Theta Functions

Manuel Eberl

March 17, 2025

Abstract

This entry defines the Ramanujan theta function

$$f(a,b) = \sum_{n=-\infty}^{\infty} a^{\frac{n(n+1)}{2}} b^{\frac{n(n-1)}{2}}$$

and derives from it the more commonly known Jacobi theta function on the unit disc

$$\vartheta_{00}(w,q) = \sum_{n=-\infty}^{\infty} w^{2n} q^{n^2},$$

its version in the complex plane

$$\vartheta_{00}(z;\tau) = \sum_{n=-\infty}^{\infty} \exp(i\pi(2nz + n^2\tau))$$

as well as its half-period variants ϑ_{01}, ϑ_{10}, and ϑ_{11}.

The most notable single result in this work is the proof of Jacobi's triple product

$$\prod_{n=1}^{\infty}(1-q^{2m})(1+q^{2m-1}w^2)(1+q^{2m-1}w^{-2}) = \sum_{k=-\infty}^{\infty} q^{k^2}w^{2k}$$

and its corollary, Euler's famous pentagonal number theorem:

$$\prod_{n=1}^{\infty}(1-q^n) = \sum_{k=-\infty}^{\infty} (-1)^k q^{k(3k-1)/2}$$

Contents

1 Auxiliary material

theory *Theta_Functions_Library*
imports
"HOL-Complex_Analysis.Complex_Analysis"
"HOL-Computational_Algebra.Computational_Algebra"
begin

1.1 Limits

abbreviation *"finite_sets_at_top ≡ finite_subsets_at_top UNIV"*

lemma *filterlim_atLeastAtMost_at_bot_at_top:*
fixes *f g :: "'a ⇒ 'b :: linorder_topology"*
assumes *"filterlim f at_bot F" "filterlim g at_top F"*
assumes *[simp]: "⋀a b. finite {a..b::'b}"*
shows *"filterlim (λx. {f x..g x}) finite_sets_at_top F"*
unfolding *filterlim_finite_subsets_at_top*
proof *safe*
fix *X :: "'b set"*
assume *X: "finite X"*
from *X* **obtain** *lb* **where** *lb: "⋀x. x ∈ X ⟹ lb ≤ x"*
by *(metis finite_has_minimal2 nle_le)*
from *X* **obtain** *ub* **where** *ub: "⋀x. x ∈ X ⟹ x ≤ ub"*
by *(metis all_not_in_conv finite_has_maximal nle_le)*
have *"eventually (λx. f x ≤ lb) F" "eventually (λx. g x ≥ ub) F"*
using *assms* **by** *(simp_all add: filterlim_at_bot filterlim_at_top)*
thus *"eventually (λx. finite {f x..g x} ∧ X ⊆ {f x..g x} ∧ {f x..g x} ⊆ UNIV) F"*
proof *eventually_elim*
case *(elim x)*
have *"X ⊆ {f x..g x}"*
proof
fix *y* **assume** *"y ∈ X"*
thus *"y ∈ {f x..g x}"*
using *lb[of y] ub[of y] elim* **by** *auto*
qed
thus *?case*
by *auto*
qed
qed

1.2 Continuity and analyticity

lemmas *[continuous_intros del] = continuous_on_power_int*

lemma *continuous_on_power_int [continuous_intros]:*
fixes *f :: "'a::topological_space ⇒ 'b::real_normed_div_algebra"*
assumes *"continuous_on s f"* **and** *"n ≥ 0 ∨ (∀x∈s. f x ≠ 0)"*
shows *"continuous_on s (λx. power_int (f x) n)"*

```
  using assms by (cases "n ≥ 0") (auto simp: power_int_def intro!: continuous_intros)

lemma analytic_on_powr [analytic_intros]:
  assumes "f analytic_on X" "g analytic_on X" "⋀x. x ∈ X ⟹ f x ∉
ℝ≤0"
  shows   "(λx. f x powr g x) analytic_on X"
proof -
  from assms(1) obtain X1 where X1: "open X1" "X ⊆ X1" "f analytic_on
X1"
    unfolding analytic_on_holomorphic by blast
  from assms(2) obtain X2 where X2: "open X2" "X ⊆ X2" "g analytic_on
X2"
    unfolding analytic_on_holomorphic by blast
  have X: "open (X2 ∩ (X1 ∩ f -‘ (-ℝ≤0)))"
    by (rule open_Int[OF _ continuous_open_preimage])
       (use X1 X2 in ‹auto intro!: holomorphic_on_imp_continuous_on analytic_imp_holomorphi
  have X': "X ⊆ X2 ∩ (X1 ∩ f -‘ (-ℝ≤0))"
    using assms(3) X1(2) X2(2) by blast
  note [holomorphic_intros] =
    analytic_imp_holomorphic[OF analytic_on_subset[OF X1(3)]]
    analytic_imp_holomorphic[OF analytic_on_subset[OF X2(3)]]
  have "(λx. exp (ln (f x) * g x)) holomorphic_on (X2 ∩ (X1 ∩ f -‘ (-ℝ≤0)))"
    by (intro holomorphic_intros) auto
  also have "?this ⟷ (λx. f x powr g x) holomorphic_on (X2 ∩ (X1 ∩
f -‘ (-ℝ≤0)))"
    by (intro holomorphic_cong) (auto simp: powr_def mult.commute)
  finally show ?thesis
    using X X' unfolding analytic_on_holomorphic by blast
qed

lemma holomorphic_on_powr [holomorphic_intros]:
  assumes "f holomorphic_on X" "g holomorphic_on X" "⋀x. x ∈ X ⟹ f
x ∉ ℝ≤0"
  shows   "(λx. f x powr g x) holomorphic_on X"
proof -
  have [simp]: "f x ≠ 0" if "x ∈ X" for x
    using assms(3)[OF that] by auto
  have "(λx. exp (ln (f x) * g x)) holomorphic_on X"
    by (auto intro!: holomorphic_intros assms(1,2)) (use assms(3) in auto)
  also have "?this ⟷ ?thesis"
    by (intro holomorphic_cong) (use assms(3) in ‹auto simp: powr_def
mult_ac›)
  finally show ?thesis .
qed

lemma continuous_powr_complex [continuous_intros]:
  assumes "continuous F f" "continuous F g"
  assumes "Re (f (netlimit F)) ≥ 0 ∨ Im (f (netlimit F)) ≠ 0"
  assumes "f (netlimit F) = 0 ⟶ Re (g (netlimit F)) > 0"
```

```
  shows   "continuous F (λz. f z powr g z :: complex)"
  using assms
  unfolding continuous_def
  by (intro tendsto_powr_complex')
     (auto simp: complex_nonpos_Reals_iff complex_eq_iff)

lemma continuous_powr_real [continuous_intros]:
  assumes "continuous F f" "continuous F g"
  assumes "f (netlimit F) = 0 ⟶ g (netlimit F) > 0 ∧ (∀F z in F. 0
≤ f z)"
  shows   "continuous F (λz. f z powr g z :: real)"
  using assms unfolding continuous_def by (intro tendsto_intros) auto
```

1.3 Formal power and Laurent series

```
lemma fps_nth_compose_linear [simp]:
  fixes f :: "'a :: comm_ring_1 fps"
  shows "fps_nth (fps_compose f (fps_const c * fps_X)) n = c ^ n * fps_nth
f n"
proof -
  have "fps_nth (fps_compose f (fps_const c * fps_X)) n =
        (∑i∈{n}. fps_nth f i * fps_nth ((fps_const c * fps_X) ^ i) n)"
    unfolding fps_compose_nth
    by (intro sum.mono_neutral_cong_right) (auto simp: power_mult_distrib)
  also have "... = c ^ n * fps_nth f n"
    by (simp add: power_mult_distrib)
  finally show ?thesis .
qed

lemma has_fps_expansionI:
  fixes f :: "'a :: {banach, real_normed_div_algebra} ⇒ 'a"
  assumes "eventually (λu. (λn. fps_nth F n * u ^ n) sums f u) (nhds
0)"
  shows   "f has_fps_expansion F"
proof -
  from assms obtain X where X: "open X" "0 ∈ X" "⋀u. u ∈ X ⟹ (λn.
fps_nth F n * u ^ n) sums f u"
    unfolding eventually_nhds by blast
  obtain r where r: "r > 0" "cball 0 r ⊆ X"
    using X(1,2) open_contains_cball by blast
  have "0 < norm (of_real r :: 'a)"
    using r(1) by simp
  also have "fps_conv_radius F ≥ norm (of_real r :: 'a)"
    unfolding fps_conv_radius_def
  proof (rule conv_radius_geI)
    have "of_real r ∈ X"
      using r by auto
    from X(3)[OF this] show "summable (λn. fps_nth F n * of_real r ^
n)"
```

```
    by (simp add: sums_iff)
  qed
  finally have "fps_conv_radius F > 0"
    by (simp_all add: zero_ereal_def)
  moreover have "(∀F z in nhds 0. eval_fps F z = f z)"
    using assms by eventually_elim (auto simp: sums_iff eval_fps_def)
  ultimately show ?thesis
    unfolding has_fps_expansion_def ..
qed

lemma fps_mult_numeral_left [simp]: "fps_nth (numeral c * f) n = numeral
c * fps_nth f n"
  by (simp add: fps_numeral_fps_const)

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

  have "(λn. fls_nth F (int n + N) * z powi (int n + N)) sums S"
    by fact
  also have "?this ⟷ (λn. fls_nth F (int (n+d) + N) * z powi (int (n+d)
+ N)) sums S"
    by (rule sums_zero_iff_shift [symmetric]) (use assms in ‹auto simp:
d_def N'_def›)
  also have "(λn. int (n+d) + N) = (λn. int n + N')"
    using assms by (auto simp: N'_def d_def)
  finally have "(λn. fls_nth F (int n + N') * z powi (int n + N')) sums
S" .
  hence "(λn. z powi (-N') * (fls_nth F (int n + N') * z powi (int n +
```

```
N'))) sums (z powi (-N') * S)"
    by (intro sums_mult)
  hence "(λn. fls_nth F (int n + N') * z ^ n) sums (z powi (-N') * S)"
    by (simp add: power_int_add power_int_minus field_simps)
  thus ?thesis
    by (simp add: eval_fls_def eval_fps_def sums_iff power_int_minus N'_def)
qed
```

1.4 Infinite sums

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

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_of_real:
  "f summable_on A ⟹ (λx. of_real (f x) :: 'a :: real_normed_algebra_1)
summable_on A"
  using summable_on_bounded_linear[of "of_real :: real ⇒ 'a", OF bounded_linear_of_real,
of f A]
  by simp

lemma has_sum_of_real_iff:
  "((λx. of_real (f x) :: 'a :: real_normed_div_algebra) has_sum (of_real
c)) A ⟷
   (f has_sum c) A"
proof -
  have "((λx. of_real (f x) :: 'a) has_sum (of_real c)) A ⟷
        (sum (λx. of_real (f x) :: 'a) ⟶ of_real c) (finite_subsets_at_top
A)"
    by (simp add: has_sum_def)
  also have "sum (λx. of_real (f x) :: 'a) = (λX. of_real (sum f X))"
    by simp
  also have "((λX. of_real (sum f X) :: 'a) ⟶ of_real c) (finite_subsets_at_top
A) ⟷
             (f has_sum c) A"
    unfolding has_sum_def tendsto_of_real_iff ..
  finally show ?thesis .
qed

lemma has_sum_of_real:
  "(f has_sum S) A ⟹ ((λx. of_real (f x) :: 'a :: real_normed_algebra_1)
has_sum of_real S) A"
  using has_sum_bounded_linear[of "of_real :: real ⇒ 'a", OF bounded_linear_of_real,
of f A S]
  by simp
```

```
lemma has_sum_finite_iff:
  fixes S :: "'a :: {topological_comm_monoid_add,t2_space}"
  assumes "finite A"
  shows   "(f has_sum S) A ⟷ S = (∑x∈A. f x)"
proof
  assume "S = (∑x∈A. f x)"
  thus "(f has_sum S) A"
    by (intro has_sum_finiteI assms)
next
  assume "(f has_sum S) A"
  moreover have "(f has_sum (∑x∈A. f x)) A"
    by (intro has_sum_finiteI assms) auto
  ultimately show "S = (∑x∈A. f x)"
    using has_sum_unique by blast
qed

lemma has_sum_finite_neutralI:
  assumes "finite B" "B ⊆ A" "⋀x. x ∈ A - B ⟹ f x = 0" "c = (∑x∈B.
f x)"
  shows   "(f has_sum c) A"
proof -
  have "(f has_sum c) B"
    by (rule has_sum_finiteI) (use assms in auto)
  also have "?this ⟷ (f has_sum c) A"
    by (intro has_sum_cong_neutral) (use assms in auto)
  finally show ?thesis .
qed

lemma has_sum_strict_mono_neutral:
  fixes f :: "'a ⇒ 'b :: {ordered_ab_group_add, topological_ab_group_add,
linorder_topology}"
  assumes ‹(f has_sum a) A› and "(g has_sum b) B"
  assumes ‹⋀x. x ∈ A∩B ⟹ f x ≤ g x›
  assumes ‹⋀x. x ∈ A-B ⟹ f x ≤ 0›
  assumes ‹⋀x. x ∈ B-A ⟹ g x ≥ 0›
  assumes ‹x ∈ B› ‹if x ∈ A then f x < g x else 0 < g x›
  shows "a < b"
proof -
  define y where "y = (if x ∈ A then f x else 0)"
  have "a - y ≤ b - g x"
  proof (rule has_sum_mono_neutral)
    show "(f has_sum (a - y)) (A - (if x ∈ A then {x} else {}))"
      by (intro has_sum_Diff assms has_sum_finiteI) (auto simp: y_def)
    show "(g has_sum (b - g x)) (B - {x})"
      by (intro has_sum_Diff assms has_sum_finiteI) (use assms in auto)
  qed (use assms in ‹auto split: if_splits›)
  moreover have "y < g x"
    using assms(3,4,5)[of x] assms(6-) by (auto simp: y_def split: if_splits)
  ultimately show ?thesis
```

```
    by (metis diff_strict_left_mono diff_strict_mono leD neqE)
qed

lemma has_sum_strict_mono:
  fixes f :: "'a ⇒ 'b :: {ordered_ab_group_add, topological_ab_group_add,
linorder_topology}"
  assumes ‹(f has_sum a) A› and "(g has_sum b) A"
  assumes ‹⋀x. x ∈ A ⟹ f x ≤ g x›
  assumes ‹x ∈ A› ‹f x < g x›
  shows "a < b"
  by (rule has_sum_strict_mono_neutral[OF assms(1,2), where x = x])
     (use assms(3-) in auto)

lemma has_sum_scaleR:
  fixes f :: "'a ⇒ 'b :: real_normed_vector"
  assumes "(f has_sum S) A"
  shows   "((λx. c *R f x) has_sum (c *R S)) A"
  using has_sum_bounded_linear[OF bounded_linear_scaleR_right[of c], of
f A S] assms by simp

lemma has_sum_scaleR_iff:
  fixes f :: "'a ⇒ 'b :: real_normed_vector"
  assumes "c ≠ 0"
  shows   "((λx. c *R f x) has_sum S) A ⟷ (f has_sum (S /R c)) A"
  using has_sum_scaleR[of f A "S /R c" c] has_sum_scaleR[of "λx. c *R
f x" A S "inverse c"] assms
  by auto

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]
  by (simp add: summable_on_def)

lemma sums_nonneg_imp_has_sum_strong:
  assumes "f sums (S::real)" "eventually (λn. f n ≥ 0) sequentially"
  shows   "(f has_sum S) UNIV"
proof -
  from assms(2) obtain N where N: "⋀n. n ≥ N ⟹ f n ≥ 0"
    by (auto simp: eventually_at_top_linorder)
  from assms(1) have "summable f"
    by (simp add: sums_iff)
  hence "summable (λn. f (n + N))"
    by (rule summable_ignore_initial_segment)
  hence "summable (λn. norm (f (n + N)))"
```

```
    using N by simp
  hence "summable (λn. norm (f n))"
    using summable_iff_shift by blast
  with assms(1) show ?thesis
    using norm_summable_imp_has_sum by blast
qed

lemma sums_nonneg_imp_has_sum:
  assumes "f sums (S::real)" and "⋀n. f n ≥ 0"
  shows   "(f has_sum S) UNIV"
  by (rule sums_nonneg_imp_has_sum_strong) (use assms in auto)

lemma summable_nonneg_imp_summable_on_strong:
  assumes "summable f" "eventually (λn. f n ≥ (0::real)) sequentially"
  shows   "f summable_on UNIV"
  using sums_nonneg_imp_has_sum_strong[OF _ assms(2)] assms(1) has_sum_imp_summable
by blast

lemma summable_nonneg_imp_summable_on:
  assumes "summable f" "⋀n. f n ≥ (0::real)"
  shows   "f summable_on UNIV"
  by (rule summable_nonneg_imp_summable_on_strong) (use assms in auto)

lemma Weierstrass_m_test_general':
  fixes f :: "'a ⇒ 'b ⇒ 'c :: banach"
  fixes M :: "'a ⇒ real"
  assumes norm_le:  "⋀x y. x ∈ X ⟹ y ∈ Y ⟹ norm (f x y) ≤ M x"
  assumes has_sum: "⋀y. y ∈ Y ⟹ ((λx. f x y) has_sum S y) X"
  assumes summable: "M summable_on X"
  shows "uniform_limit Y (λX y. ∑x∈X. f x y) S (finite_subsets_at_top
X)"
proof -
  have "uniform_limit Y (λX y. ∑x∈X. f x y) (λy. ∑∞x∈X. f x y) (finite_subsets_at_top
X)"
    using norm_le summable by (rule Weierstrass_m_test_general)
  also have "?this ⟷ ?thesis"
    by (intro uniform_limit_cong refl always_eventually allI ballI)
       (use has_sum in ‹auto simp: has_sum_iff›)
  finally show ?thesis .
qed
```

1.5 Miscellanea

```
lemma fraction_numeral_not_in_Ints [simp]:
  assumes "¬(numeral b :: int) dvd numeral a"
  shows   "numeral a / numeral b ∉ (ℤ :: 'a :: {division_ring, ring_char_0}
set)"
  using fraction_not_in_ints[of "numeral b" "numeral a", where ?'a =
'a] assms by simp
```

```
lemma fraction_numeral_not_in_Ints' [simp]:
  assumes "b ≠ Num.One"
  shows   "1 / numeral b ∉ (ℤ :: 'a :: {division_ring, ring_char_0} set)"
  using fraction_not_in_ints[of "numeral b" 1, where ?'a = 'a] assms
by simp

lemmas [simp] = not_in_Ints_imp_not_in_nonpos_Ints minus_in_Ints_iff



lemma power_int_power: "(a ^ b :: 'a :: division_ring) powi c = a powi
(int b * c)"
  by (subst power_int_mult) simp

lemma power_int_power': "(a powi b :: 'a :: division_ring) ^ c = a powi
(b * int c)"
  by (simp add: power_int_mult)



lemma real_sqrt_abs': "sqrt (abs x) = abs (sqrt x)"
  by (metis real_sqrt_abs2 real_sqrt_mult)

lemma power_int_nonneg_exp: "n ≥ 0 ⟹ x powi n = x ^ nat n"
  by (simp add: power_int_def)

lemma sin_npi_complex' [simp]: "sin (of_nat n * of_real pi) = 0"
  by (metis of_real_0 of_real_mult of_real_of_nat_eq sin_npi sin_of_real)

lemma cos_npi_complex' [simp]: "cos (of_nat n * of_real pi) = (-1) ^
n" for n
proof -
  have "cos (of_nat n * of_real pi :: 'a) = of_real (cos (real n * pi))"
    by (subst cos_of_real [symmetric]) simp
  also have "cos (real n * pi) = (-1) ^ n"
    by simp
  finally show ?thesis by simp
qed

lemma cis_power_int: "cis x powi n = cis (of_int n * x)"
  by (auto simp: power_int_def Complex.DeMoivre)

lemma complex_cnj_power_int [simp]: "cnj (x powi n) = cnj x powi n"
  by (auto simp: power_int_def)


lemma uniform_limit_singleton: "uniform_limit {x} f g F ⟷ ((λn. f
n x) ⟶ g x) F"
```

```
  by (simp add: uniform_limit_iff tendsto_iff)

lemma uniform_limit_compose':
  assumes "uniform_limit A f g F" and "h ‘ B ⊆ A"
  shows   "uniform_limit B (λn x. f n (h x)) (λx. g (h x)) F"
  unfolding uniform_limit_iff
proof safe
  fix e :: real
  assume e: "e > 0"
  from e and assms(1) have "∀⇩F n in F. ∀x∈A. dist (f n x) (g x) < e"
    by (auto simp: uniform_limit_iff)
  thus "∀⇩F n in F. ∀x∈B. dist (f n (h x)) (g (h x)) < e"
    by eventually_elim (use assms(2) in blast)
qed


lemma is_square_mult_prime_left_iff:
  assumes "prime p"
  shows   "is_square (p * x) ⟷ p dvd x ∧ is_square (x div p)"
proof
  assume *: "p dvd x ∧ is_square (x div p)"
  have [simp]: "p ≠ 0"
    using assms by auto
  from * obtain y where y: "x = y ^ 2 * p"
    by (auto elim!: dvdE is_nth_powerE simp: mult_ac)
  have "is_square ((p * y) ^ 2)"
    by auto
  also have "(p * y) ^ 2 = p * x"
    by (simp add: y power2_eq_square algebra_simps)
  finally show "is_square (p * x)" .
next
  assume *: "is_square (p * x)"
  have "p ≠ 0"
    using assms by auto
  from * obtain y where y: "p * x = y ^ 2"
    by (elim is_nth_powerE)
  have "p dvd y ^ 2"
    by (simp flip: y)
  hence "p dvd y"
    using ‹prime p› using prime_dvd_power by blast
  then obtain z where z: "y = p * z "
    by (elim dvdE)
  have "p * x = p * (p * z ^ 2)"
    by (simp add: y z algebra_simps power2_eq_square)
  hence x_eq: "x = p * z ^ 2"
    using ‹p ≠ 0› by simp
  show "p dvd x ∧ is_square (x div p)"
    using ‹p ≠ 0› by (simp add: x_eq)
qed
```

```
lemma is_square_mult2_nat_iff:
  "is_square (2 * b :: nat) ⟷ even b ∧ is_square (b div 2)"
  by (rule is_square_mult_prime_left_iff) auto

lemma is_square_mult2_int_iff:
  "is_square (2 * b :: int) ⟷ even b ∧ is_square (b div 2)"
  by (rule is_square_mult_prime_left_iff) auto

lemma is_nth_power_mult_cancel_left:
  fixes a b :: "'a :: semiring_gcd"
  assumes "is_nth_power n a" "a ≠ 0"
  shows   "is_nth_power n (a * b) ⟷ is_nth_power n b"
proof (cases "n > 0")
  case True
  show ?thesis
  proof
    assume "is_nth_power n (a * b)"
    then obtain x where x: "a * b = x ^ n"
      by (elim is_nth_powerE)
    obtain y where y: "a = y ^ n"
      using assms by (elim is_nth_powerE)
    have "y ^ n dvd x ^ n"
      by (simp flip: x y)
    hence "y dvd x"
      using ‹n > 0› by simp
    then obtain z where z: "x = y * z"
      by (elim dvdE)
    have "x ^ n = y ^ n * z ^ n"
      by (simp add: z power_mult_distrib)
    hence "b = z ^ n"
      using ‹a ≠ 0› by (simp flip: x y)
    thus "is_nth_power n b"
      by auto
  qed (use assms in ‹auto intro: is_nth_power_mult›)
qed (use assms in auto)

lemma is_nth_power_mult_cancel_right:
  fixes a b :: "'a :: semiring_gcd"
  assumes "is_nth_power n b" "b ≠ 0"
  shows   "is_nth_power n (a * b) ⟷ is_nth_power n a"
  by (subst mult.commute, subst is_nth_power_mult_cancel_left) (use assms
in auto)

end
```

2 Conversion from the complex plane to the nome

```
theory Nome
```

```
  imports "HOL-Complex_Analysis.Complex_Analysis" "HOL-Library.Going_To_Filter"
begin

definition to_nome :: "complex ⇒ complex"
  where "to_nome z = exp (i * of_real pi * z)"

lemma to_nome_nonzero [simp]: "to_nome z ≠ 0"
  by (simp add: to_nome_def)

lemma norm_to_nome: "norm (to_nome z) = exp (-pi * Im z)"
  by (simp add: to_nome_def)

lemma to_nome_add: "to_nome (z + w) = to_nome z * to_nome w"
  by (simp add: to_nome_def ring_distribs exp_add)

lemma to_nome_diff: "to_nome (z - w) = to_nome z / to_nome w"
  by (simp add: to_nome_def ring_distribs exp_diff)

lemma to_nome_minus: "to_nome (-z) = inverse (to_nome z)"
  by (simp add: to_nome_def exp_minus field_simps)

lemma to_nome_cnj: "to_nome (cnj z) = cnj (to_nome (-z))"
  by (simp add: to_nome_def exp_cnj)

lemma to_nome_power: "to_nome z ^ n = to_nome (of_nat n * z)"
  by (simp add: to_nome_def mult_ac flip: exp_of_nat_mult)

lemma to_nome_power_int: "to_nome z powi n = to_nome (of_int n * z)"
  by (auto simp: power_int_def to_nome_power simp flip: to_nome_minus)

lemma cis_conv_to_nome: "cis x = to_nome (of_real (x / pi))"
  by (simp add: cis_conv_exp to_nome_def)

lemma to_nome_powr:
  assumes "Re z ∈ {-1<..1}"
  shows   "to_nome z powr w = to_nome (z * w)"
proof -
  have "to_nome z powr w = exp (w * ln (exp (i * of_real pi * z)))"
    by (simp add: to_nome_def powr_def)
  also have "ln (exp (i * of_real pi * z)) = i * of_real pi * z"
    using mult_strict_left_mono[of "-1" "Re z" pi]
    by (subst Ln_exp) (use assms in auto)
  also have "exp (w * ...) = to_nome (z * w)"
    by (simp add: to_nome_def mult_ac)
  finally show ?thesis .
qed


lemma to_nome_0 [simp]: "to_nome 0 = 1"
```

```
  by (simp add: to_nome_def)

lemma to_nome_1 [simp]: "to_nome 1 = -1"
  and to_nome_neg1 [simp]: "to_nome (-1) = -1"
  by (simp_all add: to_nome_def exp_minus)

lemma to_nome_of_nat [simp]: "to_nome (of_nat n) = (-1) ^ n"
  by (simp add: to_nome_def complex_eq_iff Re_exp Im_exp)

lemma to_nome_of_int [simp]: "to_nome (of_int n) = (-1) powi n"
  by (simp add: to_nome_def complex_eq_iff Re_exp Im_exp)

lemma to_nome_one_half [simp]: "to_nome (1 / 2) = i"
  by (simp add: to_nome_def exp_eq_polar)

lemma to_nome_three_halves [simp]: "to_nome (3 / 2) = -i"
proof -
  have "to_nome (1 + 1 / 2) = -i"
    by (subst to_nome_add) auto
  thus ?thesis
    by simp
qed

lemma to_nome_eq_1_iff: "to_nome z = 1 ⟷ (∃n. even n ∧ z = of_int
n)"
proof -
  have "to_nome z = 1 ⟷ (∃n. z = complex_of_int (2 * n))"
    unfolding to_nome_def by (subst exp_eq_1) (auto simp: complex_eq_iff)
  also have "(∃n. z = complex_of_int (2 * n)) ⟷ (∃n. even n ∧ z =
of_int n)"
    by (metis dvd_def)
  finally show ?thesis .
qed

lemma to_nome_eq_neg1_iff: "to_nome z = -1 ⟷ (∃n. odd n ∧ z = of_int
n)"
proof -
  have "to_nome z = -1 ⟷ to_nome (z + 1) = 1"
    by (simp add: to_nome_add minus_equation_iff[of _ 1] eq_commute[of
"-1"])
  also have "... ⟷ (∃n. even n ∧ z + 1 = of_int n)"
    by (rule to_nome_eq_1_iff)
  also have "(∃n. even n ∧ z + 1 = of_int n) ⟷ (∃n. odd n ∧ z = of_int
n)"
  proof (intro iffI; elim exE)
    fix n assume "even n ∧ z + 1 = of_int n"
    thus "∃n. odd n ∧ z = of_int n"
      by (intro exI[of _ "n - 1"]) (auto simp: algebra_simps)
  next
```

```
    fix n assume "odd n ∧ z = of_int n"
    thus "∃n. even n ∧ z + 1 = of_int n"
      by (intro exI[of _ "n + 1"]) (auto simp: algebra_simps)
  qed
  finally show ?thesis .
qed

lemma to_nome_eq_1_iff': "to_nome z = 1 ⟷ (z / 2) ∈ ℤ"
proof
  assume "to_nome z = 1"
  then obtain n where "z = of_int n" "even n"
    by (subst (asm) to_nome_eq_1_iff) auto
  thus "z / 2 ∈ ℤ"
    by (auto elim!: evenE)
next
  assume "(z / 2) ∈ ℤ"
  then obtain n where "z / 2 = of_int n"
    by (auto elim!: Ints_cases)
  hence "z = of_int (2 * n)" "even (2 * n)"
    by simp_all
  thus "to_nome z = 1"
    using to_nome_eq_1_iff[of z] by blast
qed

lemma to_nome_eq_neg1_iff':"to_nome z = -1 ⟷ ((z-1) / 2) ∈ ℤ"
proof
  assume "to_nome z = -1"
  then obtain n where "z = of_int n" "odd n"
    by (subst (asm) to_nome_eq_neg1_iff) auto
  thus "((z-1) / 2) ∈ ℤ"
    by (auto elim!: oddE)
next
  assume "((z-1) / 2) ∈ ℤ"
  then obtain n where "((z-1) / 2) = of_int n"
    by (auto elim!: Ints_cases)
  hence "z = of_int (2 * n + 1)" "odd (2 * n + 1)"
    by (auto simp: algebra_simps)
  thus "to_nome z = -1"
    using to_nome_eq_neg1_iff[of z] by blast
qed

lemma to_nome_neg_one_half [simp]: "to_nome (-(1 / 2)) = -i"
  by (simp add: to_nome_def exp_eq_polar)

lemma to_nome_2 [simp]: "to_nome 2 = 1"
  by (simp add: to_nome_def exp_eq_polar mult.commute[of pi])


lemma holomorphic_to_nome [holomorphic_intros]:
```

```
  "f holomorphic_on A ⟹ (λz. to_nome (f z)) holomorphic_on A"
  unfolding to_nome_def by (intro holomorphic_intros)

lemma analytic_to_nome [analytic_intros]:
  "f analytic_on A ⟹ (λz. to_nome (f z)) analytic_on A"
  unfolding to_nome_def by (intro analytic_intros)

lemma tendsto_to_nome [tendsto_intros]:
  assumes "(f ⟶ w) F"
  shows   "((λz. to_nome (f z)) ⟶ to_nome w) F"
  using assms unfolding to_nome_def by (intro tendsto_intros)

lemma continuous_on_to_nome [continuous_intros]:
  assumes "continuous_on A f"
  shows   "continuous_on A (λz. to_nome (f z))"
  using assms unfolding to_nome_def by (intro continuous_intros)

lemma continuous_to_nome [continuous_intros]:
  assumes "continuous F f"
  shows   "continuous F (λz. to_nome (f z))"
  unfolding to_nome_def by (intro continuous_intros assms)


lemma tendsto_0_to_nome:
  assumes "filterlim (λx. Im (f x)) at_top F"
  shows   "filterlim (λx. to_nome (f x)) (nhds 0) F"
proof -
  have "((λx. exp (-(pi * x))) ⟶ 0) at_top"
    by real_asymp
  hence "((λx. exp (- (pi * Im (f x)))) ⟶ 0) F"
    by (rule filterlim_compose) fact
  hence "filterlim (λx. norm (to_nome (f x))) (nhds 0) F"
    by (simp add: norm_to_nome)
  thus ?thesis
    by (simp only: tendsto_norm_zero_iff)
qed

lemma tendsto_0_to_nome': "(to_nome ⟶ 0) (Im going_to at_top)"
  using tendsto_0_to_nome by fastforce

lemma filterlim_at_0_to_nome:
  assumes "filterlim (λx. Im (f x)) at_top F"
  shows   "filterlim (λx. to_nome (f x)) (at 0) F"
  by (intro filterlim_atI tendsto_0_to_nome assms) auto

end
```

3 General theta functions

```
theory Theta_Functions
imports
  Nome
  "Combinatorial_Q_Analogues.Q_Binomial_Identities"
  Theta_Functions_Library
begin
```

3.1 The Ramanujan theta function

We define the other theta functions in terms of the Ramanujan theta function:

$$f(a,b) = \sum_{n=-\infty}^{\infty} a^{n(n+1)/2} b^{n(n-1)/2} \qquad (\text{for } |ab| < 1)$$

This is, in some sense, more general than Jacobi's theta function: Jacobi's theta function can be expressed very easily in terms of Ramanujan's; the other direction is only straightforward in the real case. Due to the presence of square roots, the complex case becomes tedious due to branch cuts.

However, even in the complex case, results can be transferred from Jacobi's theta function to Ramanujan's by using the connection on the real line and then doing analytic continuation.

Some of the proofs below are loosely based on Ramanujan's lost notebook (as edited by Berndt [1]).

