The Rogers–Ramanujan Identities

Manuel Eberl

January 24, 2025

Abstract

This entry formalises the Rogers–Ramanujan Identities:

$$\sum_{k=-\infty}^{\infty} \frac{q^{k^2}}{\prod_{j=1}^{k}(1-q^j)} = \left(\prod_{n=0}^{\infty}(1-q^{1+5n})(1-q^{4+5n})\right)^{-1}$$
$$\sum_{k=-\infty}^{\infty} \frac{q^{k^2+k}}{\prod_{j=1}^{k}(1-q^j)} = \left(\prod_{n=0}^{\infty}(1-q^{2+5n})(1-q^{3+5n})\right)^{-1}$$

The formalisation follows the elegant proof given in Andrews and Eriksson *Integer Partitions*, using the Jacobi triple product.

1 The Rogers–Ramanujan identities

```
theory Rogers_Ramanujan
  imports "Theta_Functions.Jacobi_Triple_Product"
begin
```

Acknowledgement: I would like to thank George Andrews for giving me a crucial hint about a uniform convergence issue that I struggled with.

```
unbundle qpochhammer_inf_notation
```

First of all, we show two auxiliary results concerned with the (absolute) convergence of two infinite sums that will appear in our proof of the identities.

```
lemma summable_rogers_ramanujan_aux1:
  fixes q :: "'a :: {real_normed_field, banach}" and M :: int
  assumes q: "q ≠ 0" "norm q < 1"
  shows "(λj. norm q powi (j*(5*j+M) div 2)) summable_on UNIV"
proof -
  have *: "(λj. norm q powi (j * (5*j+M) div 2)) summable_on {0..}" for
M :: int
  proof -
    have "summable (λj. norm q powi (int j * (5 * int j + M) div 2))"
    proof (rule summable_comparison_test_bigo)
      have "eventually (λj. int j * (5 * int j + M) div 2 ≥ 0) at_top"
        by real_asymp
      hence "eventually (λj. norm q powi (int j * (5 * int j + M) div
2) =
                              norm q ^ (nat (int j * (5 * int j + M) div
2))) at_top"
        by eventually_elim (auto simp: power_int_def)
      hence "(λj. norm q powi (int j * (5 * int j + M) div 2)) ∈
            Θ(λj. norm q ^ (nat (int j * (5 * int j + M) div 2)))"
        by (rule bigthetaI_cong)
      also have "(λj. norm q ^ (nat (int j * (5 * int j + M) div 2)))
∈ O(λj. (1/2) ^ j)"
        using q by real_asymp
      finally show "(λj. norm q powi (int j * (5 * int j + M) div 2))
∈ O(λj. (1/2) ^ j)" .
    qed auto
    hence "(λj. norm q powi (int j * (5 * int j + M) div 2)) summable_on
UNIV"
      by (rule summable_nonneg_imp_summable_on) auto
    also have "?this ⟷ (λj. norm q powi (j * (5*j+M) div 2)) summable_on
{0..}"
      by (rule summable_on_reindex_bij_witness[of _ nat int]) auto
    finally show "(λj. norm q powi (j * (5*j+M) div 2)) summable_on {0..}"
.
  qed

  have "(λj. norm q powi (j * (5*j-M) div 2)) summable_on {0..}"
```

```
    using *[of "-M"] by simp
  also have "?this ⟷ (λj. norm q powi (j * (5*j+M) div 2)) summable_on
{..0}"
    by (rule summable_on_reindex_bij_witness[of _ uminus uminus]) (auto
simp: algebra_simps)
  finally have "(λj. norm q powi (j * (5*j+M) div 2)) summable_on ({..0}
∪ {0..})"
    by (intro summable_on_union *)
  also have "{..0} ∪ {0::int..} = UNIV"
    by auto
  finally show "(λj. norm q powi (j*(5*j+M) div 2)) summable_on UNIV"
.
qed

lemma summable_rogers_ramanujan_aux2:
  fixes q :: "'a :: {real_normed_field, banach}" and M :: int
  assumes q: "q ≠ 0" "norm q < 1"
  shows "summable (λj. norm (q ^ (j^2 + c * j) / qpochhammer (int j) q
q))"
proof (rule summable_comparison_test_bigo)
  have "(λj. norm (q ^ (j ^ 2 + c * j) / qpochhammer (int j) q q)) ∈

          O(λj. norm q ^ (j^2 + c * j) / (1 - norm q) ^ j)"
  proof (intro bigoI[of _ 1] always_eventually allI)
    fix j :: nat
    have [simp]: "qpochhammer (int j) q q ≠ 0"
      by (intro qpochhammer_nonneg_nonzero q)
    have "norm (norm (q ^ (j ^ 2 + c * j) / qpochhammer (int j) q q))
=
            norm q ^ (j^2 + c*j) / norm (qpochhammer (int j) q q)"
      by (simp add: norm_power norm_divide)
    also have "... ≤ norm q ^ (j^2 + c*j) / (1 - norm q) ^ j"
      by (intro divide_left_mono norm_qpochhammer_nonneg_ge mult_pos_pos)
(use q in auto)
    also have "... ≤ norm (norm q ^ (j^2 + c*j) / (1 - norm q) ^ j)"
      unfolding real_norm_def by linarith
    finally show "norm (norm (q ^ (j^2 + c * j) / qpochhammer (int j) q
q))
                    ≤ 1 * norm (norm q ^ (j^2 + c * j) / (1 - norm q) ^
j)"
      by simp
  qed
  also have "(λj. norm q ^ (j^2 + c * j) / (1 - norm q) ^ j) ∈ O(λj. (1/2)
^ j)"
    using q by real_asymp
  finally show "(λj. norm (q ^ (j ^ 2 + c * j) / qpochhammer (int j) q
q)) ∈ O(λj. (1/2) ^ j)" .
qed auto
```

