r ^ Suc k)))"
      by (subst summable_Suc_iff, intro summable_mult abs_summable_in_conv_radius)
(use r in auto)
    thus "summable (λk. 2 * norm (a (Suc k)) * r ^ Suc k)"
```

```
    using ‹r > 0› by (simp add: norm_mult norm_power mult_ac)
  qed
qed

lemma uniform_limit_lambert1:
  fixes r :: real
  assumes "0 < r" "r < min 1 (conv_radius a)"
  shows   "uniform_limit (ball 0 r) (λn q. (∑k<n. f (Suc k) q)) (lambert
a) sequentially"
proof -
  have lim: "uniform_limit (ball 0 r) (λn q. (∑k<n. f (Suc k) q)) (λq.
∑k. f (Suc k) q) sequentially"
    using assms by (rule uniform_limit_lambert1_aux)
  also have "?this ⟷ ?thesis"
  proof (intro uniform_limit_cong ballI allI refl always_eventually)
    fix q :: 'a assume q: "q ∈ ball 0 r"
    have *: "(λn. a (Suc n) * q ^ Suc n / (1 - q ^ Suc n)) sums (∑k.
f (Suc k) q)"
      using tendsto_uniform_limitI[OF lim q] unfolding f_def by (simp
add: sums_def)
    show "(∑k. f (Suc k) q) = lambert a q"
      by (rule sym, rule lambert_eqI) (fact *)
  qed
  finally show ?thesis .
qed
```

Since $a_n \frac{q^n}{1-q^n} = -a_n - a_n \frac{(\frac{1}{q})^n}{1-(\frac{1}{q})^n}$, we can substitute $q \mapsto \frac{1}{q}$ in the above uniform convergence result to deduce that uniform convergence also holds on any annulus $r \leq |q| \leq R$ with $1 < r < R$.

```
lemma uniform_limit_lambert2:
  fixes r R :: real
  assumes r: "1 < r" "r < R"
  assumes "summable a"
  defines "D ≡ cball 0 R - ball 0 r"
  shows   "uniform_limit D (λn q. (∑k<n. f (Suc k) q)) (λq. -A - lambert
a (1 / q)) sequentially"
proof -
  define g where "g = (λn q. a n * (1 / q) ^ n / (1 - (1 / q) ^ n))"
  from r(1) obtain r' where r': "1 < r'" "r' < r"
    using dense by blast
  have "uniform_limit D (λn q. ∑k<n. f (Suc k) (1 / q)) (λq. lambert
a (1 / q)) sequentially"
    using uniform_limit_lambert1
  proof (rule uniform_limit_compose[where g = "λq. 1 / q"])
    have "conv_radius a ≥ norm (of_real 1 :: 'a)"
      by (rule conv_radius_geI) (use ‹summable a› in auto)
    hence "min 1 (conv_radius a) = 1"
      by (simp add: min_def one_ereal_def)
```

```
    with r' show "1 / r' > 0" "ereal (1 / r') < min 1 (conv_radius a)"
      by auto
  next
    show "1 / q ∈ ball 0 (1 / r')" if "q ∈ D" for q
      using that r r' by (auto simp: D_def norm_divide divide_simps not_less)
  qed
  moreover have "summable (λn. a (Suc n))"
    using ‹summable a› by (simp add: summable_Suc_iff)
  hence "(λn. ∑k<n. a (Suc k)) ⟶ A"
    unfolding A_def summable_sums_iff sums_def .
  ultimately have "uniform_limit D (λn q. -(∑k<n. a (Suc k)) - (∑k<n.
f (Suc k) (1/q)))
          (λq. -A - lambert a (1 / q)) sequentially"
    by (intro uniform_limit_intros uniform_limit_const')
  also have "?this ⟷ ?thesis"
  proof (intro uniform_limit_cong refl always_eventually allI ballI)
    fix n :: nat and q :: 'a
    assume q: "q ∈ D"
    hence q': "q ≠ 0"
      using r by (auto simp: D_def)
    have q'': "q ^ k ≠ 1" if "k > 0" for k
      using q r that by (auto dest!: power_eq_1_iff simp: D_def)
    have "-(∑k<n. a (Suc k)) - (∑k<n. f (Suc k) (1 / q)) =
          (∑k<n. -a (Suc k) - f (Suc k) (1 / q))"
      by (simp add: sum_negf sum_subtractf)
    also have *: "-a k - f k (1 / q) = f k q" if "k > 0" for k
      using that q' q''[of k] by (auto simp: f_def field_simps)
    have "(∑k<n. -a (Suc k) - f (Suc k) (1 / q)) = (∑k<n. f (Suc k)
q)"
      by (intro sum.cong refl *) auto
    finally show "-(∑k<n. a (Suc k)) - (∑k<n. f (Suc k) (1 / q)) = (∑k<n.
f (Suc k) q)" .
  qed
  finally show ?thesis .
qed
```

With some more book-keeping, we show that the series converges uniformly in all compact sets that do not touch the unit circle and, if $\sum_{n=1}^{\infty} a(n)$ does not exist, lie fully within the convergence radius of $a(n)$. This is mentioned in Knopp's Theorem 259.

```
theorem uniform_limit_lambert:
  assumes "compact X" "X ⊆ eball 0 (lambert_conv_radius a) - sphere 0
1"
  shows   "uniform_limit X (λn q. (∑k<n. f (Suc k) q)) (lambert a) sequentially"
proof -
  from assms have norm_neq_1: "norm x ≠ 1" if "x ∈ X" for x
    using that by auto
  have 1: "uniform_limit (X ∩ cball 0 1) (λn q. (∑k<n. f (Suc k) q))
(lambert a) sequentially"
```

```
  proof (cases "X ∩ cball 0 1 ⊆ {0}")
    case True
    hence "X ∩ cball 0 1 = {} ∨ X ∩ cball 0 1 = {0}"
      by auto
    thus ?thesis
      by (elim disjE) (simp_all add: f_def lambert_def)
  next
    case False
    obtain r :: real
      where r: "r > 0" "r < 1" "r < conv_radius a" "⋀x. x ∈ X ∩ cball
0 1 ⟹ norm x < r"
    proof -
      define c where "c = (if lambert_conv_radius a > 1 then 1
                           else real_of_ereal (lambert_conv_radius a))"
      have "compact ((ereal ∘ norm) ` (X ∩ cball 0 1))"
        by (intro compact_continuous_image continuous_intros compact_insert)
(use assms in auto)
      hence "Sup ((ereal ∘ norm) ` (X ∩ cball 0 1)) ∈ (ereal ∘ norm)
` (X ∩ cball 0 1)"
        using False by (intro closed_contains_Sup_cl) (auto intro: compact_imp_closed)
      then obtain q where q: "q ∈ (X ∩ cball 0 1)"
        "ereal (norm q) = Sup ((ereal ∘ norm) ` (X ∩ cball 0 1))"
        unfolding o_def by force
      have q': "norm q' ≤ norm q" if "q' ∈ X ∩ cball 0 1" for q'
        using Sup_upper q that unfolding o_def by (metis ereal_less_eq(3)
imageI insertCI)
      have "q ≠ 0"
        using q' False by force
      have "conv_radius a ≥ norm (1 :: 'a)" if "summable a"
        by (rule conv_radius_geI) (use that in auto)
      hence "norm q < conv_radius a" "norm q < 1"
        using q(1) assms ‹q ≠ 0›
        by (auto simp: lambert_conv_radius_def eball_def ereal_le_less
split: if_splits)
      then obtain r where r: "norm q < r" "r < 1" "r < conv_radius a"
        by (smt (verit, ccfv_SIG) ereal_dense2 ereal_less(3) less_ereal.simps(1)
linorder_not_le order_less_le_trans)
      show ?thesis
      proof (rule that[of r])
        show "r > 0"
          using r q ‹q ≠ 0› by (smt (verit) norm_ge_zero)
        show "r < 1" "ereal r < conv_radius a"
          using r by auto
        show "norm q' < r" if "q' ∈ X ∩ cball 0 1" for q'
          using assms q q' that ‹q ≠ 0› r
          by (force simp: lambert_conv_radius_def eball_def split: if_splits)
      qed
    qed
    have "uniform_limit (ball 0 r) (λn q. (∑k<n. f (Suc k) q)) (lambert
```

```
a) sequentially"
      by (rule uniform_limit_lambert1) (use r in auto)
    thus "uniform_limit (X ∩ cball 0 1) (λn q. (∑k<n. f (Suc k) q))
(lambert a) sequentially"
      by (rule uniform_limit_on_subset) (use r in auto)
  qed

  have 2: "uniform_limit (X - ball 0 1) (λn q. (∑k<n. f (Suc k) q))
(lambert a) sequentially"
  proof (cases "∃q∈X. norm q > 1")
    case False
    hence *: "X - ball 0 1 = {}"
      using norm_neq_1 by fastforce
    show ?thesis
      unfolding * by simp
  next
    case True
    then obtain q where q: "q ∈ X" "norm q > 1"
      by auto
    with assms have "lambert_conv_radius a > 1"
      by (smt (verit, ccfv_SIG) Diff_subset dist_0_norm ereal_less(3)
in_eball_iff linorder_not_le order_less_le_trans subsetD)
    hence "summable a"
      by (simp add: lambert_conv_radius_gt_1_iff)
    obtain r where r: "r > 1" "⋀x. x ∈ X - cball 0 1 ⟹ norm x > r"
    proof -
      have compact: "compact (norm ` (X - ball 0 1))"
        by (intro compact_continuous_image compact_diff continuous_intros)
(use assms in auto)
      hence "Inf (norm ` (X - ball 0 1)) ∈ norm ` (X - ball 0 1)"
        using q by (intro closed_contains_Inf)
                   (auto intro: compact_imp_closed bounded_imp_bdd_below
compact_imp_bounded)
      then obtain q' where q': "q' ∈ X - ball 0 1" "norm q' = Inf (norm
` (X - ball 0 1))"
        by force
      have "norm q' > 1"
        using q' assms by auto
      then obtain r where r: "1 < r" "r < norm q'"
        using dense by blast

      show ?thesis
      proof (rule that[of r])
        show "r > 1"
          using q' assms r by auto
        show "norm q'' > r" if "q'' ∈ X - cball 0 1" for q''
        proof -
          have "r < Inf (norm ` (X - ball 0 1))"
            using r q' by simp
```

