

Combinatorial q -Analogues

Manuel Eberl

February 6, 2026

Abstract

This entry defines the q -analogues of various combinatorial symbols, namely:

- The q -bracket $[n]_q = \frac{1-q^n}{1-q}$ for $n \in \mathbb{Z}$
- The q -factorial $[n]_q! = [1]_q [2]_q \cdots [n]_q$ for $n \in \mathbb{Z}$
- The q -binomial coefficients $\binom{n}{k}_q = \frac{[n]_q!}{[k]_q! [n-k]_q!}$ for $n, k \in \mathbb{N}$ (also known as Gaussian binomial coefficients or Gaussian polynomials)
- The infinite q -Pochhammer symbol $(a; q)_\infty = \prod_{n=0}^{\infty} (1 - aq^n)$
- Euler's ϕ function $\phi(q) = (q; q)_\infty$
- The finite q -Pochhammer symbol $(a; q)_n = (a; q)_\infty / (aq^n; q)_\infty$ for $n \in \mathbb{Z}$

Proofs for many basic properties are provided, notably for the q -binomial theorem:

$$(-a; q)_n = \prod_{k=0}^{n-1} (1 + aq^k) = \sum_{k=0}^n \binom{n}{k}_q a^k q^{k(k-1)/2}$$

Additionally, two identities of Euler are formalised that give power series expansions for $(a; q)_\infty$ and $1/(a; q)_\infty$ in powers of a :

$$(a; q)_\infty = \prod_{k=0}^{\infty} (1 - aq^k) = \sum_{n=0}^{\infty} \frac{(-a)^n q^{n(n-1)/2}}{(1-q) \cdots (1-q^n)}$$
$$\frac{1}{(a; q)_\infty} = \prod_{k=0}^{\infty} \frac{1}{1 - aq^k} = \sum_{n=0}^{\infty} \frac{a^n}{(1-q) \cdots (1-q^n)}$$

Contents

1	Auxiliary material	3
1.1	Additional facts about infinite products	3
2	q-analogues of basic combinatorial symbols	4
2.1	The q -bracket $[n]_q$	5
2.2	The q -factorial $[n]_q!$	12
2.3	q -binomial coefficients $\binom{n}{k}_q$	16
2.4	The Gaussian polynomials	21
2.5	The finite Pochhammer symbol $(a; q)_n$	24
3	Primitive roots in an integral domain	39
4	The infinite q-Pochhammer symbol $(a; q)_\infty$	44
4.1	Definition and basic properties	44
4.2	Uniform convergence and its consequences	47
4.3	Bounds for $(a; q)_n$ and $\binom{n}{k}_q$ in terms of $(a; q)_\infty$	52
4.4	Limits of the q -binomial coefficients	54
4.5	Useful identities	56
4.6	Two series expansions by Euler	61
5	q-binomial identities	70
5.1	The q -binomial theorem	70
5.2	The infinite q -binomial theorem	72
5.3	The q -Vandermonde identity	75
6	Euler's function	80
6.1	Definition and basic properties	80
6.2	Connection to Lambert series	82
6.3	Logarithmic derivative	84