```
definition ramanujan_theta :: "'a :: {real_normed_field, banach} ⇒ 'a
⇒ 'a" where
  "ramanujan_theta a b =
     (if norm (a*b) < 1 then (∑∞n. a powi (n*(n+1) div 2) * b powi (n*(n-1)
div 2)) else 0)"

lemma ramanujan_theta_outside [simp]: "norm (a * b) ≥ 1 ⟹ ramanujan_theta
a b = 0"
  by (simp add: ramanujan_theta_def)

lemma uniform_limit_ramanujan_theta:
  fixes A :: "('a × 'a :: {real_normed_field, banach}) set"
  assumes "compact A" "⋀a b. (a, b) ∈ A ⟹ norm (a * b) < 1"
  shows   "uniform_limit A (λX (a,b). ∑n∈X. a powi (n*(n+1) div 2) *
b powi (n*(n-1) div 2))
             (λ(a,b). ∑∞n. a powi (n*(n+1) div 2) * b powi (n*(n-1)
div 2))
             (finite_subsets_at_top UNIV)"
proof (cases "A = {}")
  case False
  define f where "f = (λn ab. fst ab powi (n*(n+1) div 2) * snd ab powi
(n*(n-1) div 2) :: 'a)"
```

```
  define y where "y = max (1/2) (Sup ((λ(a,b). norm (a * b)) ‘ A))"
  define x where "x = max 2 (Sup ((λ(a,b). max (norm a) (norm b)) ‘ A))"

  have le_x: "norm a ≤ x" "norm b ≤ x" if "(a, b) ∈ A" for a b
  proof -
    have bounded: "bounded ((λ(a,b). max (norm a) (norm b)) ‘ A)"
      unfolding case_prod_unfold
      by (intro compact_imp_bounded compact_continuous_image continuous_intros
assms)
    have "(λ(a,b). max (norm a) (norm b)) (a, b) ≤ Sup ((λ(a,b). max
(norm a) (norm b)) ‘ A)"
      by (rule cSup_upper imageI)+
         (use that bounded in ‹auto intro: bounded_imp_bdd_above›)
    also have "... ≤ x"
      unfolding x_def by linarith
    finally show "norm a ≤ x" "norm b ≤ x"
      by simp_all
  qed

  have le_y: "norm (a*b) ≤ y" if "(a, b) ∈ A" for a b
  proof -
    have bounded: "bounded ((λ(a,b). norm (a * b)) ‘ A)"
      unfolding case_prod_unfold
      by (intro compact_imp_bounded compact_continuous_image continuous_intros
assms)
    have "(λ(a,b). norm (a * b)) (a, b) ≤ Sup ((λ(a,b). norm (a * b))
‘ A)"
      by (rule cSup_upper imageI)+
         (use that bounded in ‹auto intro: bounded_imp_bdd_above›)
    also have "... ≤ y"
      unfolding y_def by linarith
    finally show ?thesis
      by simp
  qed

  have "x > 1" "y > 0"
    unfolding x_def y_def by linarith+

  have "y < 1"
  proof -
    have "Sup ((λ(a,b). norm (a * b)) ‘ A) ∈ (λ(a,b). norm (a * b)) ‘
A" using ‹A ≠ {}›
      unfolding case_prod_unfold
      by (intro closed_contains_Sup compact_imp_closed compact_continuous_image
               bounded_imp_bdd_above compact_imp_bounded continuous_intros
assms) auto
    with assms show ?thesis
      by (auto simp: y_def)
  qed
```

```
  define h where "h = (λn. x ^ nat |n| * y ^ nat (min (n*(n+1) div 2)
(n*(n-1) div 2)))"

  have "uniform_limit A (λX wq. ∑n∈X. f n wq) (λwq. ∑∞n. f n wq) (finite_subsets_at_top
UNIV)"
  proof (rule Weierstrass_m_test_general, clarify)
    fix n :: int and a b :: 'a
    assume ab: "(a, b) ∈ A"
    have eq: "n * (n + 1) div 2 = n * (n - 1) div 2 + n"
      by (simp add: algebra_simps)
    have nonneg: "n * (n - 1) div 2 ≥ 0" "n * (n + 1) div 2 ≥ 0" "n *
(n - 1) ≥ 0" "n * (n + 1) ≥ 0"
      by (auto simp: zero_le_mult_iff)

    have "norm (f n (a, b)) = norm a powi (n*(n+1) div 2) * norm b powi
(n*(n-1) div 2)"
      by (simp add: f_def norm_mult norm_power_int)
    also have "(n*(n+1) div 2) = int (nat (n*(n+1) div 2))"
      by (auto simp: zero_le_mult_iff)
    also have "(n*(n-1) div 2) = int (nat (n*(n-1) div 2))"
      by (auto simp: zero_le_mult_iff)
    also have "norm a powi int (nat (n*(n+1) div 2)) * norm b powi int
(nat (n*(n-1) div 2)) =
               norm a ^ nat (n*(n+1) div 2) * norm b ^ nat (n*(n-1) div
2)"
      unfolding power_int_of_nat ..
    also have "... = (if n ≥ 0 then norm a ^ nat |n| * norm (a*b) ^ nat
(n*(n-1) div 2)
                              else norm b ^ nat |n| * norm (a*b) ^ nat
(n*(n+1) div 2))"
      using nonneg(1,2) [[linarith_split_limit = 0]] unfolding eq
      by (auto simp flip: power_add simp: power_mult_distrib norm_mult
nat_eq_iff nonneg(3,4)
               intro!: arg_cong[of _ _ "λn. x ^ n" for x] split: if_splits)
    also have "... ≤ x ^ nat |n| * norm (a*b) ^ nat (min (n*(n+1) div
2) (n*(n-1) div 2))"
      using le_x[of a b] ab ‹x > 1› le_y[of a b] ‹y < 1› [[linarith_split_limit
= 0]]
      by (auto intro!: mult_mono power_mono power_decreasing nat_mono)
    also have "... ≤ x ^ nat |n| * y ^ nat (min (n*(n+1) div 2) (n*(n-1)
div 2))"
      by (intro mult_left_mono power_mono le_y) (use ab ‹x > 1› in auto)
    also have "... = h n"
      by (simp add: h_def)
    finally show "norm (f n (a, b)) ≤ h n" .
  next
    obtain y' where y': "y < y'" "y' < 1"
      using ‹y < 1› dense by blast
```

```
    have "summable (λn. norm (h (int n)))"
    proof (rule summable_comparison_test_bigo)
      have "(λn. x ^ n * y ^ nat (min (int n * (int n + 1) div 2) (int
n * (int n - 1) div 2))) ∈ O(λn. y' ^ n)"
        using ‹x > 1› ‹y > 0› y' by real_asymp
      thus "(λn. norm (h (int n))) ∈ O(λn. y' ^ n)"
        unfolding h_def by (simp add: norm_power norm_mult nat_power_eq
power_int_def)
    next
      show "summable (λn. norm (y' ^ n))"
        unfolding norm_power by (rule summable_geometric) (use ‹y > 0›
y' in auto)
    qed
    hence "(λn. h (int n)) summable_on UNIV"
      by (rule norm_summable_imp_summable_on)
    also have "?this ⟷ h summable_on {0..}"
      by (rule summable_on_reindex_bij_witness[of _nat int]) auto
    finally have *: "h summable_on {0..}" .

    from * have "h summable_on {0<..}"
      by (rule summable_on_subset) auto
    also have "h summable_on {0<..} ⟷ h summable_on {..<0}"
      by (rule summable_on_reindex_bij_witness[of _ "λn. -n" "λn. -n"])

         (auto simp: h_def algebra_simps)
    finally have "h summable_on {..<0}" .
    from this and * have "h summable_on {..<0} ∪ {0..}"
      by (rule summable_on_union)
    also have "{..<0} ∪ {0..} = (UNIV :: int set)"
      by auto
    finally show "h summable_on UNIV" .
  qed
  thus ?thesis
    by (simp add: f_def case_prod_unfold)
qed auto

lemma has_sum_ramanujan_theta:
  assumes "norm (a*b) < 1"
  shows   "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
ramanujan_theta a b) UNIV"
proof -
  show ?thesis
    using uniform_limit_ramanujan_theta[of "{(a, b)}"] assms
    by (simp add: ramanujan_theta_def has_sum_def uniform_limit_singleton)
qed


lemma ramanujan_theta_commute: "ramanujan_theta a b = ramanujan_theta
b a"
```

```
proof (cases "norm (a * b) < 1")
  case ab: True
  have "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
ramanujan_theta a b) UNIV"
    by (intro has_sum_ramanujan_theta ab)
  also have "?this ⟷ ((λn. b powi (n*(n+1) div 2) * a powi (n*(n-1)
div 2)) has_sum ramanujan_theta a b) UNIV"
    by (intro has_sum_reindex_bij_witness[of _ uminus uminus]) (auto simp:
algebra_simps)
  finally have ... .
  moreover have "((λn. b powi (n*(n+1) div 2) * a powi (n*(n-1) div 2))
has_sum ramanujan_theta b a) UNIV"
    by (intro has_sum_ramanujan_theta) (use ab in ‹simp_all add: norm_mult
mult.commute›)
  ultimately show ?thesis
    using has_sum_unique by blast
qed (simp_all add: ramanujan_theta_def mult.commute)

lemma ramanujan_theta_0_left [simp]: "ramanujan_theta 0 b = 1 + b"
proof -
  have *: "n * (n + 1) div 2 = 0 ⟷ n ∈ {0, -1}" for n :: int
  proof -
    have "even (n * (n + 1))"
      by auto
    hence "n * (n + 1) div 2 = 0 ⟷ n * (n + 1) = 0"
      by (elim evenE) simp_all
    also have "... ⟷ n ∈ {0, -1}"
      unfolding mult_eq_0_iff by auto
    finally show ?thesis .
  qed
  have "((λn. 0 powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
(1 + b)) {0, -1}"
    by (rule has_sum_finiteI) auto
  also have "?this ⟷ ((λn. 0 powi (n*(n+1) div 2) * b powi (n*(n-1)
div 2)) has_sum (1 + b)) UNIV"
    by (intro has_sum_cong_neutral) (auto simp: *)
  finally have "((λn. 0 powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
has_sum (1 + b)) UNIV" .
  moreover have "((λn. 0 powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
has_sum ramanujan_theta 0 b) UNIV"
    by (intro has_sum_ramanujan_theta) auto
  ultimately show ?thesis
    using has_sum_unique by blast
qed

lemma ramanujan_theta_0_right [simp]: "ramanujan_theta a 0 = 1 + a"
  by (subst ramanujan_theta_commute) simp_all

lemma has_sum_ramanujan_theta1:
```

```
  assumes "norm (a*b) < 1" and [simp]: "a ≠ 0"
  shows   "((λn. a powi n * (a*b) powi (n*(n-1) div 2)) has_sum ramanujan_theta
a b) UNIV"
proof -
  have eq: "n*(n+1) div 2 = n*(n-1) div 2 + n" for n :: int
    by (cases "even n") (auto elim!: evenE oddE simp: algebra_simps)
  have "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
ramanujan_theta a b) UNIV"
    by (rule has_sum_ramanujan_theta) (use assms in auto)
  thus ?thesis
    unfolding eq by (simp add: power_int_mult_distrib power_int_add mult_ac)
qed

lemma has_sum_ramanujan_theta2:
  assumes "norm (a * b) < 1"
  shows    "((λn. (a*b) powi (n*(n-1) div 2) * (a powi n + b powi n))
has_sum
              (ramanujan_theta a b - 1)) {1..}"
proof (cases "a * b = 0")
  case True
  have "((λn. (a*b) powi (n*(n-1) div 2) * (a powi n + b powi n)) has_sum
(ramanujan_theta a b - 1)) {1}"
    using True by (intro has_sum_finiteI) auto
  also have "?this ⟷ ?thesis"
    using True by (intro has_sum_cong_neutral) (auto simp: dvd_div_eq_0_iff)

  finally show ?thesis .
next
  case False
  hence [simp]: "a ≠ 0" "b ≠ 0"
    by auto
  define S1 where "S1 = (∑∞n∈{1..}. a powi (n*(n+1) div 2) * b powi
(n*(n-1) div 2))"
  define S2 where "S2 = (∑∞n∈{..-1}. a powi (n*(n+1) div 2) * b powi
(n*(n-1) div 2))"
  have eq: "n*(n+1) div 2 = n*(n-1) div 2 + n" for n :: int
    by (cases "even n") (auto elim!: evenE oddE simp: algebra_simps)

  have 1: "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
ramanujan_theta a b) UNIV"
    by (rule has_sum_ramanujan_theta) (use assms in auto)

  have [intro]: "(λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
summable_on A" for A
    by (rule summable_on_subset_banach, rule has_sum_imp_summable[OF 1])
auto

  have S1: "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
S1) {1..}"
```

```
    unfolding S1_def by (rule has_sum_infsum) rule
  also have "(λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) =
             (λn. (a*b) powi (n*(n-1) div 2) * a powi n)"
    unfolding eq by (auto simp: power_int_mult_distrib power_int_add mult_ac)
  finally have S1': "((λn. (a * b) powi (n * (n - 1) div 2) * a powi n)
has_sum S1) {1..}" .

  have S2: "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
S2) {..-1}"
    unfolding S2_def by (rule has_sum_infsum) rule
  also have "?this ⟷ ((λn. b powi (n*(n+1) div 2) * a powi (n*(n-1)
div 2)) has_sum S2) {1..}"
    by (rule has_sum_reindex_bij_witness[of _ uminus uminus]) (auto simp:
algebra_simps)
  also have "(λn. b powi (n*(n+1) div 2) * a powi (n*(n-1) div 2)) =
             (λn. (a*b) powi (n*(n-1) div 2) * b powi n)"
    unfolding eq by (auto simp: power_int_mult_distrib power_int_add mult_ac)
  finally have S2': "((λn. (a * b) powi (n * (n - 1) div 2) * b powi n)
has_sum S2) {1..}" .

  have "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)) has_sum
(ramanujan_theta a b - 1)) (UNIV-{0})"
    using 1 by (rule has_sum_Diff) (auto simp: has_sum_finite_iff)
  also have "UNIV - {0::int} = {1..} ∪ {..-1}"
    by auto
  finally have "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
has_sum
                  (ramanujan_theta a b - 1)) ({1..} ∪ {..-1})" .
  moreover have "((λn. a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
has_sum
                  (S1 + S2)) ({1..} ∪ {..-1})"
    by (intro has_sum_Un_disjoint S1 S2) auto
  ultimately have "ramanujan_theta a b - 1 = S1 + S2"
    using has_sum_unique by blast

  moreover have "((λn. (a * b) powi (n * (n - 1) div 2) * (a powi n +
b powi n)) has_sum (S1 + S2)) {1..}"
    using has_sum_add[OF S1' S2'] by (simp add: algebra_simps)
  ultimately show "((λn. (a*b) powi (n*(n-1) div 2) * (a powi n + b powi
n))
                    has_sum (ramanujan_theta a b - 1)) {1..}"
    by simp
qed

lemma ramanujan_theta_of_real:
  "ramanujan_theta (of_real a) (of_real b) = of_real (ramanujan_theta
a b)"
proof (cases "norm (a*b) < 1")
  case ab: True
```

```
  have "((λn. of_real (a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))
:: 'a) has_sum
          of_real (ramanujan_theta a b)) UNIV"
    by (intro has_sum_of_real has_sum_ramanujan_theta) (use ab in auto)
  also have "(λn. of_real (a powi (n*(n+1) div 2) * b powi (n*(n-1) div
2)) :: 'a) =
                 (λn. of_real a powi (n*(n+1) div 2) * of_real b powi
(n*(n-1) div 2))" by simp
  finally have "((λn. of_real a powi (n * (n + 1) div 2) * of_real b powi
(n * (n - 1) div 2)) has_sum
                  (of_real (ramanujan_theta a b) :: 'a)) UNIV" .
  moreover have "((λn. of_real a powi (n*(n+1) div 2) * of_real b powi
(n*(n-1) div 2) :: 'a) has_sum
                    ramanujan_theta (of_real a) (of_real b)) UNIV"
    by (rule has_sum_ramanujan_theta) (use ab in ‹auto simp: norm_mult›)
  ultimately show ?thesis
    using has_sum_unique by blast
qed (auto simp: ramanujan_theta_def norm_mult abs_mult)

lemma ramanujan_theta_cnj:
  "ramanujan_theta (cnj a) (cnj b) = cnj (ramanujan_theta a b)"
proof (cases "norm (a*b) < 1")
  case ab: True
  have "((λn. cnj (a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2))) has_sum
cnj (ramanujan_theta a b)) UNIV"
    unfolding has_sum_cnj_iff by (intro has_sum_ramanujan_theta) (use
ab in auto)
  also have "(λn. cnj (a powi (n*(n+1) div 2) * b powi (n*(n-1) div 2)))
=
             (λn. cnj a powi (n*(n+1) div 2) * cnj b powi (n*(n-1) div
2))"
    by simp
  finally have "((λn. cnj a powi (n*(n+1) div 2) * cnj b powi (n*(n-1)
div 2)) has_sum
                  cnj (ramanujan_theta a b)) UNIV" .
  moreover have "((λn. cnj a powi (n*(n+1) div 2) * cnj b powi (n*(n-1)
div 2)) has_sum
                    ramanujan_theta (cnj a) (cnj b)) UNIV"
    by (rule has_sum_ramanujan_theta) (use ab in ‹auto simp: norm_mult›)
  ultimately show ?thesis
    using has_sum_unique by blast
qed (auto simp: ramanujan_theta_def norm_mult)

lemma ramanujan_theta_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z * g z) < 1"  "open A"
  shows   "(λz. ramanujan_theta (f z) (g z)) holomorphic_on A"
proof -
  have "(λz. ramanujan_theta (f z) (g z)) analytic_on {z}" if "z ∈ A"
```

```
for z
  proof -
    obtain r where r: "r > 0" "cball z r ⊆ A"
      using ‹open A› ‹z ∈ A› open_contains_cball_eq by blast
    define h where "h = (λX (w,q). ∑n∈X. w powi (n*(n+1) div 2) * q
powi (n*(n-1) div 2) :: complex)"
    define H where "H = (λ(w,q). ∑∞n. w powi (n*(n+1) div 2) * q powi
(n*(n-1) div 2) :: complex)"

    have lim: "uniform_limit (cball z r)
                (λX x. h X (f x, g x)) (λx. H (f x, g x)) (finite_subsets_at_top
UNIV)"
      unfolding h_def H_def
    proof (rule uniform_limit_compose'[OF uniform_limit_ramanujan_theta])
      show "compact ((λx. (f x, g x)) ‘ cball z r)" using r
        by (intro compact_continuous_image)
           (auto intro!: continuous_intros holomorphic_on_imp_continuous_on
                         assms(1,2)[THEN holomorphic_on_subset])
    qed (use r assms(3,4) in auto)

    have "(λx. H (f x, g x)) holomorphic_on ball z r"
      by (rule holomorphic_uniform_limit[OF _ lim])
         (use r in ‹auto intro!: always_eventually continuous_intros holomorphic_intros
                                 holomorphic_on_imp_continuous_on
                                 assms(1,2)[THEN holomorphic_on_subset]
assms(3)
                         simp: h_def zero_le_mult_iff›)
    also have "?this ⟷ (λx. ramanujan_theta (f x) (g x)) holomorphic_on
ball z r"
    proof (rule holomorphic_cong)
      fix w assume "w ∈ ball z r"
      hence "w ∈ A"
        using r by auto
      hence "norm (f w * g w) < 1"
        using assms(3) by auto
      thus "H (f w, g w) = ramanujan_theta (f w) (g w)"
        by (auto simp: H_def ramanujan_theta_def)
    qed auto
    finally show ?thesis
      using ‹r > 0› analytic_at_ball by blast
  qed
  hence "(λz. ramanujan_theta (f z) (g z)) analytic_on A"
    using analytic_on_analytic_at by blast
  thus ?thesis
    using analytic_imp_holomorphic by auto
qed

lemma ramanujan_theta_analytic [analytic_intros]:
  assumes "f analytic_on A" "g analytic_on A" "⋀z. z ∈ A ⟹ norm (f
```

```
z * g z) < 1"
  shows   "(λz. ramanujan_theta (f z) (g z)) analytic_on A"
proof -
  from assms(1) obtain B1 where B1: "open B1" "A ⊆ B1" "f holomorphic_on
B1"
    using analytic_on_holomorphic by metis
  from assms(2) obtain B2 where B2: "open B2" "A ⊆ B2" "g holomorphic_on
B2"
    using analytic_on_holomorphic by metis
  note [holomorphic_intros] = holomorphic_on_subset[OF B1(3)] holomorphic_on_subset[OF
B2(3)]

  define B3 where "B3 = B1 ∩ B2 ∩ (λz. f z * g z) -‘ ball 0 1"
  have "open B3" using B1 B2 unfolding B3_def
    by (intro continuous_open_preimage holomorphic_on_imp_continuous_on
          holomorphic_intros open_halfspace_Im_gt) auto
  hence B3: "open B3" "B3 ⊆ B1" "B3 ⊆ B2" "⋀z. z ∈ B3 ⟹ f z * g z
∈ ball 0 1"
    unfolding B3_def by auto

  have "(λz. ramanujan_theta (f z) (g z)) holomorphic_on B3"
    using B3 by (auto intro!: holomorphic_intros)
  moreover have "A ⊆ B3"
    using assms(3) B1 B2 by (auto simp: B3_def)
  ultimately show ?thesis
    using ‹open B3› analytic_on_holomorphic by metis
qed

lemma tendsto_ramanujan_theta [tendsto_intros]:
  fixes f g :: "'a ⇒ 'b :: {real_normed_field, banach, heine_borel}"
  assumes "(f ⟶ a) F" "(g ⟶ b) F" "norm (a * b) < 1"
  shows   "((λz. ramanujan_theta (f z) (g z)) ⟶ ramanujan_theta a
b) F"
proof -
  have "isCont (λ(w,q). ramanujan_theta w q) z" if z: "norm (fst z * snd
z) < 1" for z :: "'b × 'b"
  proof -
    have "z ∈ (λz. (fst z * snd z)) -‘ ball 0 1"
      using z by auto
    moreover have "open ((λz. (fst z * snd z :: 'b)) -‘ ball 0 1)"
      by (intro open_vimage continuous_intros) auto
    ultimately obtain r where r: "r > 0" "cball z r ⊆ (λz. (fst z *
snd z)) -‘ ball 0 1"
      by (meson open_contains_cball)

    have "continuous_on (cball z r)
          (λ(a, b). ∑∞n. a powi (n * (n + 1) div 2) * b powi (n *
(n - 1) div 2))"
    proof (rule uniform_limit_theorem)
```

```
      show "uniform_limit (cball z r)
              (λX (a, b). ∑n∈X. a powi (n * (n + 1) div 2) * b powi
(n * (n - 1) div 2))
              (λ(a, b). ∑∞n. a powi (n * (n + 1) div 2) * b powi (n
* (n - 1) div 2))
              (finite_subsets_at_top UNIV)"
        by (rule uniform_limit_ramanujan_theta) (use r in auto)
    qed (auto intro!: always_eventually continuous_intros
              simp: case_prod_unfold dist_norm zero_le_mult_iff)
    also have "?this ⟷ continuous_on (cball z r) (λ(a,b). ramanujan_theta
a b)"
      by (intro continuous_on_cong) (use r in ‹auto simp: ramanujan_theta_def›)
    finally have "continuous_on (ball z r) (λ(a,b). ramanujan_theta a b)"
      by (rule continuous_on_subset) auto
    thus ?thesis
      using ‹r > 0› centre_in_ball continuous_on_interior interior_ball
by blast
  qed
  from this[of "(a, b)"] have isCont: "isCont (λ(w,q). ramanujan_theta
w q) (a, b)"
    using assms by simp
  have lim: "((λx. (f x, g x)) ⟶ (a, b)) F"
    using assms by (intro tendsto_intros)
  show ?thesis
    using isCont_tendsto_compose[OF isCont lim] by simp
qed

lemma continuous_on_ramanujan_theta [continuous_intros]:
  fixes f g :: "'a :: topological_space ⇒ 'b :: {real_normed_field, banach,
heine_borel}"
  assumes "continuous_on A f" "continuous_on A g" "⋀z. z ∈ A ⟹ norm
(f z * g z) < 1"
  shows   "continuous_on A (λz. ramanujan_theta (f z) (g z))"
proof -
  have *: "continuous_on {z. norm (fst z * snd z) < 1} (λ(a,b). ramanujan_theta
a b :: 'b)"
    unfolding continuous_on by (auto intro!: tendsto_eq_intros simp: case_prod_unfold)
  have "continuous_on A ((λ(x,y). ramanujan_theta x y) ∘ (λx. (f x, g
x)))"
    by (intro continuous_on_compose continuous_on_subset[OF *] continuous_intros)
       (use assms in auto)
  thus ?thesis
    by (simp add: o_def)
qed

lemma continuous_ramanujan_theta [continuous_intros]:
  fixes f g :: "'a :: t2_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"
  assumes "continuous F f" "continuous F g" "norm (f (netlimit F) * g
(netlimit F)) < 1"
```

```
  shows   "continuous F (λz. ramanujan_theta (f z) (g z))"
  unfolding continuous_def
  using assms by (auto intro!: tendsto_eq_intros simp: continuous_def)

lemma ramanujan_theta_1_left:
  "ramanujan_theta 1 a = 2 * ramanujan_theta a (a ^ 3)"
proof (cases "a ≠ 0 ∧ norm a < 1")
  case False
  hence "a = 0 ∨ norm a ≥ 1"
    by auto
  thus ?thesis
  proof
    assume "norm a ≥ 1"
    thus ?thesis
      by (auto simp: ramanujan_theta_def norm_power power_less_one_iff
               simp flip: power_Suc2 power_Suc)
  qed auto
next
  case a: True
  hence [simp]: "a ≠ 0"
    by auto
  define S1 where "S1 = (∑∞n∈{0..}. a powi (n*(n+1) div 2))"
  define S2 where "S2 = (∑∞n∈{..-1}. a powi (n*(n+1) div 2))"

  have 1: "((λn. a powi (n*(n+1) div 2)) has_sum ramanujan_theta 1 a)
UNIV"
    using has_sum_ramanujan_theta[of a 1] a by (simp add: ramanujan_theta_commute)
  have summable: "(λn. a powi (n*(n+1) div 2)) summable_on A" for A
    by (rule summable_on_subset_banach, rule has_sum_imp_summable[OF 1])
auto

  have S1: "((λn. a powi (n*(n+1) div 2)) has_sum S1) {0..}"
    unfolding S1_def by (rule has_sum_infsum, rule summable)
  also have "?this ⟷ ((λn. a powi (n*(n+1) div 2)) has_sum S1) {..-1}"
    by (intro has_sum_reindex_bij_witness[of _ "λn. -n-1" "λn. -n-1"])
(auto simp: algebra_simps)
  finally have S1': "((λn. a powi (n*(n+1) div 2)) has_sum S1) {..-1}"
.

  have "((λn. a powi (n*(n+1) div 2)) has_sum (S1 + S1)) ({..-1} ∪ {0..})"
    by (intro has_sum_Un_disjoint S1 S1') auto
  also have "{..-1} ∪ {0::int..} = UNIV"
    by auto
  finally have "((λn. a powi (n * (n + 1) div 2)) has_sum (2*S1)) UNIV"
    by simp
  with 1 have "ramanujan_theta 1 a = 2 * S1"
    using has_sum_unique by blast

  define S2 where "S2 = (∑∞n | n ≥ 0 ∧ even n. a powi (n*(n+1) div
```

```
2))"
  define S3 where "S3 = (∑∞n | n ≥ 0 ∧ odd n. a powi (n*(n+1) div 2))"
  have "((λn. a powi (n*(n+1) div 2)) has_sum (S2 + S3)) ({n. n ≥ 0 ∧
even n} ∪ {n. n ≥ 0 ∧ odd n})"
    unfolding S2_def S3_def by (intro has_sum_Un_disjoint has_sum_infsum
summable) auto
  also have "{n. n ≥ 0 ∧ even n} ∪ {n. n ≥ 0 ∧ odd n} = {0::int..}"
    by auto
  finally have "S1 = S2 + S3"
    using S1 has_sum_unique by blast

  have "((λn. a powi (n*(n+1) div 2)) has_sum S2) {n. n ≥ 0 ∧ even n}"
    unfolding S2_def by (intro has_sum_Un_disjoint has_sum_infsum summable)
  also have "?this ⟷ ((λn. a powi (n*(2*n+1))) has_sum S2) {0..}"
    by (intro has_sum_reindex_bij_witness[of _ "λn. 2*n" "λn. n div 2"])
auto
  also have "(λn::int. n*(2*n+1)) = (λn. (n*(n-1) div 2) + 3*((n*(n+1))
div 2))"
  proof
    fix n :: int
    show "n*(2*n+1) = (n*(n-1) div 2) + 3*((n*(n+1)) div 2)"
      by (cases "even n") (auto elim!: evenE oddE simp: algebra_simps)
  qed
  also have "(λn. a powi ... n) = (λn. a powi (n*(n-1) div 2) * (a ^ 3)
powi (n*(n+1) div 2))"
    by (simp add: power_int_add power_int_power)
  finally have S2: "((λn. a powi (n*(n-1) div 2) * (a ^ 3) powi (n*(n+1)
div 2)) has_sum S2) {0..}" .

  have "((λn. a powi (n*(n+1) div 2)) has_sum S3) {n. n ≥ 0 ∧ odd n}"
    unfolding S3_def by (intro has_sum_Un_disjoint has_sum_infsum summable)
  also have "?this ⟷ ((λn. a powi (n*(2*n-1))) has_sum S3) {1..}"
    by (intro has_sum_reindex_bij_witness[of _ "λn. 2*n-1" "λn. (n+1)
div 2"])
       (auto elim!: oddE simp: algebra_simps)
  also have "(λn::int. n*(2*n-1)) = (λn. (n*(n+1) div 2) + 3*(n*(n-1)
div 2))"
  proof
    fix n :: int
    show "n*(2*n-1) = (n*(n+1) div 2) + 3*(n*(n-1) div 2)"
      by (cases "even n") (auto elim!: evenE oddE simp: algebra_simps)
  qed
  also have "(λn. a powi ... n) = (λn. a powi (n*(n+1) div 2) * (a^3)
powi (n*(n-1) div 2))"
    by (simp add: power_int_add power_int_power)
  finally have "((λn. a powi (n*(n+1) div 2) * (a^3) powi (n*(n-1) div
2)) has_sum S3) {1..}" .
  also have "?this ⟷ ((λn. a powi (n*(n-1) div 2) * (a^3) powi (n*(n+1)
div 2)) has_sum S3) {..-1}"
```