```
        also have "... ≤ norm q''"
          by (rule cInf_lower)
             (use that assms compact in ‹auto intro!: bounded_imp_bdd_below
compact_imp_bounded›)
        finally show ?thesis .
      qed
    qed

    obtain R where R: "R > r" "⋀x. x ∈ X ⟹ norm x < R"
    proof -
      have "bounded X"
        using assms compact_imp_bounded by blast
      then obtain R where R: "norm x ≤ R" if "x ∈ X" for x
        unfolding bounded_iff by blast
      show ?thesis
      proof (rule that[of "R + 1"])
        show "r < R + 1"
          using r(2)[of q] q R[of q] by auto
        show "norm x < R + 1" if "x ∈ X" for x
          using R[of x] that by auto
      qed
    qed

    have lim: "uniform_limit (cball 0 R - ball 0 r) (λn q. (∑k<n. f
(Suc k) q))
                (λq. -A - lambert a (1 / q)) sequentially"
      by (rule uniform_limit_lambert2) (use r R ‹summable a› in auto)
    also have "?this ⟷ uniform_limit (cball 0 R - ball 0 r) (λn q. (∑k<n.
f (Suc k) q))
                          (lambert a) sequentially"
    proof (intro uniform_limit_cong refl always_eventually allI ballI)
      fix q :: 'a assume q: "q ∈ cball 0 R - ball 0 r"
      with lim have "(λn. (∑k<n. f (Suc k) q)) ⟶ -A - lambert a
(1 / q)"
        by (rule tendsto_uniform_limitI)
      hence "(λk. f (Suc k) q) sums (-A - lambert a (1 / q))"
        by (simp add: sums_def)
      thus "-A - lambert a (1 / q) = lambert a q"
        unfolding lambert_def[of a q] by (simp add: sums_iff f_def)
    qed
    finally show ?thesis
      by (rule uniform_limit_on_subset) (use r R norm_neq_1 in force)+
  qed

  have "uniform_limit ((X ∩ cball 0 1) ∪ (X - ball 0 1))
        (λn q. (∑k<n. f (Suc k) q)) (lambert a) sequentially"
    using 1 2 by (rule uniform_limit_on_Un)
  also have "(X ∩ cball 0 1) ∪ (X - ball 0 1) = X"
```

```
    using norm_neq_1 by auto
  finally show ?thesis .
qed

lemma sums_lambert:
  assumes "norm q < lambert_conv_radius a" "norm q ≠ 1"
  shows   "(λk. f (Suc k) q) sums lambert a q"
proof -
  have "(λn. (∑k<n. f (Suc k) q)) ⟶ lambert a q"
  proof (rule tendsto_uniform_limitI[OF uniform_limit_lambert])
    show "compact {q}" "{q} ⊆ eball 0 (lambert_conv_radius a) - sphere
0 1"
      using assms by auto
  qed auto
  thus ?thesis
    by (simp add: sums_def)
qed
```

A side effect of this: the functional equation

$$L(a, \tfrac{1}{q}) = -\big(\sum_{n=1}^{\infty} a(n)\big) - L(a, q)\ ,$$

which is valid for all q with $q \neq 0$ and $|q| \neq 1$ if $\sum_{n=1}^{\infty} a(n)$ exists.

```
theorem lambert_reciprocal:
  assumes "summable a" and "q ≠ 0" and "norm q ≠ 1"
  shows   "lambert a (1 / q) = -A - lambert a q"
proof -
  have *: "lambert a (1 / q) = -A - lambert a q" if q: "norm q > 1" for
q
  proof -
    obtain r where r: "1 < r" "r < norm q"
      using q dense by blast
    have "uniform_limit (cball 0 (norm q + 1) - ball 0 r)
            (λn q. ∑k<n. f (Suc k) q)
            (λq. - A - lambert a (1 / q)) sequentially"
      by (rule uniform_limit_lambert2) (use assms r in auto)
    hence "(λk. f (Suc k) q) sums (-A - lambert a (1 / q))"
      unfolding sums_def by (rule tendsto_uniform_limitI) (use r in auto)
    moreover have "uniform_limit {q} (λn q. ∑k<n. f (Suc k) q) (lambert
a) sequentially"
      using assms r
      by (intro uniform_limit_lambert compact_diff compact_cball)
         (auto simp: lambert_conv_radius_def)
    hence "(λk. f (Suc k) q) sums lambert a q"
      unfolding sums_def by (rule tendsto_uniform_limitI) (use r in auto)
    ultimately show ?thesis
      by (simp add: sums_iff algebra_simps)
  qed
```

```
  show ?thesis
  proof (cases "norm q > 1")
    case False
    thus ?thesis using assms
      using *[of "1 / q"] by (auto simp: norm_divide)
  qed (use *[of q] in auto)
qed

lemma summable_lambert:
  assumes "norm q < lambert_conv_radius a" "norm q ≠ 1"
  shows   "summable (λk. f k q)"
  using sums_lambert[OF assms] unfolding sums_iff by (subst (asm) summable_Suc_iff)
auto
```

We have shown that the Lambert series for $a(n)$ converges everywhere except on the unit circle if $\sum_{n=1}^{\infty} a(n)$ exists, and it converges within the convergence radius of R of $a(n)$ otherwise.

We will now show that within $\min(1, R)$, this convergence is absolute.

```
lemma norm_summable_lambert:
  assumes "norm q < min 1 (conv_radius a)"
  shows   "summable (λk. norm (f k q))"
proof (cases "q = 0")
  case [simp]: True
  have "eventually (λk. norm (f k q) = 0) at_top"
    using eventually_gt_at_top[of 0] by eventually_elim (auto simp: f_def)
  thus ?thesis
    using summable_cong by fastforce
next
  case False
  define R where "R = (if conv_radius a > 1 then 1 else real_of_ereal
(conv_radius a))"
  have R: "ereal R = min 1 (conv_radius a)"
    using conv_radius_nonneg[of a] by (cases "conv_radius a") (auto simp:
R_def)
  with assms have "norm q < R"
    by (metis less_ereal.simps(1))
  then obtain r where r: "norm q < r" "r < R"
    using dense by blast
  hence "r > 0"
    using norm_ge_zero le_less_trans by blast
  have "r < 1"
    using r R by (metis ereal_less(3) less_ereal.simps(1) min_less_iff_conj)
  have "r < conv_radius a"
    using r R by (metis less_ereal.simps(1) min_less_iff_conj)
  have "norm q < 1"
    using assms by auto
  note r' = ‹r > 0› ‹r < 1› ‹r < conv_radius a›
  show ?thesis
  proof (rule summable_powser_comparison_test_bigo)
```

```
    show "summable (λn. a n * of_real r ^ n)"
      by (rule summable_in_conv_radius) (use r r' R in auto)
  next
    have "(λn. norm (f n q)) = (λn. norm (a n) * norm q ^ n / norm (1
- q ^ n))"
      by (simp add: f_def norm_mult norm_divide norm_power)
    also have "1 - norm q ^ n ≤ norm (1 - q ^ n)" for n
      by (metis norm_one norm_power norm_triangle_ineq2)
    hence "(λn. norm (a n) * norm q ^ n / norm (1 - q ^ n)) ∈
                O(λn. norm (a n) * (norm q ^ n / (1 - norm q ^ n)))"
      using ‹q ≠ 0› ‹norm q < 1›
      by (intro bigoI[of _ 1] eventually_mono[OF eventually_gt_at_top[of
0]])
         (auto intro!: mult_left_mono divide_left_mono mult_pos_pos add_pos_pos

               dest!: power_eq_1_iff simp: power_le_one power_less_1_iff)
    also have "(λn. norm (a n) * (norm q ^ n / (1 - norm q ^ n))) ∈ O(λn.
norm (a n) * norm q ^ n)"
      by (intro landau_o.big.mult_left) (use ‹q ≠ 0› ‹norm q < 1› in
real_asymp)
    also have "(λn. norm (a n) * norm q ^ n) =
                 (λn. norm (a n * of_real r ^ n * (q / of_real r) ^ n))"
      using ‹r > 0› by (intro ext) (auto simp: norm_mult norm_divide norm_power
power_divide)
    finally show "(λn. f n q) ∈ O(λn. a n * of_real r ^ n * (q / of_real
r) ^ n)"
      by (subst (asm) landau_o.big.norm_iff)
  next
    show "norm (q / of_real r) < 1"
      using r r' by (simp add: norm_divide field_simps)
  qed
qed
```

If additionally $\sum_{k=1}^{\infty} a(k)$ converges absolutely, the absolute convergence of the Lambert series also holds everywhere.