1 Auxiliary material

1.1 Additional facts about infinite products

```
theory More_Infinite_Products
  imports "HOL-Analysis.Analysis"
begin
```

```
lemma has_prod_group_nonzero:
```

```
  fixes f :: "nat  $\Rightarrow$  'a :: {semidom, t2_space}"
  assumes "f has_prod P" "k > 0" "P  $\neq$  0"
  shows "( $\lambda n. (\prod_{i \in \{n*k..<n*k+k\}}. f i))$  has_prod P"
```

```
proof -
```

```
  have "( $\lambda n. \prod_{k < n}. f k$ )  $\longrightarrow$  P"
    using assms(1) by (intro has_prod_imp_tendsto')
  hence "( $\lambda n. \prod_{k < n*k}. f k$ )  $\longrightarrow$  P"
    by (rule filterlim_compose) (use <k > 0 in real_asymp)
  also have "( $\lambda n. \prod_{k < n*k}. f k$ ) = ( $\lambda n. \prod_{m < n}. \text{prod } f \{m*k..<m*k+k\}$ )"
    by (subst prod_nat_group [symmetric]) auto
  finally have "( $\lambda n. \prod_{m \leq n}. \text{prod } f \{m*k..<m*k+k\}$ )  $\longrightarrow$  P"
    by (subst (asm) LIMSEQ_lessThan_iff_atMost)
  hence "raw_has_prod ( $\lambda n. \text{prod } f \{n*k..<n*k+k\}$ ) 0 P"
    using <P  $\neq$  0> by (auto simp: raw_has_prod_def)
  thus ?thesis
    by (auto simp: has_prod_def)
```

```
qed
```

```
lemma has_prod_group:
```

```
  fixes f :: "nat  $\Rightarrow$  'a :: real_normed_field"
  assumes "f has_prod P" "k > 0"
  shows "( $\lambda n. (\prod_{i \in \{n*k..<n*k+k\}}. f i))$  has_prod P"
```

```
proof (rule convergent_prod_tendsto_imp_has_prod)
```

```
  have "( $\lambda n. \prod_{k < n}. f k$ )  $\longrightarrow$  P"
    using assms(1) by (intro has_prod_imp_tendsto')
  hence "( $\lambda n. \prod_{k < n*k}. f k$ )  $\longrightarrow$  P"
    by (rule filterlim_compose) (use <k > 0 in real_asymp)
  also have "( $\lambda n. \prod_{k < n*k}. f k$ ) = ( $\lambda n. \prod_{m < n}. \text{prod } f \{m*k..<m*k+k\}$ )"
    by (subst prod_nat_group [symmetric]) auto
  finally show "( $\lambda n. \prod_{m \leq n}. \text{prod } f \{m*k..<m*k+k\}$ )  $\longrightarrow$  P"
    by (subst (asm) LIMSEQ_lessThan_iff_atMost)
```

```
next
```

```
  from assms obtain N P' where prod1: "raw_has_prod f N P'"
    by (auto simp: has_prod_def)
```

```
  define N' where "N' = nat [real N / real k]"
```

```
  have "k * N'  $\geq$  N"
```

```
proof -
```

```
  have "(real N / real k * real k)  $\leq$  real (N' * k)"
```

```
    unfolding N'_def of_nat_mult by (intro mult_right_mono) (use <k > 0 in auto)
```

```

    also have "real N / real k * real k = real N"
      using <k > 0> by simp
    finally show ?thesis
      by (simp only: mult.commute of_nat_le_iff)
  qed

  obtain P'' where prod2: "raw_has_prod f (k * N') P''"
    using prod1 <k * N' ≥ N> by (rule raw_has_prod_ignore_initial_segment)
  hence "P'' ≠ 0"
    by (auto simp: raw_has_prod_def)
  from prod2 have "raw_has_prod (λn. f (n + k * N')) 0 P''"
    by (simp add: raw_has_prod_def)
  hence "(λn. f (n + k * N')) has_prod P''"
    by (auto simp: has_prod_def)
  hence "(λn. ∏i=n*k..<n*k+k. f (i + k * N')) has_prod P''"
    by (rule has_prod_group_nonzero) fact+
  hence "convergent_prod (λn. ∏i=n*k..<n*k+k. f (i + k * N'))"
    using has_prod_iff by blast
  also have "(λn. ∏i=n*k..<n*k+k. f (i + k * N')) = (λn. ∏i=(n+N')*k..<(n+N')*k+k.
  f i)"
  proof
    fix n :: nat
    show "(∏i=n*k..<n*k+k. f (i + k * N')) = (∏i=(n+N')*k..<(n+N')*k+k.
  f i)"
      by (rule prod.reindex_bij_witness[of _ "λn. n - k*N'" "λn. n +
  k*N'"])
      (auto simp: algebra_simps)
  qed
  also have "convergent_prod ... ↔ convergent_prod (λn. (∏i=n*k..<n*k+k.
  f i))"
    by (rule convergent_prod_iff_shift)
  finally show "convergent_prod (λn. prod f {n * k..<n * k + k})" .
  qed

end

```

2 q -analogues of basic combinatorial symbols

```

theory Q_Analogues
imports "HOL-Complex_Analysis.Complex_Analysis"
begin

```

Various mathematical operations have generalisations in the form of q -analogues, usually in the sense that one recovers the original notion if we let $q \rightarrow 1$.