Next, we apply the Jacobi triple product to show that for $N \in \{0, 1\}$ we

have

$$\sum_{n=-\infty}^{\infty} (-1)^n q^{n(5n+2N+1)/2} = \frac{(q;q)_\infty}{\prod_{i\in I}(q^i;q^5)_\infty}$$

where $I = \{1,\ldots,4\} \setminus \{2-N, 3+N\}$.

```
lemma rogers_ramanujan_aux:
  fixes q :: complex and N :: nat
  assumes q: "norm q < 1" and N: "N < 2"
  shows "((λn. (-1) powi n * q powi (n*(5*n+2*N+1) div 2)) has_sum
          (q; q)_∞ / ((q^(1+N); q^5)_∞ * (q^(4-N); q^5)_∞)) UNIV"
proof -
  have N_cases: "N ∈ {0,1}"
    using N by auto
  have "((λn. (- (q^(3+N))) powi (n*(n+1) div 2) * (-(q^(2-N))) powi (n*(n-1)
div 2))
         has_sum ramanujan_theta (-(q^(3+N))) (-(q^(2-N)))) UNIV"
    by (rule has_sum_ramanujan_theta)
       (use q in ‹simp_all flip: power_add add: norm_power power_less_one_iff›)
  also have "ramanujan_theta (-(q^(3+N))) (-(q^(2-N))) =
               (q^(3+N) ; q^5)_∞ * (q^(2-N) ; q^5)_∞ * (q^5 ; q^5)_∞"
    using ramanujan_theta_triple_product_complex[of "-(q^(3+N))" "-(q^(2-N))"]
N
    by (simp flip: power_add add: norm_power power_less_one_iff q)
  also have "(λn. (- (q^(3+N))) powi (n*(n+1) div 2) * (-(q^(2-N))) powi
(n*(n-1) div 2)) =
             (λn. (- 1) powi n * q powi (n * (5 * n + 2 * int N + 1) div
2))" (is "?lhs = ?rhs")
  proof
    fix n :: int
    show "?lhs n = ?rhs n"
    proof (cases "q = 0")
      case True
      have *: "n * (n + 1) div 2 = 0 ⟷ n ∈ {0, -1}"
              "n * (n - 1) div 2 = 0 ⟷ n ∈ {0, 1}"
        by (auto simp: dvd_div_eq_0_iff)
      have "5 * n + 2 * int N + 1 ≠ 0"
        using N by presburger
      hence **: "n * (5 * n + 2 * int N + 1) div 2 = 0 ⟷ n = 0"
        using N by (auto simp: dvd_div_eq_0_iff)
      have "?lhs n = 0 powi (n * (n + 1) div 2) * 0 powi (n * (n - 1)
div 2)"
        using N by (simp add: True power_0_left)
      also have "... = (if n = 0 then 1 else 0)"
        unfolding power_int_0_left_If * by auto
      also have "... = ?rhs n"
        unfolding True power_int_0_left_If ** by auto
      finally show ?thesis .
    next
      case [simp]: False
```

```
      define m where "m = n*(n-1) div 2"
      have "m + (2 * n) div 2 = (n * (n - 1) + 2 * n) div 2"
        unfolding m_def by (subst div_plus_div_distrib_dvd_left [symmetric])
auto
      also have "n * (n - 1) + 2 * n = n * (n + 1)"
        by (simp add: algebra_simps)
      finally have *: "n * (n + 1) div 2 = m + n"
        by simp

      have "(- (q^(3+N))) powi (n*(n+1) div 2) * (-(q^(2-N))) powi (n*(n-1)
div 2) =
            (-1) powi n * (q powi ((3 + N) * (m+n)) * q powi ((2-N) *
m))"
        unfolding * m_def [symmetric]
        by (subst (1 2) power_int_minus_left) (auto simp: power_int_power)
      also have "q powi ((3 + N) * (m+n)) * q powi ((2-N) * m) =
                 q powi ((3 + N) * (m+n) + (2-N) * m)"
        by (simp add: power_int_add)
      also have "(3 + N) * (m+n) + (2-N) * m = N * n + 3 * n + 5 * m"
        using N by (simp add: algebra_simps of_nat_diff)
      also have "... = (2 * N * n + 6 * n + 5 * n * (n - 1)) div 2"
        by (simp add: m_def div_mult_swap)
      also have "2 * N * n + 6 * n + 5 * n * (n - 1) = n * (5 * n + 2
* int N + 1)"
        by (simp add: algebra_simps)
      finally show "?lhs n = ?rhs n" .
    qed
  qed
  finally have "((λn. (-1) powi n * q powi (n*(5*n+2*N+1) div 2)) has_sum
                  (q^(3+N) ; q^5)∞ * (q^(2-N) ; q^5)∞ * (q^5 ; q^5)∞)
UNIV"
    by simp
  also have "(q^(3+N) ; q^5)∞ * (q^(2-N) ; q^5)∞ * (q^5 ; q^5)∞ =
             (q; q)∞ / ((q^(1+N); q^5)∞ * (q^(4-N); q^5)∞)"
  proof -
    define A where "A = {2+N, 1-N, 4}"
    define B where "B = {..<5} - A"
    have q_pow_eq_1_iff: "q ^ k = 1 ⟷ k = 0" for k
    proof
      assume "q ^ k = 1"
      hence "norm (q ^ k) = 1"
        by simp
      moreover have "k > 0 ⟶ norm (q ^ k) < 1"
        using q by (simp add: norm_power power_less_one_iff)
      ultimately show "k = 0"
        by auto
    qed auto

    have "(q^(3+N) ; q^5)∞ * (q^(2-N) ; q^5)∞ * (q^5 ; q^5)∞ =
```

```
        (∏i∈A. (q ^ (i+1); q ^ 5)∞)"
    using N_cases by (auto simp: eval_nat_numeral mult_ac A_def)
  also have "... = (∏i∈{..<5} - B. (q ^ (i+1); q ^ 5)∞)"
    using N by (intro prod.cong) (auto simp: A_def B_def)
  also have "... = (∏i<5. (q ^ (i+1); q ^ 5)∞) / (∏i∈B. (q ^ (i+1);
q ^ 5)∞)"
    by (subst prod_diff)
       (simp_all add: B_def qpochhammer_inf_zero_iff norm_power power_less_one_iff

                      q q_pow_eq_1_iff flip: power_mult power_Suc power_add)
  also have "(∏i<5. (q ^ (i+1); q ^ 5)∞) = (q; q)∞"
    using prod_qpochhammer_group[of q 5 q] q by simp
  also have "B = {N, 3 - N}"
    unfolding A_def B_def using N_cases by auto
  also have "(∏i∈{N, 3 - N}. (q ^ (i + 1) ; q ^ 5)∞) = (q^(1+N); q^5)∞
* (q^(4-N); q^5)∞"
    using N by (simp flip: power_Suc flip: Suc_diff_le)
  finally show ?thesis .
 qed
 finally show ?thesis .
qed

theorem rogers_ramanujan_complex:
 fixes q :: complex
 assumes "norm q < 1"
 shows   "((λj. q ^ (j²) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞
* (q^4;q^5)∞))) UNIV"
   and   "((λj. q ^ (j² + j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
proof -
 have "((λj. q ^ (j²) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞ *
(q^4;q^5)∞))) UNIV ∧
       ((λj. q ^ (j² + j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
 proof (cases "q = 0")
   case [simp]: True
   show ?thesis
     by (auto intro!: has_sum_finite_neutralI[of "{0}"])
 next
   case [simp]: False
   note q = <norm q < 1>
   have q_pow_neq_1: "q ^ n ≠ 1" if "n > 0" for n
     using q_power_neq_1[of q n] that q by simp
   have [simp]: "(q; q)∞ ≠ 0"
     using q qpochhammer_inf_nonzero by blast
   have [simp]: "qpochhammer (int j) q q ≠ 0" for j
   proof
     assume "qpochhammer j q q = 0"
     then obtain k where k: "q * q powi k = 1" "k ∈ {0..<int j}"
```

```