```
    by (intro has_sum_reindex_bij_witness[of _ "λn. -n" "λn. -n"]) (auto
simp: algebra_simps)
  finally have S3: "((λn. a powi (n*(n-1) div 2) * (a^3) powi (n*(n+1)
div 2)) has_sum S3) {..-1}" .

  have "((λn. a powi (n*(n-1) div 2) * (a^3) powi (n*(n+1) div 2)) has_sum
(S2 + S3)) ({0..} ∪ {..-1})"
    by (intro has_sum_Un_disjoint S2 S3) auto
  also have "{0::int..} ∪ {..-1} = UNIV"
    by auto
  finally have "((λn. (a^3) powi (n*(n+1) div 2) * a powi (n*(n-1) div
2)) has_sum S2 + S3) UNIV"
    by (simp add: mult.commute)
  moreover have "((λn. (a^3) powi (n*(n+1) div 2) * a powi (n*(n-1) div
2)) has_sum
                  ramanujan_theta (a^3) a) UNIV"
    by (intro has_sum_ramanujan_theta)
       (use a in ‹auto simp: norm_power power_less_one_iff simp flip:
power_Suc2›)
  ultimately have "ramanujan_theta (a^3) a = S2 + S3"
    using has_sum_unique by blast
  also have "S2 + S3 = S1"
    by (rule sym) fact
  also have "S1 = ramanujan_theta 1 a / 2"
    using ‹ramanujan_theta 1 a = 2 * S1› by (simp add: field_simps)
  finally show ?thesis
    by (simp add: field_simps ramanujan_theta_commute)
qed

lemma ramanujan_theta_1_right: "ramanujan_theta a 1 = 2 * ramanujan_theta
a (a ^ 3)"
  by (subst ramanujan_theta_commute, rule ramanujan_theta_1_left)

lemma ramanujan_theta_neg1_left [simp]: "ramanujan_theta (-1) a = 0"
proof (cases "a ≠ 0 ∧ norm a < 1")
  case False
  hence "a = 0 ∨ norm a ≥ 1"
    by auto
  thus ?thesis
  proof
    assume "norm a ≥ 1"
    thus ?thesis
      by (auto simp: ramanujan_theta_def norm_power power_less_one_iff
               simp flip: power_Suc2 power_Suc)
  qed auto
next
  case a: True
  hence [simp]: "a ≠ 0"
    by auto
```

```
  define S1 where "S1 = (∑∞n∈{1..}. (-1) powi (n*(n+1) div 2) * a powi
(n*(n-1) div 2))"
  define S2 where "S2 = (∑∞n∈{..-2}. (-1) powi (n*(n+1) div 2) * a
powi (n*(n-1) div 2))"

  have sum: "((λn. (-1) powi (n*(n+1) div 2) * a powi (n*(n-1) div 2))
has_sum ramanujan_theta (-1) a) UNIV"
    using has_sum_ramanujan_theta[of "-1" a] a by simp
  have summable: "(λn. (-1) powi (n*(n+1) div 2) * a powi (n*(n-1) div
2)) summable_on A" for A
    by (rule summable_on_subset_banach, rule has_sum_imp_summable[OF sum])
auto

  have S1: "((λn. (-1) powi (n*(n+1) div 2) * a powi (n*(n-1) div 2))
has_sum S1) {1..}"
    unfolding S1_def by (rule has_sum_infsum, rule summable)
  also have "?this ⟷ ((λn. (-1) powi ((n-2)*(n-1) div 2) * a powi (n*(n-1)
div 2)) has_sum S1) {..0}"
    by (intro has_sum_reindex_bij_witness[of _ "λn. -n+1" "λn. -n+1"])
(auto simp: algebra_simps)
  also have "(λn. (n-2)*(n-1) div 2) = (λn::int. n*(n+1) div 2 - 2 * n
+ 1)"
    by (auto simp: algebra_simps)
  also have "(λn. (-1) powi (n*(n+1) div 2 - 2*n + 1)) =
             (λn. - ((-1) powi (n*(n+1) div 2) :: 'a))"
    by (simp add: power_int_add power_int_diff)
  finally have S1': "((λn. ((-1) powi (n*(n+1) div 2) * a powi (n*(n-1)
div 2))) has_sum (-S1)) {..0}"
    by (simp add: has_sum_uminus)

  have "((λn. ((-1) powi (n*(n+1) div 2) * a powi (n*(n-1) div 2))) has_sum
(-S1 + S1)) ({..0} ∪ {1..})"
    by (intro has_sum_Un_disjoint S1 S1') auto
  also have "{..0} ∪ {1::int..} = UNIV"
    by auto
  also have "-S1 + S1 = 0"
    by simp
  finally show ?thesis
    using sum has_sum_unique by blast
qed

lemma ramanujan_theta_neg1_right [simp]: "ramanujan_theta a (-1) = 0"
  by (subst ramanujan_theta_commute) auto

lemma ramanujan_theta_mult_power_int:
  assumes [simp]: "a ≠ 0" "b ≠ 0"
  shows   "ramanujan_theta a b =
             a powi (m*(m+1) div 2) * b powi (m*(m-1) div 2) *
             ramanujan_theta (a * (a*b) powi m) (b * (a*b) powi (-m))"
```

```
proof (cases "norm (a * b) < 1")
  case False
  thus ?thesis
    by (simp add: ramanujan_theta_def field_simps power_int_minus)
next
  case True
  hence [simp]: "a ≠ 0" "b ≠ 0"
    by auto
  define e1 e2 where "e1 = (m*(m+1) div 2)" and "e2 = (m*(m-1) div 2)"
  define a' b' where "a' = a*(a*b) powi m" and "b' = b*(a*b) powi -m"
  have eq: "n * (n + 1) div 2 = n * (n - 1) div 2 + n" for n :: int
    by (auto simp: algebra_simps)

  have "((λn. a powi e1 * b powi e2 * (a' powi (n*(n+1) div 2) * b' powi
(n*(n-1) div 2)))
          has_sum (a powi e1 * b powi e2 * ramanujan_theta a' b')) UNIV"
    by (intro has_sum_cmult_right has_sum_ramanujan_theta)
       (use True in ‹auto simp: a'_def b'_def power_int_minus field_simps›)

  also have "(λn. a powi e1 * b powi e2 * (a' powi (n*(n+1) div 2) * b'
powi (n*(n-1) div 2))) =
             (λn. a powi ((n+m)*(n+m+1) div 2) * b powi ((n+m)*(n+m-1)
div 2))" (is "?lhs = ?rhs")
  proof
    fix n :: int
    have "a powi e1 * b powi e2 * (a' powi (n*(n+1) div 2) * b' powi (n*(n-1)
div 2)) =
          a powi (e1 + (n*(n+1) div 2) + m*(n*(n+1) div 2) - m*(n*(n-1)
div 2)) *
          b powi (e2 + (n*(n-1) div 2) + m*(n*(n+1) div 2) - m*(n*(n-1)
div 2))"
      unfolding a'_def b'_def
      by (simp add: a'_def b'_def power_int_mult_distrib power_int_add
power_int_diff power_int_minus
                    power_int_divide_distrib field_simps flip: power_int_mult)

    also have "e1 + (n*(n+1) div 2) + m*(n*(n+1) div 2) - m*(n*(n-1) div
2) =
               (m * (m + 1) + 2 * m * n) div 2 + (n*(n+1) div 2)"
      unfolding eq by (simp add: algebra_simps e1_def)
    also have "... = (m * (m + 1) + 2 * m * n + n * (n + 1)) div 2"
      by (rule div_add [symmetric]) auto
    also have "(m * (m + 1) + 2 * m * n + n * (n + 1)) = (n+m)*(n+m+1)"
      by Groebner_Basis.algebra

    also have "e2 + (n*(n-1) div 2) + m*(n*(n+1) div 2) - m*(n*(n-1) div
2) =
               (m*(m-1) + 2*m*n) div 2 + (n*(n-1) div 2)"
      unfolding eq by (simp add: algebra_simps e2_def)
```

```
    also have "... = (m*(m-1) + 2*m*n + n*(n-1)) div 2"
      by (rule div_add [symmetric]) auto
    also have "m*(m-1) + 2*m*n + n*(n-1) = (n+m)*(n+m-1)"
      by Groebner_Basis.algebra

    finally show "?lhs n = ?rhs n" .
  qed

  also have "((λn. a powi ((n+m)*(n+m+1) div 2) * b powi ((n+m)*(n+m-1)
div 2)) has_sum
              (a powi e1 * b powi e2 * ramanujan_theta a' b')) UNIV ⟷

             ((λn. a powi (n * (n + 1) div 2) * b powi (n * (n - 1) div
2)) has_sum
              (a powi e1 * b powi e2 * ramanujan_theta a' b')) UNIV"
    by (intro has_sum_reindex_bij_witness[of _ "λn. n - m" "λn. n + m"])
auto
  finally have ... .
  moreover have "((λn. a powi (n * (n + 1) div 2) * b powi (n * (n -
1) div 2)) has_sum
                  ramanujan_theta a b) UNIV"
    by (rule has_sum_ramanujan_theta) (use True in auto)
  ultimately have "a powi e1 * b powi e2 * ramanujan_theta a' b' = ramanujan_theta
a b"
    using has_sum_unique by blast
  thus ?thesis
    by (simp add: e1_def e2_def a'_def b'_def)
qed

lemma ramanujan_theta_mult:
  assumes [simp]: "a ≠ 0" "b ≠ 0"
  shows   "ramanujan_theta a b = a * ramanujan_theta (a^2 * b) (1 / a)"
  using ramanujan_theta_mult_power_int[of a b 1]
  by (simp add: eval_nat_numeral field_simps)

lemma ramanujan_theta_mult':
  assumes [simp]: "a ≠ 0" "b ≠ 0"
  shows   "ramanujan_theta a b = b * ramanujan_theta (1 / b) (a * b^2)"
  using ramanujan_theta_mult[of b a] by (simp add: ramanujan_theta_commute
mult.commute)
```

3.2 The Jacobi theta function in terms of the nome

Based on Ramanujan's ϑ function, we introduce a version of Jacobi's ϑ function:

$$\vartheta(w, q) = \sum_{n=-\infty}^{\infty} w^n q^{n^2} \quad (\text{for } |q| < 1, w \neq 0)$$

Both parameters are still in terms of the nome rather than the complex plane. This has some advantages, and we can easily derive the other versions from it later.

```
definition jacobi_theta_nome :: "'a :: {real_normed_field,banach} ⇒ 'a
⇒ 'a" where
  "jacobi_theta_nome w q = (if w = 0 then 0 else ramanujan_theta (q*w)
(q/w))"

lemma jacobi_theta_nome_0_left [simp]: "jacobi_theta_nome 0 q = 0"
  by (simp add: jacobi_theta_nome_def)

lemma jacobi_theta_nome_outside [simp]:
  assumes "norm q ≥ 1"
  shows   "jacobi_theta_nome w q = 0"
proof (cases "w = 0")
  case False
  thus ?thesis using assms
    by (simp add: jacobi_theta_nome_def norm_mult ramanujan_theta_def
power_less_one_iff norm_power
             flip: power2_eq_square)
qed auto

lemma has_sum_jacobi_theta_nome:
  assumes "norm q < 1" and [simp]: "w ≠ 0"
  shows   "((λn. w powi n * q powi (n ^ 2)) has_sum jacobi_theta_nome
w q) UNIV"
proof (cases "q = 0")
  case True
  have "((λ_::int. 1) has_sum jacobi_theta_nome w q) {0}"
    by (intro has_sum_finiteI) (use True in ‹auto simp: jacobi_theta_nome_def›)
  also have "?this ⟷ ?thesis"
    using True by (intro has_sum_cong_neutral) auto
  finally show ?thesis .
next
  case False
  hence [simp]: "q ≠ 0" "w ≠ 0"
    by auto
  have "((λn. (q*w) powi (n*(n+1) div 2) * (q/w) powi (n*(n-1) div 2))
has_sum ramanujan_theta (q*w) (q/w)) UNIV"
    by (rule has_sum_ramanujan_theta)
       (use assms in ‹auto simp: norm_power power_less_one_iff simp flip:
power2_eq_square›)
  also have "(λn. (q*w) powi (n*(n+1) div 2) * (q/w) powi (n*(n-1) div
2)) =
             (λn. w powi ((n*(n+1) div 2) - (n*(n-1) div 2)) * q powi
((n*(n+1) div 2) + (n*(n-1) div 2)))"
    by (simp add: power_int_mult_distrib power_int_divide_distrib power_int_add
power_int_diff field_simps)
  also have "(λn::int. (n*(n+1) div 2) - (n*(n-1) div 2)) = (λn. n)"
```

```
    by (auto simp: fun_eq_iff algebra_simps)
  also have "(λn::int. (n*(n+1) div 2) + (n*(n-1) div 2)) = (λn. n ^ 2)"
    by (auto simp: fun_eq_iff algebra_simps power2_eq_square simp flip:
div_add)
  finally show ?thesis
    by (simp add: jacobi_theta_nome_def)
qed

lemma jacobi_theta_nome_same:
  "q ≠ 0 ⟹ jacobi_theta_nome q q = 2 * jacobi_theta_nome (1 / q^2)
(q^4)"
  by (simp add: jacobi_theta_nome_def ramanujan_theta_1_right
          flip: power_diff power2_eq_square)

lemma jacobi_theta_nome_minus_same: "q ≠ 0 ⟹ jacobi_theta_nome (-q)
q = 0"
  by (simp add: jacobi_theta_nome_def)

lemma jacobi_theta_nome_minus_same': "q ≠ 0 ⟹ jacobi_theta_nome q
(-q) = 0"
  by (simp add: jacobi_theta_nome_def)

lemma jacobi_theta_nome_0_right [simp]: "w ≠ 0 ⟹ jacobi_theta_nome
w 0 = 1"
  by (simp add: jacobi_theta_nome_def)

lemma jacobi_theta_nome_of_real:
  "jacobi_theta_nome (of_real w) (of_real q) = of_real (jacobi_theta_nome
w q)"
  by (simp add: jacobi_theta_nome_def flip: ramanujan_theta_of_real)

lemma jacobi_theta_nome_cnj:
  "jacobi_theta_nome (cnj w) (cnj q) = cnj (jacobi_theta_nome w q)"
  by (simp add: jacobi_theta_nome_def flip: ramanujan_theta_cnj)

lemma jacobi_theta_nome_minus_left:
  "jacobi_theta_nome (-w) q = jacobi_theta_nome w (-q)"
  by (simp add: jacobi_theta_nome_def)

lemma jacobi_theta_nome_quasiperiod':
  assumes [simp]: "w ≠ 0" "q ≠ 0"
  shows   "w * q * jacobi_theta_nome (q^2 * w) q = jacobi_theta_nome w
q"
proof -
  have "jacobi_theta_nome w q = ramanujan_theta (q * w) (q / w)"
    by (simp add: jacobi_theta_nome_def)
  also have "... = w * q * ramanujan_theta (q ^ 3 * w) (1 / (q * w))"
    using ramanujan_theta_mult[of "q*w" "q/w"]
    by (simp add: field_simps eval_nat_numeral)
```

```
  also have "ramanujan_theta (q ^ 3 * w) (1 / (q * w)) = jacobi_theta_nome
(q² * w) q"
    by (simp add: jacobi_theta_nome_def eval_nat_numeral field_simps)
  finally show ?thesis ..
qed

lemma jacobi_theta_nome_ii_left: "jacobi_theta_nome i q = jacobi_theta_nome
(-1) (q^4)"
proof (cases "norm q < 1")
  case q: True
  define S where "S = jacobi_theta_nome i q"
  have sum1: "((λn. i powi n * q powi n²) has_sum S) UNIV"
    unfolding S_def by (rule has_sum_jacobi_theta_nome) (use q in auto)
  also have "?this ⟷ ((λn. i powi (-n) * q powi (-n)²) has_sum S) UNIV"
    by (rule has_sum_reindex_bij_witness[of _ uminus uminus]) auto
  finally have sum2: "((λn. (-i) powi n * q powi n²) has_sum S) UNIV"
    by (simp add: power_int_minus flip: power_int_inverse)
  have "((λn. (i powi n + (-i) powi n) * q powi n²) has_sum (S + S)) UNIV"

    unfolding ring_distribs by (intro has_sum_add sum1 sum2)
  also have "?this ⟷ ((λn. 2 * i powi n * q powi n²) has_sum (S + S))
{n. even n}"
    by (intro has_sum_cong_neutral) auto
  also have "... ⟷ ((λn. 2 * (i powi (2*n) * q powi (2*n)²)) has_sum
(S + S)) UNIV"
    by (intro has_sum_reindex_bij_witness[of _ "λn. 2*n" "λn. n div 2"])
auto
  finally have sum3: "((λn. 2 * (i powi (2*n) * (q^4) powi n ^ 2)) has_sum
(2 * S)) UNIV"
    by (simp flip: mult_2[of S] power_int_mult add: power_int_mult)
  have "((λn. i powi (2*n) * (q^4) powi n ^ 2) has_sum S) UNIV"
    using has_sum_cmult_right[OF sum3, of "1/2"] by simp
  also have "(λn. i powi (2*n)) = (λn. (-1) powi n)"
    by (simp add: power_int_mult)
  finally have "((λn. (-1) powi n * (q^4) powi n²) has_sum S) UNIV" .
  moreover have "((λn. (-1) powi n * (q^4) powi n²) has_sum jacobi_theta_nome
(-1) (q^4)) UNIV"
    by (rule has_sum_jacobi_theta_nome) (use q in <auto simp: norm_power
power_less_one_iff>)
  ultimately show ?thesis
    using has_sum_unique unfolding S_def by blast
qed (auto simp: norm_power power_less_one_iff)

lemma jacobi_theta_nome_quasiperiod:
  assumes [simp]: "w ≠ 0" "q ≠ 0"
  shows   "jacobi_theta_nome (q² * w) q = jacobi_theta_nome w q / (w
* q)"
  using jacobi_theta_nome_quasiperiod'[of w q] by (simp add: field_simps)
```

```
lemma jacobi_theta_nome_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z) ≠ 0" "⋀z. z ∈ A ⟹ norm (g z)
< 1" "open A"
  shows   "(λz. jacobi_theta_nome (f z) (g z)) holomorphic_on A"
proof -
  have "(λz. ramanujan_theta (g z * f z) (g z / f z)) holomorphic_on A"
    by (intro holomorphic_intros)
       (use assms in ‹auto simp: norm_power power_less_one_iff simp flip:
power2_eq_square›)
  also have "?this ⟷ ?thesis"
    by (intro holomorphic_cong) (use assms(3,4) in ‹auto simp: jacobi_theta_nome_def›)
  finally show ?thesis .
qed

lemma jacobi_theta_nome_analytic [analytic_intros]:
  assumes "f analytic_on A" "g analytic_on A"
  assumes "⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"
  shows   "(λz. jacobi_theta_nome (f z) (g z)) analytic_on A"
proof -
  from assms(1) obtain B1 where B1: "open B1" "A ⊆ B1" "f holomorphic_on
B1"
    using analytic_on_holomorphic by metis
  from assms(2) obtain B2 where B2: "open B2" "A ⊆ B2" "g holomorphic_on
B2"
    using analytic_on_holomorphic by metis
  note [holomorphic_intros] = holomorphic_on_subset[OF B1(3)] holomorphic_on_subset[OF
B2(3)]

  define B3 where "B3 = B1 ∩ B2 ∩ (λz. (f z, g z)) -‘ ((-{0}) × ball
0 1)"
  have "open B3" using B1 B2 unfolding B3_def
    by (intro continuous_open_preimage holomorphic_on_imp_continuous_on
              holomorphic_intros continuous_intros open_halfspace_Im_gt)
(auto intro!: open_Times)
  hence B3: "open B3" "B3 ⊆ B1" "B3 ⊆ B2" "⋀z. z ∈ B3 ⟹ f z ≠ 0
∧ g z ∈ ball 0 1"
    unfolding B3_def by auto

  have "(λz. jacobi_theta_nome (f z) (g z)) holomorphic_on B3"
    using B3 by (auto intro!: holomorphic_intros)
  moreover have "A ⊆ B3"
    using assms(3,4) B1 B2 by (auto simp: B3_def)
  ultimately show ?thesis
    using ‹open B3› analytic_on_holomorphic by metis
qed

lemma tendsto_jacobi_theta_nome [tendsto_intros]:
  fixes f g :: "'a ⇒ 'b :: {real_normed_field, banach, heine_borel}"
```

```
  assumes "(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1"
  shows   "((λz. jacobi_theta_nome (f z) (g z)) ⟶ jacobi_theta_nome
w q) F"
proof -
  have "((λz. jacobi_theta_nome (f z) (g z)) ⟶ ramanujan_theta (q
* w) (q / w)) F"
  proof (rule Lim_transform_eventually)
    show "((λz. ramanujan_theta (g z * f z) (g z / f z)) ⟶
            ramanujan_theta (q * w) (q / w)) F"
      by (intro tendsto_intros assms)
         (use assms(3,4) in <simp_all flip: power2_eq_square add: norm_power
power_less_one_iff>)
  next
    have "eventually (λx. f x ∈ -{0}) F"
      by (rule topological_tendstoD[OF assms(1)]) (use assms(3) in auto)
    thus "eventually (λz. ramanujan_theta (g z * f z) (g z / f z) = jacobi_theta_nome
(f z) (g z)) F"
      by eventually_elim (simp add: jacobi_theta_nome_def)
  qed
  thus ?thesis
    using assms(3) by (simp add: jacobi_theta_nome_def)
qed

lemma continuous_on_jacobi_theta_nome [continuous_intros]:
  fixes f g :: "'a :: topological_space ⇒ 'b :: {real_normed_field, banach,
heine_borel}"
  assumes "continuous_on A f" "continuous_on A g"
  assumes "⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"
  shows   "continuous_on A (λz. jacobi_theta_nome (f z) (g z))"
proof -
  have *: "continuous_on {z. fst z ≠ 0 ∧ norm (snd z) < 1} (λ(a,b). jacobi_theta_nome
a b :: 'b)"
    unfolding continuous_on by (auto intro!: tendsto_eq_intros simp: case_prod_unfold)
  have "continuous_on A ((λ(x,y). jacobi_theta_nome x y) ∘ (λx. (f x,
g x)))"
    by (intro continuous_on_compose continuous_on_subset[OF *] continuous_intros)
       (use assms in auto)
  thus ?thesis
    by (simp add: o_def)
qed

lemma continuous_jacobi_theta_nome [continuous_intros]:
  fixes f g :: "'a :: t2_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"
  assumes "continuous F f" "continuous F g" "f (netlimit F) ≠ 0" "norm
(g (netlimit F)) < 1"
  shows   "continuous F (λz. jacobi_theta_nome (f z) (g z))"
  unfolding continuous_def
  using assms by (auto intro!: tendsto_eq_intros simp: continuous_def)
```

3.3 The Jacobi theta function in the upper half of the complex plane

We now define the more usual version of the Jacobi ϑ function, which takes two complex parameters z and t where z is arbitrary and t must lie in the upper half of the complex plane.

definition *jacobi_theta_00 :: "complex ⇒ complex ⇒ complex"* **where**
"jacobi_theta_00 z t = jacobi_theta_nome (to_nome z ^ 2) (to_nome t)"

lemma *jacobi_theta_00_outside: "Im t ≤ 0 ⟹ jacobi_theta_00 z t = 0"*
by *(simp add: jacobi_theta_00_def mult_le_0_iff to_nome_def)*

lemma *has_sum_jacobi_theta_00:*
assumes *"Im t > 0"*
shows *"((λn. to_nome (of_int n ^ 2 * t + 2 * of_int n * z)) has_sum jacobi_theta_00 z t) UNIV"*
using *has_sum_jacobi_theta_nome[of "exp (i * of_real pi * t)" "exp (2 * i * of_real pi * z)"] assms*
by *(simp add: jacobi_theta_00_def algebra_simps exp_add exp_power_int to_nome_def*
flip: exp_of_nat_mult)

lemma *sums_jacobi_theta_00:*
assumes *"Im t > 0"*
shows *"((λn. if n = 0 then 1 else 2 * to_nome t ^ n^2 *
cos (2 * of_nat n * of_real pi * z)) sums jacobi_theta_00 z t)"*
proof -
define *f* **where** *"f = (λn::int. to_nome (of_int n ^ 2 * t + 2 * of_int n * z))"*
define *S1* **where** *"S1 = ($\sum_\infty$n∈{1..}. f n)"*
define *S2* **where** *"S2 = ($\sum_\infty$n∈{..-1}. f n)"*

have *sum: "(f has_sum jacobi_theta_00 z t) UNIV"*
unfolding *f_def* **by** *(rule has_sum_jacobi_theta_00) fact*
have *[simp]: "f summable_on A"* **for** *A*
by *(rule summable_on_subset_banach, rule has_sum_imp_summable[OF sum]) auto*

have *"(f has_sum S1) {1..}" "(f has_sum S2) {..-1}"*
unfolding *S1_def S2_def* **by** *(rule has_sum_infsum; simp)+*
moreover have *"(f has_sum 1) {0}"*
by *(rule has_sum_finiteI) (auto simp: f_def)*
ultimately have *"(f has_sum (1 + S1 + S2)) ({0} ∪ {1..} ∪ {..-1})"*
by *(intro has_sum_Un_disjoint) auto*
also have *"{0} ∪ {1..} ∪ {..-1::int} = UNIV"*
by *auto*
finally have *"(f has_sum 1 + S1 + S2) UNIV"* .
with *sum* **have** *eq: "jacobi_theta_00 z t = 1 + S1 + S2"*

```
    using has_sum_unique by blast

  note ‹(f has_sum S2) {..-1}›
  also have "(f has_sum S2) {..-1} ⟷ ((λn. f (-n)) has_sum S2) {1..}"
    by (intro has_sum_reindex_bij_witness[of _ uminus uminus]) auto
  finally have "((λn. f n + f (-n)) has_sum (S1 + S2)) {1..}"
    using ‹(f has_sum S1) {1..}› by (intro has_sum_add)
  also have "?this ⟷ ((λn. f (int n) + f (-int n)) has_sum (S1 + S2))
{1..}"
    by (rule has_sum_reindex_bij_witness[of _ int nat]) auto
  also have "(λn::nat. f (int n) + f (-int n)) =
             (λn. 2 * to_nome t ^ (n ^ 2) * cos (2 * of_nat n * of_real
pi * z))"
    by (auto simp: f_def exp_add exp_diff ring_distribs to_nome_def mult_ac
cos_exp_eq
             simp flip: exp_of_nat_mult)
  also have "(... has_sum (S1 + S2)) {1..} ⟷
             ((λn. if n = 0 then 1 else 2 * to_nome t ^ (n ^ 2) *
                    cos (2 * of_nat n * of_real pi * z)) has_sum (S1
+ S2)) {1..}"
    by (intro has_sum_cong) auto
  finally have "((λn. if n = 0 then 1 else 2 * to_nome t ^ (n ^ 2) * cos
(2 * of_nat n * of_real pi * z))
                   has_sum (1 + (S1 + S2))) ({0} ∪ {1..})"
    by (intro has_sum_Un_disjoint) (auto intro: has_sum_finiteI)
  also have "1 + (S1 + S2) = jacobi_theta_00 z t"
    using eq by (simp add: add_ac)
  also have "{0} ∪ {1::nat..} = UNIV"
    by auto
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

lemma jacobi_theta_00_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A" "⋀z. z ∈ A ⟹ Im
(g z) > 0" "open A"
  shows   "(λz. jacobi_theta_00 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_00_def using assms(3,4)
  by (auto intro!: holomorphic_intros assms(1,2) simp: to_nome_def)

lemma jacobi_theta_00_analytic [analytic_intros]:
  assumes "f analytic_on A" "g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z)
> 0"
  shows   "(λz. jacobi_theta_00 (f z) (g z)) analytic_on A"
  unfolding jacobi_theta_00_def using assms(3)
  by (auto intro!: analytic_intros assms(1,2) simp: to_nome_def)

lemma jacobi_theta_00_plus_half_left:
  "jacobi_theta_00 (z + 1 / 2) t = jacobi_theta_00 z (t + 1)"
```

```
proof -
  define q where "q = exp (i * of_real pi * t)"
  define w where "w = exp (2 * i * of_real pi * z)"
  have "jacobi_theta_00 (z + 1 / 2) t = jacobi_theta_nome (-w) q"
    by (simp add: jacobi_theta_00_def w_def q_def algebra_simps exp_add
to_nome_def flip: exp_of_nat_mult)
  also have "... = jacobi_theta_nome w (-q)"
    by (simp add: jacobi_theta_nome_minus_left)
  also have "... = jacobi_theta_00 z (t + 1)"
    by (simp add: jacobi_theta_00_def algebra_simps exp_add q_def w_def
to_nome_def flip: exp_of_nat_mult)
  finally show ?thesis .
qed

lemma jacobi_theta_00_plus_2_right: "jacobi_theta_00 z (t + 2) = jacobi_theta_00
z t"
  by (simp add: jacobi_theta_00_def algebra_simps exp_add to_nome_def)

interpretation jacobi_theta_00_left: periodic_fun_simple' "λz. jacobi_theta_00
z t"
proof
  fix z :: complex
  have "jacobi_theta_00 (z + 1) t = jacobi_theta_00 (z + 1/2 + 1/2) t"
    by (simp add: add.commute)
  also have "... = jacobi_theta_00 z (t + 2)"
    unfolding jacobi_theta_00_plus_half_left by (simp add: add.commute)
  also have "jacobi_theta_00 z (t + 2) = jacobi_theta_00 z t"
    by (rule jacobi_theta_00_plus_2_right)
  finally show "jacobi_theta_00 (z + 1) t = jacobi_theta_00 z t" .
qed

interpretation jacobi_theta_00_right: periodic_fun_simple "λt. jacobi_theta_00
z t" 2
proof
  fix t :: complex
  show "jacobi_theta_00 z (t + 2) = jacobi_theta_00 z t"
    by (rule jacobi_theta_00_plus_2_right)
qed

lemma jacobi_theta_00_plus_quasiperiod:
  "jacobi_theta_00 (z + t) t = jacobi_theta_00 z t / to_nome (t + 2 *
z)"
proof -
  define q where "q = exp (i * of_real pi * t)"
  define w where "w = exp (2 * i * of_real pi * z)"
  have "jacobi_theta_00 (z + t) t = jacobi_theta_nome (q^2 * w) q"
    by (simp add: w_def q_def jacobi_theta_00_def algebra_simps exp_add
to_nome_def
            flip: exp_of_nat_mult)
```