2.1 The q -bracket $[n]_q$

The q -bracket $[n]_q = \frac{1-q^n}{1-q}$ is the q -analogue of an integer n . The q -bracket has a removable singularity at $q = 1$ with $\lim_{q \rightarrow 1} [n]_q = n$.

definition `qbracket` :: "'a \Rightarrow int \Rightarrow 'a :: field" **where**

`"qbracket q n = (if q = 1 then of_int n else (1 - q powi n) / (1 - q))"`

lemma `qbracket_1_left [simp]`: `"qbracket 1 n = of_int n"`

`by (simp add: qbracket_def)`

lemma `qbracket_0_0 [simp]`: `"qbracket 0 0 = 0"`

`by (auto simp: qbracket_def power_int_0_left_if)`

lemma `qbracket_0_nonneg [simp]`: `"n \neq 0 \implies qbracket 0 n = 1"`

`by (auto simp: qbracket_def power_int_0_left_if)`

lemma `qbracket_0_left`: `"qbracket 0 n = (if n = 0 then 0 else 1)"`

`by auto`

lemma `qbracket_0 [simp]`: `"qbracket q 0 = 0"`

`by (simp add: qbracket_def)`

lemma `qbracket_1 [simp]`: `"qbracket q 1 = 1"`

`by (simp add: qbracket_def)`

lemma `qbracket_2 [simp]`: `"qbracket q 2 = 1 + q"`

`by (simp add: qbracket_def field_simps power2_eq_square)`

lemma `qbracket_of_real`: `"qbracket (of_real q :: 'a :: real_field) n = of_real (qbracket q n)"`

`by (simp add: qbracket_def)`

lemma `qbracket_minus`:

`assumes "q = 0 \longrightarrow n = 0"`

`shows "qbracket q (-n) = -qbracket (inverse q) n / q"`

proof (cases "q = 1")

`case True`

`thus ?thesis by auto`

next

`case False`

`have "qbracket q (-n) = qbracket (inverse q) n * (1 - 1 / q) / (1 - q)"`

`using assms False by (auto simp add: qbracket_def power_int_minus divide_simps)`

`also have "... = -qbracket (inverse q) n / q"`

`using assms False by (simp add: divide_simps) (auto simp: field_simps qbracket_0_left)`

`finally show ?thesis .`

qed

```

lemma qbracket_inverse:
  assumes "q = 0  $\longrightarrow$  n = 0"
  shows "qbracket (inverse q) n = -q * qbracket q (-n)"
  using assms by (cases "q = 0") (auto simp: qbracket_minus qbracket_0_left)

```

```

lemma qbracket_nonneg_altdef: "n  $\geq$  0  $\implies$  qbracket q n = ( $\sum$  k<nat n. q ^ k)"
  by (auto simp: qbracket_def sum_gp_strict power_int_def)

```

```

lemma qbracket_nonpos_altdef:
  assumes n: "n  $\leq$  0" and [simp]: "q  $\neq$  0"
  shows "qbracket q n = -(q powi n * ( $\sum$  k<nat (-n). q ^ k))"
proof -
  have "qbracket q n = qbracket q (-(-n))"
    by simp
  also have "... = -qbracket (inverse q) (-n) / q"
    by (intro qbracket_minus) auto
  also have "... = -( $\sum$  k<nat (-n). inverse q ^ k) / q"
    using n by (subst qbracket_nonneg_altdef) auto
  also have "... = -( $\sum$  k<nat (-n). q powi (-(k+1)))"
    by (simp add: sum_divide_distrib field_simps power_int_diff)
  also have "( $\sum$  k<nat (-n). q powi (-(k+1))) = ( $\sum$  k<nat (-n). q powi (n + k))"
    by (intro sum.reindex_bij_witness[of _ "\lambda k. nat (-n) - k - 1" "\lambda k. nat (-n) - k - 1"])
    (auto simp: of_nat_diff)
  also have "... = q powi n * ( $\sum$  k<nat (-n). q ^ k)"
    by (simp add: power_int_add sum_distrib_left sum_distrib_right)
  finally show ?thesis .
qed