      using q by (auto simp: qpochhammer_eq_0_iff)
    show False
      using q_pow_neq_1[of "nat (k+1)"] k by (auto simp: nat_add_distrib
power_int_def)
  qed

  define B :: "int ⇒ int ⇒ complex"
    where "B = (λn k. if n ≥ 0 ∧ k ≥ 0 then qbinomial q (nat n) (nat
k) else 0)"
  have B_eq: "B (int n) (int k) = qbinomial q n k" for n k
    by (simp add: B_def)
  have [simp]: "B n k = 0" if "n < 0 ∨ k < 0 ∨ k > n" for n k
    using that by (auto simp: B_def)
  have B_sym: "B n k = B n (n - k)" for n k
    using qbinomial_symmetric[of q "nat k" "nat n"] q
    by (auto simp: B_def nat_diff_distrib)

  have B_rec: "B n k = q powi k * B (n-1) k + B (n-1) (k-1)" if "n ≠
0 ∨ k ≠ 0" for n k :: int
  proof (cases "n > 0 ∧ k > 0")
    case True
    define n' k' where "n' = nat (n-1)" and "k' = nat (k-1)"
    have [simp]: "n = int (Suc n')" "k = int (Suc k')"
      using True unfolding B_def by (auto simp: n'_def k'_def)
    show ?thesis
      by (auto simp: B_def nat_add_distrib power_int_def qbinomial_Suc_Suc)
  qed (use that in ‹auto simp: B_def qbinomial_0_middle›)

  have B_rec': "B n k = B (n-1) k + q powi (n-k) * B (n-1) (k-1)" if
"n ≠ 0 ∨ k ≠ 0" for n k :: int
  proof (cases "n > 0 ∧ k > 0")
    case True
    define n' k' where "n' = nat (n-1)" and "k' = nat (k-1)"
    have [simp]: "n = int (Suc n')" "k = int (Suc k')"
      using True unfolding B_def by (auto simp: n'_def k'_def)
    show ?thesis using q
      by (auto simp: B_def nat_add_distrib power_int_def qbinomial_Suc_Suc'
nat_diff_distrib)
  qed (use that in ‹auto simp: B_def qbinomial_0_middle›)

  have B_rec3: "B n k * (1 - q ^ (n - k)) = (1 - q ^ n) * B (n - 1)
k" for n k
  proof (cases "n > 0 ∧ k > 0 ∧ k < n")
    case True
    thus ?thesis
      using qbinomial_rec2[of q "nat n" "nat k"] q
      by (auto simp: B_def q_pow_neq_1 nat_diff_distrib)
  qed (auto simp: B_def)
```

```
    define e :: "int ⇒ int ⇒ int" where "e = (λN j. j*(5 * j + 1 -
4 * N) div 2)"
    define S where "S = (λN n. ∑j≤n. q^(j^2 + nat N * j) * B n j)"
    define T where "T = (λN n. ∑∞j. (-1) powi j * q powi (e N j) *
B (2 * int n + N) (int n + 2*j))"

    have e0_conv_e1: "e 0 j = e 1 j + 2 * j" for j
    proof -
      have "e 0 j = j * (5 * j + 1) div 2"
        by (simp add: e_def)
      also have "j * (5 * j + 1) = j * (5 * j - 3) + 4 * j"
        by (simp add: algebra_simps)
      also have "... div 2 = e 1 j + 2 * j"
        by (subst div_plus_div_distrib_dvd_left) (auto simp: e_def)
      finally show ?thesis .
    qed

    have has_sum_aux:
      "(λj. (-1) powi j * q powi (e a j) * B (2 * int n + a+b) (int (n+b)
+ 2*j)) summable_on UNIV"
      (is "?f summable_on _") for a b n
    proof -
      have "finite {j. 2 * j ∈ {-int (n+b)..int n + a}}"
      proof (rule finite_subset)
        show "{j. 2 * j ∈ {-int (n+b)..int n + a}} ⊆
                {-(n+b+1) div 2..(int n + a + 1) div 2}"
          by (auto simp: minus_le_iff)
      qed auto
      hence "?f summable_on {j. 2 * j ∈ {-int (n+b)..int n + a}}"
        by (rule summable_on_finite)
      also have "?this ⟷ ?f summable_on UNIV"
        by (rule summable_on_cong_neutral) auto
      finally show ?thesis .
    qed

    have has_sum_T: "((λj. (-1) powi j * q powi (e N j) * B (2*n+N) (n+2*j))
has_sum T N n) UNIV"
      for n N using has_sum_infsum[OF has_sum_aux[of N n 0]] by (simp
add: T_def)

    have has_sum_T0: "((λj. (-1) powi j * q powi (e 0 j) * B (2*n+1) (n+1+2*j))
has_sum T 0 n) UNIV"
      for n
    proof -
      define T' where
        "T' = (λn. ∑∞j. (-1) powi j * q powi (e 0 j) * B (2 * int n
+ 1) (int n+1+2*j))"
      have 1: "((λj::int. (-1) powi j * q powi e 0 j * B (2*n+1) (n+1+2*j))
has_sum T' n) UNIV"
```

```
      using has_sum_infsum[OF has_sum_aux[of 0 n 1]] unfolding T'_def
by (simp add: algebra_simps)
    also have "(λj::int. (-1) powi j * q powi e 0 j * B (2*n+1) (n+1+2*j))
=
               (λj::int. (-1) powi j * q powi e 0 j *
                 (B (2*n) (n+2*j) + q powi (n+1+2*j) * B (2*n) (n+1+2*j)))"
      by (subst B_rec) auto
    finally have 2: "(... has_sum (T' n)) UNIV" .

    have 3: "((λj::int. (-1) powi j * q powi (e 0 j + 2 * j) * q powi
(n+1) * B (2*n) (n+2*j+1))
               has_sum (T' n - T 0  n)) UNIV"
      using has_sum_diff[OF 2 has_sum_T[of 0 n]] by (simp add: algebra_simps
power_int_add)
    also have "?this ⟷ ((λj::int. -((-1) powi j * q powi (e 0 (-j-1)
- 2*j-2) * q powi (n+1) * B (2*n) (n-2*j-1)))
            has_sum (T' n - T 0 n)) UNIV"
      by (intro has_sum_reindex_bij_witness[of _ "λj. -j-1" "λj. -j-1"])
         (auto simp: algebra_simps power_int_diff power_int_minus power_int_minus_left)
    also have "(λj::int. -((-1) powi j * q powi (e 0 (-j-1) - 2*j-2)
* q powi (n+1) * B (2*n) (n-2*j-1))) =
               (λj::int. -((-1) powi j * q powi (e 0 j + 2*j) * q powi
(n+1) * B (2*n) (n+2*j+1)))"
       (is "?lhs = ?rhs")
    proof
      fix j :: int
      have "?lhs j = - ((- 1) powi j * q powi (e 0 (-j-1) - (4*j+2)
+ 2 * j) *
                        q powi int (n + 1) * B (2*n) (n+2*j+1))"
        by (subst B_sym) (simp_all add: algebra_simps)
      also have "e 0 (-j-1) - (4*j+2) = ((j+1)*(4+5*j) - 2*(4*j+2))
div 2"
        unfolding e_def by (subst div_diff) (auto simp: algebra_simps)
      also have "(j+1)*(4+5*j) - 2*(4*j+2) = j * (5*j + 1)"
        by (simp add: algebra_simps)
      also have "... div 2 = e 0 j"
        by (simp add: e_def algebra_simps)
      finally show "?lhs j = ?rhs j" .
    qed
    finally have "((λj::int. (-1) powi j * q powi (e 0 j + 2*j) * q powi
(n+1) * B (2*n) (n+2*j+1))
                  has_sum (T 0 n - T' n)) UNIV"
      by (subst (asm) has_sum_uminus) simp_all
    with 3 have "T' n - T 0 n = T 0 n - T' n"
      using has_sum_unique by blast
    hence "T 0 n = T' n"
      by simp
    with 1 show ?thesis
      by (simp add: algebra_simps)
```

```
  qed
```