```
  also have "... = jacobi_theta_nome w q / (w * q)"
    by (subst jacobi_theta_nome_quasiperiod) (auto simp: w_def q_def)
  also have "... = exp (-pi * i * (t + 2 * z)) * jacobi_theta_00 z t"
    by (simp add: w_def q_def jacobi_theta_00_def field_simps exp_add
exp_minus exp_diff to_nome_def
            flip: exp_of_nat_mult)
  finally show ?thesis
    by (simp add: exp_minus exp_diff exp_add to_nome_def field_simps)
qed

lemma jacobi_theta_00_quasiperiodic:
  "jacobi_theta_00 (z + of_int m + of_int n * t) t =
     jacobi_theta_00 z t / to_nome (of_int (n^2) * t + 2 * of_int n *
z)"
proof -
  write jacobi_theta_00 ("ϑ")
  have "ϑ (z + of_int m + of_int n * t) t =
        ϑ (z  + of_int n * t + of_int m) t"
    by (simp add: add_ac)
  also have "... = ϑ (z + of_int n * t) t"
    by (rule jacobi_theta_00_left.plus_of_int)
  also have "... = ϑ z t / to_nome (of_int (n^2) * t + 2 * of_int n *
z)"
  proof -
    have *: "ϑ (z + of_nat n * t) t = ϑ z t / to_nome (of_nat (n^2) *
t + 2 * of_nat n * z)"
      for z :: complex and n :: nat
    proof (induction n)
      case (Suc n)
      have "ϑ (z + of_nat (Suc n) * t) t = ϑ (z + of_nat n * t + t) t"
        by (simp add: algebra_simps)
      also have "... = ϑ z t / (to_nome (of_nat (n^2) * t + 2 * of_nat
n * z) *
                                to_nome (t + 2 * (z + of_nat n * t)))"
        by (subst jacobi_theta_00_plus_quasiperiod, subst Suc.IH) auto
      also have "to_nome (of_nat (n^2) * t + 2 * of_nat n * z) * to_nome
(t + 2 * (z + of_nat n * t)) =
                 to_nome ((of_nat (n^2) * t + 2 * of_nat n * z) + (t +
2 * (z + of_nat n * t)))"
        by (rule to_nome_add [symmetric])
      also have "(of_nat (n^2) * t + 2 * of_nat n * z) + (t + 2 * (z +
of_nat n * t)) =
                 of_nat ((Suc n)^2) * t + 2 * of_nat (Suc n) * z"
        by (simp add: algebra_simps power2_eq_square)
      finally show ?case .
    qed auto
    show ?thesis
    proof (cases "n ≥ 0")
      case True
```

```
      thus ?thesis
        using *[of z "nat n"] by simp
    next
      case False
      thus ?thesis
        using *[of "z + of_int n * t" "nat (-n)"] False
        by (simp add: field_simps power2_eq_square to_nome_add to_nome_diff
to_nome_minus)
    qed
  qed
  finally show ?thesis .
qed

lemma jacobi_theta_00_onequarter_left:
  "jacobi_theta_00 (1/4) t = jacobi_theta_00 (1/2) (4 * t)"
  by (simp add: jacobi_theta_00_def to_nome_power jacobi_theta_nome_ii_left)

lemma jacobi_theta_00_eq_0: "jacobi_theta_00 ((t + 1) / 2) t = 0"
proof -
  have "jacobi_theta_00 ((t + 1) / 2) t = jacobi_theta_nome (to_nome (t
+ 1)) (to_nome t)"
    by (simp add: jacobi_theta_00_def to_nome_power add_divide_distrib)
  also have "... = 0"
    by (simp add: to_nome_add jacobi_theta_nome_minus_same)
  finally show ?thesis .
qed

lemma jacobi_theta_00_eq_0': "jacobi_theta_00 ((of_int m + 1/2) + (of_int
n + 1/2) * t) t = 0"
proof -
  have "jacobi_theta_00 ((of_int m + 1/2) + (of_int n + 1/2) * t) t =
        jacobi_theta_00 ((t + 1) / 2 + of_int m + of_int n * t) t"
    by (simp add: algebra_simps add_divide_distrib)
  also have "... = 0"
    by (simp only: jacobi_theta_00_quasiperiodic jacobi_theta_00_eq_0)
auto
  finally show ?thesis .
qed

lemma tendsto_jacobi_theta_00 [tendsto_intros]:
  assumes "(f ⟶ w) F" "(g ⟶ q) F" "Im q > 0"
  shows   "((λz. jacobi_theta_00 (f z) (g z)) ⟶ jacobi_theta_00 w
q) F"
  unfolding jacobi_theta_00_def
  by (intro tendsto_intros assms(1,2)) (use assms(3) in ‹auto simp: norm_to_nome›)

lemma continuous_on_jacobi_theta_00 [continuous_intros]:
  assumes "continuous_on A f" "continuous_on A g"
  assumes "⋀z. z ∈ A ⟹ Im (g z) > 0"
```

shows *"continuous_on A (λz. jacobi_theta_00 (f z) (g z))"*
unfolding *jacobi_theta_00_def*
by *(intro continuous_intros assms(1,2)) (use assms(3)* **in** *<auto simp: norm_to_nome>)*

lemma *continuous_jacobi_theta_00 [continuous_intros]:*
assumes *"continuous F f" "continuous F g" "Im (g (netlimit F)) > 0"*
shows *"continuous F (λz. jacobi_theta_00 (f z) (g z))"*
unfolding *jacobi_theta_00_def*
by *(intro continuous_intros assms(1,2)) (use assms(3)* **in** *<auto simp: norm_to_nome>)*

3.4 The auxiliary theta functions in terms of the nome

definition *jacobi_theta_nome_00 :: "'a :: {real_normed_field, banach} ⇒ 'a ⇒ 'a"* **where**
"jacobi_theta_nome_00 w q = jacobi_theta_nome (w^2) q"

definition *jacobi_theta_nome_01 :: "'a :: {real_normed_field, banach} ⇒ 'a ⇒ 'a"* **where**
"jacobi_theta_nome_01 w q = jacobi_theta_nome (-(w^2)) q"

definition *jacobi_theta_nome_10 :: "'a :: {real_normed_field, banach, ln} ⇒ 'a ⇒ 'a"* **where**
*"jacobi_theta_nome_10 w q = w * q powr (1/4) * jacobi_theta_nome (w ^ 2 * q) q"*

definition *jacobi_theta_nome_11 :: "complex ⇒ complex ⇒ complex"* **where**
"jacobi_theta_nome_11 w q = i ** w * q powr (1/4) * jacobi_theta_nome (-(w ^ 2) * q) q"*

lemmas *jacobi_theta_nome_xx_defs =*
jacobi_theta_nome_00_def jacobi_theta_nome_01_def
jacobi_theta_nome_10_def jacobi_theta_nome_11_def

lemma *jacobi_theta_nome_00_outside [simp]: "norm q ≥ 1 ⟹ jacobi_theta_nome_00 w q = 0"*
and *jacobi_theta_nome_01_outside [simp]: "norm q ≥ 1 ⟹ jacobi_theta_nome_01 w q = 0"*
and *jacobi_theta_nome_10_outside [simp]: "norm q' ≥ 1 ⟹ jacobi_theta_nome_10 w' q' = 0"*
and *jacobi_theta_nome_11_outside [simp]: "norm q'' ≥ 1 ⟹ jacobi_theta_nome_11 w'' q'' = 0"*
by *(simp_all add: jacobi_theta_nome_xx_defs)*

lemma *jacobi_theta_nome_01_conv_00: "jacobi_theta_nome_01 w' q' = jacobi_theta_nome_00 w' (-q')"*
and *jacobi_theta_nome_11_conv_10: "jacobi_theta_nome_11 w q = jacobi_theta_nome_10 (*i ** w) q"*

```
  by (simp_all add: jacobi_theta_nome_xx_defs power_mult_distrib jacobi_theta_nome_minus_le

lemma jacobi_theta_nome_00_0_right [simp]: "w ≠ 0 ⟹ jacobi_theta_nome_00
w 0 = 1"
  and jacobi_theta_nome_01_0_right [simp]: "w ≠ 0 ⟹ jacobi_theta_nome_01
w 0 = 1"
  and jacobi_theta_nome_10_0_right [simp]: "jacobi_theta_nome_10 w' 0
= 0"
  and jacobi_theta_nome_11_0_right [simp]: "jacobi_theta_nome_11 w'' 0
= 0"
  by (simp_all add: jacobi_theta_nome_xx_defs)

lemma jacobi_theta_nome_00_of_real:
        "jacobi_theta_nome_00 (of_real w :: 'a :: {banach, real_normed_field})
(of_real q) =
         of_real (jacobi_theta_nome_00 w q)"
  and jacobi_theta_nome_01_of_real:
        "jacobi_theta_nome_01 (of_real w :: 'a) (of_real q) = of_real
(jacobi_theta_nome_01 w q)"
  and jacobi_theta_nome_10_complex_of_real:
        "q ≥ 0 ⟹ jacobi_theta_nome_10 (complex_of_real w) (of_real
q) =
                    of_real (jacobi_theta_nome_10 w q)"
  by (simp_all add: jacobi_theta_nome_xx_defs flip: jacobi_theta_nome_of_real
powr_of_real)

lemma jacobi_theta_nome_00_cnj:
        "jacobi_theta_nome_00 (cnj w) (cnj q) = cnj (jacobi_theta_nome_00
w q)"
  and jacobi_theta_nome_01_cnj:
        "jacobi_theta_nome_01 (cnj w) (cnj q) = cnj (jacobi_theta_nome_01
w q)"
  and jacobi_theta_nome_10_cnj:
        "(Im q = 0 ⟹ Re q ≥ 0) ⟹
           jacobi_theta_nome_10 (cnj w) (cnj q) = cnj (jacobi_theta_nome_10
w q)"
  and jacobi_theta_nome_11_cnj:
        "(Im q = 0 ⟹ Re q ≥ 0) ⟹
           jacobi_theta_nome_11 (cnj w) (cnj q) = -cnj (jacobi_theta_nome_11
w q)"
  by (simp_all add: jacobi_theta_nome_xx_defs cnj_powr flip: jacobi_theta_nome_cnj)


lemma has_sum_jacobi_theta_nome_00:
  assumes "norm q < 1" "w ≠ 0"
  shows   "((λn. w powi (2*n) * q powi n^2) has_sum jacobi_theta_nome_00
w q) UNIV"
  using has_sum_jacobi_theta_nome[of q "w^2"] assms
  by (simp add: jacobi_theta_nome_00_def power_int_mult_distrib power_int_mult
```

```
power_mult_distrib)

lemma has_sum_jacobi_theta_nome_01:
  assumes "norm q < 1" "w ≠ 0"
  shows   "((λn. (-1) powi n * w powi (2*n) * q powi n^2) has_sum jacobi_theta_nome_01
w q) UNIV"
  using has_sum_jacobi_theta_nome[of q "-(w^2)"] assms
  by (simp add: jacobi_theta_nome_01_def power_int_mult power_mult_distrib

          flip: power_int_mult_distrib)

lemma has_sum_jacobi_theta_nome_10':
  assumes q: "norm q < 1" and [simp]: "w ≠ 0" "q ≠ 0"
  shows   "((λn. w powi (2*n+1) * q powi (n*(n+1))) has_sum
             (jacobi_theta_nome_10 w q / q powr (1/4))) UNIV"
proof -
  have "((λn. w * ((w^2 * q) powi n * q powi (n ^ 2))) has_sum
         (w * jacobi_theta_nome (w ^ 2 * q) q)) UNIV"
    by (intro has_sum_cmult_right has_sum_jacobi_theta_nome) (use q in
auto)
  also have "(λn. w * ((w^2 * q) powi n * q powi (n ^ 2))) = (λn. w powi
(2*n+1) * q powi (n*(n+1)))"
    by (simp add: power_int_mult_distrib power_int_power power_int_add
ring_distribs)
       (simp_all add: algebra_simps power2_eq_square)?
  finally show ?thesis
    by (simp add: jacobi_theta_nome_10_def)
qed

lemma has_sum_jacobi_theta_nome_10:
  fixes q :: "'a :: {real_normed_field, banach, ln}"
  assumes q: "norm q < 1" and [simp]: "w ≠ 0" "exp (ln q) = q"
  shows   "((λn. w powi (2*n+1) * q powr (of_int n + 1 / 2) ^ 2) has_sum
             (jacobi_theta_nome_10 w q)) UNIV"
proof -
  have "exp (ln q) ≠ 0"
    by (rule exp_not_eq_zero)
  hence [simp]: "q ≠ 0"
    by auto
  have "((λn. q powr (1/4) * (w powi (2*n+1) * q powi (n*(n+1)))) has_sum
            (q powr (1/4) * (jacobi_theta_nome_10 w q / q powr (1/4))))
UNIV"
    by (intro has_sum_cmult_right has_sum_jacobi_theta_nome_10') fact+
  also have "(q powr (1/4) * (jacobi_theta_nome_10 w q / q powr (1/4)))
= jacobi_theta_nome_10 w q"
    by simp
  also have "(λn::int. q powr (1/4) * (w powi (2*n+1) * q powi (n*(n+1))))
=
            (λn::int. w powi (2*n+1) * q powr ((of_int n + 1/2) ^ 2))"
```

```
  proof
    fix n :: int
    have "q powr (1/4) * (w powi (2*n+1) * q powi (n*(n+1))) =
          w powi (2*n+1) * (q powr (1/4) * q powi (n*(n+1)))"
      by (simp add: mult_ac)
    also have "... = w powi (2*n+1) * (q powr (1/4) * q powr (of_int (n*(n+1))))"
    proof -
      have "q powr (of_int (n*(n+1))) = exp (of_int (n*(n+1)) * ln q)"
        by (simp add: powr_def)
      also have "... = q powi (n * (n + 1))"
        by (subst exp_power_int [symmetric]) auto
      finally show ?thesis
        by simp
    qed
    also have "q powr (1/4) * q powr of_int (n*(n+1)) =
               q powr (1/4 + of_int (n*(n+1)))"
      by (simp add: powr_def field_simps flip: exp_add)
    also have "1/4 + of_int (n*(n+1)) = (of_int n + 1/2 :: 'a) ^ 2"
      by (simp add: field_simps power2_eq_square)
    finally show "q powr (1/4) * (w powi (2*n+1) * q powi (n*(n+1))) =
                  w powi (2*n+1) * q powr ((of_int n + 1/2) ^ 2)" .
  qed
  finally show ?thesis .
qed

lemma has_sum_jacobi_theta_nome_11':
  assumes q: "norm q < 1" and [simp]: "w ≠ 0" "q ≠ 0"
  shows   "((λn. (-1) powi n * w powi (2*n+1) * q powi (n*(n+1))) has_sum
             (jacobi_theta_nome_11 w q / (i * q powr (1/4)))) UNIV"
proof -
  have "((λn. w * ((-(w^2)) * q) powi n * q powi (n ^ 2))) has_sum
          (w * jacobi_theta_nome (-(w ^ 2) * q) q)) UNIV"
    by (intro has_sum_cmult_right has_sum_jacobi_theta_nome) (use q in
auto)
  also have "(λn. (-(w^2)) * q) powi n) = (λn. (-1) powi n * (w ^ 2 * q)
powi n)"
    by (subst power_int_mult_distrib [symmetric]) auto
  also have "(λn. w * ((-1) powi n * (w^2 * q) powi n * q powi (n ^ 2)))
=
             (λn. (-1) powi n * w powi (2*n+1) * q powi (n*(n+1)))"
    by (simp add: power_int_mult_distrib power_int_power power_int_add
ring_distribs)
       (simp_all add: algebra_simps power2_eq_square)?
  finally show ?thesis
    by (simp add: jacobi_theta_nome_11_def mult_ac)
qed

lemma has_sum_jacobi_theta_nome_11:
  assumes q: "norm q < 1" and [simp]: "w ≠ 0" "q ≠ 0"
```

```
  shows   "((λn. i * (-1) powi n * w powi (2*n+1) * q powr (of_int n +
1/2) ^ 2) has_sum
             (jacobi_theta_nome_11 w q)) UNIV"
proof -
  have "((λn. (i*w) powi (2*n+1) * q powr (of_int n + 1 / 2) ^ 2) has_sum
             (jacobi_theta_nome_10 (i*w) q)) UNIV"
    by (intro has_sum_jacobi_theta_nome_10) (use q in auto)
  also have "(λn. (i*w) powi (2*n+1)) = (λn. i * i powi (2*n) * w powi
(2*n+1))"
    by (simp add: power_int_mult_distrib power_int_add mult_ac)
  also have "(λn. i powi (2*n)) = (λn. (-1) powi n)"
    by (subst power_int_mult) auto
  finally show ?thesis
    by (simp add: jacobi_theta_nome_11_conv_10)
qed


lemma jacobi_theta_nome_00_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z) ≠ 0" "⋀z. z ∈ A ⟹ norm (g z)
< 1" "open A"
  shows   "(λz. jacobi_theta_nome_00 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_nome_00_def
  by (intro holomorphic_intros assms(1,2)) (use assms(3-) in auto)

lemma jacobi_theta_nome_01_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z) ≠ 0" "⋀z. z ∈ A ⟹ norm (g z)
< 1" "open A"
  shows   "(λz. jacobi_theta_nome_01 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_nome_01_def
  by (intro holomorphic_intros assms(1,2)) (use assms(3-) in auto)

lemma jacobi_theta_nome_10_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z) ≠ 0"
  assumes "⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ g z ∉ ℝ≤0" "open A"
  shows   "(λz. jacobi_theta_nome_10 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_nome_10_def
  by (intro holomorphic_intros assms(1,2)) (use assms(3-) in force)+

lemma jacobi_theta_nome_11_holomorphic [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A"
  assumes "⋀z. z ∈ A ⟹ norm (f z) ≠ 0"
  assumes "⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ g z ∉ ℝ≤0" "open A"
  shows   "(λz. jacobi_theta_nome_11 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_nome_11_def
  by (intro holomorphic_intros assms(1,2)) (use assms(3-) in force)+
```

lemma *jacobi_theta_nome_00_analytic [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (f z) ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"(λz. jacobi_theta_nome_00 (f z) (g z)) analytic_on A"*
unfolding *jacobi_theta_nome_00_def*
by *(intro analytic_intros assms(1,2)) (use assms(3-)* **in** *auto)*

lemma *jacobi_theta_nome_01_analytic [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (f z) ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"(λz. jacobi_theta_nome_01 (f z) (g z)) analytic_on A"*
unfolding *jacobi_theta_nome_01_def*
by *(intro analytic_intros assms(1,2)) (use assms(3-)* **in** *auto)*

lemma *jacobi_theta_nome_10_analytic [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (f z) ≠ 0"*
assumes *"⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ g z ∉ ℝ≤0"*
shows *"(λz. jacobi_theta_nome_10 (f z) (g z)) analytic_on A"*
unfolding *jacobi_theta_nome_10_def*
by *(intro analytic_intros assms(1,2)) (use assms(3-)* **in** *force)+*

lemma *jacobi_theta_nome_11_analytic [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A"*
assumes *"⋀z. z ∈ A ⟹ norm (f z) ≠ 0"*
assumes *"⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ g z ∉ ℝ≤0"*
shows *"(λz. jacobi_theta_nome_11 (f z) (g z)) analytic_on A"*
unfolding *jacobi_theta_nome_11_def*
by *(intro analytic_intros assms(1,2)) (use assms(3-)* **in** *force)+*

lemma *tendsto_jacobi_theta_nome_00 [tendsto_intros]:*
fixes *f g :: "'a ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1"*
shows *"((λz. jacobi_theta_nome_00 (f z) (g z)) ⟶ jacobi_theta_nome_00 w q) F"*
unfolding *jacobi_theta_nome_00_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_00 [continuous_intros]:*
fixes *f g :: "'a :: topological_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"continuous_on A (λz. jacobi_theta_nome_00 (f z) (g z))"*
unfolding *jacobi_theta_nome_00_def*

by *(intro continuous_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *continuous_jacobi_theta_nome_00 [continuous_intros]:*
fixes *f g :: "'a :: t2_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0" "norm (g (netlimit F)) < 1"*
shows *"continuous F (λz. jacobi_theta_nome_00 (f z) (g z))"*
unfolding *jacobi_theta_nome_00_def*
by *(intro continuous_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *tendsto_jacobi_theta_nome_01 [tendsto_intros]:*
fixes *f g :: "'a ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1"*
shows *"((λz. jacobi_theta_nome_01 (f z) (g z)) ⟶ jacobi_theta_nome_01 w q) F"*
unfolding *jacobi_theta_nome_01_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_01 [continuous_intros]:*
fixes *f g :: "'a :: topological_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1"*
shows *"continuous_on A (λz. jacobi_theta_nome_01 (f z) (g z))"*
unfolding *jacobi_theta_nome_01_def*
by *(intro continuous_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *continuous_jacobi_theta_nome_01 [continuous_intros]:*
fixes *f g :: "'a :: t2_space ⇒ 'b :: {real_normed_field, banach, heine_borel}"*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0" "norm (g (netlimit F)) < 1"*
shows *"continuous F (λz. jacobi_theta_nome_01 (f z) (g z))"*
unfolding *jacobi_theta_nome_01_def*
by *(intro continuous_intros assms(1,2)) (use assms(3,4)* **in** *auto)*

lemma *tendsto_jacobi_theta_nome_10_complex [tendsto_intros]:*
fixes *f g :: "complex ⇒ complex"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1" "q ∉ $\mathbb{R}_{\leq 0}$"*
shows *"((λz. jacobi_theta_nome_10 (f z) (g z)) ⟶ jacobi_theta_nome_10 w q) F"*
unfolding *jacobi_theta_nome_10_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3-5)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_10_complex [continuous_intros]:*
fixes *f g :: "complex ⇒ complex"*
assumes *"continuous_on A f" "continuous_on A g"*

assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ (Re (g z) > 0 ∨ Im (g z) ≠ 0)"*
shows *"continuous_on A (λz. jacobi_theta_nome_10 (f z) (g z))"*
unfolding *jacobi_theta_nome_10_def*
by *(intro continuous_intros assms(1,2); use assms(3,4)* **in** *force)*

lemma *continuous_jacobi_theta_nome_10_complex [continuous_intros]:*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0"*
assumes *"norm (g (netlimit F)) < 1" "Re (g (netlimit F)) > 0 ∨ Im (g (netlimit F)) ≠ 0"*
shows *"continuous F (λz. jacobi_theta_nome_10 (f z) (g z))"*
unfolding *jacobi_theta_nome_10_def*
by *(intro continuous_intros assms(1,2); use assms(3-)* **in** *force)*

lemma *tendsto_jacobi_theta_nome_10_real [tendsto_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1" "q > 0"*
shows *"((λz. jacobi_theta_nome_10 (f z) (g z)) ⟶ jacobi_theta_nome_10 w q) F"*
unfolding *jacobi_theta_nome_10_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3-5)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_10_real [continuous_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ g z ∈ {0<..<1}"*
shows *"continuous_on A (λz. jacobi_theta_nome_10 (f z) (g z))"*
unfolding *jacobi_theta_nome_10_def*
by *(intro continuous_intros assms(1,2); use assms(3,4)* **in** *force)*

lemma *continuous_jacobi_theta_nome_10_real [continuous_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0" "g (netlimit F) ∈ {0<..<1}"*
shows *"continuous F (λz. jacobi_theta_nome_10 (f z) (g z))"*
unfolding *jacobi_theta_nome_10_def*
by *(intro continuous_intros assms(1,2); use assms(3-)* **in** *auto)*

lemma *tendsto_jacobi_theta_nome_11_complex [tendsto_intros]:*
fixes *f g :: "complex ⇒ complex"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1" "q ∉ $\mathbb{R}_{\leq 0}$"*
shows *"((λz. jacobi_theta_nome_11 (f z) (g z)) ⟶ jacobi_theta_nome_11 w q) F"*
unfolding *jacobi_theta_nome_11_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3-5)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_11_complex [continuous_intros]:*
fixes *f g :: "complex ⇒ complex"*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ norm (g z) < 1 ∧ (Re (g z) > 0 ∨ Im (g z) ≠ 0)"*
shows *"continuous_on A (λz. jacobi_theta_nome_11 (f z) (g z))"*
unfolding *jacobi_theta_nome_11_def*
by *(intro continuous_intros assms(1,2); use assms(3,4)* **in** *force)*

lemma *continuous_jacobi_theta_nome_11_complex [continuous_intros]:*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0"*
assumes *"norm (g (netlimit F)) < 1" "Re (g (netlimit F)) > 0 ∨ Im (g (netlimit F)) ≠ 0"*
shows *"continuous F (λz. jacobi_theta_nome_11 (f z) (g z))"*
unfolding *jacobi_theta_nome_11_def*
by *(intro continuous_intros assms(1,2); use assms(3-)* **in** *force)*

lemma *tendsto_jacobi_theta_nome_11_real [tendsto_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "w ≠ 0" "norm q < 1" "q > 0"*
shows *"((λz. jacobi_theta_nome_11 (f z) (g z)) ⟶ jacobi_theta_nome_11 w q) F"*
unfolding *jacobi_theta_nome_11_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3-5)* **in** *auto)*

lemma *continuous_on_jacobi_theta_nome_11_real [continuous_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ f z ≠ 0" "⋀z. z ∈ A ⟹ g z ∈ {0<..<1}"*
shows *"continuous_on A (λz. jacobi_theta_nome_11 (f z) (g z))"*
unfolding *jacobi_theta_nome_11_def*
by *(intro continuous_intros assms(1,2); use assms(3,4)* **in** *force)*

lemma *continuous_jacobi_theta_nome_11_real [continuous_intros]:*
fixes *f g :: "real ⇒ real"*
assumes *"continuous F f" "continuous F g" "f (netlimit F) ≠ 0" "g (netlimit F) ∈ {0<..<1}"*
shows *"continuous F (λz. jacobi_theta_nome_11 (f z) (g z))"*
unfolding *jacobi_theta_nome_11_def*
by *(intro continuous_intros assms(1,2); use assms(3-)* **in** *auto)*

3.5 The auxiliary theta functions in the complex plane

definition *jacobi_theta_01 :: "complex ⇒ complex ⇒ complex"* **where**
"jacobi_theta_01 z t = jacobi_theta_00 (z + 1/2) t"

definition *jacobi_theta_10 :: "complex ⇒ complex ⇒ complex"* **where**
*"jacobi_theta_10 z t = to_nome (z + t/4) * jacobi_theta_00 (z + t/2) t"*

```
definition jacobi_theta_11 :: "complex ⇒ complex ⇒ complex" where
  "jacobi_theta_11 z t = to_nome (z + t/4 + 1/2) * jacobi_theta_00 (z
+ t/2 + 1/2) t"

lemma jacobi_theta_00_conv_nome:
  "jacobi_theta_00 z t = jacobi_theta_nome_00 (to_nome z) (to_nome t)"
  by (simp add: jacobi_theta_00_def jacobi_theta_nome_00_def)

lemma jacobi_theta_01_conv_nome:
  "jacobi_theta_01 z t = jacobi_theta_nome_01 (to_nome z) (to_nome t)"
  by (simp add: jacobi_theta_01_def jacobi_theta_nome_01_def jacobi_theta_00_conv_nome

                jacobi_theta_nome_00_def to_nome_add power_mult_distrib)

lemma jacobi_theta_10_conv_nome:
  assumes "Re t ∈ {-1<..1}"
  shows   "jacobi_theta_10 z t = jacobi_theta_nome_10 (to_nome z) (to_nome
t)"
  using assms
  by (simp add: jacobi_theta_10_def jacobi_theta_nome_10_def jacobi_theta_00_conv_nome

                jacobi_theta_nome_00_def to_nome_add power_mult_distrib
to_nome_power to_nome_powr)

lemma jacobi_theta_11_conv_nome:
  assumes "Re t ∈ {-1<..1}"
  shows   "jacobi_theta_11 z t = jacobi_theta_nome_11 (to_nome z) (to_nome
t)"
  using assms
  by (simp add: jacobi_theta_11_def jacobi_theta_nome_11_def jacobi_theta_00_conv_nome

                jacobi_theta_nome_00_def to_nome_add power_mult_distrib
to_nome_power to_nome_powr)


lemma has_sum_jacobi_theta_01:
  assumes "Im t > 0"
  shows   "((λn. (-1) powi n * to_nome (of_int n ^ 2 * t + 2 * of_int
n * z))
             has_sum jacobi_theta_01 z t) UNIV"
proof -
  have "((λn. to_nome (of_int n ^ 2 * t + 2 * of_int n * (z + 1 / 2)))
has_sum jacobi_theta_01 z t) UNIV"
    unfolding jacobi_theta_01_def by (intro has_sum_jacobi_theta_00 assms)
  also have "(λn. to_nome (of_int n ^ 2 * t + 2 * of_int n * (z + 1 /
2))) =
             (λn. (-1) powi n * to_nome (of_int n ^ 2 * t + 2 * of_int
n * z))"
```

```
    by (simp add: ring_distribs exp_add mult_ac to_nome_add)
  finally show ?thesis .
qed

lemma sums_jacobi_theta_01:
  assumes "Im t > 0"
  shows   "((λn. if n = 0 then 1 else 2 * (-1) ^ n *  to_nome t ^ n²
*
             cos (2 * of_nat n * of_real pi * z)) sums jacobi_theta_01
z t)"
proof -
  have [simp]: "sin (of_nat n * of_real pi :: complex) = 0" for n
    by (metis of_real_0 of_real_mult of_real_of_nat_eq sin_npi sin_of_real)
  have [simp]: "cos (of_nat n * of_real pi :: complex) = (-1) ^ n" for
n
  proof -
    have "cos (of_nat n * of_real pi) = complex_of_real (cos (real n *
pi))"
      by (subst cos_of_real [symmetric]) simp
    also have "cos (real n * pi) = (-1) ^ n"
      by simp
    finally show ?thesis by simp
  qed
  show ?thesis
    using sums_jacobi_theta_00[of t "z + 1/2"] assms
    by (simp add: jacobi_theta_01_def  ring_distribs cos_add mult_ac cong:
if_cong)
qed


interpretation jacobi_theta_01_left: periodic_fun_simple' "λz. jacobi_theta_01
z t"
proof
  fix z :: complex
  show "jacobi_theta_01 (z + 1) t = jacobi_theta_01 z t"
    using jacobi_theta_00_left.plus_1[of "z + 1/2" t] by (simp add: jacobi_theta_01_def)
qed

interpretation jacobi_theta_01_right: periodic_fun_simple "λt. jacobi_theta_01
z t" 2
proof
  fix t :: complex
  show "jacobi_theta_01 z (t + 2) = jacobi_theta_01 z t"
    using jacobi_theta_00_right.plus_period[of "z + 1/2" t] by (simp add:
jacobi_theta_01_def)
qed

lemma jacobi_theta_10_plus1_left: "jacobi_theta_10 (z + 1) t = -jacobi_theta_10
z t"
```