```

```

lemma norm_qbracket_le:
  fixes q :: "'a :: real_normed_field"
  assumes "n  $\geq$  0" "norm q  $\leq$  1"
  shows "norm (qbracket q n)  $\leq$  real_of_int n"
proof -
  have "norm (qbracket q n) = norm (sum (\lambda k. q ^ k) {.. $\text{nat } n$ })"
    using assms by (simp add: qbracket_nonneg_altdef)
  also have "...  $\leq$  ( $\sum$  k<nat n. norm q ^ k)"
    by (rule sum_norm_le) (simp_all add: norm_power)
  also have "...  $\leq$  ( $\sum$  k<nat n. 1 ^ k)"
    using assms by (intro sum_mono power_mono) auto
  finally show ?thesis
    using assms by simp
qed

```

```

lemma qbracket_add:
  assumes "q  $\neq$  0  $\vee$  (k + 1 = 0  $\longrightarrow$  k = 0)"

```

```

shows "qbracket q (k + 1) = qbracket q 1 * q powi k + qbracket q k"
using assms
by (cases "q = 0")
  (auto simp: qbracket_def divide_simps power_int_add power_int_diff
    power_int_0_left_if add_eq_0_iff,
    (simp add: algebra_simps)?)

lemma qbracket_diff:
  assumes "q ≠ 0 ∨ (k = 1 → k = 0)"
  shows "qbracket q (k - 1) = qbracket q (-1) * q powi k + qbracket q
k"
  using assms qbracket_add[of q k "-1"] by simp

lemma qbracket_diff':
  assumes "q ≠ 0 ∨ (k = 1 → k = 0)"
  shows "qbracket q (k - 1) = qbracket q k * q powi -1 + qbracket q
(-1)"
  using assms qbracket_add[of q "-1" k] by simp

lemma qbracket_plus1: "q ≠ 0 ∨ n ≠ -1 ⇒ qbracket q (n + 1) = qbracket
q n + q powi n"
  by (subst qbracket_add) (auto simp: add_eq_0_iff)

lemma qbracket_rec: "q ≠ 0 ∨ n ≠ 0 ⇒ qbracket q n = qbracket q (n-1)
+ q powi (n-1)"
  using qbracket_plus1[of q "n-1"] by simp

lemma qbracket_eq_0_iff:
  fixes q :: "'a :: field"
  shows "qbracket q n = 0 ↔ (q = 1 ∧ of_int n = (0 :: 'a)) ∨ (q
≠ 1 ∧ q powi n = 1)"
  by (auto simp: qbracket_def)

lemma continuous_on_qbracket [continuous_intros]:
  fixes q :: "'a::topological_space ⇒ 'b :: real_normed_field"
  assumes [continuous_intros]: "continuous_on A q"
  assumes "∧x. n < 0 ⇒ x ∈ A ⇒ q x ≠ 0"
  shows "continuous_on A (λx. qbracket (q x) n)"
proof (cases "n ≥ 0")
  case True
  thus ?thesis
  by (auto simp: qbracket_nonneg_altdef intro!: continuous_intros)
next
  case False
  thus ?thesis using assms(2)
  by (auto simp: qbracket_nonpos_altdef intro!: continuous_intros)
qed

lemma tendsto_qbracket [tendsto_intros]:

```

```

fixes q :: "'a::topological_space ⇒ 'b :: real_normed_field"
assumes "(q ⟶ Q) F"
assumes "n < 0 ⟹ Q ≠ 0"
shows "((λx. qbracket (q x) n) ⟶ qbracket Q n) F"
proof -
  have "continuous_on (if n < 0 then -{0} else UNIV) (λx. qbracket x n
  :: 'b)"
    by (intro continuous_intros) auto
  moreover have "Q ∈ (if n < 0 then -{0} else UNIV)"
    using assms(2) by auto
  moreover have "open (if n < 0 then -{0::'b} else UNIV)"
    by auto
  ultimately have "isCont (λx. qbracket x n :: 'b) Q"
    using continuous_on_eq_continuous_at by blast
  with assms(1) show ?thesis
    using continuous_within_tendsto_compose' by force
qed

lemma continuous_qbracket [continuous_intros]:
  fixes q :: "'a::t2_space ⇒ 'b :: real_normed_field"
  assumes "continuous F q"
  assumes "n < 0 ⟹ q (netlimit F) ≠ 0"
  shows "continuous F (λx. qbracket (q x) n)"
  using assms unfolding continuous_def by (intro tendsto_intros) auto

lemma has_field_derivative_qbracket_real [derivative_intros]:
  fixes q :: real
  assumes "q ≠ 0 ∨ n ≥ 0"
  defines "D ≡ (if q = 1 then of_int (n * (n - 1)) / 2
    else (1 - q powi n)/(1-q)^2 - of_int n * q powi (n-1)
  / (1-q))"
  shows "((λq. qbracket q n) has_field_derivative D) (at q within A)"
proof (cases "q = 1")
  case False
  have "((λq. (1 - q powi n) / (1 - q)) has_field_derivative D) (at q
  within A)"
    unfolding D_def using assms(1) False
    by (auto intro!: derivative_eq_intros simp: divide_simps power2_eq_square)
  also have ev: "eventually (λq. q ≠ 1) (at q within A)"
    using False eventually_neq_at_within by blast
  have "((λq. (1 - q powi n) / (1 - q)) has_field_derivative D) (at q
  within A) ⟷
    ((λq. qbracket q n) has_field_derivative D) (at q within A)"
    by (intro has_field_derivative_cong_eventually eventually_mono[OF
  ev]) (auto simp: qbracket_def False)
  finally show ?thesis .
next
  case True
  have ev: "eventually (λq::real. q > 0) (at 1)"