```
lemma norm_summable_lambert':
  assumes "summable (λk. norm (a k))" and "norm q ≠ 1"
  shows   "summable (λk. norm (f k q))"
proof -
  have *: "summable (λk. norm (f k q))" if q: "norm q < 1" for q
  proof -
    have "conv_radius a ≥ norm (1 :: 'a)"
      using assms(1) by (intro conv_radius_geI) (auto dest: summable_norm_cancel)
    with q have "ereal (norm q) < conv_radius a"
      by (simp add: ereal_le_less)
    thus ?thesis
      using assms(2) q by (intro norm_summable_lambert) auto
  qed
```

```
  show ?thesis
  proof (cases "norm q < 1")
    case True
    thus ?thesis using *[of q] by simp
  next
    case False
    hence [simp]: "q ≠ 0"
      by auto
    have [simp]: "q ^ k = 1 ⟷ k = 0" for k
      using assms(2) by (auto dest: power_eq_1_iff)
    have "summable (λk. norm (a k + f k (inverse q)))"
      using False assms(2) by (intro summable_norm_add assms *) (auto
simp: norm_divide field_simps)
    moreover have "eventually (λk. f k q = -a k - f k (inverse q)) at_top"
      using eventually_gt_at_top[of 0]
      by eventually_elim (use False assms(2) in ‹auto simp: fun_eq_iff
field_simps f_def›)
    hence "eventually (λk. norm (f k q) = norm (a k + f k (inverse q)))
at_top"
      by eventually_elim (auto simp: norm_uminus_minus)
    ultimately show ?thesis
      using summable_cong by fast
  qed
qed

lemma abs_summable_on_lambert:
  assumes "norm q < min 1 (conv_radius a)"
  shows   "(λk. f k q) abs_summable_on {1..}"
proof -
  have "summable (λk. norm (f (Suc k) q))"
    by (subst summable_Suc_iff, rule norm_summable_lambert) (use assms
in auto)
  hence "(λk. f (Suc k) q) abs_summable_on UNIV"
    by (subst summable_on_UNIV_nonneg_real_iff) auto
  also have "?this ⟷ ?thesis"
    by (intro summable_on_reindex_bij_witness[of _ "λk. k - 1" Suc]) auto
  finally show ?thesis .
qed

lemma abs_summable_on_lambert':
  assumes "summable (λk. norm (a k))" and "norm q ≠ 1"
  shows   "(λk. f k q) abs_summable_on {1..}"
proof -
  have "summable (λk. norm (f (Suc k) q))"
    by (subst summable_Suc_iff, rule norm_summable_lambert') (use assms
in auto)
  hence "(λk. f (Suc k) q) abs_summable_on UNIV"
    by (subst summable_on_UNIV_nonneg_real_iff) auto
  also have "?this ⟷ ?thesis"
```

```
    by (intro summable_on_reindex_bij_witness[of _ "λk. k - 1" Suc]) auto
  finally show ?thesis .
qed

lemma summable_on_lambert:
  assumes "norm q < min 1 (conv_radius a)"
  shows   "(λk. f k q) summable_on {1..}"
  using abs_summable_summable[OF abs_summable_on_lambert[OF assms]] .

lemma has_sum_lambert:
  assumes "norm q < min 1 (conv_radius a)"
  shows   "((λk. f k q) has_sum lambert a q) {1..}"
proof -
  have "((λk. f (Suc k) q) has_sum lambert a q) UNIV"
  proof (rule norm_summable_imp_has_sum)
    show "summable (λn. norm (f (Suc n) q))"
      using norm_summable_lambert[OF assms] by (subst summable_Suc_iff)
    show "(λk. f (Suc k) q) sums lambert a q"
      by (rule sums_lambert) (use assms in ‹auto simp: lambert_conv_radius_def›)
  qed
  also have "?this ⟷ ?thesis"
    by (intro has_sum_reindex_bij_witness[of _ "λk. k - 1" Suc]) auto
  finally show ?thesis .
qed
```

We can also show a more precise convergence result that essentially fully reduces the question of convegence of a Lambert series to that of its "corresponding" power series: $\sum_{k=1}^{\infty} a(k)\frac{q^k}{1-q^k}$ converges if and only if the "corresponding" power series $\sum_{k=1}^{\infty} a(k)q^k$ converges or if $\sum_{k=1}^{\infty} a(k)$ converges. This is Theorem 259 in Knopp's book. A key ingredient, aside from the results we have amassed so far, is the du-Bois Reymond summation test.

```
theorem summable_lambert_iff:
  assumes "norm q ≠ 1"
  shows   "summable (λk. f k q) ⟷ summable a ∨ summable (λk. a k *
q ^ k)"
proof (cases "summable a")
  case True
  hence "summable (λk. f k q)" using assms
    by (intro summable_lambert) (auto simp: lambert_conv_radius_def)
  with True show ?thesis
    by auto
next
  case not_summable: False
  have [simp]: "q ^ k ≠ 1" if "k > 0" for k
    using that assms by (auto dest: power_eq_1_iff)
  have "summable (λk. f k q) ⟷ summable (λk. a k * q ^ k)"
  proof (cases "norm q < 1")
    case False
```

```
    with assms have q: "norm q > 1"
      by simp
    have "¬summable (λk. a k * q ^ k)"
      using q not_summable conv_radius_geI[of a q] summable_in_conv_radius[of
1 a]
      by (auto simp: ereal_le_less one_ereal_def)
    moreover have "¬summable (λk. f k q)"
    proof
      assume "summable (λk. f k q)"
      hence *: "summable (λk. a k / (1 - q ^ k) * q ^ k)"
        by (auto simp: f_def)
      hence "summable (λk. a k / (1 - q ^ k) * 1 ^ k)"
        by (rule powser_inside) (use q in auto)
      hence "summable (λk. a k / (1 - q ^ k) - a k / (1 - q ^ k) * q ^
k)"
        by (intro summable_diff *) auto
      moreover have "eventually (λk. a k / (1 - q ^ k) - a k / (1 - q
^ k) * q ^ k = a k) at_top"
        using eventually_gt_at_top[of 0]
        by eventually_elim (simp add: divide_simps, simp add: algebra_simps)
      ultimately have "summable a"
        using summable_cong by fast
      with not_summable show False
        by contradiction
    qed
    ultimately show ?thesis
      by simp
  next
    case q: True
    show ?thesis
    proof
      assume "summable (λk. f k q)"
      hence "summable (λk. f k q * (1 - q ^ k))"
      proof (rule dubois_reymond_summation_test)
        have "summable (λk. norm (q - 1) * norm q ^ k)"
          using q by (intro summable_mult summable_geometric) auto
        also have "(λk. norm (q - 1) * norm q ^ k) = (λk. norm ((q - 1)
* q ^ k))"
          by (simp add: norm_mult norm_power)
        also have "... = (λk. norm (1 - q ^ k - (1 - q ^ Suc k)))"
          by (simp add: algebra_simps)
        finally show "summable (λk. norm (1 - q ^ k - (1 - q ^ Suc k)))"
.
      qed
      moreover have "eventually (λk. f k q * (1 - q ^ k) = a k * q ^
k) at_top"
        using eventually_gt_at_top[of 0] by eventually_elim (auto simp:
divide_simps f_def)
      ultimately show "summable (λk. a k * q ^ k)"
```

```
      using summable_cong by fast
  next
    assume "summable (λk. a k * q ^ k)"
    hence "summable (λk. a k * q ^ k * (1 / (1 - q ^ k)))"
    proof (rule dubois_reymond_summation_test)
      show "summable (λk. norm (1 / (1 - q ^ k) - 1 / (1 - q ^ Suc
k)))"
      proof (rule summable_comparison_test_ev)
        show "summable (λk. 2 / (1 - norm q) ^ 2 * norm q ^ k)"
          using q by (intro summable_mult summable_geometric) auto
      next
        show "eventually (λk. norm (norm (1 / (1 - q ^ k) - 1 / (1
- q ^ Suc k))) ≤
                                    2 / (1 - norm q) ^ 2 * norm q ^ k) at_top"
          using eventually_gt_at_top[of 0]
        proof eventually_elim
          case k: (elim k)
          have "norm (1 - q) ≤ norm (1 :: 'a) + 1"
            using q by norm
          hence 1: "norm (1 - q) ≤ 2"
            by simp
          have 2: "norm (1 - q ^ l) ≥ 1 - norm q" if l: "l > 0" for
l
          proof -
            have "norm (1 - q ^ l) ≥ norm (1 :: 'a) - norm (q ^ l)"
              by norm
            moreover have "norm (q ^ l) ≤ norm q ^ 1"
              using q l unfolding norm_power by (intro power_decreasing)
auto
            ultimately show ?thesis
              by simp
          qed

          have "1 / (1 - q ^ k) - 1 / (1 - q ^ Suc k) =
                  (q ^ k - q ^ Suc k) / ((1 - q ^ k) * (1 - q ^ Suc
k))"
            using k by (simp add: divide_simps del: power_Suc)
          also have "... = (1 - q) * q ^ k / ((1 - q ^ k) * (1 - q ^
Suc k))"
            by (simp add: algebra_simps)
          also have "norm ... = norm (1 - q) * norm q ^ k / (norm (1
- q ^ k) * norm (1 - q ^ Suc k))"
            by (simp add: norm_mult norm_divide norm_power)
          also have "... ≤ 2 * norm q ^ k / ((1 - norm q) * (1 - norm
q))" using q k
            by (intro frac_le mult_mono mult_pos_pos zero_le_power norm_ge_zero
1 2)
               (auto simp del: power_Suc)
          finally show ?case
```