```
  using jacobi_theta_00_left.plus_1[of "z + t / 2" t]
  by (simp add: jacobi_theta_10_def to_nome_add algebra_simps)

lemma jacobi_theta_11_plus1_left: "jacobi_theta_11 (z + 1) t = -jacobi_theta_11
z t"
  using jacobi_theta_00_left.plus_1[of "z + t / 2 + 1 / 2" t]
  by (simp add: jacobi_theta_11_def to_nome_add algebra_simps)

lemma jacobi_theta_10_plus2_right: "jacobi_theta_10 z (t + 2) = i * jacobi_theta_10
z t"
  using jacobi_theta_00_right.plus_1[of "z + t / 2" t]
        jacobi_theta_00_left.plus_1[of "z + t / 2" "t + 2"]
  by (simp add: jacobi_theta_10_def to_nome_add algebra_simps add_divide_distrib)

lemma jacobi_theta_11_plus2_right: "jacobi_theta_11 z (t + 2) = i * jacobi_theta_11
z t"
  using jacobi_theta_00_right.plus_1[of "z + t / 2 + 1 / 2" t]
        jacobi_theta_00_left.plus_1[of "z + t / 2 + 1 / 2" "t + 2"]
  by (simp add: jacobi_theta_11_def to_nome_add algebra_simps add_divide_distrib)



lemma jacobi_theta_00_plus_half_left': "jacobi_theta_00 (z + 1/2) t =
jacobi_theta_01 z t"
  by (simp add: jacobi_theta_01_def to_nome_add algebra_simps)

lemma jacobi_theta_01_plus_half_left: "jacobi_theta_01 (z + 1/2) t =
jacobi_theta_00 z t"
  using jacobi_theta_00_left.plus_1[of z t]
  by (simp add: jacobi_theta_01_def to_nome_add algebra_simps)

lemma jacobi_theta_10_plus_half_left': "jacobi_theta_10 (z + 1/2) t =
jacobi_theta_11 z t"
  by (simp add: jacobi_theta_10_def jacobi_theta_11_def to_nome_add algebra_simps)

lemma jacobi_theta_11_plus_half_left': "jacobi_theta_11 (z + 1/2) t =
-jacobi_theta_10 z t"
  using jacobi_theta_00_left.plus_1[of "z + t / 2" t]
  by (simp add: jacobi_theta_10_def jacobi_theta_11_def to_nome_add algebra_simps)



lemma tendsto_jacobi_theta_01 [tendsto_intros]:
  assumes "(f ⟶ w) F" "(g ⟶ q) F" "Im q > 0"
  shows   "((λz. jacobi_theta_01 (f z) (g z)) ⟶ jacobi_theta_01 w
q) F"
  unfolding jacobi_theta_01_def
  by (intro tendsto_intros assms(1,2)) (use assms(3) in ‹auto simp: norm_to_nome›)
```

lemma *continuous_on_jacobi_theta_01 [continuous_intros]:*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ Im (g z) > 0"*
shows *"continuous_on A (λz. jacobi_theta_01 (f z) (g z))"*
unfolding *jacobi_theta_01_def*
by *(intro continuous_intros assms(1,2)) (use assms(3)* **in** *‹auto simp: norm_to_nome›)*

lemma *continuous_jacobi_theta_01 [continuous_intros]:*
assumes *"continuous F f" "continuous F g" "Im (g (netlimit F)) > 0"*
shows *"continuous F (λz. jacobi_theta_01 (f z) (g z))"*
unfolding *jacobi_theta_01_def*
by *(intro continuous_intros assms(1,2)) (use assms(3)* **in** *‹auto simp: norm_to_nome›)*

lemma *holomorphic_jacobi_theta_01 [holomorphic_intros]:*
assumes *"f holomorphic_on A" "g holomorphic_on A" "⋀z. z ∈ A ⟹ Im (g z) > 0" "open A"*
shows *"(λz. jacobi_theta_01 (f z) (g z)) holomorphic_on A"*
unfolding *jacobi_theta_01_def* **by** *(intro holomorphic_intros assms(1,2)) (use assms(3-)* **in** *auto)*

lemma *analytic_jacobi_theta_01 [analytic_intros]:*
assumes *"f analytic_on A" "g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z) > 0"*
shows *"(λz. jacobi_theta_01 (f z) (g z)) analytic_on A"*
unfolding *jacobi_theta_01_def* **by** *(intro analytic_intros assms(1,2)) (use assms(3-)* **in** *auto)*

lemma *tendsto_jacobi_theta_10 [tendsto_intros]:*
assumes *"(f ⟶ w) F" "(g ⟶ q) F" "Im q > 0"*
shows *"((λz. jacobi_theta_10 (f z) (g z)) ⟶ jacobi_theta_10 w q) F"*
unfolding *jacobi_theta_10_def*
by *(intro tendsto_intros assms(1,2)) (use assms(3)* **in** *‹auto simp: norm_to_nome›)*

lemma *continuous_on_jacobi_theta_10 [continuous_intros]:*
assumes *"continuous_on A f" "continuous_on A g"*
assumes *"⋀z. z ∈ A ⟹ Im (g z) > 0"*
shows *"continuous_on A (λz. jacobi_theta_10 (f z) (g z))"*
unfolding *jacobi_theta_10_def*
by *(intro continuous_intros assms(1,2)) (use assms(3)* **in** *‹auto simp: norm_to_nome›)*

lemma *continuous_jacobi_theta_10 [continuous_intros]:*
assumes *"continuous F f" "continuous F g" "Im (g (netlimit F)) > 0"*
shows *"continuous F (λz. jacobi_theta_10 (f z) (g z))"*
unfolding *jacobi_theta_10_def*

```
  by (intro continuous_intros continuous_divide assms(1,2))
     (use assms(3) in ‹auto simp: norm_to_nome›)

lemma holomorphic_jacobi_theta_10 [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A" "⋀z. z ∈ A ⟹ Im
(g z) > 0" "open A"
  shows   "(λz. jacobi_theta_10 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_10_def by (intro holomorphic_intros assms(1,2))
(use assms(3-) in auto)

lemma analytic_jacobi_theta_10 [analytic_intros]:
  assumes "f analytic_on A" "g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z)
> 0"
  shows   "(λz. jacobi_theta_10 (f z) (g z)) analytic_on A"
  unfolding jacobi_theta_10_def by (intro analytic_intros assms(1,2))
(use assms(3-) in auto)


lemma tendsto_jacobi_theta_11 [tendsto_intros]:
  assumes "(f ⟶ w) F" "(g ⟶ q) F" "Im q > 0"
  shows   "((λz. jacobi_theta_11 (f z) (g z)) ⟶ jacobi_theta_11 w
q) F"
  unfolding jacobi_theta_11_def
  by (intro tendsto_intros assms(1,2)) (use assms(3) in ‹auto simp: norm_to_nome›)

lemma continuous_on_jacobi_theta_11 [continuous_intros]:
  assumes "continuous_on A f" "continuous_on A g"
  assumes "⋀z. z ∈ A ⟹ Im (g z) > 0"
  shows   "continuous_on A (λz. jacobi_theta_11 (f z) (g z))"
  unfolding jacobi_theta_11_def
  by (intro continuous_intros assms(1,2)) (use assms(3) in ‹auto simp:
norm_to_nome›)

lemma continuous_jacobi_theta_11 [continuous_intros]:
  assumes "continuous F f" "continuous F g" "Im (g (netlimit F)) > 0"
  shows   "continuous F (λz. jacobi_theta_11 (f z) (g z))"
  unfolding jacobi_theta_11_def
  by (intro continuous_intros continuous_divide assms(1,2))
     (use assms(3) in ‹auto simp: norm_to_nome›)

lemma holomorphic_jacobi_theta_11 [holomorphic_intros]:
  assumes "f holomorphic_on A" "g holomorphic_on A" "⋀z. z ∈ A ⟹ Im
(g z) > 0" "open A"
  shows   "(λz. jacobi_theta_11 (f z) (g z)) holomorphic_on A"
  unfolding jacobi_theta_11_def by (intro holomorphic_intros assms(1,2))
(use assms(3-) in auto)

lemma analytic_jacobi_theta_11 [analytic_intros]:
  assumes "f analytic_on A" "g analytic_on A" "⋀z. z ∈ A ⟹ Im (g z)
```

```
> 0"
  shows   "(λz. jacobi_theta_11 (f z) (g z)) analytic_on A"
  unfolding jacobi_theta_11_def by (intro analytic_intros assms(1,2))
(use assms(3-) in auto)
```

end

4 The Jacobi Triple Product

```
theory Jacobi_Triple_Product
  imports Theta_Functions "Lambert_Series.Lambert_Series_Library"
begin
```

4.1 Versions for Jacobi's theta function

```
unbundle qpochhammer_inf_notation
```

The following follows the short proof given by Andrews [**?**], which makes use of two series expansions for $(a;b)_\infty$ and $1/(a;b)\infty$ due to Euler.

We prove it for Jacobi's theta function and derive a version for Ramanujan's later on. One could possibly also adapt the prove to work for Ramanujan's version directly, which makes the transfer to Jacobi's function a bit easier. However, I chose to do it for Jacobi's version first in order to stay closer to the text by Andrews.

The proof is fairly straightforward; the only messy part is proving the absolute convergence of the double sum (which Andrews does not bother doing). This is necessary in order to allow the exchange of the order of summation.

```
theorem jacobi_theta_nome_triple_product_complex:
  fixes w q :: complex
  assumes "w ≠ 0" "norm q < 1"
  shows   "jacobi_theta_nome w q = (q² ; q²)∞ * (-q*w ; q²)∞ * (-q/w
; q²)∞"
proof -
```

We first establish the identity for real w and q with the somewhat arbitrarily chosen bounds $q \in (0, \frac{1}{4})$ and $w \in (\frac{1}{2}, 1)$. This is considerably more restrictive than Andrews's version, but I was not able to prove absolute convergence of the sum for his bounds.

It does not matter anyway, since we will extend it to the full domain of ϑ by analytic continuation later anyway.

```
  have eq_real: "jacobi_theta_nome w q = (q²; q²)∞ * (-q*w; q²)∞ * (-q/w;
q²)∞"
    if wq: "0 < q" "q < 1/4" "1/2 < w" "w < 1" for q w :: real
```

```
  proof -
    have q2: "q^2 < 1"
      using wq by (simp add:  power_less_one_iff)
    define tri where "tri = (λn::nat. n*(n+1) div 2)"
    have [simp]: "2 * (n * (n - Suc 0) div 2) = n * (n - Suc 0)" for n
      by (subst dvd_mult_div_cancel) auto
    from wq have [simp]: "w ≠ 0" "q ≠ 0"
      by auto
    have [simp]: "(q^2; q^2)_∞ ≠ 0"
    proof
      assume "(q^2; q^2)_∞ = 0"
      then obtain n where "q^2 * q ^ (2*n) = 1"
        using q2 wq by (auto simp: qpochhammer_inf_zero_iff power_mult)
      hence "q ^ (2*n + 2) = 1"
        by (simp add: eval_nat_numeral)
      moreover have "q ^ (2*n + 2) < 1"
        by (subst power_less_one_iff) (use wq in auto)
      ultimately show False
        by linarith
    qed

    have "((λn. (q^2)^(n*(n-1) div 2) * (-w*q)^n / (∏k=1..n. (q^2)^k -
1)) has_sum (-w*q;q^2)_∞) UNIV"
      by (intro norm_summable_imp_has_sum sums_qpochhammer_inf_real norm_summable_qpochhamm
          (use wq q2 mult_strict_mono[of w 1 q 1] in auto)
    hence "((λn. q ^ (n*(n-1)) * (q^n * (-w)^n) / (∏k=1..n. q^(2*k)
- 1)) has_sum (-w*q; q^2)_∞) UNIV"
      by (simp add: norm_power power_less_one_iff power_minus' power_mult_distrib
mult_ac
                flip: power_mult)
    also have "(λn. q ^ (n*(n-1)) * (q^n * (-w)^n) / (∏k=1..n. q^(2*k)
- 1)) =
                (λn. q ^ (n^2) * w^n * (q^(2*n+2); q^2)_∞ / (q^2;q^2)_∞)"
(is "?lhs = ?rhs")
    proof
      fix n :: nat
      have "(q^2 ; q^2)_∞ = (∏k<n. 1 - q^(2*k+2)) * (q^(2*n+2) ; q^2)_∞"
        using qpochhammer_inf_mult_power_q[of "q^2" "q^2" n] q2
        by (simp add: power_add power2_eq_square power_mult_distrib mult_ac
mult_2_right
                      qpochhammer_nonneg_def)
      also have "(∏k<n. 1 - q^(2*k+2)) = (∏k=1..n. 1 - q^(2*k))"
        by (rule prod.reindex_bij_witness[of _ "λk. k-1" Suc]) auto
      also have "(∏k=1..n. 1 - q^(2*k)) = (-1)^n * (∏k=1..n. q^(2*k)-1)"
        by (subst prod_diff_swap) auto
      finally have "?lhs n = q ^ (n*(n-1)) * q^n * w^n * (q^(2*n+2); q^2)_∞
/ (q^2;q^2)_∞"
        by (auto simp: divide_simps power_minus')
      also have "q^(n*(n-1)) * q^n = q^(n*(n-1) + n)"
```

```
      by (simp add: power_add)
    also have "n*(n-1) + n = n^2"
      by (simp add: power2_eq_square algebra_simps)
    finally show "?lhs n = ?rhs n" .
  qed
  finally have "((λn. (q²; q²)∞ * (q^(n^2) * w^n * (q^(2*n+2); q²)∞
/ (q²;q²)∞))
                  has_sum (q²; q²)∞ * (-w*q;q²)∞) UNIV"
    by (rule has_sum_cmult_right)
  hence "((λn. q^(n^2) * w^n * (q^(2*n+2); q²)∞) has_sum (q²;q²)∞
* (-w*q;q²)∞) UNIV"
    by simp
  also have "?this ⟷ ((λn. q powi (n^2) * w powi n * (q powi (2*n+2);
q²)∞) has_sum
                          (q²;q²)∞ * (-w*q;q²)∞) {0..}"
    by (intro has_sum_reindex_bij_witness[of _ nat int])
       (auto simp: power_int_def nat_add_distrib nat_mult_distrib nat_power_eq)
  also have "... ⟷ ((λn. q powi (n^2) * w powi n * (q powi (2*n+2);
q²)∞) has_sum
                          (q²;q²)∞ * (-w*q;q²)∞) UNIV"
  proof (rule has_sum_cong_neutral)
    fix n :: int
    assume "n ∈ UNIV - {0..}"
    hence n: "n < 0"
      by simp
    define k where "k = nat (-n-1)"
    have "q powi (2 * int k) = (q ^ 2) ^ k"
      by (auto simp: power_int_def power_mult nat_mult_distrib)
    hence "q powi (2 * n + 2) * q² ^ k = q powi (2*(n + 1 + int k))"
      by (simp add: power_int_add flip: power_mult power_int_mult)
    also have "n + 1 + int k = 0"
      using n by (auto simp: k_def)
    finally have "∃k. q powi (2 * n + 2) * q² ^ k = 1"
      by (intro exI[of _ k]) auto
    thus "q powi (n^2) * w powi n * (q powi (2 * n + 2) ; q²)∞ = 0"
      using q2 by (auto simp: qpochhammer_inf_zero_iff intro!: exI[of
_ "-1"])
  qed auto
  finally have "((λn. q powi (n^2) * w powi n * (q powi (2*n+2); q²)∞)
has_sum
                  (q²;q²)∞ * (-w*q;q²)∞) UNIV" .
```

Brace yourselves: now we have to prove absolute convergence of the double sum. We use a crude bound for the inner sum, at which point the outer sum is just a geometric one that obviously converges.

```
  have "((λ(n,m). q powi n² * w powi n * ((-1)^m * q powi (m^2+m+2*n*m)
/ (∏ i=1..m. 1 - q^(2*i)))) has_sum
          (q²; q²)∞ * (-w*q ; q²)∞) (UNIV × UNIV)"
  proof (rule has_sum_SigmaI; (unfold prod.case)?)
```

```
      show "((λn. q powi (n^2) * w powi n * (q powi (2*n+2); q^2)_∞)
              has_sum (q^2;q^2)_∞ * (-w*q;q^2)_∞) UNIV"
        by fact
    next
      fix n :: int
      have "((λm. q^2 ^ (m*(m-1) div 2) * (q powi (2*n+2))^m / (∏k=1..m.
q^2^k-1)) has_sum
                    (q powi (2*n+2);q^2)_∞) UNIV"
        by (intro norm_summable_imp_has_sum sums_qpochhammer_inf_real
                  norm_summable_qpochhammer_inf) (use q2 in auto)
      also have "(λm. (q^2) ^ (m*(m-1) div 2) * (q powi (2*n+2))^m / (∏k=1..m.
q^2^k-1)) =
                 (λm. (-1)^m * q powi (m^2+m+2*n*m) / (∏k=1..m. 1-q^(2*k)))"
(is "?lhs = ?rhs")
      proof
        fix m :: nat
        have "(q^2) ^ (m*(m-1) div 2) * (q powi (2*n+2))^m =
              q powi (int m * int (m-1) + int m * (2*n+2))"
          by (simp add: power_int_add power_int_nonneg_exp nat_mult_distrib
power_int_power'
                        power_mult_distrib ring_distribs mult_ac
                  del: power_Suc flip: power_mult)
        also have "int m * int (m-1) + int m * (2*n+2) = int m ^ 2 + int
m + 2 * n * int m"
          by (cases m) (simp_all add: algebra_simps power2_eq_square)
        finally have "q^2 ^ (m * (m - 1) div 2) * (q powi (2 * n + 2)) ^
m =
                      q powi (int m ^ 2 + int m + 2 * n * int m)" .
        moreover have "(∏k=1..m. q^2^k-1) = (-1)^m * (∏k=1..m. 1 - q^(2*k))"
        proof -
          have "(∏k=1..m. q^2^k-1) = (∏k=1..m. q^(2*k)-1)"
            by (simp add: power_mult)
          also have "... = (-1)^m * (∏k=1..m. 1 - q^(2*k))"
            by (subst prod_diff_swap) auto
          finally show ?thesis .
        qed
        ultimately show "?lhs m = ?rhs m"
          by (simp add: divide_simps)
      qed
      finally show "((λm. q powi (n^2) * w powi n * ((-1)^m * q powi (m^2+m+2*n*m)
/ (∏k=1..m. 1 - q^(2*k)))) has_sum
                    (q powi (n^2) * w powi n * (q powi (2*n+2);q^2)_∞))
UNIV"
        by (rule has_sum_cmult_right)
    next
      have "(λ(n, m). q powi n^2 * w powi n * (q powi ((int m)^2 + int m
+ 2 * n * int m) /
              (∏i = 1..m. 1 - q ^ (2 * i)))) summable_on UNIV × UNIV"
      proof (rule summable_on_comparison_test; (safe)?)
```

```
      fix n :: int and m :: nat
      have "q powi (n^2) * w powi n * (q powi (m^2 + m + 2*n*m) / (∏i=1..m.
1-q^(2*i))) ≤
            q powi (n^2) * w powi n * (q powi (m^2 + m + 2*n*m) / (∏i=1..m.
1-q^1))"
        by (intro mult_left_mono divide_left_mono prod_mono zero_le_power_int
conjI
                  diff_left_mono power_decreasing mult_nonneg_nonneg
prod_pos mult_pos_pos)
           (use wq in <auto simp: power_less_one_iff power_le_one_iff>)
      also have "... = q powi (n^2) * w powi n * (q powi (m^2 + m + 2*n*m)
/ (1-q)^m)"
        by simp
      also have "... = q powi (n+m)^2 * w powi n * (q / (1-q))^m"
        by (simp add: power2_eq_square algebra_simps power_int_add power_divide)
      also have "... = q powi (n+m)^2 * w powi (n+m) * (q / (w * (1
- q)))^m"
        using wq by (simp add: power_int_add divide_simps mult_ac)
      also have "... ≤ w powi (n+m)^2 * w powi (n+m) * (q / (w * (1
- q)))^m"
        by (rule mult_right_mono power_int_mono)+ (use wq in auto)
      also have "... = w powi ((n+m)^2+(n+m)) * (q/(w*(1-q)))^m"
        by (simp add: power2_eq_square power_int_add)
      finally show "q powi n^2 * w powi n * (q powi (int m ^ 2 + int
m + 2*n * int m) /
                        (∏i=1..m. 1-q^(2*i)))
                      ≤ (case (n, m) of (n, m) ⇒ w powi ((n+m)^2+(n+m))
* (q/(w*(1-q)))^m)"
        by simp
    next
      have "(λ(n,m). w powi (n^2+n) * (q/(w*(1-q)))^m) summable_on UNIV
× UNIV"
      proof (rule summable_on_SigmaI; (unfold prod.case)?)
        fix n :: int
        have "q < (1 / 2) * (3 / 4)"
          using wq by simp
        also have "... ≤ w * (1 - q)"
          by (intro mult_mono) (use wq in auto)
        finally have "(λm. (q/(w*(1-q)))^m) sums (1 / (1 - q/(w*(1-q))))"
          by (intro geometric_sums) (use wq in <auto simp: power_le_one_iff
power_less_one_iff>)
        hence "((λm. (q/(w*(1-q)))^m) has_sum (1 / (1 - q/(w*(1-q)))))
UNIV"
          by (intro sums_nonneg_imp_has_sum) (use wq in auto)
        thus "((λm. w powi (n^2 + n) * (q / (w * (1 - q))) ^ m)
                has_sum (w powi (n^2 + n)) * (1/(1-q/(w*(1-q))))) UNIV"
          by (intro has_sum_cmult_right)
      next
        have "summable (λn. w ^ n)"
```

```
          by (intro summable_mult2 summable_geometric) (use wq in auto)
        hence "(λn. w ^ n) summable_on UNIV"
          by (rule summable_nonneg_imp_summable_on) (use wq in auto)
        hence "(λn. w ^ n * (1 / (1 - q / (w * (1 - q))))) summable_on
UNIV"
          by (intro summable_on_cmult_left)
        hence "(λn. w ^ n * (1 / (1 - q / (w * (1 - q))))) summable_on
range (λn. n*(n+1))"
          by (rule summable_on_subset) auto
        also have "?this ⟷ (λn. w ^ (n*(n+1)) * (1 / (1 - q/(w*(1-q)))))
summable_on UNIV"
        proof (subst summable_on_reindex)
          have "strict_mono (λn::nat. n * (n + 1))"
            by (intro strict_monoI_Suc) auto
          thus "inj (λn::nat. n * (n + 1))"
            by (rule strict_mono_imp_inj_on)
        qed (simp_all only: o_def)
        also have "... ⟷ (λn. w powi (n*(n+1)) * (1 / (1 - q/(w*(1-q)))))
summable_on {0..}"
          by (rule summable_on_reindex_bij_witness[of _ nat int])
             (auto simp: power_int_def nat_mult_distrib nat_add_distrib)
        finally have summable1: "(λn. w powi (n*(n+1)) * (1/(1-q/(w*(1-q)))))
summable_on {0..}" .
        also have "?this ⟷ (λn. w powi (n*(n+1)) * (1/(1-q/(w*(1-q)))))
summable_on {..-1}"
          by (rule summable_on_reindex_bij_witness[of _ "λn. -n-1"
"λn. -n-1"])
             (auto simp: algebra_simps)
        finally have summable2: "(λn. w powi (n*(n+1)) * (1/(1-q/(w*(1-q)))))
summable_on {..-1}" .
        have "(λn. w powi (n*(n+1)) * (1/(1-q/(w*(1-q))))) summable_on
({..-1} ∪ {0..})"
          by (intro summable_on_union summable1 summable2)
        also have "{..-1} ∪ {0::int..} = UNIV"
          by auto
        finally show "(λn. w powi (n^2+n) * (1/(1-q/(w*(1-q))))) summable_on
UNIV"
          by (simp add: power2_eq_square algebra_simps)
      qed (use wq in auto)
      also have "?this ⟷ ((λ(n,m). w powi ((n + int m)² + (n + int
m)) * (q/(w*(1-q))) ^ m)
                                summable_on UNIV × UNIV)"
        by (rule summable_on_reindex_bij_witness
              [of _ "λ(n,m). (n + int m, m)" "λ(n,m). (n - int m, m)"])
auto
      finally show "(λ(n,m). w powi ((n+m)² + (n+m)) * (q/(w*(1-q)))^m)
summable_on UNIV × UNIV" .
    qed (use wq in ‹auto intro!: mult_nonneg_nonneg divide_nonneg_pos
prod_pos
```

```
                        simp: power_less_one_iff›)
      hence "(λ(n, m). q powi n² * w powi n * ((-1) ^ m * q powi ((int
m)² + int m + 2 * n * int m) /
              (∏ i=1..m. 1 - q ^ (2 * i)))) abs_summable_on UNIV × UNIV"
        using wq by (simp add: case_prod_unfold abs_mult power_abs
                               power_int_abs abs_prod power_le_one_iff)
      thus "(λ(n, m). q powi n² * w powi n * ((- 1) ^ m * q powi ((int
m)² + int m + 2 * n * int m) /
              (∏ i = 1..m. 1 - q ^ (2 * i)))) summable_on UNIV × UNIV"
        by (rule abs_summable_summable)
    qed

    also have "(λ(n,m). q powi n² * w powi n * ((-1)^m * q powi (int m
^ 2 + int m + 2 * n * int m) /
                  (∏ i=1..m. 1 - q^(2*i)))) =
               (λ(n,m). ((-1)^m * (q/w)^m / (∏ i=1..m. 1 - q^(2*i))) *
                        (q powi ((n+m)^2) * w powi (n+m)))"
      by (simp add: power2_eq_square field_simps power_int_add)
    also have "(... has_sum ((q²; q²)∞ * (-w*q; q²)∞)) (UNIV × UNIV)
⟷
                  ((λ(n,m). ((-1)^m * (q/w)^m / (∏ i=1..m. 1 - q^(2*i)))
* (q powi (n^2) * w powi n))
                     has_sum ((q²; q²)∞ * (-w*q; q²)∞)) (UNIV × UNIV)"
      by (rule has_sum_reindex_bij_witness[of _ "λ(n, m). (n - int m,
m)" "λ(n, m). (n + int m, m)"])
         auto
    finally have sum:
      "((λ(n,m). ((-q/w)^m / (∏ i=1..m. 1 - q^(2*i))) * (q powi (n^2)
* w powi n))
         has_sum ((q²; q²)∞ * (-w*q; q²)∞)) (UNIV × UNIV)"
      by (simp add: power_minus')

    have "((λn. inverse (-q/w; q²)∞ * (q powi (n^2) * w powi n))
           has_sum (q² ; q²)∞ * (- w * q ; q²)∞) UNIV"
    proof (rule has_sum_SigmaD[OF sum]; unfold prod.case)
      fix n :: int
      have "((λm. (-q/w)^m / (∏ i=1..m. 1 - (q²)^i)) has_sum (inverse
(-q/w; q²)∞)) UNIV"
        by (intro norm_summable_imp_has_sum sums_inverse_qpochhammer_inf_real
                  norm_summable_inverse_qpochhammer_inf)
           (use q2 wq in ‹auto simp: norm_divide›)
      also have "(λi. (q^2)^i) = (λi. q^(2*i))"
        by (simp add: power_mult)
      finally show "((λm. ((-q/w)^m / (∏ i=1..m. 1 - q^(2*i))) * (q powi
(n^2) * w powi n))
                       has_sum (inverse (-q/w; q²)∞ * (q powi (n^2) *
w powi n))) UNIV"
        by (rule has_sum_cmult_left)
    qed
```

```
    hence "((λn. (-q/w; q²)∞ * (inverse (-q/w; q²)∞ * (q powi (n^2)
* w powi n)))
             has_sum (-q/w;q²)∞ * ((q²;q²)∞ * (-w*q;q²)∞)) UNIV"
      by (rule has_sum_cmult_right)
    moreover have "(-q/w; q²)∞ ≠ 0"
    proof
      assume "(-q/w; q²)∞ = 0"
      then obtain n where "q ^ (2*n+1) / w = -1"
        using q2 by (auto simp: qpochhammer_inf_zero_iff power_mult)
      have "norm (q ^ (2 * n + 1) / w) = q ^ (2*n+1) / w"
        using wq by (simp add: norm_divide norm_power)
      also have "... ≤ q ^ 1 / w"
        by (intro divide_right_mono power_decreasing) (use wq in auto)
      also have "... < 1"
        using wq by simp
      also note ‹q ^ (2*n+1) / w = -1›
      finally show False
        by simp
    qed
    ultimately have "((λn. w powi n * q powi (n^2)) has_sum
                       (q²; q²)∞ * (-q*w; q²)∞ * (-q/w; q²)∞) UNIV"
      by (simp add: divide_simps mult_ac)
    moreover have "((λn. w powi n * q powi (n^2)) has_sum jacobi_theta_nome
w q) UNIV"
      by (rule has_sum_jacobi_theta_nome) (use wq in auto)
    ultimately show "jacobi_theta_nome w q = (q²; q²)∞ * (-q*w; q²)∞
* (-q/w; q²)∞"
      using has_sum_unique by blast
  qed
```

We perform analytic continuation to extend the identity to all w:

```
  have eq_real2:
     "jacobi_theta_nome w (complex_of_real q) =
        (of_real q ^ 2; of_real q ^ 2)∞ * (-of_real q * w; of_real q
^ 2)∞ *
        (-of_real q / w; of_real q ^ 2)∞"
    if wq: "w ≠ 0" "q ∈ {0<..<1/4}" for w :: complex and q :: real
  proof -
    define r where "r = (2/3 :: real)"
    have r: "q < r" "1/2 < r" "r < 1"
      using wq by (simp_all add: r_def)
    define f where
      "f = (λw. jacobi_theta_nome w (complex_of_real q) -
                (of_real q ^ 2; of_real q ^ 2)∞ * (-of_real q * w; of_real
q ^ 2)∞ *
                (-of_real q / w; of_real q ^ 2)∞)"
    have "f w = 0"
    proof (rule analytic_continuation[of f])
      show "f holomorphic_on (-{0})"
```

```
      unfolding f_def using wq
      by (auto intro!: holomorphic_intros simp: norm_power power_less_one_iff)
    next
      show "of_real ` {1/2<..<1} ⊆ -{0}"
      using wq by auto
    next
      show "of_real r islimpt complex_of_real ` {1/2<..<1}" using r
      by (intro islimpt_isCont_image continuous_intros)
         (auto simp: eventually_at_filter open_imp_islimpt simp del:
of_real_add)
    next
      show "f w = 0" if "w ∈ complex_of_real ` {1/2<..<1}" for w :: complex
      proof -
        from that obtain x where x: "w = complex_of_real x" "x ∈ {1/2<..<1}"
          by auto
        have "0 = complex_of_real (jacobi_theta_nome x q - (q^2 ; q^2)∞
* (-q*x; q^2)∞ * (-q/x ; q^2)∞)"
          by (subst eq_real) (use wq x in auto)
        also have "... = f w" using x wq
          by (simp add: f_def power_less_one_iff
                   flip: jacobi_theta_nome_of_real qpochhammer_inf_of_real)
        finally show ?thesis ..
      qed
    qed (use wq r in ‹auto simp: connected_punctured_universe›)
    thus ?thesis
      by (simp add: f_def)
  qed
```

And another analytic continuation to extend it to all q:

```
  show "jacobi_theta_nome w q = (q ^ 2; q ^ 2)∞ * (-q * w; q ^ 2)∞ *
(-q / w; q ^ 2)∞"
  proof -
    note wq = assms
    define f where
      "f = (λq. jacobi_theta_nome w q -
                (q ^ 2; q ^ 2)∞ * (-q * w; q ^ 2)∞ * (-q / w; q ^ 2)∞)"
    have "f q = 0"
    proof (rule analytic_continuation[of f])
      show "f holomorphic_on (ball 0 1)"
        unfolding f_def using wq
        by (auto intro!: holomorphic_intros simp: norm_power power_less_one_iff)
    next
      show "of_real ` {0<..<1/4} ⊆ ball (0::complex) 1"
        using wq by auto
    next
      show "of_real (1/8) islimpt complex_of_real ` {0<..<1/4}"
        by (intro islimpt_isCont_image continuous_intros)
           (auto simp: eventually_at_filter open_imp_islimpt complex_eq_iff)
    next
```

```
    show "f q = 0" if "q ∈ complex_of_real ` {0<..<1/4}" for q :: complex
    proof -
      from that obtain x where x: "q = complex_of_real x" "x ∈ {0<..<1/4}"
        by auto
      have "0 = jacobi_theta_nome w x - (of_real x ^ 2 ; of_real x ^
2)∞ *
                  (-of_real x * w; of_real x ^ 2)∞ * (-of_real x / w
; of_real x ^ 2)∞"
        by (subst eq_real2) (use wq x in auto)
      also have "... = f q" using x wq
        by (simp add: f_def power_less_one_iff
                 flip: jacobi_theta_nome_of_real qpochhammer_inf_of_real)
      finally show ?thesis ..
    qed
  qed (use wq in auto)
  thus ?thesis
    by (simp add: f_def)
 qed
qed

lemma jacobi_theta_nome_triple_product_real:
 fixes w q :: real
 assumes "w ≠ 0" "|q| < 1"
 shows   "jacobi_theta_nome w q = (q^2 ; q^2)∞ * (-q*w ; q^2)∞ * (-q/w
; q^2)∞"
proof -
 define q' w' where "q' = complex_of_real q" and "w' = complex_of_real
w"
 have "complex_of_real (jacobi_theta_nome w q) = jacobi_theta_nome w'
q'"
   by (simp add: jacobi_theta_nome_of_real w'_def q'_def)
 also have "... = (q'^2 ; q'^2)∞ * (-q'*w' ; q'^2)∞ * (-q'/w' ; q'^2)∞"
   by (rule jacobi_theta_nome_triple_product_complex)
      (use assms in ‹simp_all add: q'_def w'_def›)
 also have "... = of_real ((q^2 ; q^2)∞ * (-q*w ; q^2)∞ * (-q/w ; q^2)∞)"
using assms
   by (simp add: q'_def w'_def power_less_one_iff abs_square_less_1 flip:
qpochhammer_inf_of_real)
 finally show ?thesis
   by (simp only: of_real_eq_iff)
qed
```

4.2 Version of Ramanujan's theta function

The triple product for Ramanujan's theta function, which follows easily from the above one, has a particularly elegant form:

$$f(a,b) = (-a;ab)_\infty\,(-b;ab)_\infty\,(ab;ab)_\infty$$

It follows directly from the version for Jacobi's theta function, although I again have to employ analytic continuation to avoid dealing with the branch cuts when converting Ramanujan's theta function to Jacobi's.