The initial conditions $S_i(0) = T_i(0) = 1$ are easily determined.

```
  have [simp]: "S N 0 = 1" for N
    by (simp_all add: S_def B_def)

  have [simp]: "T N 0 = 1" if N: "N ∈ {0,1}" for N
  proof -
    have "((λj. (-1) powi j * q powi e N j * B N (2 * j)) has_sum 1)
{0}"
      by (intro has_sum_finiteI) (use N in ‹auto simp: e_def B_def›)
    also have "?this ⟷ ((λj. (-1) powi j * q powi e N j * B N (2
* j)) has_sum 1) UNIV"
      using N by (intro has_sum_cong_neutral) (auto simp: B_def)
    finally show ?thesis
      by (simp add: has_sum_iff T_def)
  qed
```

Next, we prove that the S_i satisfy the recurrences $S_0(n+1) = S_0(n) + q^{n+1}S_1(n)$ and $S_1(n+1) = q^{n+1}S_0(n+1) + (1 - q^{n+1}S_1(n)$. This requires just a few manipulations of finite sums involving the two recurrences for the q-binomial coefficients.

```
  have S0_rec: "S 0 (Suc n) = S 0 n + q ^ Suc n * S 1 n" for n
  proof -
    define n' where "n' = Suc n"
    have n': "n' > 0" and [simp]: "n' - 1 = n" "int n' - 1 = int n"
      by (simp_all add: n'_def)

    have "S 0 n' = (∑j≤n'. q ^ j² * B (int n') (int j))"
      by (simp add: S_def)
    also have "... = (∑j≤n'. q^(j²) * B (int n) (int j)) +
                     (∑j≤n'. q powi (int j ^ 2 - int j) * q ^ n' *
B (int n) (int j - 1))"
      by (subst B_rec')
         (use n' in ‹simp_all add: algebra_simps power_add sum.distrib
power_int_diff
                           power_int_add flip: of_nat_power›)
    also have "(∑j≤n'. q^(j²) * B (int n) (int j)) = S 0 n"
      unfolding S_def by (intro sum.mono_neutral_cong_right) (auto simp:
n'_def)
    also have "(∑j≤n'. q powi (int j ^ 2 - int j) * q ^ n' * B (int
n) (int j - 1)) =
                 q^n' * (∑j≤n'. q powi (int j ^ 2 - int j) * B (int
n) (int j - 1))"
      by (simp add: sum_distrib_left sum_distrib_right mult_ac)
    also have "(∑j≤n'. q powi (int j ^ 2 - int j) * B (int n) (int
j - 1)) =
                 (∑j=1..n'. q powi (int j ^ 2 - int j) * B (int n) (int
j - 1))"
```

```
      by (intro sum.mono_neutral_right) auto
    also have "... = (∑j≤n. q powi (int j ^ 2 + int j) * B (int n)
(int j))"
      by (intro sum.reindex_bij_witness[of _ "λj. j+1" "λj. j-1"])
         (auto simp: n'_def of_nat_diff power2_eq_square algebra_simps)
    also have "... = (∑j≤n. q ^ (j^2+j) * B (int n) (int j))"
      by (rule sum.cong) (auto simp: power_int_def nat_add_distrib nat_power_eq)
    also have "q ^ n' * ... = q ^ n' * S 1 n"
      by (simp add: S_def)
    finally show ?thesis
      unfolding n'_def .
  qed

  have S1_rec: "S 1 (Suc n) = q ^ (Suc n) * S 0 (Suc n) + (1 - q ^ Suc
n) * S 1 n" for n
  proof -
    define n' where "n' = Suc n"
    have n': "n' > 0" and [simp]: "n' - 1 = n" "int n' - 1 = int n"
      by (simp_all add: n'_def)
    have "S 1 n' - q^n' * S 0 n' = (∑j≤n'. q ^ (j^2 + j) * (B (int
n') (int j) * (1 - q ^ (n' - j))))"
      by (simp add: S_def sum_distrib_left power_int_add ring_distribs
power_diff
          power_add mult_ac flip: sum_subtractf)
    also have "... = (∑j≤n. q ^ (j^2 + j) * (B (int n') (int j) * (1
- q ^ (n' - j))))"
      by (rule sum.mono_neutral_right) (auto simp: n'_def)
    also have "... = (∑j≤n. q ^ (j^2 + j) * B (int n' - 1) (int j)
* (1 - q ^ n'))"
      by (subst B_rec3) (auto simp: mult_ac)
    also have "... = (1 - q ^ n') * S 1 n"
      by (simp add: S_def sum_distrib_left sum_distrib_right mult_ac)
    finally show ?thesis
      unfolding n'_def by (simp add: algebra_simps)
  qed
```