```
            by (simp add: power2_eq_square)
        qed
      qed
    qed
    thus "summable (λk. f k q)"
      by (simp add: f_def)
  qed
 qed
 with not_summable show ?thesis
  by blast
qed

end


lemma holomorphic_lambert [holomorphic_intros]:
 assumes "X ⊆ eball 0 (lambert_conv_radius a) - sphere 0 1"
 shows   "lambert a holomorphic_on X"
proof -
 have "lambert a holomorphic_on eball 0 (lambert_conv_radius a) - sphere
0 1"
 proof (rule holomorphic_uniform_sequence)
  show "open (eball 0 (lambert_conv_radius a) - sphere (0 :: complex)
1)"
   by (intro open_Diff) auto
 next
  fix q :: complex
  assume q: "q ∈ eball 0 (lambert_conv_radius a) - sphere 0 1"
  have "open (eball 0 (lambert_conv_radius a) - sphere 0 1 :: complex
set)"
   by auto
  then obtain r where r: "r > 0" "cball q r ⊆ eball 0 (lambert_conv_radius
a) - sphere 0 1"
   unfolding open_contains_cball using q by blast
  define f where "f ≡ λk q. a k * q ^ k / (1 - q ^ k)"
  show "∃d>0. cball q d ⊆ eball 0 (lambert_conv_radius a) - sphere
0 1 ∧
          uniform_limit (cball q d) (λn q. ∑k<n. f (Suc k) q) (lambert
a) sequentially"
  proof (intro exI[of _ r] conjI)
   show "uniform_limit (cball q r) (λn q. ∑k<n. f (Suc k) q) (lambert
a) sequentially"
    unfolding f_def by (rule uniform_limit_lambert) (use r in auto)
  qed (use r in auto)
 next
  show "(λq. ∑k<n. a (Suc k) * q ^ Suc k / (1 - q ^ Suc k)) holomorphic_on
        eball 0 (lambert_conv_radius a) - sphere 0 1" for n :: nat
   by (intro holomorphic_intros) (auto simp del: power_Suc dest!: power_eq_1_iff)
 qed
```

```
  thus ?thesis
    by (rule holomorphic_on_subset) fact
qed

lemma holomorphic_lambert' [holomorphic_intros]:
  assumes "f holomorphic_on A" "⋀z. z ∈ A ⟹ f z ∈ eball 0 (lambert_conv_radius
a) - sphere 0 1"
  shows   "(λz. lambert a (f z)) holomorphic_on A"
  by (rule holomorphic_on_compose_gen[OF assms(1) holomorphic_lambert[OF
order.refl],
           unfolded o_def])
     (use assms(2) in auto)

lemma analytic_lambert [analytic_intros]:
  fixes a :: "nat ⇒ complex"
  assumes "A ⊆ eball 0 (lambert_conv_radius a) - sphere 0 1"
  shows   "lambert a analytic_on A"
proof -
  have "open (eball 0 (lambert_conv_radius a) - sphere 0 1 :: complex
set)"
    by auto
  hence "lambert a analytic_on eball 0 (lambert_conv_radius a) - sphere
0 1"
    using holomorphic_lambert[OF order.refl, of a]
    by (auto simp: analytic_on_open)
  thus ?thesis
    by (rule analytic_on_subset) fact
qed

lemma analytic_lambert' [analytic_intros]:
  assumes "f analytic_on A" "⋀z. z ∈ A ⟹ f z ∈ eball 0 (lambert_conv_radius
a) - sphere 0 1"
  shows   "(λz. lambert a (f z)) analytic_on A"
  by (rule analytic_on_compose_gen[OF assms(1) analytic_lambert[OF order.refl],
unfolded o_def])
     (use assms(2) in auto)

lemma continuous_on_lambert [continuous_intros]:
  fixes a :: "nat ⇒ 'a :: {real_normed_field, banach, heine_borel}"
  assumes "A ⊆ eball 0 (lambert_conv_radius a) - sphere 0 1"
  shows   "continuous_on A (lambert a)"
proof -
  have "isCont (lambert a) q" if q: "q ∈ eball 0 (lambert_conv_radius
a) - sphere 0 1" for q
  proof -
    have "open (eball 0 (lambert_conv_radius a) - sphere 0 1 :: 'a set)"
      by auto
    with q obtain r where r: "r > 0" "cball q r ⊆ eball 0 (lambert_conv_radius
a) - sphere 0 1"
```

```
    unfolding open_contains_cball by blast
  have "continuous_on (cball q r) (lambert a)"
  proof (rule uniform_limit_theorem)
    show "uniform_limit (cball q r)
            (λn q. ∑k<n. a (Suc k) * q ^ Suc k / (1 - q ^ Suc k)) (lambert
a) at_top"
      by (intro uniform_limit_lambert) (use r in auto)
    show "∀F n in sequentially. continuous_on (cball q r)
            (λq. ∑k<n. a (Suc k) * q ^ Suc k / (1 - q ^ Suc k))" us-
ing q r
      by (auto intro!: always_eventually continuous_intros simp del:
power_Suc dest!: power_eq_1_iff)
  qed auto
  hence "continuous_on (ball q r) (lambert a)"
    by (rule continuous_on_subset) auto
  thus ?thesis using ‹r > 0›
    by (subst (asm) continuous_on_eq_continuous_at) auto
qed
with assms show ?thesis
  by (intro continuous_at_imp_continuous_on) auto
qed

lemma continuous_on_lambert' [continuous_intros]:
  fixes a :: "nat ⇒ 'a :: {real_normed_field, banach, heine_borel}"
  assumes "continuous_on A f" "⋀z. z ∈ A ⟹ f z ∈ eball 0 (lambert_conv_radius
a) - sphere 0 1"
  shows   "continuous_on A (λz. lambert a (f z))"
  by (rule continuous_on_compose2[OF continuous_on_lambert[OF order.refl]
assms(1)])
     (use assms(2) in auto)

lemma tendsto_lambert [tendsto_intros]:
  fixes a :: "nat ⇒ 'a :: {real_normed_field, banach, heine_borel}"
  assumes "(f ⟶ c) F" "c ∈ eball 0 (lambert_conv_radius a) - sphere
0 1"
  shows   "((λx. lambert a (f x)) ⟶ lambert a c) F"
proof -
  have "open (eball 0 (lambert_conv_radius a) - sphere 0 1 :: 'a set)"
    by (intro open_Diff closed_sphere) auto
  hence "isCont (lambert a) c"
    using continuous_on_lambert[OF order.refl]
    by (intro continuous_on_interior) (use assms in ‹auto simp: interior_open›)
  thus ?thesis
    using assms(1) by (simp add: isCont_tendsto_compose)
qed
```

If $\sum_{n=1}^{\infty} a(n)$ exists, the Lambert series of $a(n)$ tends to it for $q \to \infty$.

```
lemma tendsto_lambert_at_infinity:
  assumes "summable (a :: nat ⇒ 'a :: {real_normed_field, banach, heine_borel})"
```

```
  shows   "(lambert a ⟶ -(∑n. a (Suc n))) at_infinity"
proof (rule Lim_transform_eventually)
  have "((λq. -(∑n. a (Suc n)) - lambert a (1 / q)) ⟶ -(∑n. a (Suc
n)) - lambert a 0) at_infinity"
    by (rule tendsto_diff tendsto_lambert tendsto_intros tendsto_divide_0
filterlim_ident)+
       (use assms in ‹auto simp: lambert_conv_radius_def›)
  thus "((λq. -(∑n. a (Suc n)) - lambert a (1 / q)) ⟶ -(∑n. a (Suc
n))) at_infinity"
    by simp
next
  have "eventually (λq :: 'a. norm q > 1) at_infinity"
    by (metis dual_order.strict_trans1 eventually_at_infinity gt_ex)
  thus "∀F x in at_infinity. -(∑n. a (Suc n)) - lambert a (1 / x) =
lambert a x"
    by eventually_elim (subst lambert_reciprocal, use assms in auto)
qed
```

4.3 Power series expansion

By exchanging the order of summation, we can prove the power series expansion of $L(a, q)$ as

$$L(a, q) = \sum_{n=1}^{\infty} (a * 1)(n) q^n$$

where $*$ denotes the Dirichlet product, i.e. $(a * 1)(n) = \sum_{d|n} a(d)$.
This gives particularly nice results when $a(n)$ is a number-theoretic function.

```
theorem has_sum_lambert_powser:
  assumes "norm q < min 1 (conv_radius a)"
  assumes "dirichlet_prod a (λ_. 1) = b"
  shows   "((λn. b n * q ^ n) has_sum lambert a q) {1..}"
proof -
  have q: "norm q < 1" "norm q < conv_radius a"
    using assms by auto
  have q': "norm q < lambert_conv_radius a"
    using q by (auto simp: lambert_conv_radius_def)
  have "((λ(n, k). a n * q ^ (k * n)) has_sum lambert a q) ({1..} × {1..})"
  proof (rule has_sum_SigmaI; (unfold prod.case)?)
    show "((λn. a n * q ^ n / (1 - q ^ n)) has_sum lambert a q) {1..}"
      by (intro has_sum_lambert) (use assms in ‹auto simp: lambert_conv_radius_def›)
  next
    show "(λ(n, k). a n * q ^ (k * n)) summable_on {1..} × {1..}"
    proof (rule abs_summable_summable)
      show "(λ(n, k). a n * q ^ (k * n)) abs_summable_on {1..} × {1..}"
      proof (rule summable_on_SigmaI; (unfold prod.case)?)
        fix n :: nat
        assume n: "n ∈ {1..}"
        have "((λk. norm (a n) * (norm q ^ n) ^ k) has_sum
```

```
            (norm (a n) * (norm q ^ n / (1 - norm q ^ n)))) {1..}"
        using has_sum_geometric[of "norm q ^ n" 1] q n
        by (intro has_sum_cmult_right) (auto simp: norm_power power_less_1_iff)
      thus "((λk. norm (a n * q ^ (k * n))) has_sum
            (norm (a n) * norm q ^ n / (1 - norm q ^ n))) {1..}"
        by (simp add: norm_mult norm_power norm_divide mult_ac flip:
power_mult)
    next
      show "(λn. norm (a n) * norm q ^ n / (1 - norm q ^ n)) summable_on
{1..}"
      proof (rule abs_summable_summable)
        show "(λx. norm (a x) * norm q ^ x / (1 - norm q ^ x)) abs_summable_on
{1..}"
          using abs_summable_on_lambert[of "of_real (norm q)" "λn. norm
(a n)"] q' q
          by (cases "summable (λn. norm (a n))")
             (auto simp: lambert_conv_radius_def split: if_splits)
      qed
    qed auto
  qed
  next
    fix n :: nat
    assume n: "n ∈ {1..}"
    have "((λk. a n * (q ^ n) ^ k) has_sum a n * (q ^ n / (1 - q ^ n)))
{1..}"
      using has_sum_geometric[of "q ^ n" 1] q n
      by (intro has_sum_cmult_right) (auto simp: power_less_1_iff norm_power)
    thus "((λk. a n * q ^ (k * n)) has_sum a n * q ^ n / (1 - q ^ n))
{1..}"
      by (simp add: mult_ac flip: power_mult)
  qed
  also have "?this ⟷ ((λ(n, d). a d * q ^ n) has_sum lambert a q) (SIGMA
n:{1..}. {d. d dvd n})"
    by (rule has_sum_reindex_bij_witness[of _ "λ(m, d). (d, m div d)"
"λ(n,k). (n * k, n)"])
       (auto simp: mult_ac)
  finally have 1: "((λ(n, d). a d * q ^ n) has_sum lambert a q) (SIGMA
n:{1..}. {d. d dvd n})" .