```

```

    by real_asymp
    have "(λq::real. ((1 - q powr n) / (1 - q) - of_int n) / (q - 1)) -1→
of_int (n * (n - 1)) / 2"
    by real_asymp
    also have "?this ↔ (λq::real. ((1 - q powi n) / (1 - q) - of_int
n) / (q - 1)) -1→ of_int (n * (n - 1)) / 2"
    by (intro tendsto_cong) (use ev in eventually_elim, auto simp: powr_real_of_int')
    also have "... ↔ ((λy. (qbracket y n - qbracket q n) / (y - q)) →
D) (at q)"
    unfolding D_def True
    by (intro filterlim_cong eventually_mono[OF eventually_neq_at_within[of
1]])
        (auto simp: qbracket_def)
    finally show ?thesis
    unfolding has_field_derivative_iff using Lim_at_imp_Lim_at_within
by blast
qed

lemma has_field_derivative_qbracket_complex [derivative_intros]:
  fixes q :: complex
  assumes "q ≠ 0 ∨ n ≥ 0"
  defines "D ≡ (if q = 1 then of_int (n * (n - 1)) / 2
                else (1 - q powi n)/(1-q)^2 - of_int n * q powi (n-1)
/ (1-q))"
  shows "(λq. qbracket q n) has_field_derivative D) (at q within A)"
proof (cases "q = 1")
  case False
  have "(λq. (1 - q powi n) / (1 - q)) has_field_derivative D) (at q
within A)"
    unfolding D_def using assms(1) False
    by (auto intro!: derivative_eq_intros simp: divide_simps power2_eq_square)
  also have ev: "eventually (λq. q ≠ 1) (at q within A)"
    using False eventually_neq_at_within by blast
  have "(λq. (1 - q powi n) / (1 - q)) has_field_derivative D) (at q
within A) ↔
    ((λq. qbracket q n) has_field_derivative D) (at q within A)"
    by (intro has_field_derivative_cong_eventually eventually_mono[OF
ev]) (auto simp: qbracket_def False)
  finally show ?thesis .
next
case True
define F :: "complex fps"
  where "F = fps_binomial (of_int n) - 1 - of_int n * fps_X"
have F: "(λw. ((1 - (1+w) powi n) / (1 - (1+w)) - of_int n) / ((1+w)
- 1)) has_laurent_expansion
    fls_shift 2 (fps_to_fls F)"
  by (rule has_laurent_expansion_schematicI, (rule laurent_expansion_intros)+)
  (simp_all flip: fls_of_int fls_divide_fps_to_fls
    add: fls_times_fps_to_fls fls_X_times_conv_shift one_plus_fls_X_powi_eq