Next, we show that the T_i satisfy equivalent recurrence relations. This again consists of some conceptually simply manipulations of sums. The sums are again finite in the sense that all but finitely many summands are 0, but it is is a bit easier to simply sum over all integers and not worry about the precise summation range.

```
  have T0_rec: "T 0 (Suc n) = T 0 n + q ^ Suc n * T 1 n" for n
  proof -
    have "((λj::int. (-1) powi j * q powi e 0 j * B (2*n+2) (n+1+2*j)
-
                (-1) powi j * q powi e 0 j * B (2*n+1) (n+1+2*j))
                  has_sum (T 0 (Suc n) - T 0 n)) UNIV"
      using has_sum_T[of 0 "Suc n"] has_sum_T0[of n]
      by (intro has_sum_diff) (simp_all add: algebra_simps)
```

```
      hence "((λj::int. (-1) powi j * q powi e 0 j * (B (2*n+2) (n+1+2*j)
- B (2*n+1) (n+1+2*j)))
               has_sum (T 0 (Suc n) - T 0 n)) UNIV"
        by (simp add: algebra_simps)
      also have "(λj::int. B (2*n+2) (n+1+2*j) - B (2*n+1) (n+1+2*j))
=
                 (λj::int. B (2*n+1) (n+2*j) * q powi (n+1-2*j))"
        by (subst B_rec') (simp_all add: algebra_simps)
      finally have "((λj::int. (-1) powi j * q powi e 0 j * B (2*n+1) (n+2*j)
* q powi (n+1-2*j))
                     has_sum T 0 (Suc n) - T 0 n) UNIV"
        by (simp only: mult.assoc)
      also have "(λj::int. (-1) powi j * q powi e 0 j * B (2*n+1) (n+2*j)
* q powi (n+1-2*j)) =
                 (λj::int. q powi (n+1) * ((-1) powi j * q powi (e 1 j)
* B (2*n+1) (n+2*j)))"
        (is "?lhs = ?rhs")
      proof
        fix j :: int
        have "?lhs j = q powi (n+1) * ((-1) powi j * q powi (e 0 j - 2
* j) * B (2*n+1) (n+2*j))"
          by (simp add: power_int_add power_int_diff field_simps)
        also have "e 0 j - 2 * j = e 1 j"
          by (simp add: e0_conv_e1)
        finally show "?lhs j = ?rhs j" .
      qed
      finally have "((λj::int. q powi (n+1) * ((-1) powi j * q powi (e
1 j) * B (2*n+1) (n+2*j)))
                     has_sum (T 0 (Suc n) - T 0 n)) UNIV" .
      moreover have "((λj::int. q powi (n+1) * ((-1) powi j * q powi
(e 1 j) * B (2*n+1) (n+2*j))) has_sum
                       (q powi (n+1) * T 1 n)) UNIV"
        using has_sum_T[of 1 n] by (intro has_sum_cmult_right has_sum_T)
(simp_all add: add_ac)
      ultimately have "T 0 (Suc n) - T 0 n = q powi (n+1) * T 1 n"
        using has_sum_unique by blast
      thus ?thesis
        by (simp add: algebra_simps power_int_add)
    qed

    have T1_rec: "T 1 (Suc n) = q ^ Suc n * T 0 (Suc n) + (1 - q ^ Suc
n) * T 1 n" for n
    proof -
      have "((λj::int. (-1) powi j * q powi e 1 j * B (2*n+3) (n+1+2*j)
-
                       q ^ Suc n * ((-1) powi j * q powi e 0 j * B (2*n+2)
(n+1+2*j)))
                       has_sum (T 1 (Suc n) - q ^ Suc n * T 0 (Suc n)))
UNIV"
```

```
        using has_sum_T[of 1 "Suc n"] has_sum_T[of 0 "Suc n"]
        by (intro has_sum_diff has_sum_cmult_right) (simp_all add: algebra_simps)
      also have "(λj::int. (-1) powi j * q powi e 1 j * B (2*n+3) (n+1+2*j)
-
                          q ^ Suc n * ((-1) powi j * q powi e 0 j * B (2*n+2)
(n+1+2*j))) =
                 (λj::int. (-1) powi j * q powi e 1 j *
                      (B (2*n+1) (n+2*j) + q powi (n+2-2*j) * B (2*n+1)
(n+2*j-1)))"
         (is "?lhs = ?rhs")
      proof
        fix j :: int
        have "?lhs j = (-1) powi j * (q powi e 1 j * B (2*n+3) (n+1+2*j)
-
                            q powi (e 0 j - 2*j) * q powi (n+1+2*j) * B
(2*n+2) (n+1+2*j))"
          by (simp add: algebra_simps power_int_add power_int_diff)
        also have "e 0 j - 2 * j = e 1 j"
          by (simp add: e0_conv_e1)
        also have "(-1) powi j * (q powi e 1 j * B (2*n+3) (n+1+2*j) -
                     q powi e 1 j * q powi (n+1+2*j) * B (2*n+2) (n+1+2*j))
=
                   (-1) powi j * q powi e 1 j *
                     (B (2*n+3) (n+1+2*j) - q powi (n+1+2*j) * B (2*n+2)
(n+1+2*j))"
          by (simp add: algebra_simps)
        also have "... = (-1) powi j * q powi e 1 j * B (2*n+2) (n+2*j)"
          by (subst B_rec) (simp_all add: algebra_simps)
        also have "... = (-1) powi j * q powi e 1 j *
                          (B (2*n+1) (n+2*j) + q powi (n+2-2*j) * B (2*n+1)
(n+2*j-1))"
          by (subst B_rec') (simp_all add: algebra_simps)
        finally show "?lhs j = ?rhs j" .
      qed
      finally have 1: "((λj::int. (-1) powi j * q powi e 1 j *
                           (B (2*n+1) (n+2*j) + q powi (n+2-2*j) * B (2*n+1)
(n+2*j-1)))
                       has_sum (T 1 (Suc n) - q ^ Suc n * T 0 (Suc n)))
UNIV" .

      have 2: "((λj::int. (-1) powi j * q powi e 1 j * B (2*n+1) (n+2*j))
has_sum T 1 n) UNIV"
        using has_sum_T[of 1 n] by (simp add: add_ac)
      have "((λj::int. q^n * ((-1) powi j * q powi (e 1 j - 2*j + 2) *