  have 2: "((λd. a d * q ^ n) has_sum (b n * q ^ n)) {d. d dvd n}" if
n: "n > 0" for n
  proof -
    have "((λd. a d * q ^ n) has_sum ((∑ d | d dvd n. a d) * q ^ n))
{d. d dvd n}"
      by (intro has_sum_cmult_left has_sum_finite finite_divisors_nat
n)
    also have "(∑ d | d dvd n. a d) = dirichlet_prod a (λ_. 1) n"
      by (simp add: dirichlet_prod_def)
    also have "... = b n"
```

```
    using assms by simp
  finally show ?thesis .
qed

show "((λn. b n * q ^ n) has_sum lambert a q) {1..}"
  using has_sum_SigmaD[OF 1, of "λn. b n * q ^ n"] 2 by simp
qed

lemma sums_lambert_powser:
  assumes "norm q < min 1 (conv_radius a)"
  assumes "dirichlet_prod a (λ_. 1) = b"
  shows   "(λn. b n * q ^ n) sums lambert a q"
proof -
  from assms(2) have [simp]: "b 0 = 0"
    using dirichlet_prod_0 by blast
  have "((λn. b n * q ^ n) has_sum lambert a q) {1..}"
    by (rule has_sum_lambert_powser) fact+
  also have "?this ⟷ ((λn. b n * q ^ n) has_sum lambert a q) UNIV"
    by (intro has_sum_cong_neutral) (auto simp: not_le)
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

lemma conv_radius_dirichlet_prod_1_ge:
  fixes a b :: "nat ⇒ 'a :: {real_normed_field, banach}"
  defines "b ≡ dirichlet_prod a (λ_. 1)"
  shows   "conv_radius b ≥ min 1 (conv_radius a)"
proof (rule conv_radius_geI_ex)
  fix r :: real
  assume r: "0 < r" "r < min 1 (conv_radius a)"
  have "(λn. b n * of_real r ^ n) sums lambert a (of_real r)"
    using r by (intro sums_lambert_powser) (auto simp: b_def)
  thus "∃z. norm z = r ∧ summable (λn. b n * z ^ n)"
    using r(1) by (intro exI[of _ "of_real r"]) (auto simp: sums_iff)
qed

lemma sums_lambert_powser':
  assumes "norm q < min 1 (conv_radius a)"
  assumes "fds b = fds a * fds_zeta" "b 0 = 0"
  shows   "(λn. b n * q ^ n) sums lambert a q"
  using assms(1)
proof (rule sums_lambert_powser)
  have "fds_nth (fds a * fds_zeta) = dirichlet_prod a (λ_. 1)"
    by (auto simp: fds_nth_mult)
  also have "fds a * fds_zeta = fds b"
    using assms(2) by (simp only: )
  also have "fds_nth (fds b) = b"
    using assms(3) by (auto simp: fun_eq_iff fds_nth_fds)
  finally show "dirichlet_prod a (λ_. 1) = b" ..
```

qed

4.3.1 Divisor σ function

For any q with $|q| < 1$ and any $\alpha \in \mathbb{C}$, we have

$$\sum_{n=1}^{\infty} \sigma_\alpha(n) q^n = \sum_{n=1}^{\infty} n^\alpha \frac{q^n}{1-q^n}$$

where $\sigma_\alpha(n)$ is the divisor σ function, i.e. $sigma_\alpha(n) = \sum_{d|n} d^\alpha$.

```
lemma divisor_sigma_powser_conv_lambert:
  fixes α q :: "'a :: {nat_power_normed_field, banach}"
  assumes q: "norm q < 1"
  shows   "(λn. divisor_sigma α n * q ^ n) sums lambert (λn. nat_power
n α) q"
proof (rule sums_lambert_powser')
  have "conv_radius (λn. nat_power n α) = 1"
  proof (rule tendsto_imp_conv_radius_eq)
    have "(λn. ereal ((real n powr (α · 1)) powr (1 / real n))) ⟶
1"
      unfolding one_ereal_def by (rule tendsto_ereal) real_asymp
    also have "?this ⟷ (λn. ereal (norm (nat_power n α) powr (1 / real
n))) ⟶ 1"
      by (intro filterlim_cong refl eventually_mono[OF eventually_gt_at_top[of
0]])
         (simp add: norm_nat_power)
    finally show "(λn. ereal (norm (nat_power n α) powr (1 / real n)))
⟶ 1"
      by (simp add: norm_nat_power)
  qed auto
  thus "ereal (norm q) < min 1 (conv_radius (λn. nat_power n α))"
    using q by simp
next
  have "fds (divisor_sigma α) = fds_zeta * fds_shift α fds_zeta"
    by (rule fds_divisor_sigma)
  also have "fds_shift α fds_zeta = fds (λn. nat_power n α)"
    by (simp add: fds_eq_iff)
  finally show "fds (divisor_sigma α) = fds (λn. nat_power n α) * fds_zeta"
    by (simp add: mult.commute)
qed auto

lemma divisor_count_powser_conv_lambert:
  fixes q :: "'a :: {nat_power_normed_field, banach}"
  assumes q: "norm q < 1"
  shows   "(λn. of_nat (divisor_count n) * q ^ n) sums lambert (λ_. 1)
q"
  using divisor_sigma_powser_conv_lambert[OF assms, of 0]
  by (simp add: divisor_sigma_0_left)
```

4.3.2 Möbius μ function

For any q with $|q| < 1$, we have

$$\sum_{n=1}^{\infty} \mu(n) \frac{q^n}{1-q^n} = q$$

where $\mu(n)$ is Möbus' μ function, which is 0 if n is not squarefree (i.e. contains the same prime factor more than once) and otherwise equal to $(-1)^k$, where k is the number of prime factors of n.

```
lemma lambert_moebius_mu:
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes q: "norm q < 1"
  shows   "lambert moebius_mu q = q"
proof -
  have "(λn. indicator {1} n * q ^ n) sums lambert moebius_mu q"
  proof (rule sums_lambert_powser')
    have "fds moebius_mu * fds_zeta = (1 :: 'a fds)"
      using fds_zeta_times_moebius_mu[where ?'a = 'a] by (simp only:
mult.commute)
    also have "(1 :: 'a fds) = fds (indicator {1})"
      by (auto simp: fds_eq_iff fds_nth_one)
    finally show "fds (indicator {1} :: nat ⇒ 'a) = fds moebius_mu * fds_zeta"
..
  qed (use q in ‹auto simp: conv_radius_moebius_mu›)
  also have "(λn. indicator {1} n * q ^ n) = (λn. (if n = 1 then 1 else
0) * q ^ n)"
    by auto
  finally have "... sums lambert moebius_mu q" .
  moreover have "(λn. (if n = 1 then 1 else 0) * q ^ n) sums (q ^ 1)"
    by (rule powser_sums_if)
  ultimately show "lambert moebius_mu q = q"
    by (simp add: sums_iff)
qed

lemma lambert_conv_radius_moebius_mu:
  "lambert_conv_radius (moebius_mu :: nat ⇒ 'a :: {real_normed_field,
banach}) = 1"
proof -
  have "¬summable (moebius_mu :: nat ⇒ 'a)"
    using not_convergent_moebius_mu[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  thus ?thesis
    by (simp add: lambert_conv_radius_def conv_radius_moebius_mu)
qed
```

4.3.3 Euler's totient function φ

For any q with $|q| < 1$, we have

$$\frac{q}{(1-q)^2} = \sum_{n=1}^{\infty} nq^n = \sum_{n=1}^{\infty} \varphi(n) \frac{q^n}{1-q^n}$$

where $\varphi(n)$ is Euler's totient function, i.e. the number of positive integers not greater than n that are coprime to n.

```
lemma lambert_totient:
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes q: "norm q < 1"
  shows   "lambert (λn. of_nat (totient n) :: 'a) q = q / (1 - q) ^ 2"
proof -
  have "(λn. of_nat n * q ^ n) sums lambert (λn. of_nat (totient n) ::
'a) q"
  proof (rule sums_lambert_powser')
    show "fds (of_nat :: nat ⇒ 'a) = fds (λn. of_nat (totient n)) *
fds_zeta"
      by (rule fds_totient_times_zeta [symmetric])
  qed (use q in ‹auto simp: conv_radius_totient›)
  from sums_unique2[OF this n_powser_sums[OF q]] show ?thesis .
qed

lemma lambert_conv_radius_totient:
  "lambert_conv_radius (λn. of_nat (totient n) :: 'a :: {real_normed_field,
banach}) = 1"
proof -
  have "¬summable (λn. of_nat (totient n) :: 'a :: {real_normed_field,
banach})"
    using not_convergent_totient[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  thus ?thesis
    by (simp add: lambert_conv_radius_def conv_radius_totient)
qed
```

4.3.4 Mangoldt's Λ function

For any q with $|q| < 1$, we have

$$\sum_{n=1}^{\infty} \ln n \; q^n = \sum_{n=1}^{\infty} \Lambda(n) \frac{q^n}{1-q^n}$$

where $\Lambda(n)$ is Mangoldt's function, which is defined to be equal to $\log n$ if n is prime and 0 otherwise.