```
theorem ramanujan_theta_triple_product_complex:
  fixes a b :: complex
  assumes "norm (a * b) < 1"
  shows   "ramanujan_theta a b = (-a ; a*b)∞ * (-b ; a*b)∞ * (a*b ;
a*b)∞"
proof -
  have real_eq1: "ramanujan_theta a b = (-a ; a * b)∞ * (-b ; a * b)∞
* (a * b ; a * b)∞"
    if ab: "a > 0" "b > 0" "a * b < 1" for a b :: real
  proof -
    define w q where "w = sqrt (a/b)" and "q = sqrt (a*b)"
    have [simp]: "w ≠ 0"
      using ab by (auto simp: w_def sgn_if)
    have q: "|q| < 1"
      using ab by (simp_all add: q_def abs_mult flip: real_sqrt_abs')

    have "ramanujan_theta a b = jacobi_theta_nome w q"
      using ab by (auto simp: sgn_if jacobi_theta_nome_def w_def q_def
real_sqrt_mult real_sqrt_divide)
    also have "... = (q^2 ; q^2)∞ * (- q * w ; q^2)∞ * (- q / w ; q^2)∞"
      by (rule jacobi_theta_nome_triple_product_real) (use q in auto)
    also have "... = (-a ; a * b)∞ * (-b ; a * b)∞ * (a * b ; a * b)∞"
      using ab by (simp add: mult_ac q_def w_def real_sqrt_mult real_sqrt_divide
power_mult_distrib)
    finally show ?thesis .
  qed

  have real_eq2: "ramanujan_theta a b = (-a ; a * b)∞ * (-b ; a * b)∞
* (a * b ; a * b)∞"
    if ab: "a > 0" "norm (a * b) < 1" for a :: real and b :: complex
  proof -
    define f :: "complex ⇒ complex"
      where "f = (λb. ramanujan_theta (of_real a) b - (-of_real a ; of_real
a * b)∞ *
                        (-b ; of_real a * b)∞ * (of_real a * b ; of_real
a * b)∞)"

    have "f b = 0"
    proof (rule analytic_continuation[where f = f])
      show "f holomorphic_on (ball 0 (1 / a))"
        unfolding f_def using ab by (intro holomorphic_intros) (auto
simp: norm_mult field_simps)
    next
      show "complex_of_real ` {0<..<1/a} ⊆ ball 0 (1/a)"
       and "complex_of_real (1/(2*a)) ∈ ball 0 (1/a)"
        using ab by (auto simp: norm_mult norm_divide field_simps)
```

```
    next
      show "complex_of_real (1 / (2 * a)) islimpt
            complex_of_real ` {0<..<1 / a}"
        by (intro islimpt_isCont_image continuous_intros open_imp_islimpt)
           (use ab in <auto simp: complex_eq_iff eventually_at_filter
field_simps>)
    next
      fix b assume "b ∈ complex_of_real ` {0<..<1/a}"
      then obtain x where x: "b = complex_of_real x" "x ∈ {0<..<1/a}"
        by auto
      show "f b = 0"
        unfolding f_def using real_eq1[of a x] x ab
        by (simp add: field_simps ramanujan_theta_of_real flip: qpochhammer_inf_of_real)
    qed (use ab in <auto simp: norm_mult field_simps>)
    thus ?thesis
      by (simp add: f_def)
  qed

  show "ramanujan_theta a b = (-a ; a * b)∞ * (-b ; a * b)∞ * (a * b
; a * b)∞"
  proof (cases "b = 0")
    case True
    thus ?thesis by auto
  next
    case False
    note ab = assms
    define f :: "complex ⇒ complex"
      where "f = (λa. ramanujan_theta a b - (-a ; a * b)∞ * (-b ; a *
b)∞ * (a * b ; a * b)∞)"

    have "f a = 0"
    proof (rule analytic_continuation[where f = f])
      show "f holomorphic_on (ball 0 (1 / norm b))"
        unfolding f_def using ab <b ≠ 0>
        by (intro holomorphic_intros) (auto simp: field_simps norm_mult)
    next
      show "complex_of_real ` {0<..<1/norm b} ⊆ ball 0 (1 / norm b)"
       and "complex_of_real (1/(2*norm b)) ∈ ball 0 (1 / norm b)"
        using ab <b ≠ 0> by (auto simp: norm_mult norm_divide field_simps)
    next
      show "complex_of_real (1 / (2 * norm b)) islimpt
            complex_of_real ` {0<..<1 / norm b}"
        by (intro islimpt_isCont_image continuous_intros open_imp_islimpt)
           (use ab <b ≠ 0> in <auto simp: complex_eq_iff eventually_at_filter
field_simps>)
    next
      fix a assume "a ∈ complex_of_real ` {0<..<1/norm b}"
      then obtain x where x: "a = complex_of_real x" "x ∈ {0<..<1/norm
b}"
```

```
    by auto
  show "f a = 0"
    unfolding f_def using real_eq2[of x b] x ab ‹b ≠ 0›
    by (simp add: norm_mult field_simps ramanujan_theta_of_real flip:
qpochhammer_inf_of_real)
 qed (use ab ‹b ≠ 0› in ‹auto simp: norm_mult field_simps›)
 thus ?thesis
  by (simp add: f_def)
qed
qed

lemma ramanujan_theta_triple_product_real:
 fixes a b :: real
 assumes ab: "|a * b| < 1"
 shows   "ramanujan_theta a b = (-a ; a * b)∞ * (-b ; a * b)∞ * (a
* b ; a * b)∞"
proof -
 define a' b' where "a' = complex_of_real a" and "b' = complex_of_real
b"
 have "complex_of_real (ramanujan_theta a b) = ramanujan_theta a' b'"
  by (simp add: a'_def b'_def ramanujan_theta_of_real)
 also have "... = (-a' ; a' * b')∞ * (-b' ; a' * b')∞ * (a' * b' ; a'
* b')∞"
  by (rule ramanujan_theta_triple_product_complex)
    (use assms in ‹auto simp: a'_def b'_def abs_mult norm_mult›)
 also have "... = complex_of_real ((-a ; a * b)∞ * (-b ; a * b)∞ * (a
* b ; a * b)∞)"
  using assms by (simp flip: qpochhammer_inf_of_real add: a'_def b'_def)
 finally show ?thesis
  by (simp only: of_real_eq_iff)
qed
```

4.3 The pentagonal number theorem

An easy corollary of this is the Pentagonal Number Theorem, which, in our notation, simply reads:

$$(q;q)_\infty = f(-q,-q^2) = \theta(-\sqrt{q}, q\sqrt{q})$$

```
corollary pentagonal_number_theorem_complex:
 fixes q :: complex
 assumes q: "norm q < 1"
 shows   "(q; q)∞ = ramanujan_theta (-q) (-(q²))"
proof -
 have "ramanujan_theta (-q) (-(q²)) = (∏ i<3. (q * q ^ i; q^3)∞)"
  by (subst ramanujan_theta_triple_product_complex)
    (use q in ‹simp_all flip: power_Suc add: norm_power power_less_one_iff
eval_nat_numeral›)
 also have "... = (q ; q)∞"
```

```
    by (rule prod_qpochhammer_group) (use q in auto)
  finally show ?thesis ..
qed

lemma pentagonal_number_theorem_real:
  fixes q :: real
  assumes q: "|q| < 1"
  shows   "(q; q)∞ = ramanujan_theta (-q) (-(q²))"
proof -
  have "complex_of_real ((q; q)∞) = (of_real q; of_real q)∞"
    by (subst qpochhammer_inf_of_real) (use q in auto)
  also have "... = complex_of_real (ramanujan_theta (-q) (-(q^2)))"
    by (subst pentagonal_number_theorem_complex)
       (use q in ‹auto simp flip: ramanujan_theta_of_real›)
  finally show ?thesis
    by (simp only: of_real_eq_iff)
qed
```

The following is the more explicit form of the Pentagonal Number Theorem usually found in textbooks:

$$\prod_{n=1}^{\infty}\left(1-q^n\right) = \sum_{k=-\infty}^{\infty}(-1)^k q^{k(3k-1)/2}$$

The exponents $g_k = k(3k-1)/2$ (for $k \in \mathbb{Z}$) are called the *generalised pentagonal numbers*.

```
corollary pentagonal_number_theorem_complex':
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "abs_convergent_prod (λn. 1 - q^(n+1))" (is ?th1)
    and "(λk. (-1) powi k * q powi (k*(3*k-1) div 2)) abs_summable_on
UNIV" (is ?th2)
    and "(∏n::nat. 1 - q^(n+1)) = (∑∞(k::int). (-1) powi k * q powi
(k * (3*k-1) div 2))" (is ?th3)
proof -
  have "?th1 ∧ ?th2 ∧ ?th3"
  proof (cases "q = 0")
    case [simp]: True
    have "((λn. (-1) powi n * q powi (n*(3*n-1) div 2))
            has_sum ramanujan_theta (-q) (-q²)) {0}"
      by (intro has_sum_finiteI) auto
    also have "?this ⟷ ((λn. (-1) powi n * q powi (n*(3*n-1) div 2))

                        has_sum ramanujan_theta (-q) (-q²)) UNIV"
      by (intro has_sum_cong_neutral) (auto simp: dvd_div_eq_0_iff)
    finally show ?thesis
      by (auto simp: abs_convergent_prod_def has_sum_iff summable_on_iff_abs_summable_on_co
  next
    case [simp]: False
```

```
    have "(λn. 1 + norm q ^ Suc n) has_prod (-norm q; norm q)∞"
      using has_prod_qpochhammer_inf[of "norm q" "-norm q"] q by simp
    hence th1: "abs_convergent_prod (λn. 1 - q ^ (n+1))"
      by (simp add: abs_convergent_prod_def norm_mult norm_power has_prod_iff)

    have prod: "(λn. 1 - q ^ Suc n) has_prod (q; q)∞"
      using has_prod_qpochhammer_inf[of q q] q by simp
    have "((λn. (-q) powi (n*(n+1) div 2) * (-(q^2)) powi (n*(n-1) div
2))
            has_sum ramanujan_theta (-q) (-(q^2))) UNIV"
      by (rule has_sum_ramanujan_theta)
         (auto simp: norm_power power_less_one_iff q simp flip: power_Suc)
    also have "(λn. (-q) powi (n*(n+1) div 2) * (-(q^2)) powi (n*(n-1)
div 2)) =
               (λn. (- 1) powi n * q powi (n*(3*n-1) div 2))"
      (is "?lhs = ?rhs")
    proof
      fix n :: int
      have "(-q) powi (n*(n+1) div 2) * (-(q^2)) powi (n*(n-1) div 2)
=
              (-1) powi (n*(n+1) div 2 + n*(n-1) div 2) *
              (q powi (n*(n+1) div 2) * (q^2) powi (n*(n-1) div 2))"
        by (auto simp: power_int_minus_left)
      also have "n*(n+1) div 2 + n*(n-1) div 2 = (n*(n+1) + n*(n-1)) div
2"
        by (rule div_plus_div_distrib_dvd_left [symmetric]) auto
      also have "(n*(n+1) + n*(n-1)) div 2 = n ^ 2"
        by (simp add: algebra_simps power2_eq_square)
      also have "(-1) powi (n ^ 2) = (-1::complex) powi n"
        by (auto simp: power_int_minus_left)
      also have "(q^2) powi (n*(n-1) div 2) = q powi (n*(n-1))"
        by (simp add: power_int_power)
      also have "q powi (n * (n + 1) div 2) * q powi (n * (n - 1)) =
                 q powi (n * (n + 1) div 2 + (2 * n * (n - 1)) div 2)"
        by (simp add: power_int_add)
      also have "n * (n + 1) div 2 + (2 * n * (n - 1)) div 2 = (n*(n+1)
+ 2*n*(n-1)) div 2"
        by (rule div_plus_div_distrib_dvd_left [symmetric]) auto
      also have "n*(n+1) + 2*n*(n-1) = n * (3 * n - 1)"
        by (simp add: algebra_simps)
      finally show "?lhs n = ?rhs n" .
    qed
    finally have sum: "((λn. (-1) powi n * q powi (n*(3*n-1) div 2))
                        has_sum ramanujan_theta (-q) (-q^2)) UNIV" .

    have "(λn. (-1) powi n * q powi (n*(3*n-1) div 2)) summable_on UNIV"
      using sum  by (rule has_sum_imp_summable)
    hence th2: "(λn. (-1) powi n * q powi (n*(3*n-1) div 2)) abs_summable_on
UNIV"
```

```
    by (simp add: summable_on_iff_abs_summable_on_complex)

  have th3: "(∏n. 1 - q ^ (n+1)) = (∑_∞(k::int). (-1) powi k * q powi
(k * (3*k-1) div 2))"
    using sum_prod pentagonal_number_theorem_complex[OF q]
    by (simp add: has_prod_iff has_sum_iff)

  show ?thesis
    using th1 th2 th3 by blast
 qed
 thus ?th1 ?th2 ?th3
  by blast+
qed
```

4.4 (Non-)vanishing of theta functinos

A corollary of the Jacobi triple product: the Jacobi theta function has no zeros apart from the "obvious" ones, i.e. the ones at the center of the cells of the lattice generated by the period 1 and the quasiperiod t.

```
corollary jacobi_theta_00_eq_0_iff_complex:
 fixes z t :: complex
 assumes "Im t > 0"
 shows   "jacobi_theta_00 z t = 0 ⟷ (∃m n. z = (of_int m + 1/2) +
(of_int n + 1/2) * t)"
proof
 assume "∃m n. z = (of_int m + 1/2) + (of_int n + 1/2) * t"
 then obtain m n where mn: "z = (of_int m + 1/2) + (of_int n + 1/2)
* t"
  by blast
 show "jacobi_theta_00 z t = 0"
  unfolding mn by (rule jacobi_theta_00_eq_0')
next
 assume "jacobi_theta_00 z t = 0"
 define w q where "w = to_nome (2*z)" and "q = to_nome t"
 have [simp]: "w ≠ 0" "q ≠ 0"
  by (auto simp: w_def q_def)
 have q: "norm q < 1"
  using assms by (auto simp: q_def norm_to_nome)
 have q2: "norm (q ^ 2) < 1"
  by (simp add: norm_power power_less_one_iff q)

 note ‹jacobi_theta_00 z t = 0›
 also have "jacobi_theta_00 z t = jacobi_theta_nome w q"
  by (simp add: jacobi_theta_00_def w_def q_def to_nome_power)
 also have "... = (q² ; q²)_∞ * (- q * w ; q²)_∞ * (- q / w ; q²)_∞"
  by (rule jacobi_theta_nome_triple_product_complex) (use q in auto)
 also have "... = 0 ⟷ ((∃n. q^(2*n+2) = 1) ∨ (∃n. (w*q^(2*n+1)) =
-1) ∨ (∃n. (q^(2*n+1)/w) = -1))"
```

```
    using q q2 by (simp add: qpochhammer_inf_zero_iff power_add power_mult
mult.assoc
                              power2_eq_square power_mult_distrib mult.commute[of
w]
                              minus_equation_iff[of _ 1] eq_commute[of
"-1"])
  also have "(λn. q^(2*n+2) = 1) = (λ_. False)"
  proof
    fix n :: nat
    have "norm (q ^ (2*n+2)) < 1"
      unfolding norm_power by (subst power_less_one_iff) (use q in auto)
    thus "q^(2*n+2) = 1 ⟷ False"
      by auto
  qed
  finally show "∃m n. z = (of_int m + 1/2) + (of_int n + 1/2) * t"
  proof (elim disjE exE FalseE)
    fix n :: nat
    assume "w * q ^ (2*n+1) = -1"
    also have "w * q ^ (2*n+1) = to_nome (2 * z + (2 * of_nat n + 1) *
t)"
      by (simp add: w_def q_def to_nome_power algebra_simps flip: to_nome_add)
    finally have "(2 * z + (2 * of_nat n + 1) * t - 1) / 2 ∈ ℤ"
      unfolding to_nome_eq_neg1_iff' .
    then obtain m where m: "(2 * z + (2 * of_nat n + 1) * t - 1) / 2
= of_int m"
      by (elim Ints_cases)
    have "z = (of_int m + 1/2) + (of_int (-int (n+1)) + 1/2) * t"
      by (subst m [symmetric]) (auto simp: field_simps)
    thus ?thesis
      by blast
  next
    fix n :: nat
    assume "q ^ (2*n+1) / w = -1"
    also have "q ^ (2*n+1) / w = to_nome (-2 * z + (2 * of_nat n + 1)
* t)"
      by (simp add: w_def q_def to_nome_power algebra_simps flip: to_nome_add
to_nome_diff)
    finally have "(-2 * z + (2 * of_nat n + 1) * t - 1) / 2 ∈ ℤ"
      unfolding to_nome_eq_neg1_iff' .
    then obtain m where m: "(-2 * z + (2 * of_nat n + 1) * t - 1) / 2
= of_int m"
      by (elim Ints_cases)
    have "z = (of_int (-m-1) + 1/2) + (of_int (int n) + 1/2) * t"
      unfolding of_int_diff of_int_minus by (subst m [symmetric]) (auto
simp: field_simps)
    thus ?thesis
      by blast
  qed
qed
```

```
lemma jacobi_theta_00_nonzero:
  assumes z: "Im t > 0" and "Im z / Im t - 1 / 2 ∉ ℤ"
  shows   "jacobi_theta_00 z t ≠ 0"
proof
  assume "jacobi_theta_00 z t = 0"
  then obtain m n where *: "z = (of_int m + 1/2) + (of_int n + 1/2) *
t"
    by (subst (asm) jacobi_theta_00_eq_0_iff_complex) (use <Im t > 0>
in auto)
  hence "Im z = (of_int n + 1/2) * Im t"
    by simp
  hence "Im z / Im t - 1 / 2 = of_int n"
    using z by (auto simp: field_simps)
  also have "... ∈ ℤ"
    by auto
  finally show False
    using assms by simp
qed

lemma jacobi_theta_00_0_left_nonzero:
  assumes "Im t > 0"
  shows   "jacobi_theta_00 0 t ≠ 0"
  by (rule jacobi_theta_00_nonzero) (use assms in auto)

lemma jacobi_theta_nome_nonzero_complex:
  fixes q w :: complex
  assumes [simp]: "w ≠ 0" "norm q < 1"
  assumes "q = 0 ∨ (ln (norm w) / ln (norm q) - 1) / 2 ∉ ℤ"
  shows   "jacobi_theta_nome w q ≠ 0"
proof (cases "q = 0")
  case [simp]: False
  define z where "z = -i * ln w / pi"
  define t where "t = -i * ln q / pi"
  have [simp]: "to_nome z = w" "to_nome t = q"
    using assms by (simp_all add: z_def t_def to_nome_def)
  have "Im t > 0"
    by (auto simp: t_def field_simps)

  have *: "Im z / (2 * Im t) - 1 / 2 = (ln (norm w) / ln (norm q) - 1)
/ 2"
    by (auto simp: z_def t_def)
  have "jacobi_theta_nome w q = jacobi_theta_00 (z/2) t"
    by (simp add: jacobi_theta_00_def to_nome_power)
  also have "... ≠ 0"
  proof (rule jacobi_theta_00_nonzero)
    have "Im (z / 2) / Im t - 1/2 = ln (norm w) / (2 * ln (norm q)) -
1 / 2"
      by (simp add: z_def t_def)
```

```
    also have "... = (ln (norm w) / ln (norm q) - 1) / 2"
      by (simp add: field_simps)
    also have "... ∉ ℤ"
      using assms by auto
    finally show "Im (z / 2) / Im t - 1/2 ∉ ℤ" .
  qed (use * ‹Im t > 0› in auto)
  finally show ?thesis .
qed auto

lemma jacobi_theta_nome_q_q_nonzero_complex:
  assumes "norm (q::complex) < 1" "q ≠ 0"
  shows   "jacobi_theta_nome q q ≠ 0"
proof
  assume "jacobi_theta_nome q q = 0"
  define t where "t = -i * ln q / pi"
  have [simp]: "to_nome t = q"
    using assms by (simp_all add: t_def to_nome_def)
  have "Im t > 0"
    using assms by (auto simp: t_def field_simps)

  note ‹jacobi_theta_nome q q = 0›
  also have "jacobi_theta_nome q q = jacobi_theta_00 (t/2) t"
    by (simp add: jacobi_theta_00_def to_nome_power)
  finally obtain m n where *: "t / 2 = (of_int m + 1/2) + (of_int n +
1/2) * t"
    by (subst (asm) jacobi_theta_00_eq_0_iff_complex) (use ‹Im t > 0›
in auto)
  have "t = (2 * of_int m + 1) + (2 * of_int n + 1) * t"
    using arg_cong[OF *, of "λx. x * 2"] by (simp add: ring_distribs
mult_ac)
  hence **: "2 * of_int n * t = -(2 * of_int m + 1)"
    by (Groebner_Basis.algebra)
  have "n = 0"
    using ‹Im t > 0› arg_cong[OF **, of Im] by simp
  with ** have "-2 * complex_of_int m = of_int 1"
    by simp
  also have "-2 * complex_of_int m = of_int (-2 * m)"
    by simp
  finally have "-2 * m = 1"
    by (simp only: of_int_eq_iff)
  thus False
    by presburger
qed

lemma jacobi_theta_nome_nonzero_real:
  fixes q w :: real
  assumes [simp]: "w ≠ 0" "norm q < 1" and "(ln |w| / ln |q| - 1) / 2
∉ ℤ"
  shows   "jacobi_theta_nome w q ≠ 0"
```

```
proof -
  have "jacobi_theta_nome (complex_of_real w) (complex_of_real q) ≠ 0"
    by (rule jacobi_theta_nome_nonzero_complex) (use assms in auto)
  thus ?thesis
    by (simp add: jacobi_theta_nome_of_real)
qed

lemma jacobi_theta_nome_1_left_nonzero_complex:
  assumes "norm (q :: complex) < 1"
  shows   "jacobi_theta_nome 1 q ≠ 0"
  by (rule jacobi_theta_nome_nonzero_complex) (use assms in auto)

lemma jacobi_theta_nome_1_left_nonzero_real:
  assumes "|q::real| < 1"
  shows   "jacobi_theta_nome 1 q ≠ 0"
  by (metis assms jacobi_theta_nome_1_left_nonzero_complex jacobi_theta_nome_of_real

           norm_of_real of_real_0 of_real_1)

unbundle no_qpochhammer_inf_notation

end
```

5 The theta nullwert functions

```
theory Theta_Nullwert
  imports "Sum_Of_Squares_Count.Sum_Of_Squares_Count" Jacobi_Triple_Product
begin
```

The theta nullwert function (nullwert being German for "zero value") are the four functions $\vartheta_{xy}(z;\tau)$ with $z = 0$. However, they are very commonly denoted in terms of the nome instead, corresponding to $\vartheta_{xy}(w, q)$ with $w = 1$.

It is easy to see that $\vartheta_{11}(0;\tau) = \vartheta_{11}(1, q)$ is identically zero and therefore uninteresting. The remaining three functions $\vartheta_{10}(0, q)$, $\vartheta_{00}(0, q)$, and $\vartheta_{01}(0, q)$ are denoted $\vartheta_2(q)$, $\vartheta_3(q)$, and $\vartheta_4(q)$.

It is also not hard to see that in fact $\vartheta_4(q) = \vartheta_3(-q)$, but we still introduce separate notation for ϑ_4 since it is very commonly used in the literature.

```
lemma jacobi_theta_nome_11_1_left [simp]: "jacobi_theta_nome_11 1 q =
0"
  using jacobi_theta_nome_minus_same[of q] by (auto simp: jacobi_theta_nome_11_def)

abbreviation jacobi_theta_nw_10 :: "'a :: {real_normed_field, banach,
ln} ⇒ 'a" where
  "jacobi_theta_nw_10 q ≡ jacobi_theta_nome_10 1 q"

abbreviation jacobi_theta_nw_00 :: "'a :: {real_normed_field, banach}
⇒ 'a" where
```

```
  "jacobi_theta_nw_00 q ≡ jacobi_theta_nome_00 1 q"

abbreviation jacobi_theta_nw_01 :: "'a :: {real_normed_field, banach}
⇒ 'a" where
  "jacobi_theta_nw_01 q ≡ jacobi_theta_nome_01 1 q"


bundle jacobi_theta_nw_notation
begin
notation jacobi_theta_nw_10 ("ϑ₂")
notation jacobi_theta_nw_00 ("ϑ₃")
notation jacobi_theta_nw_01 ("ϑ₄")
end

bundle no_jacobi_theta_nw_notation
begin
no_notation jacobi_theta_nw_10 ("ϑ₂")
no_notation jacobi_theta_nw_00 ("ϑ₃")
no_notation jacobi_theta_nw_01 ("ϑ₄")
end

unbundle jacobi_theta_nw_notation


lemma jacobi_theta_nw_10_0 [simp]: "ϑ₂ 0 = 0"
  and jacobi_theta_nw_00_0 [simp]: "ϑ₃ 0 = 1"
  and jacobi_theta_nw_01_0 [simp]: "ϑ₄ 0 = 1"
  by simp_all

lemma jacobi_theta_nw_01_conv_00: "ϑ₄ q = ϑ₃ (-q)"
  by (simp add: jacobi_theta_nome_01_conv_00)


lemma jacobi_theta_nw_10_of_real:
       "y ≥ 0 ⟹ ϑ₂ (complex_of_real y) = of_real (ϑ₂ y)"
  and jacobi_theta_nw_00_of_real: "ϑ₃ (of_real x) = of_real (ϑ₃ x)"
  and jacobi_theta_nw_01_of_real: "ϑ₄ (of_real x) = of_real (ϑ₄ x)"
  by (simp_all flip: jacobi_theta_nome_10_complex_of_real jacobi_theta_nome_00_of_real
                     jacobi_theta_nome_01_of_real)

lemma jacobi_theta_nw_10_cnj:
       "(Im q = 0 ⟹ Re q ≥ 0) ⟹ ϑ₂ (cnj q) = cnj (ϑ₂ q)"
  and jacobi_theta_nw_00_cnj: "ϑ₃ (cnj q) = cnj (ϑ₃ q)"
  and jacobi_theta_nw_01_cnj: "ϑ₄ (cnj q) = cnj (ϑ₄ q)"
  by (simp_all flip: jacobi_theta_nome_10_cnj jacobi_theta_nome_00_cnj
jacobi_theta_nome_01_cnj)
```

The nullwerte have the following definitions as infinite sums:

$$\vartheta_2(q) = \sum_{-\infty}^{\infty} q^{(n+\frac{1}{2})^2}$$

$$\vartheta_3(q) = \sum_{-\infty}^{\infty} q^{n^2}$$

$$\vartheta_4(q) = \sum_{-\infty}^{\infty} (-1)^n q^{n^2}$$

```
lemma has_sum_jacobi_theta_nw_10_complex:
  assumes "norm (q :: complex) < 1"
  shows   "((λn. q powr ((of_int n + 1 / 2) ^ 2)) has_sum ϑ₂ q) UNIV"
proof (cases "q = 0")
  case [simp]: False
  show ?thesis
    using has_sum_jacobi_theta_nome_10[of q 1] assms by simp
qed auto

lemma has_sum_jacobi_theta_nw_10_real:
  assumes "q ∈ {0<..<1::real}"
  shows   "((λn. q powr ((of_int n + 1 / 2) ^ 2)) has_sum ϑ₂ q) UNIV"
proof (cases "q = 0")
  case [simp]: False
  show ?thesis
    using has_sum_jacobi_theta_nome_10[of q 1] assms by simp
qed auto

lemma has_sum_jacobi_theta_nw_00:
  assumes "norm q < 1"
  shows   "((λn. q powi (n ^ 2)) has_sum ϑ₃ q) UNIV"
  using has_sum_jacobi_theta_nome_00[of q 1] assms by simp

lemma has_sum_jacobi_theta_nw_01:
  assumes "norm q < 1"
  shows   "((λn. (-1) powi n * q powi (n ^ 2)) has_sum ϑ₄ q) UNIV"
  using has_sum_jacobi_theta_nome_01[of q 1] assms by simp
```

The theta nullwert functions do not vanish (except for $\vartheta_2(0) = 0$).

```
lemma jacobi_theta_00_nw_nonzero_complex: "norm (q::complex) < 1 ⟹
ϑ₃ q ≠ 0"
  by (simp add: jacobi_theta_nome_00_def jacobi_theta_nome_1_left_nonzero_complex)

lemma jacobi_theta_01_nw_nonzero_complex: "norm (q::complex) < 1 ⟹
ϑ₄ q ≠ 0"
  by (simp add: jacobi_theta_nw_01_conv_00 jacobi_theta_00_nw_nonzero_complex)
```

```
lemma jacobi_theta_10_nw_nonzero_complex:
  assumes "q ≠ 0" "norm (q::complex) < 1"
  shows   "ϑ₂ q ≠ 0"
  using jacobi_theta_nome_q_q_nonzero_complex[of q] assms
  by (auto simp: jacobi_theta_nome_10_def)

lemma jacobi_theta_00_nw_nonzero_real: "|q::real| < 1 ⟹ ϑ₃ q ≠ 0"
  and jacobi_theta_01_nw_nonzero_real: "|q::real| < 1 ⟹ ϑ₄ q ≠ 0"
  and jacobi_theta_10_nw_nonzero_real: "q ∈ {0..<1} ⟹ q ≠ 0 ⟹ ϑ₂
q ≠ 0"
  using jacobi_theta_00_nw_nonzero_complex[of "of_real q"]
        jacobi_theta_01_nw_nonzero_complex[of "of_real q"]
        jacobi_theta_10_nw_nonzero_complex[of "of_real q"]
  by (simp_all add: jacobi_theta_nw_00_of_real jacobi_theta_nw_01_of_real
                    jacobi_theta_nw_10_of_real)
```

5.1 The Maclaurin series of ϑ_3 and ϑ_4

It is easy to see from the above infinite sums that $\vartheta_3(q)$ and $\vartheta_4(q)$ have the Maclaurin series

$$1 + 2\sum_{n=1}^{\infty}[\exists i.\ n = i^2]c^n q^n$$

for $c = 1$ and $c = -1$, respectively.

In other words, $\vartheta_3(q)$ is the generating function of the number $r_1(n)$ of ways to write an integer n as a square of an integer – 1 for $n = 0$, 2 if n is a non-zero perfect square, and 0 otherwise.

Consequently, $\vartheta_3(q)^k$ is the generating function of the number $r_k(n)$ of ways to write an integer n as a square of k integers. The function $r_k(n)$ is written as `count_sos k n` in Isabelle.