B (2*n+1) (n+2*j-1)))
                       has_sum (T 1 (Suc n) - q ^ Suc n * T 0 (Suc n)
- T 1 n)) UNIV"
        using has_sum_diff[OF 1 2]
        by (simp add: algebra_simps power_int_add power_int_diff power2_eq_square)
```

```
      also have "?this ⟷ ((λj::int. -(q^(Suc n) * ((-1) powi j * q
powi (e 1 (1-j) + 2*j - 1) *
                                        B (2*n+1) (n-2*j+1))))
                            has_sum (T 1 (Suc n) - q ^ Suc n * T 0 (Suc
n) - T 1 n)) UNIV"
        by (intro has_sum_reindex_bij_witness[of _ "λj. 1 - j" "λj. 1
- j"])
           (auto simp: power_int_diff power_int_add power_int_minus_left
algebra_simps power2_eq_square)
      also have "(λj. e 1 (1 - j) + 2 * j - 1) = (λj. e 1 j)"
      proof
        fix j :: int
        have "e 1 (1 - j) + 2 * j - 1 = ((1 - j) * (2 - 5 * j) + (4 *
j - 2)) div 2"
          unfolding e_def by (subst div_plus_div_distrib_dvd_left) auto
        also have "(1 - j) * (2 - 5 * j) + (4 * j - 2) = j * (5 * j -
3)"
          by (simp add: algebra_simps)
        also have "... div 2 = e 1 j"
          by (simp add: algebra_simps e_def)
        finally show "e 1 (1 - j) + 2 * j - 1 = e 1 j" .
      qed
      also have "(λj::int. B (int (2 * n + 1)) (int n - 2 * j + 1)) =

                 (λj. B (int (2*n+1)) (int n + 2*j))"
        by (subst B_sym) (auto simp: algebra_simps)
      finally have "((λj::int. (q^(Suc n) * ((-1) powi j * q powi e 1 j
* B (2*n+1) (n+2*j))))
                       has_sum -(T 1 (Suc n) - q ^ Suc n * T 0 (Suc n)
- T 1 n)) UNIV"
        by (simp only: has_sum_uminus)

      moreover have "((λj::int. (q^(Suc n) * ((-1) powi j * q powi e
1 j * B (2*n+1) (n+2*j))))
              has_sum (q^(Suc n) * T 1 n)) UNIV"
        using has_sum_T[of 1 n] by (intro has_sum_cmult_right) (simp_all
add: add_ac)
      ultimately have "-(T 1 (Suc n) - q ^ Suc n * T 0 (Suc n) - T 1 n)
= q^(Suc n) * T 1 n"
        using has_sum_unique by blast
      thus ?thesis
        by (simp add: algebra_simps)
    qed

    have S0_eq: "S 0 = T 0" and S1_eq: "S 1 = T 1"
    proof -
      have "S 0 n = T 0 n ∧ S 1 n = T 1 n" for n
        by (induction n) (simp_all add: S0_rec T0_rec S1_rec T1_rec)
      thus "S 0 = T 0" "S 1 = T 1"
```

```
      by blast+
  qed

  have B_lim1: "(λn. B (f n) (int m)) ⟶ 1 / qpochhammer m q q"
    if "filterlim f at_top at_top" for f :: "nat ⇒ int" and m :: nat
  proof -
    have f: "eventually (λn. f n ≥ 0) at_top"
      using that by (simp add: filterlim_at_top)
    have "(λn. qbinomial q n m) ⟶ 1 / qpochhammer m q q"
      by (rule tendsto_qbinomial1) (use q in auto)
    moreover have "filterlim (λn. nat (f n)) at_top at_top"
      by (rule filterlim_compose[OF _ that]) real_asymp
    ultimately have "(λn. qbinomial q (nat (f n)) m) ⟶ 1 / qpochhammer
m q q"
      by (rule filterlim_compose)
    also have "eventually (λn. qbinomial q (nat (f n)) m = B (f n) (int
m)) at_top"
      using f by eventually_elim (auto simp: B_def)
    hence "(λn. qbinomial q (nat (f n)) m) ⟶ 1 / qpochhammer m
q q ⟷ ?thesis"
      by (intro filterlim_cong) auto
    finally show ?thesis .
  qed

  have B_lim2: "(λn. B (f n) (g n)) ⟶ 1 / (q; q)∞"
    if "filterlim (λn. f n - g n) at_top at_top" "filterlim g at_top
at_top"
    for f g :: "nat ⇒ int"
  proof -
    have g_pos: "eventually (λn. g n > 0) at_top"
      using that(2) eventually_compose_filterlim eventually_gt_at_top
by blast
    have "eventually (λn. f n - g n > 0) at_top"
      using that(1) eventually_compose_filterlim eventually_gt_at_top
by blast
    hence fg: "eventually (λn. f n ≥ g n) at_top"
      by eventually_elim auto

    have "(λn. qbinomial q (nat (f n)) (nat (g n))) ⟶ 1 / (q;q)∞"
    proof (rule tendsto_qbinomial2)
      show "filterlim (λn. nat (g n)) at_top at_top"
        by (rule filterlim_compose[OF _ that(2)]) real_asymp
      have "filterlim (λn. nat (f n - g n)) at_top at_top"
        by (rule filterlim_compose[OF _ that(1)]) real_asymp
      also have "eventually (λn. nat (f n - g n) = nat (f n) - nat (g
n)) at_top"
        using fg g_pos by eventually_elim (auto simp: nat_diff_distrib)
      hence "filterlim (λn. nat (f n - g n)) at_top at_top ⟷
            filterlim (λn. nat (f n) - nat (g n)) at_top at_top"
```

```
        by (intro filterlim_cong) auto
      finally show "filterlim (λn. nat (f n) - nat (g n)) at_top at_top"
.
    qed (use q in auto)
    also have "eventually (λn. qbinomial q (nat (f n)) (nat (g n)) =
B (f n) (g n)) at_top"
      using g_pos fg by eventually_elim (auto simp: B_def)
    hence "(λn. qbinomial q (nat (f n)) (nat (g n))) ⟶ 1 / (q;q)∞
⟷
          (λn. B (f n) (g n)) ⟶ 1 / (q;q)∞"
      by (intro filterlim_cong) auto
    finally show ?thesis .
  qed
```