```
lemma lambert_mangoldt:
  fixes q :: "'a :: {real_normed_field, banach}"
```

```
  assumes q: "norm q < 1"
  shows   "(λn. of_real (ln (Suc n)) * q ^ (Suc n)) sums lambert mangoldt
q"
proof -
  have "(λn. (if n = 0 then 0 else of_real (ln n)) * q ^ n) sums lambert
mangoldt q"
    by (rule sums_lambert_powser')
       (use q fds_mangoldt_times_zeta[where ?'a = 'a] in ‹auto simp: conv_radius_mangoldt›)
  also have "?this ⟷ (λn. (if Suc n = 0 then 0 else of_real (ln (Suc
n))) * q ^ Suc n) sums lambert mangoldt q"
    by (subst sums_Suc_iff) auto
  also have "... ⟷ ?thesis"
    by simp
  finally show ?thesis .
qed

lemma lambert_conv_radius_mangoldt:
  "lambert_conv_radius (mangoldt :: nat ⇒ 'a :: {real_normed_field, banach})
= 1"
proof -
  have "¬summable (mangoldt :: nat ⇒ 'a :: {real_normed_field, banach})"
    using not_convergent_mangoldt[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  thus ?thesis
    by (simp add: lambert_conv_radius_def conv_radius_mangoldt)
qed
```

4.3.5 Liouville's λ function

For any q with $|q| < 1$, we have

$$\sum_{n=1}^{\infty} q^{n^2} = \sum_{n=1}^{\infty} \lambda(n) \frac{q^n}{1-q^n}$$

where $\lambda(n)$ is Liouville's function, which is defined as the number of prime factors of n (taking multiplicity into account).

```
lemma lambert_liouville_lambda:
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes q: "norm q < 1"
  shows   "(λn. ind is_square n * q ^ n) sums lambert liouville_lambda
q"
  by (rule sums_lambert_powser')
     (use q fds_liouville_lambda_times_zeta[where ?'a = 'a]
       in ‹auto simp: conv_radius_liouville_lambda mult_ac›)

lemma lambert_liouville_lambda':
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes q: "norm q < 1"
```

```
  shows   "(λn. q ^ ((n+1) ^ 2)) sums lambert liouville_lambda q"
proof -
  have "(λn. ind is_square ((n+1)^2) * q ^ ((n+1) ^ 2)) sums lambert liouville_lambda
q ⟷
        (λn. ind is_square n * q ^ n) sums lambert liouville_lambda q"
  proof (rule sums_mono_reindex)
    show "strict_mono (λn::nat. (n + 1)^2)"
      by (intro strict_monoI power_strict_mono) auto
    show "ind is_square n * q ^ n = 0" if "n ∉ range (λn. (n + 1)^2)" for
n
    proof (rule ccontr)
      assume "ind is_square n * q ^ n ≠ 0"
      hence "is_square n" "n > 0"
        by (auto simp: ind_def split: if_splits)
      then obtain m where "n = m ^ 2" "m > 0"
        by (elim is_nth_powerE) auto
      hence "n = ((m - 1) + 1) ^ 2"
        by auto
      with that show False by blast
    qed
  qed
  thus ?thesis
    using lambert_liouville_lambda[OF assms] by simp
qed

lemma lambert_conv_radius_liouville_lambda:
  "lambert_conv_radius (liouville_lambda :: nat ⇒ 'a :: {real_normed_field,
banach}) = 1"
proof -
  have "¬summable (liouville_lambda :: nat ⇒ 'a)"
    using not_convergent_liouville_lambda[where ?'a = 'a] summable_LIMSEQ_zero
    by (auto simp: convergent_def)
  thus ?thesis
    by (simp add: lambert_conv_radius_def conv_radius_liouville_lambda)
qed
```

4.4 Expressing a Lambert series in terms of a power series

Let $a(n)$ be a sequence of numbers. Then we can express the value of the Lambert series as an infinite sum in terms of the "normal" power series $f(q) = \sum_{k=1}^{\infty} a(k)q^k$:

$$L(a, q) = \sum_{n=1}^{\infty} f(q^n)$$

The proof is quite obvious, by expanding $f(q^n)$ into its power series and then switching the order of summation.

This gives us a number of interesting relationships, including a connection between $L(n^a, q)$ and the polylogarithm function Li_{-a}.

```
theorem lambert_conv_powser_has_sum:
  assumes q: "norm q < min 1 (conv_radius a)" and [simp]: "a 0 = 0"
  defines "f ≡ (λq. ∑n. a n * q ^ n)"
  shows   "((λn. f (q ^ n)) has_sum lambert a q) {1..}"
proof -
  have "((λ(k, n). a k * (q ^ k) ^ n) has_sum lambert a q) ({1..} × {1..})"
  proof (rule has_sum_SigmaI; (unfold prod.case)?)
    fix k :: nat
    assume k: "k ∈ {1..}"
    show "((λy. a k * (q ^ k) ^ y) has_sum (a k * (q ^ k / (1 - q ^ k))))
{1..}"
      by (intro has_sum_cmult_right has_sum_geometric_from_1)
         (use q k in ‹auto simp: power_less_1_iff norm_power›)
  next
    have "((λk. a k * q ^ k / (1 - q ^ k)) has_sum lambert a q) {1..}"
      using q by (intro has_sum_lambert) (auto simp: lambert_conv_radius_def
split: if_splits)
    thus "((λk. a k * (q ^ k / (1 - q ^ k))) has_sum lambert a q) {1..}"
      by (simp add: field_simps)
  next
    show "(λ(k, n). a k * (q ^ k) ^ n) summable_on {1..} × {1..}"
    proof (rule abs_summable_summable, rule summable_on_SigmaI;
           (unfold prod.case norm_divide norm_power norm_mult norm_one
norm_of_nat)?)
      fix k :: nat assume k: "k ∈ {1..}"
      show "((λn. norm (a k) * (norm q ^ k) ^ n) has_sum
              (norm (a k) * (norm q ^ k / (1 - norm q ^ k)))) {1..}"
        by (intro has_sum_cmult_right has_sum_geometric_from_1)
           (use k q in ‹auto simp: power_less_1_iff›)
    next
      show "(λn. norm (a n) * (norm q ^ n / (1 - norm q ^ n))) summable_on
{1..}"
        using summable_on_lambert[of "norm q" "λn. norm (a n)"] q
        by (auto simp: lambert_conv_radius_def split: if_splits)
    qed auto
  qed
  hence *: "((λ(n, k). a k * q ^ (k * n)) has_sum lambert a q) ({1..}
× {1..})"
    by (subst (asm) has_sum_swap) (simp_all flip: power_mult add: mult.commute)

  show ?thesis
  proof (rule has_sum_SigmaD [OF *]; unfold prod.case)
    fix n :: nat assume n: "n ∈ {1..}"
    have "ereal (norm q ^ n) ≤ ereal (norm q ^ 1)"
      unfolding ereal_less_eq using n q by (intro power_decreasing) auto
    also have "... < conv_radius a"
      using q by (simp add: less_eq_ereal_def)
    finally have norm_q_n: "norm q ^ n < conv_radius a" .
```

```
    have "((λk. a k * (q ^ n) ^ k) has_sum f (q ^ n)) UNIV"
    proof (rule norm_summable_imp_has_sum)
      from norm_q_n show "((λk. a k * (q ^ n) ^ k) sums f (q ^ n))"
        unfolding f_def by (intro summable_sums summable_in_conv_radius)
(auto simp: norm_power)
    next
      show "summable (λk. norm (a k * (q ^ n) ^ k))"
        by (rule abs_summable_in_conv_radius) (use norm_q_n in ‹auto simp:
norm_power›)
    qed
    also have "?this ⟷ ((λk. a k * q ^ (k * n)) has_sum f (q ^ n)) {1..}"

      by (intro has_sum_cong_neutral) (auto simp: mult.commute not_le
simp flip: power_mult)
    finally show ... .
  qed
qed

lemma lambert_conv_powser_has_sum':
  assumes "norm q < r" and "r ≤ 1"
  assumes "⋀q. norm q < r ⟹ (λn. a (Suc n) * q ^ Suc n) sums f q"
  shows   "((λn. f (q ^ n)) has_sum lambert a q) {1..}"
proof -
  define a' where "a' = (λk. if k = 0 then 0 else a k)"
  have "norm q < r"
    by fact
  also have "r ≤ conv_radius a"
  proof (rule conv_radius_geI_ex)
    fix r' assume r': "0 < r'" "ereal r' < ereal r"
    show "∃x. norm x = r' ∧ summable (λn. a n * x ^ n)"
      by (rule exI[of _ "of_real r'"], subst summable_Suc_iff [symmetric])

         (use assms(3)[of "of_real r'"] r' in ‹simp add: sums_iff›)
  qed
  also have "conv_radius a = conv_radius a'"
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of
0]]) (auto simp: a'_def)
  finally have "((λn. (∑k. a' k * (q ^ n) ^ k)) has_sum lambert a' q)
{1..}"
    using assms(1,2) by (intro lambert_conv_powser_has_sum) (auto simp:
a'_def)
  also have "?this ⟷ ((λn. f (q ^ n)) has_sum lambert a' q) {1..}"
  proof (rule has_sum_cong)
    fix n :: nat assume n: "n ∈ {1..}"
    have "norm q ^ n ≤ norm q ^ 1"
      using assms(1,2) n by (intro power_decreasing) auto
    also have "... < r"
      using assms(1,2) by simp
    finally have "(λk. a (Suc k) * (q ^ n) ^ Suc k) sums f (q ^ n)"
```