```
definition fps_jacobi_theta_nw :: "'a :: field ⇒ 'a fps" where
  "fps_jacobi_theta_nw c = Abs_fps (λn. if n = 0 then 1 else if is_square
n then 2 * c ^ n else 0)"

theorem fps_jacobi_theta_power_eq:
  "fps_jacobi_theta_nw c ^ k = Abs_fps (λn. of_nat (count_sos k n) * c
^ n)"
proof (induction k)
  case (Suc k)
  have "fps_jacobi_theta_nw (c::'a) ^ Suc k =
          fps_jacobi_theta_nw c * Abs_fps (λn. of_nat (count_sos k n)
* c ^ n)"
    by (simp add: Suc.IH mult.commute)
  also have "... = Abs_fps (λn. of_nat (count_sos (Suc k) n) * c ^ n)"
(is "?lhs = ?rhs")
  proof (rule fps_ext)
    fix n :: nat
```

```
    have "fps_nth (fps_jacobi_theta_nw (c::'a) * Abs_fps (λn. of_nat (count_sos
k n) * c ^ n)) n =
            (∑i=0..n. fps_nth (fps_jacobi_theta_nw c) i * of_nat (count_sos
k (n - i)) * c ^ (n - i))"
      by (simp add: fps_mult_nth sum_distrib_left sum_distrib_right algebra_simps)
    also have "... = of_nat (count_sos k n) * c ^ n +
                  (∑i∈{0<..n}. fps_nth (fps_jacobi_theta_nw c) i *

                                  of_nat (count_sos k (n - i)) * c ^ (n
- i))"
      (is "_ = _ + ?S")
      by (subst sum.head) (auto simp: fps_jacobi_theta_nw_def)
    also have "?S =  (∑i | i ∈ {0<..n} ∧ is_square i.
                     2 * of_nat (count_sos k (n - i)) * c ^ n)"
      by (rule sum.mono_neutral_cong_right) (auto simp: fps_jacobi_theta_nw_def
simp flip: power_add)
    also have "... = (∑i ∈ {1..floor_sqrt n}.
                     2 * of_nat (count_sos k (n - i ^ 2)) * c ^ n)"
      by (intro sum.reindex_bij_witness[of _ "λi. i ^ 2" floor_sqrt])
         (auto elim!: is_nth_powerE simp: le_floor_sqrt_iff)
    also have "of_nat (count_sos k n) * c ^ n + ... = of_nat (count_sos
(Suc k) n) * c ^ n"
      by (simp add: count_sos_Suc sum_distrib_left sum_distrib_right power_add
algebra_simps)
    finally show "fps_nth ?lhs n = fps_nth ?rhs n"
      by simp
  qed
  finally show ?case .
qed (auto intro!: fps_ext)

corollary fps_jacobi_theta_altdef:
  "fps_jacobi_theta_nw c = Abs_fps (λn. of_nat (count_sos 1 n) * c ^ n)"
  using fps_jacobi_theta_power_eq[of c 1] by simp

lemma norm_summable_fps_jacobi_theta:
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes "norm (c * q) < 1"
  shows   "summable (λn. norm (fps_nth (fps_jacobi_theta_nw c) n * q
^ n))"
proof (rule summable_comparison_test)
  show "summable (λn. 2 * norm (c * q) ^ n)"
    by (intro summable_mult summable_geometric) (use assms in auto)
  show "∃N. ∀n≥N. norm (norm (fps_nth (fps_jacobi_theta_nw c) n * q
^ n)) ≤ 2 * norm (c * q) ^ n"
    by (rule exI[of _ 0]) (auto simp: fps_jacobi_theta_nw_def norm_mult
norm_power power_mult_distrib)
qed

lemma summable_fps_jacobi_theta:
```

```
  fixes q :: "'a :: {real_normed_field, banach}"
  assumes "norm (c * q) < 1"
  shows   "summable (λn. fps_nth (fps_jacobi_theta_nw c) n * q ^ n)"
  using norm_summable_fps_jacobi_theta[OF assms] by (rule summable_norm_cancel)

lemma summable_ints_symmetric:
  fixes f :: "int ⇒ 'a :: {real_normed_vector, banach}"
  assumes "summable (λn. norm (f (int n)))"
  assumes "⋀n. f (-n) = f n"
  shows   "f abs_summable_on UNIV" and "summable (λn. norm ((if n = 0
then 1 else 2) *R f (int n)))"
proof -
  show "summable (λn. norm ((if n = 0 then 1 else 2) *R f (int n)))"
  proof (rule summable_comparison_test)
    show "summable (λn. 2 * norm (f n))"
      by (intro summable_mult assms)
  qed (auto intro!: exI[of _ 0])
next
  have "(λn. f (int n)) abs_summable_on UNIV"
    using assms by (subst summable_on_UNIV_nonneg_real_iff) auto
  also have "?this ⟷ f abs_summable_on {0..}"
    by (rule summable_on_reindex_bij_witness[of _ nat int]) auto
  finally have 1: "f abs_summable_on {0<..}"
    by (rule summable_on_subset) auto
  also have "?this ⟷ f abs_summable_on {..<0}"
    by (rule summable_on_reindex_bij_witness[of _ uminus uminus]) (use
assms(2) in auto)
  finally have "f abs_summable_on ({..<0} ∪ {0} ∪ {0<..})"
    by (intro summable_on_Un_disjoint 1) auto
  also have "({..<(0::int)} ∪ {0} ∪ {0<..}) = UNIV"
    by auto
  finally show "f abs_summable_on UNIV" .
qed

lemma has_sum_ints_symmetric_iff:
  fixes f :: "int ⇒ 'a :: {banach, real_normed_vector}"
  assumes "⋀n. f (-n) = f n"
  shows   "(f has_sum S) UNIV ⟷ ((λn. (if n = 0 then 1 else 2) *R
f (int n)) has_sum S) UNIV"
proof
  assume *: "((λn. (if n = 0 then 1 else 2) *R f (int n)) has_sum S)
UNIV"
  have "((λn. (if n = 0 then 1 else 2) *R f (int n)) has_sum (S - f 0))
(UNIV - {0})"
    using has_sum_Diff[OF * has_sum_finite[of "{0}"]] by simp
  also have "?this ⟷ ((λn. (if n = 0 then 1 else 2) *R f n) has_sum
(S - f 0)) {0<..}"
    by (intro has_sum_reindex_bij_witness[of _ nat int]) auto
  finally have "((λn. (if n = 0 then 1 else 2) *R f n) has_sum S - f 0)
```

```
{0<..}" .
  also have "?this ⟷ ((λn. 2 *R f n) has_sum (S - f 0)) {0<..}"
    by (intro has_sum_cong) auto
  also have "... ⟷ (f has_sum (S - f 0) /R 2) {0<..}"
    by (rule has_sum_scaleR_iff) auto
  finally have 1: "(f has_sum (S - f 0) /R 2) {0<..}" .

  have "(f has_sum (S - f 0) /R 2) {0<..} ⟷ (f has_sum (S - f 0) /R
2) {..<0}"
    by (intro has_sum_reindex_bij_witness[of _ uminus uminus]) (use assms
in auto)
  with 1 have 2: "(f has_sum (S - f 0) /R 2) {..<0}"
    by simp

  have "(f has_sum ((S - f 0) /R 2 + (S - f 0) /R 2 + f 0)) ({..<0} ∪
{0<..} ∪ {0})"
    by (intro has_sum_Un_disjoint 1 2) (auto simp: has_sum_finite_iff)
  also have "{..<0} ∪ {0<..} ∪ {0::int} = UNIV"
    by auto
  also have "(S - f 0) /R 2 + (S - f 0) /R 2 + f 0 = S"
    by (simp flip: mult_2 scaleR_2)
  finally show "(f has_sum S) UNIV" .
next
  assume *: "(f has_sum S) UNIV"
  define S' where "S' = (∑∞n∈{0<..}. f n)"
  have "f summable_on {0<..}"
    by (rule summable_on_subset_banach[of _ UNIV]) (use * in ‹auto dest:
has_sum_imp_summable›)
  hence 1: "(f has_sum S') {0<..}"
    unfolding S'_def by (rule has_sum_infsum)

  have "(f has_sum S') {0<..} ⟷ (f has_sum S') {..<0}"
    by (rule has_sum_reindex_bij_witness[of _ uminus uminus]) (use assms
in auto)
  with 1 have 2: "(f has_sum S') {..<0}"
    by simp

  have "(f has_sum (S' + S' + f 0)) ({..<0} ∪ {0<..} ∪ {0})"
    by (intro has_sum_Un_disjoint 1 2) (auto simp: has_sum_finite_iff)
  also have "({..<0} ∪ {0<..} ∪ {0::int}) = UNIV"
    by auto
  also have "S' + S' + f 0 = 2 *R S' + f 0"
    by (simp add: algebra_simps flip: scaleR_2)
  finally have 3: "S = 2 *R S' + f 0"
    using * has_sum_unique by blast

  have "((λn. 2 *R f n) has_sum (2 *R S')) {0<..}"
    by (intro has_sum_scaleR 1)
  also have "?this ⟷ ((λn. (if n = 0 then 1 else 2) *R f n) has_sum
```

```
(2 *R S')) {0<..}"
    by (intro has_sum_cong) auto
  finally have "((λn. (if n = 0 then 1 else 2) *R f n) has_sum (f 0 + 2
*R S')) ({0} ∪ {0<..})"
    by (intro has_sum_Un_disjoint) (auto simp: has_sum_finite_iff)
  also have "?this ⟷ ((λn. (if n = 0 then 1 else 2) *R f (int n)) has_sum
(f 0 + 2 *R S')) UNIV"
    by (rule has_sum_reindex_bij_witness[of _ int nat]) auto
  finally show "((λn. (if n = 0 then 1 else 2) *R f (int n)) has_sum S)
UNIV"
    using 3 by (simp add: add.commute)
qed

lemma sums_jacobi_theta_nw_00:
  assumes "norm q < 1"
  shows   "(λn. fps_nth (fps_jacobi_theta_nw 1) n * q ^ n) sums ϑ3 q"
proof -
  define S where "S = (∑n. if is_square n ∧ n > 0 then q ^ n else 0)"
  have "((λn. (if n = 0 then 1 else 2) *R q powi (int n ^ 2)) has_sum
(ϑ3 q)) UNIV"
  proof (subst has_sum_ints_symmetric_iff [symmetric])
    show "((λn. q powi n^2) has_sum ϑ3 q) UNIV"
      by (rule has_sum_jacobi_theta_nw_00) fact
  qed auto
  also have "?this ⟷ ((λn. fps_nth (fps_jacobi_theta_nw 1) n * q ^
n) has_sum
                          ϑ3 q) {n. is_square n}"
    by (rule has_sum_reindex_bij_witness[of _ floor_sqrt "λi. i ^ 2"])
       (auto simp: fps_jacobi_theta_nw_def power_int_def scaleR_conv_of_real
nat_power_eq
             elim!: is_nth_powerE)
  also have "... ⟷ ((λn. fps_nth (fps_jacobi_theta_nw 1) n * q ^ n)
has_sum
                          ϑ3 q) UNIV"
    by (intro has_sum_cong_neutral) (auto simp: fps_jacobi_theta_nw_def
intro: Nat.gr0I)
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

lemma sums_jacobi_theta_nw_01:
  assumes "norm q < 1"
  shows   "(λn. fps_nth (fps_jacobi_theta_nw (-1)) n * q ^ n) sums ϑ4
q"
proof -
  have "(λn. fps_nth (fps_jacobi_theta_nw 1) n * (-q) ^ n) sums ϑ3 (-q)"
    by (rule sums_jacobi_theta_nw_00) (use assms in auto)
  also have "(λn. fps_nth (fps_jacobi_theta_nw 1) n * (-q) ^ n) =
             (λn. fps_nth (fps_jacobi_theta_nw (-1)) n * q ^ n)"
```

```
    by (auto simp: fun_eq_iff fps_jacobi_theta_nw_def power_minus')
  also have "ϑ3 (-q) = ϑ4 q"
    by (simp add: jacobi_theta_nw_01_conv_00)
  finally show ?thesis .
qed

lemma has_fps_expansion_jacobi_theta_3 [fps_expansion_intros]:
  "ϑ3 has_fps_expansion fps_jacobi_theta_nw 1"
proof (rule has_fps_expansionI)
  have "eventually (λq. q ∈ ball 0 1) (nhds (0 :: 'a))"
    by (rule eventually_nhds_in_open) auto
  thus "eventually (λq. (λn. fps_nth (fps_jacobi_theta_nw 1) n * q ^ n
:: 'a) sums ϑ3 q) (nhds 0)"
    by eventually_elim (rule sums_jacobi_theta_nw_00, auto)
qed

lemma has_fps_expansion_jacobi_theta_4 [fps_expansion_intros]:
  "ϑ4 has_fps_expansion fps_jacobi_theta_nw (-1)"
proof (rule has_fps_expansionI)
  have "eventually (λq. q ∈ ball 0 1) (nhds (0 :: 'a))"
    by (rule eventually_nhds_in_open) auto
  thus "eventually (λq. (λn. fps_nth (fps_jacobi_theta_nw (-1)) n * q
^ n :: 'a) sums ϑ4 q) (nhds 0)"
    by eventually_elim (rule sums_jacobi_theta_nw_01, auto)
qed

lemma fps_conv_radius_jacobi_theta_nw [simp]:
  fixes c :: "'a :: {banach, real_normed_field}"
  shows "fps_conv_radius (fps_jacobi_theta_nw c) = 1 / ereal (norm c)"
proof -
  have "fps_conv_radius (fps_jacobi_theta_nw c) =
          inverse (limsup (λn. ereal (root n (norm (fps_nth (fps_jacobi_theta_nw
c) n)))))"
    by (simp add: fps_conv_radius_def conv_radius_def)
  also have "limsup (λn. ereal (root n (norm (fps_nth (fps_jacobi_theta_nw
c) n)))) = norm c"
    (is "?lhs = _")
  proof (rule antisym)
    have "?lhs ≤ limsup (λn. root n 2 * norm c)"
      by (intro Limsup_mono always_eventually)
         (auto simp: fps_jacobi_theta_nw_def norm_power real_root_mult
real_root_power)
    also have "(λn. ereal (root n 2 * norm c)) ⟶ ereal (1 * norm
c)"
      by (intro tendsto_intros LIMSEQ_root_const) auto
    hence "limsup (λn. root n 2 * norm c) = ereal (1 * norm c)"
      by (intro lim_imp_Limsup) auto
    finally show "?lhs ≤ norm c"
      by simp
```

```
  next
    have "limsup ((λn. ereal (root n (norm (fps_nth (fps_jacobi_theta_nw
c) n)))) ∘ (λn. n ^ 2)) ≤ ?lhs"
      by (rule limsup_subseq_mono) (auto intro!: strict_monoI power_strict_mono)
    also have "limsup ((λn. ereal (root n (norm (fps_nth (fps_jacobi_theta_nw
c) n)))) ∘ (λn. n ^ 2)) =
                limsup ((λn. ereal (root (n^2) 2 * norm c)))"
      by (rule Limsup_eq, rule eventually_mono[OF eventually_gt_at_top[of
0]])
         (auto simp: o_def fps_jacobi_theta_nw_def norm_power real_root_mult
real_root_power)
    also have "(λn. ereal (root (n^2) 2 * norm c)) ⟶ ereal (1 * norm
c)"
      by (intro tendsto_intros filterlim_compose[OF LIMSEQ_root_const]
                filterlim_subseq[of "λn. n ^ 2"] strict_monoI power_strict_mono)
auto
    hence "limsup (λn. ereal (root (n^2) 2 * norm c)) = ereal (1 * norm
c)"
      by (intro lim_imp_Limsup) auto
    finally show "norm c ≤ ?lhs"
      by simp
  qed
  finally show ?thesis
    by (simp add: divide_ereal_def)
qed
```

Recall that $\vartheta_2(q) = q^{1/4}\vartheta(q,q)$. Since the factor $q^{1/4}$ has a branch cut, it is somewhat unpleasant to deal with and we will concentrate only on the factor $\vartheta(q,q)$ for now. This is a holomorphic function on the unit disc except for a removable singularity at $q = 0$.

For $q \neq 0$ and $|q| < 1$, $\vartheta(q,q)$ has following the power series expansion:

$$\vartheta(q,q) = \sum_{n=-\infty}^{\infty} q^{n(n+1)} = \sum_{n=0}^{\infty} 2q^{n(n+1)}$$

Note that $n(n+1)$ is twice the triangular number $n(n+1)/2$, so we can also see this as a series expansion in terms of powers of q^2.

```
lemma has_sum_jacobi_theta_nw_10_aux:
  assumes q: "norm q < 1" "q ≠ 0"
  shows   "((λn. 2 * q ^ (n*(n+1))) has_sum jacobi_theta_nome q q) UNIV"
proof -
  define S where "S = jacobi_theta_nome q q"
  have 1: "((λn. q powi (n*(n+1))) has_sum S) UNIV"
    using has_sum_jacobi_theta_nome[of q q]
    using q by (simp add: algebra_simps power2_eq_square power_int_add
S_def)
  have summable: "(λn. q powi (n * (n + 1))) summable_on I" for I
    by (rule summable_on_subset_banach, rule has_sum_imp_summable[OF 1])
auto
```

```
  define S1 where "S1 = (∑∞n∈{0..}. q powi (n*(n+1)))"
  have S1: "((λn. q powi (n*(n+1))) has_sum S1) {0..}"
    unfolding S1_def by (rule has_sum_infsum[OF summable])
  have "((λn. q powi (n*(n+1))) has_sum S1) {0..} ⟷
        ((λn. q powi (n*(n+1))) has_sum S1) {..<0}"
    by (intro has_sum_reindex_bij_witness[of _ "λn. -n-1" "λn. -n-1"])
       (auto simp: algebra_simps)
  with S1 have S1': "((λn. q powi (n*(n+1))) has_sum S1) {..<0}"
    by simp
  have "((λn. q powi (n*(n+1))) has_sum (S1 + S1)) ({..<0} ∪ {0..})"
    by (intro has_sum_Un_disjoint S1 S1') auto
  also have "{..<(0::int)} ∪ {0..} = UNIV"
    by auto
  finally have 2: "((λn. q powi (n*(n+1))) has_sum (2 * S1)) UNIV"
    by simp

  from this and 1 have 3: "2 * S1 = S"
    using has_sum_unique by blast

  have "((λn. q powi (n*(n+1))) has_sum S1) {0..} ⟷ ((λn. q ^ (n*(n+1)))
has_sum S1) UNIV"
    by (rule has_sum_reindex_bij_witness[of _ int nat])
       (auto simp: power_int_def algebra_simps power_add nat_add_distrib
nat_mult_distrib)
  with S1 show "((λn. 2 * q ^ (n*(n+1))) has_sum S) UNIV"
    unfolding 3 [symmetric] by (intro has_sum_cmult_right) auto
qed

lemma sums_jacobi_theta_nw_10_aux:
  assumes q: "norm q < 1" "q ≠ 0"
  shows   "(λn. if ∃i. n = i*(i+1) then 2 * q ^ n else 0) sums jacobi_theta_nome
q q"
proof -
  have inj: "inj (λi::nat. i * (i + 1))"
    by (rule strict_mono_imp_inj_on) (auto simp: strict_mono_Suc_iff)
  have "((λn. 2 * q ^ (n*(n+1))) has_sum jacobi_theta_nome q q) UNIV"
    by (rule has_sum_jacobi_theta_nw_10_aux) fact+
  also have "?this ⟷ ((λn. 2 * q ^ n) has_sum jacobi_theta_nome q q)
(range (λi. i*(i+1)))"
    by (subst has_sum_reindex[OF inj]) (auto simp: o_def)
  also have "... ⟷ ((λn. if ∃i. n = i*(i+1) then 2 * q ^ n else 0)
has_sum jacobi_theta_nome q q) UNIV"
    by (rule has_sum_cong_neutral) auto
  finally show ?thesis
    by (rule has_sum_imp_sums)
qed

definition fps_jacobi_theta_nw_10 :: "'a :: field fps" where
```

```
  "fps_jacobi_theta_nw_10 = Abs_fps (λn. if ∃i. n = i*(i+1) then 2 else
0)"

lemma fps_conv_radius_jacobi_theta_2 [simp]: "fps_conv_radius fps_jacobi_theta_nw_10
= 1"
proof -
  have "fps_conv_radius (fps_jacobi_theta_nw_10 :: 'a fps) =
          inverse (limsup (λn. ereal (root n (norm (fps_nth fps_jacobi_theta_nw_10
n :: 'a)))))"
    unfolding fps_conv_radius_def conv_radius_def ..
  also have "limsup (λn. ereal (root n (norm (fps_nth fps_jacobi_theta_nw_10
n :: 'a)))) = 1"
    (is "?lhs = _")
  proof (rule antisym)
    have "?lhs ≤ limsup (λn. root n 2)"
      by (intro Limsup_mono always_eventually)
         (auto simp: fps_jacobi_theta_nw_10_def norm_power real_root_ge_zero)
    also have "(λn. ereal (root n 2)) ⟶ ereal 1"
      by (intro tendsto_intros LIMSEQ_root_const) auto
    hence "limsup (λn. root n 2) = ereal 1"
      by (intro lim_imp_Limsup) auto
    finally show "?lhs ≤ 1"
      by (simp add: one_ereal_def)
  next
    define h where "h = (λn::nat. n * (n + 1))"
    have h: "strict_mono h"
      by (rule strict_monoI_Suc) (auto simp: algebra_simps h_def)
    have "limsup ((λn. ereal (root n (norm (fps_nth fps_jacobi_theta_nw_10
n :: 'a)))) ∘ h) ≤ ?lhs"
      using h by (rule limsup_subseq_mono)
    also have "limsup ((λn. ereal (root n (norm (fps_nth fps_jacobi_theta_nw_10
n :: 'a)))) ∘ h) =
                 limsup ((λn. ereal (root (h n) 2)))"
      by (rule Limsup_eq, rule eventually_mono[OF eventually_gt_at_top[of
0]])
         (auto simp: o_def fps_jacobi_theta_nw_10_def h_def algebra_simps)
    also have "(λn. ereal (root (h n) 2)) ⟶ ereal 1"
      by (intro tendsto_intros filterlim_compose[OF LIMSEQ_root_const]
                filterlim_subseq[of h] h) auto
    hence "limsup (λn. ereal (root (h n) 2)) = ereal 1"
      by (intro lim_imp_Limsup) auto
    finally show "1 ≤ ?lhs"
      by (simp add: one_ereal_def)
  qed
  finally show ?thesis
    by simp
qed

lemma has_laurent_expansion_jacobi_theta_2 [laurent_expansion_intros]:
```

```
  "(λq. jacobi_theta_nome q q) has_laurent_expansion fps_to_fls fps_jacobi_theta_nw_10"
  unfolding has_laurent_expansion_def
proof safe
  show "fls_conv_radius (fps_to_fls fps_jacobi_theta_nw_10 :: complex
fls) > 0"
    unfolding fls_conv_radius_fps_to_fls by simp
next
  have "eventually (λq. q ∈ ball 0 1 - {0}) (at (0 :: complex))"
    by (rule eventually_at_in_open) auto
  thus "eventually (λq. eval_fls (fps_to_fls fps_jacobi_theta_nw_10) q
=
        jacobi_theta_nome q q) (at (0::complex))"
  proof eventually_elim
    case q: (elim q)
    have "eval_fls (fps_to_fls fps_jacobi_theta_nw_10) q = eval_fps fps_jacobi_theta_nw_10
q"
      by (subst eval_fps_to_fls) (use q in auto)
    also have "eval_fps fps_jacobi_theta_nw_10 q = (∑n. fps_nth fps_jacobi_theta_nw_10
n * q ^ n)"
      by (simp add: eval_fps_def)
    also have "(λn. fps_nth fps_jacobi_theta_nw_10 n * q ^ n) =
               (λn. if ∃i. n = i*(i+1) then 2 * q ^ n else 0)"
      by (auto simp: fun_eq_iff fps_jacobi_theta_nw_10_def)
    also have "(∑n. ... n) = jacobi_theta_nome q q"
      using sums_jacobi_theta_nw_10_aux[of q] q by (simp add: sums_iff)
    finally show ?case .
  qed
qed
```

For $\vartheta(q,q)^2$, we can find the following expansion into a double sum:

$$\vartheta(q,q)^2 = \sum_{n=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} q^{i(i+1)+j(j+1)}$$

```
lemma has_sum_jacobi_theta_nw_10_aux_square:
  fixes q :: complex
  assumes q: "norm q < 1" "q ≠ 0"
  shows "((λ(i,j). q powi (i*(i+1) + j*(j+1))) has_sum jacobi_theta_nome
q q ^ 2) UNIV"
proof -
  define S where "S = jacobi_theta_nome q q"
  have 1: "((λn. q powi (n*(n+1))) has_sum S) UNIV"
    using has_sum_jacobi_theta_nome[of q q]
    using q by (simp add: algebra_simps power2_eq_square power_int_add
S_def)
  have summable: "(λn. q powi (n * (n + 1))) summable_on I" for I
    by (rule summable_on_subset_banach, rule has_sum_imp_summable[OF 1])
auto
```

```
  define S' where "S' = jacobi_theta_nome (norm q) (norm q)"
  have 2: "((λn. norm q powi (n*(n+1))) has_sum S') UNIV"
    using has_sum_jacobi_theta_nome[of "norm q" "norm q"]
    using q by (simp add: algebra_simps power2_eq_square power_int_add
S'_def)

  have "((λ(i,j). q powi (i*(i+1) + j*(j+1))) has_sum S^2) (UNIV × UNIV)"
  proof (rule has_sum_SigmaI; (unfold prod.case)?)
    show "((λi. S * q powi (i*(i+1))) has_sum S^2) UNIV"
      unfolding power2_eq_square by (intro has_sum_cmult_right 1)
  next
    fix i :: int
    show "((λj. q powi (i * (i + 1) + j * (j + 1))) has_sum S * q powi
(i * (i + 1))) UNIV"
      using has_sum_cmult_left[OF 1, of "q powi (i * (i + 1))"] q
      by (simp add: power_int_add mult_ac)
  next
    have "(λij. norm (case ij of (i,j) ⇒ q powi (i * (i + 1) + j * (j
+ 1)))) summable_on UNIV × UNIV"
    proof (rule summable_on_SigmaI; (unfold prod.case)?)
      show "(λj. S' * norm q powi (j * (j + 1))) summable_on UNIV"
        using has_sum_imp_summable[OF 2] by (intro summable_on_cmult_right)
    next
      fix i :: int
      show "((λj. norm (q powi (i*(i+1) + j*(j+1)))) has_sum (S' * norm
q powi (i*(i+1)))) UNIV"
        using has_sum_cmult_left[OF 2, of "norm q powi (i*(i+1))"] q
        by (simp add: norm_power_int norm_mult power_int_add mult_ac)
    qed auto
    thus "(λ(i, j). q powi (i * (i + 1) + j * (j + 1))) summable_on UNIV
× UNIV"
      by (rule abs_summable_summable)
  qed
  thus ?thesis
    by (simp add: S_def)
qed
```

With some creative reindexing, we find the following power series expansion:

$$q\vartheta(q^2,q^2)^2 = \sum_{n=0}^{\infty} r_2(2n+1)q^{2n+1}$$

```
lemma sums_q_times_jacobi_theta_nw_10_aux_square_square:
  fixes q :: complex
  assumes q: "q ≠ 0" "norm q < 1"
  shows "(λn. (if odd n then of_nat (count_sos 2 n) else 0) * q ^ n)
sums
          (q * jacobi_theta_nome (q^2) (q^2) ^ 2)"
proof -
```

```
  define IJ where "IJ = (λn. {(i, j). i^2 + j^2 = int n})"
  have [simp, intro]: "finite (IJ n)" for n
    using bij_betw_finite[OF bij_betw_sos_decomps_2[of n]] by (simp add:
IJ_def)

  have aux: "1 + x ≠ y" if "even x" "even y" for x y :: int
    using that by presburger

  define S where "S = q * jacobi_theta_nome (q^2) (q^2) ^ 2"
  have "((λ(i,j). (q^2) powi (i*(i+1) + j*(j+1))) has_sum jacobi_theta_nome
(q^2) (q^2) ^ 2) UNIV"
    by (intro has_sum_jacobi_theta_nw_10_aux_square)
       (use q in ‹auto simp: norm_power power_less_one_iff›)
  hence "((λ(i,j). q * (q^2) powi (i*(i+1) + j*(j+1))) has_sum S) (UNIV
× UNIV)"
    unfolding S_def case_prod_unfold by (intro has_sum_cmult_right) auto
  also have "((λ(i,j). q * (q^2) powi (i*(i+1) + j*(j+1)))) =
             ((λ(i,j). q powi (1 + 2 * (i*(i+1) + j*(j+1)))))" using q
    by (auto simp: power_int_add power2_eq_square fun_eq_iff power_int_mult_distrib
algebra_simps)
       (auto simp flip: power_int_add simp: algebra_simps)?
  also have "(... has_sum S) (UNIV × UNIV) ⟷
             ((λ(i,j). q powi (i ^ 2 + j ^ 2)) has_sum S) {(i,j). odd
(i^2+j^2)}"
    by (rule has_sum_reindex_bij_witness[of _
             "λ(i,j). ((j+i-1) div 2, (j-i-1) div 2)" "λ(i,j). (i-j,
i+j+1)"])
       (auto elim!: evenE oddE simp: algebra_simps power2_eq_square aux
             intro!: arg_cong[of _ _ "λa. q powi a"])
  also have "... ⟷ ((λ(n,(i,j)). q powi n) has_sum S) (SIGMA n:{n. odd
n}. IJ n)"
  proof (rule has_sum_reindex_bij_witness[of _ snd "λ(i,j). (nat (i^2+j^2),
(i,j))"])
    fix nij assume nij: "nij ∈ Sigma {n. odd n} IJ"
    obtain n i j where [simp]: "nij = (n, (i, j))"
      using prod_cases3 by blast
    from nij have n: "odd n" and ij: "i ^ 2 + j ^ 2 = int n"
      by (auto simp: IJ_def)
    have "odd (int n)"
      using n by simp
    also have "int n = i ^ 2 + j ^ 2"
      by (rule ij [symmetric])
    finally show "snd nij ∈ {(i, j). odd (i^2 + j^2)}"
      by auto
  qed (auto simp: IJ_def even_nat_iff)
  finally have *: "((λ(n,(i,j)). q powi n) has_sum S) (SIGMA n:{n. odd
n}. IJ n)" .
  have "((λn. count_sos 2 n * q ^ n) has_sum S) {n. odd n}"
  proof (rule has_sum_SigmaD[OF *]; unfold prod.case)
```

```
    fix n :: nat assume n: "n ∈ {n. odd n}"
    have "count_sos 2 n = card (IJ n)"
      by (simp add: IJ_def count_sos_2)
    thus "((λ(i,j). q powi int n) has_sum complex_of_nat (count_sos 2
n) * q ^ n) (IJ n)"
      using q by (simp add: has_sum_finite_iff)
  qed
  also have "?this ⟷ ((λn. (if odd n then of_nat (count_sos 2 n) else
0) * q ^ n) has_sum S) UNIV"
    by (intro has_sum_cong_neutral) auto
  finally show "(λn. (λn. if odd n then of_nat (count_sos 2 n) else 0)
n * q ^ n) sums S"
    by (rule has_sum_imp_sums)
qed

lemma has_laurent_expansion_q_times_jacobi_theta_nw_10_aux_square_square:
  defines "F ≡ Abs_fps (λn. if odd n then of_nat (count_sos 2 n) else
0)"
  shows "(λq. q * jacobi_theta_nome (q^2) (q^2) ^ 2) has_laurent_expansion
fps_to_fls F"
  unfolding has_laurent_expansion_def
proof
  have "0 < norm (1/2::complex)"
    by simp
  also have "fls_conv_radius (fps_to_fls F) ≥ norm (1 / 2 :: complex)"
    unfolding fls_conv_radius_fps_to_fls fps_conv_radius_def
    by (rule conv_radius_geI) (use sums_q_times_jacobi_theta_nw_10_aux_square_square[of
"1/2"]
        in ‹auto simp: sums_iff F_def›)
  finally show "fls_conv_radius (fps_to_fls F) > 0"
    by - (simp_all add: zero_ereal_def)
next
  have "eventually (λq. q ∈ ball 0 1 - {0}) (at (0::complex))"
    by (rule eventually_at_in_open) auto
  thus "∀_F q in at 0. eval_fls (fps_to_fls F) q = q * (jacobi_theta_nome
(q^2) (q^2))^2"
  proof eventually_elim
    case q: (elim q)
    have "(λn. fps_nth F n * q ^ n) sums (q * (jacobi_theta_nome (q^2)
(q^2))^2)"
      using sums_q_times_jacobi_theta_nw_10_aux_square_square[of q] q
by (simp add: F_def)
    thus "eval_fls (fps_to_fls F) q = q * (jacobi_theta_nome (q^2) (q^2))^2"
      using eval_fls_eq[of 0 "fps_to_fls F" q "q * (jacobi_theta_nome
(q^2) (q^2))^2"]
      by (simp add: fls_subdegree_fls_to_fps_gt0)
  qed
qed
```

5.2 Identities

Lastly, we derive a variety of identities between the different theta functions.