The q-binomial coefficient is bounded uniformly for all n, k (with a fixed q):

```
  define c where "c = (-norm q; norm q)∞ / (norm q ; norm q)∞"
  have B_bound: "norm (B n k) ≤ c" for n k
  proof (cases "0 ≤ k ∧ k ≤ n")
    case True
    thus ?thesis
      using norm_qbinomial_le_qpochhammer_inf[of q "nat n" "nat k"]
q
      by (auto simp: B_def c_def)
  next
    case False
    hence "norm (B n k) = 0"
      by auto
    also have "0 ≤ c"
      unfolding c_def using q by (auto intro!: divide_nonneg_nonneg
qpochhammer_inf_nonneg)
    finally show ?thesis .
  qed

  have uniform_limit1:
    "uniform_limit UNIV (λX n. ∑j∈X. q ^ (j² + N * j) * B (int n)
(int j)) (S N) finite_sets_at_top"
    for N
  proof (rule Weierstrass_m_test_general')
    fix n :: nat
    have "((λj. q ^ (j² + N * j) * B (int n) (int j)) has_sum S N n)
{..n}"
      by (intro has_sum_finiteI) (auto simp: S_def)
    also have "?this ⟷ ((λj. q ^ (j² + N * j) * B (int n) (int j))
has_sum S N n) UNIV"
      by (intro has_sum_cong_neutral) auto
    finally show "((λj. q ^ (j² + N * j) * B (int n) (int j)) has_sum
S N n) UNIV" .
  next
    fix j n :: nat
```

```
      have "norm (q ^ (j^2 + N*j) * B n j) = norm q ^ (j^2 + N*j) * norm
(B n j)"
        by (simp add: norm_power norm_mult)
      also have "... ≤ norm q ^ (j^2+N*j) * c"
        by (intro mult_left_mono norm_qbinomial_le B_bound) auto
      finally show "norm (q ^ (j^2+N*j) * B (int n) (int j)) ≤ c * norm
q ^ (j^2+N*j)"
        by (simp add: algebra_simps)
    next
      show "(λj. c * norm q ^ (j^2+N*j)) summable_on UNIV"
      proof (intro summable_on_cmult_right summable_nonneg_imp_summable_on)
        show "summable (λj. norm q ^ (j^2 + N*j))"
        proof (rule summable_comparison_test_bigo)
          show "(λj. norm q ^ (j^2+N*j)) ∈ O(λn. (1/2) ^ n)"
            using q by real_asymp
        qed auto
      qed auto
    qed

    have uniform_limit2:
      "uniform_limit UNIV (λJ n. ∑j∈J. (-1) powi j * q powi e N j *
B (2*n+N) (n+2*j)) (T N)
       finite_sets_at_top" for N
    proof (rule Weierstrass_m_test_general'[OF _ has_sum_T])
      fix j :: int and n :: nat
      have "norm ((-1) powi j * q powi e N j * B (2*int n+N) (int n +
2*j)) =
              norm q powi e N j * norm (B (2*int n+N) (int n + 2*j))"
        by (simp add: norm_mult norm_power_int)
      also have "... ≤ norm q powi e N j * c"
        by (intro mult_left_mono B_bound) auto
      finally show "norm ((-1) powi j * q powi e N j * B (2*int n+N) (int
n + 2*j)) ≤
                      c * norm q powi e N j"
        by (simp add: mult_ac)
    next
      have "(λj. norm q powi e N j) summable_on UNIV"
        using summable_rogers_ramanujan_aux1[of q "1-4*N"] q by (simp
add: e_def algebra_simps)
      thus "(λj. c * norm q powi e N j) summable_on UNIV"
        by (rule summable_on_cmult_right)
    qed

    have tendsto_S: "S N ⟶ (∑∞j. q ^ (j^2 + N*j) / qpochhammer j
q q)" for N
    proof (rule swap_uniform_limit'[OF _ _ uniform_limit1]; (intro always_eventually
allI)?)
    next
      fix X :: "nat set"
```

```
    have "(λn. ∑j∈X. q ^ (j²+N*j) * B (int n) (int j)) ⟶
            (∑j∈X. q ^ (j²+N*j) * (1 / qpochhammer j q q))"
      by (intro tendsto_intros B_lim1)
    thus "(λn. ∑j∈X. q ^ (j²+N*j) * B (int n) (int j)) ⟶
            (∑j∈X. q ^ (j²+N*j) / qpochhammer j q q)"
      by simp
  next
    have "summable (λj. norm (q ^ (j²+N*j) / qpochhammer (int j) q q))"
      using summable_rogers_ramanujan_aux2[of q N] q by simp
    hence "((λj. q ^ (j²+N*j) / qpochhammer j q q) has_sum
             (∑∞j. q ^ (j²+N*j) / qpochhammer j q q)) UNIV"
      by (intro has_sum_infsum norm_summable_imp_summable_on)
    thus "(sum (λj. q ^ (j²+N*j) / qpochhammer (int j) q q) ⟶
            (∑∞j. q ^ (j²+N*j) / qpochhammer j q q)) finite_sets_at_top"
      by (simp add: has_sum_def)
  qed auto

  have tendsto_T0: "T 0 ⟶ (1 / ((q; q^5)∞ * (q^4; q^5)∞))"
  proof (rule swap_uniform_limit'[OF _ _ uniform_limit2]; (intro always_eventually
allI)?)
    fix X :: "int set"
    show "(λn. ∑j∈X. (-1) powi j * q powi e 0 j * B (2*n+0) (n+2*j))
⟶
            (∑j∈X. (-1) powi j * q powi e 0 j * (1 / (q; q)∞))"
      by (intro tendsto_intros B_lim2) real_asymp+
  next
    have "((λx. (-1) powi x * q powi e 0 x) has_sum
            ((q; q)∞ / ((q; q^5)∞ * (q^4; q^5)∞))) UNIV"
      unfolding e_def using rogers_ramanujan_aux[of q 0] q by simp
    hence "((λj::int. (-1) powi j * q powi e 0 j * (1 / (q ; q)∞))

              has_sum ((q; q)∞ / ((q; q^5)∞ * (q^4; q^5)∞) * (1 /
(q ; q)∞))) UNIV"
      by (rule has_sum_cmult_left)
    also have "(q; q)∞ / ((q; q^5)∞ * (q^4; q^5)∞) * (1 / (q ; q)∞)
=
                1 / ((q; q^5)∞ * (q^4; q^5)∞)"
      by simp
    finally show "(sum (λj. (-1) powi j * q powi e 0 j * (1 / (q ; q)∞))
⟶
                  (1 / ((q; q^5)∞ * (q^4; q^5)∞))) finite_sets_at_top"
      by (simp add: has_sum_def)
  qed auto

  have tendsto_T1: "T 1 ⟶ (1 / ((q^2; q^5)∞ * (q^3; q^5)∞))"
  proof (rule swap_uniform_limit'[OF _ _ uniform_limit2]; (intro always_eventually
allI)?)
    fix X :: "int set"
    show "(λn. ∑j∈X. (-1) powi j * q powi e 1 j * B (2*n+1) (n+2*j))
```

```
⟶
            (∑j∈X. (-1) powi j * q powi e 1 j * (1 / (q; q)_∞))"
        by (intro tendsto_intros B_lim2) real_asymp+
    next
      have "((λj. (-1) powi j * q powi ((j*(5*j+3)) div 2)) has_sum
              ((q; q)_∞ / ((q^2; q^5)_∞ * (q^3; q^5)_∞))) UNIV"
        unfolding e_def using rogers_ramanujan_aux[of q 1] q
        by (simp add: algebra_simps eval_nat_numeral)
      also have "?this ⟷ ((λx. (-1) powi x * q powi e 1 x) has_sum
                           ((q; q)_∞ / ((q^2; q^5)_∞ * (q^3; q^5)_∞)))
UNIV"
      proof (intro has_sum_reindex_bij_witness[of _ uminus uminus] refl)
        fix j :: int
        have "(-1) powi j * q powi (j*(5*j+3) div 2) = (-1) powi (-j)
* q powi (j*(5*j+3) div 2)"
          by (auto simp: power_int_minus field_simps)
        moreover have "(j * (5 * j + 3)) div 2 = e 1 (-j)"
          unfolding e_def by (rule arg_cong[of _ _ "λn. n div 2"]) (simp_all
add: algebra_simps)
        ultimately show "(-1) powi -j * q powi e 1 (-j) = (-1) powi j
* q powi (j*(5*j+3) div 2)"
          by simp
      qed auto
      finally have "((λx. (-1) powi x * q powi e 1 x) has_sum
                     ((q; q)_∞ / ((q^2; q^5)_∞ * (q^3; q^5)_∞))) UNIV"
.
      hence "((λj::int. (-1) powi j * q powi e 1 j * (1 / (q ; q)_∞))

                has_sum ((q; q)_∞ / ((q^2; q^5)_∞ * (q^3; q^5)_∞) * (1
/ (q ; q)_∞))) UNIV"
        by (rule has_sum_cmult_left)
      also have "(q; q)_∞ / ((q^2; q^5)_∞ * (q^3; q^5)_∞) * (1 / (q ; q)_∞)
=
                   1 / ((q^2; q^5)_∞ * (q^3; q^5)_∞)"
        by simp
      finally show "(sum (λj. (-1) powi j * q powi e 1 j * (1 / (q ; q)_∞))
⟶
                     (1 / ((q^2; q^5)_∞ * (q^3; q^5)_∞))) finite_sets_at_top"
        by (simp add: has_sum_def)
    qed auto
```