```
    by (intro assms) (auto simp: norm_power)
  hence "(λk. a' (Suc k) * (q ^ n) ^ Suc k) sums f (q ^ n)"
    by (simp add: a'_def)
  hence "(λk. a' k * (q ^ n) ^ k) sums f (q ^ n)"
    by (subst (asm) sums_Suc_iff) (simp add: a'_def)
  thus "(∑k. a' k * (q ^ n) ^ k) = f (q ^ n)"
    by (simp add: sums_iff)
qed
also have "lambert a' q = lambert a q"
  by (simp add: a'_def)
finally show ?thesis .
qed

lemma lambert_conv_powser_sums:
  assumes q: "norm q < min 1 (conv_radius a)" and [simp]: "a 0 = 0"
  defines "f ≡ (λq. ∑n. a n * q ^ n)"
  shows   "(λn. f (q ^ Suc n)) sums lambert a q"
proof -
  have "((λn. f (q ^ n)) has_sum lambert a q) {1..}"
    unfolding f_def by (rule lambert_conv_powser_has_sum) fact+
  also have "?this ⟷ ((λn. f (q ^ Suc n)) has_sum lambert a q) UNIV"
    by (rule has_sum_reindex_bij_witness[of _ Suc "λn. n - 1"]) auto
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

lemma lambert_conv_powser_sums':
  assumes "norm q < r" and "r ≤ 1"
  assumes "⋀q. norm q < r ⟹ (λn. a (Suc n) * q ^ Suc n) sums f q"
  shows   "(λn. f (q ^ Suc n)) sums lambert a q"
proof -
  have "((λn. f (q ^ n)) has_sum lambert a q) {1..}"
    by (rule lambert_conv_powser_has_sum') fact+
  also have "?this ⟷ ((λn. f (q ^ Suc n)) has_sum lambert a q) UNIV"
    by (rule has_sum_reindex_bij_witness[of _ Suc "λn. n - 1"]) auto
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

lemma lambert_mult_exp_conv_powser_has_sum:
  assumes "norm q < r" and "r ≤ 1" and c: "norm c ≤ 1"
  assumes "⋀q. norm q < r ⟹ (λn. a (Suc n) * q ^ Suc n) sums f q"
  shows   "((λn. f (c * q ^ n)) has_sum lambert (λn. c ^ n * a n) q)
{1..}"
proof (rule lambert_conv_powser_has_sum')
  fix q :: 'a assume q: "norm q < r"
  have "norm c * norm q ≤ 1 * norm q"
    using q c by (intro mult_right_mono) auto
  hence cq: "norm c * norm q < r"
```

```
    using q by simp
  have "(λn. a (Suc n) * (c * q) ^ Suc n) sums f (c * q)"
    by (rule assms) (use cq in ‹simp add: norm_mult›)
  thus "(λn. c ^ Suc n * a (Suc n) * q ^ Suc n) sums f (c * q)"
    by (simp add: power_mult_distrib algebra_simps)
qed (use assms in auto)

lemma lambert_mult_exp_conv_powser_sums:
  assumes "norm q < r" and "r ≤ 1" and c: "norm c ≤ 1"
  assumes "⋀q. norm q < r ⟹ (λn. a (Suc n) * q ^ Suc n) sums f q"
  shows   "((λn. f (c * q ^ Suc n)) sums lambert (λn. c ^ n * a n) q)"
proof (rule lambert_conv_powser_sums')
  fix q :: 'a assume q: "norm q < r"
  have "norm c * norm q ≤ 1 * norm q"
    using q c by (intro mult_right_mono) auto
  hence cq: "norm c * norm q < r"
    using q by simp
  have "(λn. a (Suc n) * (c * q) ^ Suc n) sums f (c * q)"
    by (rule assms) (use cq in ‹simp add: norm_mult›)
  thus "(λn. c ^ Suc n * a (Suc n) * q ^ Suc n) sums f (c * q)"
    by (simp add: power_mult_distrib algebra_simps)
qed (use assms in auto)

lemma lambert_power_int_has_sum_polylog_gen:
  fixes q :: complex
  assumes q: "norm q < 1" and c: "norm c ≤ 1"
  shows "((λn. polylog (-a) (c * q ^ n)) has_sum lambert (λn. c ^ n *
of_nat n powi a) q) {1..}"
  using q
proof (rule lambert_mult_exp_conv_powser_has_sum)
  show "(λn. of_nat (Suc n) powi a * q ^ Suc n) sums polylog (-a) q"
    if "norm q < 1" for q :: complex
    using sums_polylog[of q "-a"] that by simp
qed (use assms in auto)

lemma has_sum_lambert_recip_complex_gen:
  fixes q :: complex
  assumes q: "norm q < 1" and c: "norm c ≤ 1"
  shows   "((λk. -ln (1 - c * q ^ k)) has_sum lambert (λn. c ^ n / of_nat
n) q) {1..}"
proof -
  have "((λn. polylog 1 (c * q ^ n)) has_sum lambert (λn. c ^ n * of_nat
n powi - 1) q) {1..}"
    using lambert_power_int_has_sum_polylog_gen[OF q, of c "-1"] q c by
simp
  also have "?this ⟷ ((λn::nat. -ln (1 - c * q ^ n)) has_sum
                       lambert (λn. c ^ n * of_nat n powi -1) q) {1..}"
  proof (intro has_sum_cong)
    fix n :: nat assume n: "n ∈ {1..}"
```

```
    have "norm c * norm (q ^ n) ≤ 1 * norm (q ^ n)"
      using q c by (intro mult_right_mono) auto
    also have "norm (q ^ n) ≤ norm q ^ 1"
      using n q unfolding norm_power by (intro power_decreasing) auto
    also have "1 * norm q ^ 1 < 1"
      using q by simp
    finally have cq: "norm (c * q ^ n) < 1"
      by (simp_all add: norm_mult)
    thus "polylog 1 (c * q ^ n) = -ln (1 - c * q ^ n)"
      by (subst polylog_1) auto
  qed
  also have "(λn. c ^ n * of_nat n powi -1) = (λn. c ^ n / of_nat n ::
complex)"
    by (auto simp: field_simps)
  finally show ?thesis .
qed

lemma has_sum_lambert_recip_complex:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows   "((λk. -ln (1 - q ^ k)) has_sum lambert (λn. 1 / of_nat n)
q) {1..}"
  using has_sum_lambert_recip_complex_gen[OF assms, of 1] by simp

lemma has_sum_lambert_recip_complex':
  fixes q :: complex
  assumes q: "norm q < 1"
  shows   "((λk. -ln (1 + q ^ k)) has_sum lambert (λn. (-1) ^ n / of_nat
n) q) {1..}"
  using has_sum_lambert_recip_complex_gen[OF assms, of "-1"] by simp

lemma has_sum_lambert_poly_complex:
  fixes q :: complex and a :: nat
  assumes q: "norm q < 1" and a: "a > 0"
  defines "E ≡ poly (eulerian_poly a)"
  shows   "((λn. E (q ^ n) * q ^ n / (1 - q ^ n) ^ (a + 1)) has_sum
             lambert (λn. complex_of_nat n ^ a) q) {1..}"
proof -
  have "((λn. polylog (-a) (q ^ n)) has_sum lambert (λn. of_nat n ^ a)
q) {1..}"
    using lambert_power_int_has_sum_polylog_gen[OF q, of 1 a] q by simp
  also have "?this ⟷ ((λn. E (q ^ n) * q ^ n / (1 - q ^ n) ^ (a+1))
                       has_sum lambert (λn. of_nat n ^ a) q) {1..}"
  proof (rule has_sum_cong)
    fix n :: nat assume n: "n ∈ {1..}"
    have "norm (q ^ n) ≤ norm q ^ 1"
      using n q unfolding norm_power by (intro power_decreasing) auto
    also have "... < 1"
      using q by simp
```

```
    finally show "polylog (-a) (q ^ n) = E (q ^ n) * q ^ n / (1 - q ^ n)
^ (a+1)"
      using a by (subst polylog_neg_int_left)
                 (auto simp: E_def power_int_diff power_int_minus divide_simps)
  qed
  finally show ?thesis .
qed

lemma lambert_minus1_power_has_sum:
  assumes q: "norm q < 1"
  shows   "((λn. q ^ n / (1 + q ^ n)) has_sum lambert (λn. (-1) ^ Suc
n) q) {1..}"
  using q
proof (rule lambert_conv_powser_has_sum')
  show "(λn. (-1) ^ Suc (Suc n) * q ^ Suc n) sums (q / (1 + q))" if "norm
q < 1" for q :: 'a
  proof -
    have "(λn. (-1) ^ Suc (Suc n) * q ^ Suc n) sums (1 - 1 / (1 + q))"
      using sums_minus[OF geometric_sums[of "-q"]] that
      by (subst sums_Suc_iff) (auto simp: field_simps power_minus')
    also have "1 - 1 / (1 + q) = q / (1 + q)"
      using that by (auto simp: divide_simps add_eq_0_iff)
    finally show ?thesis .
  qed
qed auto

lemma lambert_exp_has_sum:
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes q: "norm q < 1" and a: "norm a ≤ 1"
  shows   "((λn. a * q ^ n / (1 - a * q ^ n)) has_sum lambert (λn. a
^ n) q) {1..}"
proof -
  have "((λn. a * q ^ n / (1 - a * q ^ n)) has_sum lambert (λn. a ^ n
* 1) q) {1..}"
    using q
  proof (rule lambert_mult_exp_conv_powser_has_sum)
    show "(λn. 1 * q ^ Suc n) sums (q / (1 - q))" if "norm q < 1" for
q :: 'a
      using geometric_sums_gen[of q 1] that by simp
  qed (use a in auto)
  thus ?thesis
    by simp
qed
```

4.5 Connection to Euler's function

In this section, we show a connection between Lambert series and Euler's function:

$$\varphi(q) = \prod_{k=1}^{\infty}(1-q^k)$$

(not to be confused with Euler's totient function, commonly denoted with $\varphi(n)$)

For this, we apply the results from the previous section to $a(n) = \frac{1}{n}$ to obtain:

$$\sum_{k=1}^{\infty}\ln(1-q^k) = -L\left(\tfrac{1}{n}, q\right)$$