```
theorem jacobi_theta_nw_00_plus_01_complex: "ϑ₃ q + ϑ₄ q = 2 * ϑ₃ (q
^ 4 :: complex)"
proof (cases "norm q < 1")
  case q: True
  define f :: "complex ⇒ complex" where "f = (λq. ϑ₃ q + ϑ₄ q - 2 *
ϑ₃ (q ^ 4))"
  define F :: "complex fps"
    where "F = fps_jacobi_theta_nw 1 + fps_jacobi_theta_nw (-1) -
               2 * (fps_jacobi_theta_nw 1 oo fps_X ^ 4)"
  have [simp]: "is_square (4 :: nat)"
    unfolding is_nth_power_def by (rule exI[of _ 2]) auto
  have "fps_jacobi_theta_nw 1 + fps_jacobi_theta_nw (-1 :: complex) =
        2 * Abs_fps (λn. if n = 0 then 1 else if even n ∧ is_square n
then 2 else 0)"
    by (intro fps_ext) (auto simp: fps_jacobi_theta_nw_def intro: Nat.gr0I)
  also have "Abs_fps (λn. if n = 0 then 1 else if even n ∧ is_square n
then 2 else 0) =
               fps_compose (fps_jacobi_theta_nw 1) (fps_X ^ 4)"
    by (auto simp: fps_eq_iff fps_jacobi_theta_nw_def fps_nth_compose_X_power
is_square_mult2_nat_iff
                   is_nth_power_mult_cancel_left elim!: dvdE)
  finally have "F = 0"
    by (simp add: F_def)

  have "f q = 0"
  proof (rule has_fps_expansion_0_analytic_continuation[of f])
    have "(λq. ϑ₃ q + ϑ₄ q - 2 * (ϑ₃ ∘ (λq. q ^ 4)) q) has_fps_expansion
F"
      unfolding F_def by (intro fps_expansion_intros has_fps_expansion_compose)
auto
    also have "F = 0"
      by fact
    finally show "f has_fps_expansion 0"
      by (simp add: f_def)
  next
    show "f holomorphic_on ball 0 1"
      unfolding f_def by (auto intro!: holomorphic_intros simp: norm_power
power_less_one_iff)
  qed (use q in auto)
  thus ?thesis
    by (simp add: f_def)
qed (auto simp: norm_power power_less_one_iff)

lemma jacobi_theta_nw_00_plus_01_real: "ϑ₃ q + ϑ₄ q = 2 * ϑ₃ (q ^ 4 ::
real)"
  by (subst of_real_eq_iff [where ?'a = complex, symmetric],
```

```
      unfold of_real_add of_real_mult of_real_diff)
   (use jacobi_theta_nw_00_plus_01_complex[of q]
    in ‹simp_all flip: jacobi_theta_nome_00_of_real jacobi_theta_nome_01_of_real›)


theorem jacobi_theta_nw_00_plus_01_square_complex:
  "ϑ₃ q ^ 2 + ϑ₄ q ^ 2 = 2 * ϑ₃ (q ^ 2 :: complex) ^ 2"
proof (cases "norm q < 1")
  case q: True
  define F :: "complex fps"
    where "F = 2 * fps_compose (fps_jacobi_theta_nw 1 ^ 2) (fps_X ^ 2)
-
              fps_jacobi_theta_nw 1 ^ 2 - fps_jacobi_theta_nw (-1) ^
2"
  have "(λz. 2 * ((λz. ϑ₃ z ^ 2) ∘ (λz. z ^ 2)) z - ϑ₃ z ^ 2 - ϑ₄ z ^
2) has_fps_expansion F"
    unfolding F_def by (intro fps_expansion_intros has_fps_expansion_compose)
auto
  also have "F = 0"
  proof -
    have "2 * fps_compose (Abs_fps (λn. of_nat (count_sos 2 n) :: complex))
(fps_X ^ 2) =
           Abs_fps (λn. if even n then of_nat (2 * count_sos 2 n) else
0)"
      by (auto elim!: evenE simp: fps_nth_compose_X_power fps_eq_iff count_sos_2_double)
    also have "... = fps_jacobi_theta_nw 1 ^ 2 + fps_jacobi_theta_nw (-1)
^ 2"
      by (auto simp: fps_eq_iff fps_jacobi_theta_power_eq)
    also have "Abs_fps (λn. of_nat (count_sos 2 n)) = fps_jacobi_theta_nw
1 ^ 2"
      by (simp add: fps_jacobi_theta_power_eq)
    finally show "F = 0"
      by (simp add: algebra_simps F_def)
  qed
  finally have "(λz::complex. 2 * ϑ₃ (z²) ^ 2 - ϑ₃ z ^ 2 - ϑ₄ z ^ 2) has_fps_expansion
0"
    by simp
  hence "2 * (ϑ₃ (q²))² - (ϑ₃ q)² - (ϑ₄ q)² = 0"
    by (rule has_fps_expansion_0_analytic_continuation[where A = "ball
0 1"])
       (use q in ‹auto intro!: holomorphic_intros simp: norm_power power_less_one_iff›)
  thus ?thesis
    by (simp add: algebra_simps)
qed (auto simp: norm_power power_less_one_iff)

corollary midpoint_jacobi_theta_nw_00_01_square_complex:
  "midpoint (ϑ₃ q ^ 2) (ϑ₄ q ^ 2) = ϑ₃ (q ^ 2 :: complex) ^ 2"
  using jacobi_theta_nw_00_plus_01_square_complex[of q] by (simp add:
midpoint_def)
```

```
lemma jacobi_theta_nw_00_plus_01_square_real: "ϑ3 q ^ 2 + ϑ4 q ^ 2 =
2 * ϑ3 (q ^ 2 :: real) ^ 2"
  by (subst of_real_eq_iff [where ?'a = complex, symmetric],
        unfold of_real_add of_real_mult of_real_diff)
     (use jacobi_theta_nw_00_plus_01_square_complex[of q]
      in ‹simp_all flip: jacobi_theta_nome_00_of_real jacobi_theta_nome_01_of_real›)

theorem jacobi_theta_nw_00_times_01_complex: "ϑ3 q * ϑ4 q = (ϑ4 (q ^
2) ^ 2 :: complex)"
proof -
  have "2 * ϑ3 q * ϑ4 q = (ϑ3 q + ϑ4 q) ^ 2 - (ϑ3 q ^ 2 + ϑ4 q ^ 2)"
    by Groebner_Basis.algebra
  also have "... = 2 * (2 * ϑ3 (q ^ 4) ^ 2 - ϑ3 (q²) ^ 2)"
    unfolding jacobi_theta_nw_00_plus_01_complex jacobi_theta_nw_00_plus_01_square_complex
    by Groebner_Basis.algebra
  also have "2 * ϑ3 (q ^ 4) ^ 2 - ϑ3 (q²) ^ 2 = ϑ4 (q²) ^ 2"
    using jacobi_theta_nw_00_plus_01_square_complex[of "q ^ 2"]
    by (simp add: algebra_simps)
  finally show ?thesis
    by simp
qed

lemma jacobi_theta_nw_00_times_01_real: "ϑ3 q * ϑ4 q = (ϑ4 (q ^ 2) ^
2 :: real)"
  by (subst of_real_eq_iff [where ?'a = complex, symmetric],
        unfold of_real_add of_real_mult of_real_diff)
     (use jacobi_theta_nw_00_times_01_complex[of q]
      in ‹simp_all flip: jacobi_theta_nome_00_of_real jacobi_theta_nome_01_of_real›)

lemma jacobi_theta_nw_00_plus_10_square_square_aux:
  fixes q :: complex
  shows "ϑ3 q ^ 2 - ϑ3 (q²) ^ 2 = q * jacobi_theta_nome (q²) (q²) ^ 2"
proof (cases "q ≠ 0 ∧ norm q < 1")
  case True
  define f :: "complex ⇒ complex"
    where "f = (λq. ϑ3 q ^ 2 - ((λq. ϑ3 q ^ 2) ∘ (λq. q ^ 2)) q - q *
jacobi_theta_nome (q²) (q²) ^ 2)"
  define F where "F = (fps_to_fls (fps_jacobi_theta_nw 1))² -
    fps_to_fls ((fps_jacobi_theta_nw 1)² oo fps_X²) -
    fps_to_fls (Abs_fps (λn. if odd n then complex_of_nat (count_sos 2
n) else 0))"
  have "f has_laurent_expansion F"
    unfolding F_def f_def
    by (intro laurent_expansion_intros fps_expansion_intros
              has_laurent_expansion_q_times_jacobi_theta_nw_10_aux_square_square
              has_laurent_expansion_fps) auto
  also have "F = fps_to_fls 0"
    unfolding F_def fps_to_fls_power [symmetric] fps_to_fls_minus [symmetric]
```

```
fps_to_fls_eq_iff
    by (auto simp: fps_eq_iff fps_jacobi_theta_power_eq fps_nth_compose_X_power
count_sos_2_double
             elim!: evenE)
  finally have "f has_laurent_expansion 0"
    by simp

  have "f q = 0"
  proof (rule has_laurent_expansion_0_analytic_continuation[of f])
    show "f has_laurent_expansion 0"
      by fact
    show "f holomorphic_on ball 0 1 - {0}"
      by (auto simp: f_def o_def norm_power power_less_one_iff intro!:
holomorphic_intros)
  qed (use True in auto)
  thus ?thesis
    by (simp add: f_def)
qed (auto simp: norm_power power_less_one_iff)

theorem jacobi_theta_nw_00_plus_10_square_square_complex:
  fixes q :: complex
  assumes "Re q ≥ 0 ∧ (Re q = 0 ⟶ Im q ≥ 0)"
  shows   "ϑ3 (q^2) ^ 2 + ϑ2 (q^2) ^ 2 = ϑ3 q ^ 2"
proof -
  have "ϑ3 q ^ 2 - ϑ3 (q^2) ^ 2 = q * jacobi_theta_nome (q^2) (q^2) ^ 2"
    by (rule jacobi_theta_nw_00_plus_10_square_square_aux)
  also have "q = ((q ^ 2) powr (1 / 4)) ^ 2"
  proof -
    have "((q ^ 2) powr (1 / 4)) ^ 2 = csqrt (q ^ 2)"
      using assms by (auto simp add: powr_def csqrt_exp_Ln simp flip:
exp_of_nat_mult)
    also have "csqrt (q ^ 2) = q"
      by (rule csqrt_unique) (use assms in ‹auto simp: not_less›)
    finally show ?thesis ..
  qed
  hence "q * jacobi_theta_nome (q^2) (q^2) ^ 2 = ϑ2 (q ^ 2) ^ 2"
    by (simp add: jacobi_theta_nome_10_def power_mult_distrib)
  finally show ?thesis by (Groebner_Basis.algebra)
qed

lemma jacobi_theta_nw_00_plus_10_square_square_real:
  assumes "q ≥ (0::real)"
  shows   "ϑ3 (q^2) ^ 2 + ϑ2 (q^2) ^ 2 = ϑ3 q ^ 2"
  by (subst of_real_eq_iff [where ?'a = complex, symmetric],
        unfold of_real_add of_real_mult of_real_diff)
     (use jacobi_theta_nw_00_plus_10_square_square_complex[of q] assms

      in ‹simp_all flip: jacobi_theta_nome_00_of_real jacobi_theta_nome_10_complex_of_real
```

```
theorem jacobi_theta_nw_00_minus_10_square_square_complex:
  assumes "0 ≤ Re q ∧ (Re q = 0 ⟶ 0 ≤ Im q)"
  shows   "ϑ₃ (q²) ^ 2 - ϑ₂ (q²) ^ 2 = ϑ₄ (q :: complex) ^ 2"
  using jacobi_theta_nw_00_plus_01_square_complex[of q]
        jacobi_theta_nw_00_plus_10_square_square_complex[OF assms]
  by Groebner_Basis.algebra

lemma jacobi_theta_nw_00_minus_10_square_square_real:
  assumes "q ≥ (0::real)"
  shows   "ϑ₃ (q²) ^ 2 - ϑ₂ (q²) ^ 2 = ϑ₄ q ^ 2"
  using jacobi_theta_nw_00_plus_01_square_real[of q]
        jacobi_theta_nw_00_plus_10_square_square_real[OF assms]
  by Groebner_Basis.algebra
```

The following shows that the theta nullwerte provide a parameterisation of the Fermat curve $X^4 + Y^4 = Z^4$:

```
theorem jacobi_theta_nw_pow4_complex: "ϑ₂ q ^ 4 + ϑ₄ q ^ 4 = (ϑ₃ q ^
4 :: complex)"
proof (cases "norm q < 1")
  case q: True
  define r where "r = csqrt q"
  have q_eq: "q = r ^ 2"
    by (simp add: r_def)
  have "norm r < 1"
    using q by (auto simp: r_def)
  have "0 ≤ Re r ∧ (Re r = 0 ⟶ 0 ≤ Im r)"
    using csqrt_principal[of q] by (auto simp: r_def simp del: csqrt.sel)
  note r = <norm r < 1> this

  have "ϑ₃ q ^ 4 - ϑ₂ q ^ 4 = (ϑ₃ q ^ 2 + ϑ₂ q ^ 2) * (ϑ₃ q ^ 2 - ϑ₂ q
^ 2)"
    by Groebner_Basis.algebra
  also have "... = (ϑ₃ r * ϑ₄ r) ^ 2"
    using jacobi_theta_nw_00_plus_10_square_square_complex[OF r(2)]
          jacobi_theta_nw_00_minus_10_square_square_complex[OF r(2)]
    unfolding q_eq by Groebner_Basis.algebra
  also have "ϑ₃ r * ϑ₄ r = ϑ₄ q ^ 2"
    unfolding q_eq using jacobi_theta_nw_00_times_01_complex[of r] .
  finally have "ϑ₃ q ^ 4 - ϑ₂ q ^ 4 = ϑ₄ q ^ 4"
    by simp
  thus ?thesis
    by Groebner_Basis.algebra
qed auto

lemma jacobi_theta_nw_pow4_real: "q ≥ 0 ⟹ ϑ₂ q ^ 4 + ϑ₄ q ^ 4 = (ϑ₃
q ^ 4 :: real)"
  by (subst of_real_eq_iff [where ?'a = complex, symmetric],
        unfold of_real_add of_real_mult of_real_diff)
```

```
    (use jacobi_theta_nw_pow4_complex[of q]
     in ‹simp_all flip: jacobi_theta_nome_00_of_real jacobi_theta_nome_01_of_real

                        jacobi_theta_nome_10_complex_of_real›)
```

5.3 Properties of the nullwert functions on the real line

```
lemma has_field_derivative_jacobi_theta_nw_00:
  fixes q :: "'a :: {real_normed_field,banach}"
  assumes q: "norm q < 1"
  defines "a ≡ (λn. 2 * (of_nat n + 1)^2 * q ^ (n * (n + 2)))"
  shows "summable a" "(ϑ3 has_field_derivative (∑n. a n)) (at q)"
proof -
  define F :: "'a fps" where "F = fps_jacobi_theta_nw 1"
  define F' where [simp]: "F' = fps_deriv F"
  define f' :: "'a ⇒ 'a" where "f' = eval_fps F"
  have [simp]: "fps_conv_radius F = 1"
    unfolding F_def using fps_conv_radius_jacobi_theta_nw[of "1::'a"]

    by (simp add: one_ereal_def)

  have "(λn. fps_nth F' n * q ^ n) sums eval_fps F' q"
    by (rule sums_eval_fps)
       (use q in ‹auto intro!: less_le_trans[OF _ fps_conv_radius_deriv]›)
  moreover have "bij_betw (λn. (n+1)^2 - 1) UNIV {n. is_square (Suc n)}"
    by (rule bij_betwI[of _ _ _ "λn. floor_sqrt (n+1) - 1"]) (auto elim!:
is_nth_powerE)
  moreover have "strict_mono (λn::nat. (n+1)^2 - 1)"
    by (intro strict_monoI_Suc) (auto simp: power2_eq_square)
  ultimately have "(λn. fps_nth (fps_deriv F) ((n+1)^2 - 1) * q ^ ((n+1)^2
- 1)) sums
                    eval_fps (fps_deriv F) q"
    by (subst sums_mono_reindex) (auto simp: F_def fps_jacobi_theta_nw_def
bij_betw_def)
  also have "(λn. fps_nth (fps_deriv F) ((n+1)^2 - 1) * q ^ ((n+1)^2 -
1)) =
             (λn. 2 * (of_nat n + 1)^2 * q ^ ((n+1)^2-1))"
    by (auto simp: F_def fps_jacobi_theta_nw_def add_ac)
  also have "(λn::nat. (n+1) ^ 2 - 1) = (λn. n * (n + 2))"
    by (simp add: algebra_simps power2_eq_square)
  finally have "a sums eval_fps (fps_deriv F) q"
    by (simp only: a_def)

  moreover have "(ϑ3 has_field_derivative (eval_fps F' q)) (at q)"
  proof -
    have "ereal (norm q) < fps_conv_radius F"
      using q by (auto simp: one_ereal_def)
    hence "(eval_fps F has_field_derivative (eval_fps F' q)) (at q)"
      unfolding F'_def by (rule has_field_derivative_eval_fps)
```

```
    also have "?this ⟷ ?thesis"
    proof (intro DERIV_cong_ev)
      have "eventually (λt. t ∈ ball 0 1) (nhds q)"
        by (rule eventually_nhds_in_open) (use q in auto)
      thus "eventually (λt. eval_fps F t = ϑ3 t) (nhds q)"
      proof eventually_elim
        case (elim t)
        thus ?case
          using sums_jacobi_theta_nw_00[of t] by (simp add: sums_iff eval_fps_def
F_def)
      qed
    qed auto
    finally show ?thesis .
  qed

  ultimately show "summable a" "(ϑ3 has_field_derivative (∑n. a n))
(at q)"
    by (simp_all add: sums_iff)
qed

lemma jacobi_theta_nw_10_le_00:
  assumes "q ≥ (0::real)"
  shows   "ϑ2 q ≤ ϑ3 q"
proof (cases "q < 1")
  case True
  with assms have q: "q ≥ 0" "q < 1"
    by auto
  define r where "r = sqrt q"
  have "0 ≤ ϑ3 q"
    using has_sum_jacobi_theta_nw_00[of q] by (rule has_sum_nonneg) (use
q in auto)
  have "(ϑ3 q)² - (ϑ2 q)² = (ϑ4 r)²"
    using jacobi_theta_nw_00_minus_10_square_square_real[of r] q
    by (simp add: r_def)
  also have "... ≥ 0"
    by simp
  finally have "(ϑ3 q)² ≥ (ϑ2 q)²"
    by simp
  thus "ϑ3 q ≥ ϑ2 q"
    by (rule power2_le_imp_le) (fact ‹ϑ3 q ≥ 0›)
qed auto

lemma jacobi_theta_nw_00_pos:
  fixes q :: real
  assumes "q ∈ {-1<..<1}"
  shows   "ϑ3 q > 0"
proof -
  have pos: "ϑ3 q > 0" if "q ∈ {0..<1}" for q :: real
    using has_sum_0 has_sum_jacobi_theta_nw_00
```

```
  proof (rule has_sum_strict_mono)
    show "0 < q powi 0 ^ 2"
      by auto
  qed (use that in auto)

  have "ϑ₄ q > 0" if q: "q ∈ {0..<1}" for q :: real
  proof -
    have eq: "ϑ₃ q * ϑ₄ q = (ϑ₄ (q ^ 2) ^ 2)"
      by (rule jacobi_theta_nw_00_times_01_real)
    have "ϑ₃ q * ϑ₄ q ≥ 0"
      by (subst eq) auto
    with pos[of q] q have "ϑ₄ q ≥ 0"
      by (simp add: zero_le_mult_iff)

    have zero_iff: "ϑ₄ q = 0 ⟷ ϑ₄ (q ^ 2) = 0" if "q ∈ {0..<1}" for
q :: real
      using jacobi_theta_nw_00_times_01_real[of q] pos[of q] that by auto

    have "ϑ₄ q ≠ 0"
    proof
      assume "ϑ₄ q = 0"
      have "ϑ₄ (q ^ (2 ^ n)) = 0" for n
      proof (induction n)
        case (Suc n)
        have "ϑ₄ (q ^ (2 ^ Suc n)) = ϑ₄ ((q ^ (2 ^ n)) ^ 2)"
          by (simp flip: power_mult add: mult_ac)
        also have "... = 0 ⟷ ϑ₄ (q ^ (2 ^ n)) = 0"
          by (subst zero_iff [symmetric]) (use q in ‹auto simp: power_less_one_iff›)
        finally show ?case
          using Suc.IH by simp
      qed (use ‹ϑ₄ q = 0› in auto)
      hence "(λn. ϑ₄ (q ^ (2 ^ n))) ⟶ 0"
        by simp
      moreover have "(λn. ϑ₄ (q ^ (2 ^ n))) ⟶ ϑ₄ 0"
      proof (rule continuous_on_tendsto_compose[of _ ϑ₄])
        show "continuous_on {0..<1::real} ϑ₄"
          by (intro continuous_intros) auto
        show "(λn. q ^ (2 ^ n)) ⟶ 0"
        proof (cases "q = 0")
          case False
          thus ?thesis
            using q by real_asymp
        qed (auto simp: power_0_left)
      qed (use q in ‹auto simp: power_less_one_iff›)
      ultimately have "ϑ₄ (0::real) = 0"
        using LIMSEQ_unique by blast
      thus False
        by simp
    qed
```

```
    with ‹ϑ₄ q ≥ 0› show "ϑ₄ q > 0"
      by simp
  qed
  from this[of "-q"] and pos[of q] show ?thesis
    using assms by (cases "q ≥ 0") (auto simp: jacobi_theta_nw_01_conv_00)
qed

lemma jacobi_theta_nw_01_pos: "q ∈ {-1<..<1} ⟹ ϑ₄ q > (0::real)"
  using jacobi_theta_nw_00_pos[of "-q"]
  by (simp add: jacobi_theta_nw_01_conv_00)

lemma jacobi_theta_nw_00_nonneg: "ϑ₃ q ≥ (0::real)"
  using jacobi_theta_nw_00_pos[of q] by (cases "norm q < 1") (auto simp:
abs_less_iff)

lemma jacobi_theta_nw_01_nonneg: "ϑ₄ q ≥ (0::real)"
  by (simp add: jacobi_theta_nw_01_conv_00 jacobi_theta_nw_00_nonneg)

lemma strict_mono_jacobi_theta_nw_00: "strict_mono_on {-1<..<1::real}
ϑ₃"
proof -
  have theta3_less: "ϑ₃ x < ϑ₃ y" if xy: "0 ≤ x" "x < y" "y < 1" for x
y :: real
  proof (rule has_sum_strict_mono)
    show "((λn. x powi n²) has_sum ϑ₃ x) UNIV" "((λn. y powi n²) has_sum
ϑ₃ y) UNIV"
      by (rule has_sum_jacobi_theta_nw_00; use xy in simp)+
    show "x powi (n^2) ≤ y powi (n^2)" for n :: int
      by (intro power_int_mono) (use xy in auto)
    show "x powi (1^2) < y powi (1^2)"
      using xy by auto
  qed auto

  have theta4_less: "ϑ₄ x < ϑ₄ y" if xy: "0 ≤ y" "y < x" "x < 1" for x
y :: real
  proof -
    include qpochhammer_inf_notation
    have "ϑ₄ x = jacobi_theta_nome (-1) x"
      by (simp add: jacobi_theta_nome_01_def)
    also have "... = (x² ; x²)∞ * ((x ; x²)∞) ^ 2"
      by (subst jacobi_theta_nome_triple_product_real) (use xy in ‹simp_all
add: power2_eq_square›)
    also have "... < (x² ; x²)∞ * ((y ; y²)∞) ^ 2"
    proof (intro mult_strict_left_mono power_strict_mono)
      show "(x² ; x²)∞ > 0" "(x ; x²)∞ ≥ 0"
        using xy by (auto intro!: qpochhammer_inf_pos qpochhammer_inf_nonneg
simp: power_less_one_iff)
    next
      show "(x ; x²)∞ < (y ; y²)∞"
```

```
  proof (rule has_prod_less)
    show "(λn. 1 - x * (x^2)^n) has_prod (x ; x²)∞"
         "(λn. 1 - y * (y^2)^n) has_prod (y ; y²)∞"
      by (rule has_prod_qpochhammer_inf; use xy in ‹simp add: power_less_one_iff›)+
  next
    show "1 - x * (x^2)^0 < 1 - y * (y^2)^0"
      using xy by simp
  next
    fix n :: nat
    have "x * (x²)^n = x^(2*n+1)"
      by (simp add: power_mult power_add)
    also have "... < 1"
      by (subst power_less_one_iff) (use xy in auto)
    finally show "1 - x * (x²)^n > 0"
      by simp
  next
    fix n :: nat
    have "x^(2*n+1) ≥ y^(2*n+1)"
      by (intro power_mono) (use xy in auto)
    thus "1 - x * (x²) ^ n ≤ 1 - y * (y²) ^ n"
      by (simp add: power_mult)
  qed
qed auto
also have "... ≤ (y² ; y²)∞ * ((y ; y²)∞) ^ 2"
proof (intro mult_right_mono zero_le_power)
  show "(y ; y²)∞ ≥ 0"
    by (intro qpochhammer_inf_nonneg) (use xy in ‹auto simp: power_less_one_iff›)
next
  show "(x² ; x²)∞ ≤ (y² ; y²)∞"
  proof (rule has_prod_le[OF _ _ conjI])
    show "(λn. 1 - x² * (x^2)^n) has_prod (x² ; x²)∞"
         "(λn. 1 - y² * (y^2)^n) has_prod (y² ; y²)∞"
      by (rule has_prod_qpochhammer_inf; use xy in ‹simp add: power_less_one_iff›)+
  next
    fix n :: nat
    have "x² * (x²)^n = x^(2*n+2)"
      by (simp add: power_mult power_add power2_eq_square)
    also have "... ≤ 1"
      by (subst power_le_one_iff) (use xy in auto)
    finally show "1 - x² * (x²)^n ≥ 0"
      by simp
  next
    fix n :: nat
    have "x^(2*n+2) ≥ y^(2*n+2)"
      by (intro power_mono) (use xy in auto)
    thus "1 - x² * (x²) ^ n ≤ 1 - y² * (y²) ^ n"
      by (simp add: power_mult power2_eq_square)
  qed
qed
```

```
    also have "... = jacobi_theta_nome (-1) y"
      by (subst jacobi_theta_nome_triple_product_real) (use xy in ‹simp_all
add: power2_eq_square›)
    also have "... = ϑ₄ y"
      by (simp add: jacobi_theta_nome_01_def)
    finally show "ϑ₄ x < ϑ₄ y" .
  qed

  show "strict_mono_on {-1<..<1::real} ϑ₃"
  proof (rule strict_mono_onI)
    fix x y :: real
    assume xy: "x ∈ {-1<..<1}" "y ∈ {-1<..<1}" "x < y"
    consider "x ≥ 0" | "y ≤ 0" | "x < 0" "y > 0"
      using xy by linarith
    thus "ϑ₃ x < ϑ₃ y"
    proof cases
      assume "x ≥ 0"
      thus ?thesis by (intro theta3_less) (use xy in auto)
    next
      assume "y ≤ 0"
      hence "ϑ₄ (-x) < ϑ₄ (-y)"
        by (intro theta4_less) (use xy in auto)
      thus ?thesis
        by (simp add: jacobi_theta_nw_01_conv_00)
    next
      assume xy': "x < 0" "y > 0"
      have "ϑ₄ (-x) < ϑ₄ 0"
        by (rule theta4_less) (use xy xy' in auto)
      moreover have "ϑ₃ 0 < ϑ₃ y"
        by (rule theta3_less) (use xy xy' in auto)
      ultimately show ?thesis
        by (simp add: jacobi_theta_nw_01_conv_00)
    qed
  qed
qed

lemma strict_antimono_jacobi_theta_nw_01: "strict_antimono_on {-1<..<1::real}
ϑ₄"
  by (auto intro!: monotone_onI strict_mono_onD[OF strict_mono_jacobi_theta_nw_00]

          simp: jacobi_theta_nw_01_conv_00)

lemma jacobi_theta_nw_10_nonneg:
  assumes "x ≥ 0"
  shows   "ϑ₂ x ≥ (0::real)"
proof -
  consider "x = 0" | "x ≥ 1" | "x ∈ {0<..<1}"
    using assms by force
  thus ?thesis
```

```
  proof cases
    assume x: "x ∈ {0<..<1}"
    show ?thesis
      using has_sum_jacobi_theta_nw_10_real
      by (rule has_sum_nonneg) (use x in auto)
  qed auto
qed

lemma strict_mono_jacobi_theta_nw_10: "strict_mono_on {0::real..<1} ϑ₂"
proof (rule strict_mono_onI)
  fix x y :: real
  assume xy: "x ∈ {0..<1}" "y ∈ {0..<1}" "x < y"
  note mono_rules = strict_mono_jacobi_theta_nw_00 strict_antimono_jacobi_theta_nw_01

  have "ϑ₂ x ^ 4 = ϑ₃ x ^ 4 - ϑ₄ x ^ 4"
    by (subst jacobi_theta_nw_pow4_real [symmetric]) (use xy in auto)
  also have "... < ϑ₃ y ^ 4 - ϑ₄ y ^ 4"
    by (intro diff_strict_mono power_strict_mono mono_rules[THEN monotone_onD]
              jacobi_theta_nw_00_nonneg jacobi_theta_nw_01_nonneg)
       (use xy in auto)
  also have "... = ϑ₂ y ^ 4"
    by (subst jacobi_theta_nw_pow4_real [symmetric]) (use xy in auto)
  finally show "ϑ₂ x < ϑ₂ y"
    by (rule power_less_imp_less_base) (use xy in ‹auto intro!: jacobi_theta_nw_10_nonneg›)
qed

lemma jacobi_theta_nw_10_pos:
  assumes "x ∈ {0<..<1}"
  shows   "ϑ₂ x > (0::real)"
  using strict_mono_onD[OF strict_mono_jacobi_theta_nw_10, of 0 x] assms
by simp

end
```

References

[1] B. C. Berndt. *Ramanujans Notebooks*. Springer New York, 1991.