Now we need only combine everything we have shown and we're done.

```
    have "(∑∞j. q ^ (j²) / qpochhammer j q q) = 1 / ((q;q^5)_∞ * (q^4;q^5)_∞)"
      using tendsto_S[of 0] tendsto_T0 LIMSEQ_unique
      unfolding of_nat_0 mult_0 add_0_right S0_eq S1_eq
      by blast
    moreover have "(λj. q ^ (j²) / qpochhammer j q q) summable_on UNIV"
      by (rule norm_summable_imp_summable_on)
         (use summable_rogers_ramanujan_aux2[of q 0] q in simp_all)
```

```
    ultimately have th1:
        "((λj. q ^ (j²) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞
* (q^4;q^5)∞))) UNIV"
      by (simp add: has_sum_iff)

    have "(∑∞j. q ^ (j² + j) / qpochhammer j q q) = 1 / ((q^2 ; q^5)∞
* (q^3 ; q^5)∞)"
      using tendsto_S[of 1] tendsto_T1 LIMSEQ_unique
      unfolding of_nat_1 mult_1 S0_eq S1_eq
      by blast
    moreover have "(λj. q ^ (j² + j) / qpochhammer j q q) summable_on
UNIV"
      by (rule norm_summable_imp_summable_on)
         (use summable_rogers_ramanujan_aux2[of q 1] q in simp_all)
    ultimately have th2:
       "((λj. q ^ (j² + j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
      by (simp add: has_sum_iff)

    from th1 and th2 show ?thesis
      by blast
  qed
  thus "((λj. q ^ (j²) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞ *
(q^4;q^5)∞))) UNIV"
   and "((λj. q ^ (j² + j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
    by blast+
qed

lemma rogers_ramanujan_real:
  fixes q :: real
  assumes "|q| < 1"
  shows   "((λj. q ^ (j²) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞
* (q^4;q^5)∞))) UNIV"
    and   "((λj. q ^ (j² + j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
proof -
  define q' where "q' = complex_of_real q"
  have [simp]: "norm q' < 1"
    using assms by (simp add: q'_def)

  have "((λj. q' ^ (j²) / qpochhammer j q' q') has_sum (1 / ((q';q'^5)∞
* (q'^4;q'^5)∞))) UNIV"
    by (rule rogers_ramanujan_complex) auto
  also have "(λj. q' ^ (j²) / qpochhammer j q' q') = (λj. of_real (q ^
(j²) / qpochhammer j q q))"
    by (simp add: q'_def qpochhammer_of_real)
  also have "(1 / ((q';q'^5)∞ * (q'^4;q'^5)∞)) = of_real (1 / ((q;q^5)∞
* (q^4;q^5)∞))"
```

```
    using assms by (simp add: q'_def power_abs power_less_one_iff flip:
qpochhammer_inf_of_real)
  finally show "((λj. q ^ (j^2) / qpochhammer j q q) has_sum (1 / ((q;q^5)∞
* (q^4;q^5)∞))) UNIV"
    by (subst (asm) has_sum_of_real_iff)

  have "((λj. q' ^ (j^2 + j) / qpochhammer j q' q') has_sum (1 / ((q'^2;q'^5)∞
* (q'^3;q'^5)∞))) UNIV"
    by (rule rogers_ramanujan_complex) auto
  also have "(λj. q' ^ (j^2 + j) / qpochhammer j q' q') = (λj. of_real
(q ^ (j^2+j) / qpochhammer j q q))"
    by (simp add: q'_def qpochhammer_of_real)
  also have "(1 / ((q'^2;q'^5)∞ * (q'^3;q'^5)∞)) = of_real (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))"
    using assms by (simp add: q'_def power_abs power_less_one_iff flip:
qpochhammer_inf_of_real)
  finally show "((λj. q ^ (j^2+j) / qpochhammer j q q) has_sum (1 / ((q^2;q^5)∞
* (q^3;q^5)∞))) UNIV"
    by (subst (asm) has_sum_of_real_iff)
qed

unbundle no_qpochhammer_inf_notation

end
```

References

[1] G. Andrews and K. Eriksson. *Integer Partitions*. Cambridge University Press, 2004.