```
lemma sums_lambert_recip_complex:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows   "((λk. -ln (1 - q ^ Suc k)) sums lambert (λn. 1 / of_nat n)
q)"
  using q
proof (rule lambert_conv_powser_sums')
  show "(λk. 1 / of_nat (Suc k) * q ^ Suc k) sums - Ln (1 - q)" if "norm
q < 1" for q
    using sums_minus[OF Ln_series[of "-q"]] that
    by (subst sums_Suc_iff) (simp_all add: power_minus')
qed auto

lemma sums_lambert_recip_complex':
  fixes q :: complex
  assumes q: "norm q < 1"
  shows   "((λk. -ln (1 + q ^ Suc k)) sums lambert (λn. (-1)^n / of_nat
n) q)"
  using q
proof (rule lambert_conv_powser_sums')
  show "(λk. (-1)^Suc k / of_nat (Suc k) * q ^ Suc k) sums - Ln (1 +
q)" if "norm q < 1" for q
    using sums_minus[OF Ln_series[of "q"]] that
    by (subst sums_Suc_iff) (simp_all add: power_minus')
qed auto
```

By exponentiating this, we get:

$$\varphi(q) \stackrel{\text{def}}{=} \prod_{n=1}^{\infty}\left(1-q^n\right) = \exp\left(-\sum_{n=1}^{\infty}\frac{1}{n}\frac{q^n}{1-q^n}\right)$$

In other words, the Lambert sum $\sum \frac{1}{n}\frac{q^n}{1-q^n}$ is a logarithm of Euler's function $\varphi(q)$.

Note that this does not show that this is *the* logarithm of $\varphi(q)$, but merely that it is *one* of the branches of the multi-valued logarithm of $\varphi(q)$. Nevertheless, we will – just like is typically in textbooks – ignore this in our informal explanations and write $\ln \varphi(q)$.

```
theorem euler_phi_conv_lambert:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "(λn. 1 - q ^ Suc n) has_prod exp (-lambert (λn. 1 / of_nat n)
q)"
proof -
  have not_1: "q ^ n ≠ 1" if "n > 0" for n
    using that q by (auto dest: power_eq_1_iff)
  have "(λn. exp (-ln (1 - q ^ Suc n))) has_prod exp (lambert (λn. 1 /
of_nat n) q)"
    by (intro sums_imp_has_prod_exp sums_lambert_recip_complex q)
  also have "(λn. exp (-ln (1 - q ^ Suc n))) = (λn. inverse (1 - q ^ Suc
n))"
    using q unfolding exp_minus by (subst exp_Ln) (auto simp del: power_Suc
simp: not_1)
  finally have "(λn. inverse (inverse (1 - q ^ Suc n))) has_prod
                  inverse (exp (lambert (λn. 1 / complex_of_nat n) q))"
    by (intro has_prod_inverse)
  thus ?thesis
    using q by (simp del: power_Suc add: exp_minus)
qed
```

With our general results on Lambert series, we also know that $\ln \varphi(q)$ has the power series expansion

$$\ln \varphi(q) = -\sum_{n=1}^{\infty} \sigma_{-1}(n) q^n = -\sum_{n=1}^{\infty} \frac{\sigma_1(n)}{n} q^n \ .$$

```
lemma ln_euler_phi_powser:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "(λn. divisor_sigma (-1) n * q ^ n) sums lambert (λn. 1 / of_nat
n) q"
  using divisor_sigma_powser_conv_lambert[OF q, of "-1"]
  by (simp add: powr_minus divide_inverse)

lemma ln_euler_phi_powser':
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "(λn. divisor_sum n / n * q ^ n) sums lambert (λn. 1 / of_nat
n) q"
  using ln_euler_phi_powser[OF q]
  by (simp add: divisor_sigma_minus divisor_sigma_1_left mult_ac)
```

We also show the following variant of the above, also mentioned by Knopp:

```
theorem euler_phi_variant_conv_lambert:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "(λn. 1 + q ^ Suc n) has_prod exp (-lambert (λn. (-1) ^ n / of_nat
n) q)"
proof -
  have not_1: "q ^ n ≠ -1" if "n > 0" for n
  proof
    assume "q ^ n = -1"
    hence "norm q ^ n = 1"
      by (simp flip: norm_power)
    thus False
      using that q by (auto dest: power_eq_1_iff)
  qed
  have "(λn. exp (-ln (1 + q ^ Suc n))) has_prod exp (lambert (λn. (-1)^n
/ of_nat n) q)"
    by (intro sums_imp_has_prod_exp sums_lambert_recip_complex' q)
  also have "(λn. exp (-ln (1 + q ^ Suc n))) = (λn. inverse (1 + q ^ Suc
n))"
    using q unfolding exp_minus
    by (subst exp_Ln) (auto simp del: power_Suc simp: not_1 add_eq_0_iff)
  finally have "(λn. inverse (inverse (1 + q ^ Suc n))) has_prod
                  inverse (exp (lambert (λn. (-1)^n / complex_of_nat
n) q))"
    by (intro has_prod_inverse)
  thus ?thesis
    using q by (simp del: power_Suc add: exp_minus)
qed
```

4.6 Application: Fibonacci numbers

Lastly, we show a connection between the Fibonacci numbers and Lambert series, namely that:

$$\sum_{n=1}^{\infty} \frac{1}{F_n} = \sqrt{5}\left[L\big(1, \tfrac{1}{2}(3-\sqrt{5})\big) - L\big(1, \tfrac{1}{2}(7-3\sqrt{5})\big)\right]$$

```
lemma fib_closed_form_alt:
  defines "φ ≡ (1 + sqrt 5) / 2"
  shows   "real (fib n) = (φ ^ n - (-1 / φ) ^ n) / sqrt 5"
proof -
  have "real (fib n) = (φ ^ n - ((1 - sqrt 5) / 2) ^ n) / sqrt 5"
    unfolding φ_def by (rule fib_closed_form)
  also have "1 + sqrt 5 > 0"
    by (intro add_pos_pos) auto
  hence "(1 - sqrt 5) / 2 = -(1 / φ)"
    by (simp add: φ_def field_simps)
  finally show ?thesis
```

```
    by simp
qed

theorem sum_inv_even_fib_conv_lambert:
  defines "L ≡ lambert (λ_. 1)"
  shows   "((λn. 1 / real (fib (2*n))) has_sum
             (sqrt 5 * (L ((3 - sqrt 5) / 2) - L ((7 - 3 * sqrt 5) / 2))))
{1..}"
proof -
  define φ :: real where "φ = (1 + sqrt 5) / 2"
  have "1 + sqrt 5 > 0"
    by (intro add_pos_pos) auto
  hence [simp]: "1 + sqrt 5 ≠ 0"
    by auto
  have pos: "φ > 1"
    by (auto simp: φ_def intro: add_pos_pos)
  have [simp]: "φ ^ k = 1 ⟷ k = 0" for k
    by (auto simp: φ_def dest: power_eq_1_iff)
  have "((λn. 1 * (1/φ^2)^n / (1 - (1/φ^2)^n) -  1 * (1/φ^4)^n / (1 -
(1/φ^4)^n))
          has_sum (L (1/φ^2) - L(1/φ^4))) {1..}"
    unfolding L_def using pos
    by (intro has_sum_diff has_sum_lambert) (auto simp: field_simps intro!:
one_less_power)
  also have "(λn. 1 * (1/φ^2)^n / (1 - (1/φ^2)^n) -  1 * (1/φ^4)^n /
(1 - (1/φ^4)^n)) =
             (λn. 1 / (φ ^ (2 * n) - (1 / φ) ^ (2 * n)))" using pos
    by (simp add: divide_simps fun_eq_iff flip: power_mult)
       (simp add: algebra_simps flip: power_add)?
  also have "(... has_sum (L (1/φ^2) - L(1/φ^4))) {1..} ⟷
             ((λn. 1 / sqrt 5 * (1 / real (fib (2 * n)))) has_sum (L (1/φ^2)
- L(1/φ^4))) {1..}"
  proof (intro has_sum_cong)
    fix n :: nat assume n: "n ∈ {1..}"
    have "φ ^ (2 * n) - (1 / φ) ^ (2 * n) = (φ ^ (2 * n) - (-1 / φ) ^
(2 * n))"
      by simp
    also have "... = real (fib (2 * n)) * sqrt 5"
      by (subst fib_closed_form_alt) (simp add: φ_def)
    finally show "1 / (φ ^ (2 * n) - (1 / φ) ^ (2 * n)) = 1 / sqrt 5 *
(1 / real (fib (2 * n)))"
      by simp
  qed
  also have "1 / φ ^ 2 = (3 - sqrt 5) / 2"
    by (simp add: φ_def power_divide power2_eq_square divide_simps) (auto
simp: algebra_simps)?
  also have "1 / φ ^ 4 = (7 - 3 * sqrt 5) / 2"
    by (simp add: φ_def power_divide eval_nat_numeral divide_simps) (auto
simp: algebra_simps)?
```

```
  finally have "((λn. sqrt 5 * (1 / sqrt 5 * (1 / real (fib (2 * n)))))
has_sum
                  sqrt 5 * (L ((3 - sqrt 5) / 2) - L ((7 - 3 * sqrt 5)
/ 2))) {1..}"
    by (intro has_sum_cmult_right)
  thus ?thesis
    by simp
qed

end
```

References

[1] K. Knopp. *Theorie und Anwendung der Unendlichen Reihen*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2013.