```

```

F_def)
  have F': "fls_subdegree (fls_shift 2 (fps_to_fls F)) ≥ 0"
  proof (cases "F = 0")
    case [simp]: False
      hence "subdegree F ≥ 2"
        by (intro subdegree_geI) (auto simp: F_def numeral_2_eq_2 less_Suc_eq)
      thus ?thesis
        by (intro fls_shift_nonneg_subdegree) (simp add: fls_subdegree_fls_to_fps)
    qed auto

  have "(λw. ((1 - w powi n) / (1 - w) - complex_of_int n) / (w - 1))
  -1→
    fls_nth (fls_shift 2 (fps_to_fls F)) 0"
    using has_laurent_expansion_imp_tendsto[OF F F'] .
  also have "fls_nth (fls_shift 2 (fps_to_fls F)) 0 = of_int (n * (n -
  1)) / 2"
    by (simp add: F_def numeral_2_eq_2 gbinomial_Suc_rec)
  finally have "(λq :: complex. ((1 - q powi n) / (1 - q) - of_int n) /
  (q - 1)) -1→ of_int (n * (n - 1)) / 2" .
  also have "?this ↔ ((λy. (qbracket y n - qbracket q n) / (y - q))
  → D) (at q)"
    unfolding D_def True
    by (intro filterlim_cong eventually_mono[OF eventually_neq_at_within[of
  1]])
    (auto simp: qbracket_def)
  finally show ?thesis
    unfolding has_field_derivative_iff using Lim_at_imp_Lim_at_within
  by blast
  qed

lemma holomorphic_on_qbracket [holomorphic_intros]:
  assumes "q holomorphic_on A"
  assumes "∧x. n < 0 ⇒ x ∈ A ⇒ q x ≠ 0"
  shows "(λx. qbracket (q x) n) holomorphic_on A"
  proof -
    have "(λx. qbracket x n) holomorphic_on (if n < 0 then -{0} else UNIV)"
      by (subst holomorphic_on_open) (auto intro!: derivative_eq_intros)
    hence "(λx. qbracket x n) ∘ q holomorphic_on A"
      by (intro holomorphic_on_compose_gen) (use assms in auto)
    thus ?thesis
      by (simp add: o_def)
  qed

lemma analytic_on_qbracket [analytic_intros]:
  assumes "q analytic_on A"
  assumes "∧x. n < 0 ⇒ x ∈ A ⇒ q x ≠ 0"
  shows "(λx. qbracket (q x) n) analytic_on A"
  proof -
    have "(λx. qbracket x n) holomorphic_on (if n < 0 then -{0} else UNIV)"

```

```

    by (intro holomorphic_intros) auto
  hence "( $\lambda x. \text{qbracket } x \ n$ ) analytic_on (if  $n < 0$  then  $\{-0\}$  else UNIV)"
    by (subst analytic_on_open) auto
  hence "( $(\lambda x. \text{qbracket } x \ n) \circ q$ ) analytic_on A"
    by (intro analytic_on_compose_gen) (use assms in auto)
  thus ?thesis
    by (simp add: o_def)
qed

lemma meromorphic_on_qbracket [meromorphic_intros]:
  assumes "q meromorphic_on A"
  shows "( $\lambda x. \text{qbracket } (q \ x) \ n$ ) meromorphic_on A"
proof -
  have "( $\lambda x. \text{qbracket } (q \ x) \ n$ ) meromorphic_on {z}" if z: "z  $\in$  A" for z
  proof -
    have [meromorphic_intros]: "q meromorphic_on {z}"
      using assms by (rule meromorphic_on_subset) (use z in auto)
    show "( $\lambda x. \text{qbracket } (q \ x) \ n$ ) meromorphic_on {z}"
    proof (cases "eventually ( $\lambda x. q \ x \neq 1$ ) (at z)")
      case True
      have "( $\lambda x. (1 - q \ x \text{ powi } n) / (1 - q \ x)$ ) meromorphic_on {z}"
        by (intro meromorphic_intros)
      also have "eventually ( $\lambda x. (1 - q \ x \text{ powi } n) / (1 - q \ x) = \text{qbracket}$ 
(q x) n) (at z)"
        using True by eventually_elim (auto simp: qbracket_def)
      hence "( $\lambda x. (1 - q \ x \text{ powi } n) / (1 - q \ x)$ ) meromorphic_on {z}  $\longleftrightarrow$ 
( $\lambda x. \text{qbracket } (q \ x) \ n$ ) meromorphic_on {z}"
        by (intro meromorphic_on_cong) auto
      finally show ?thesis .
    case False
    next
      case False
      have "( $\lambda z. q \ z - 1$ ) meromorphic_on {z}"
        by (intro meromorphic_intros)
      with False have "eventually ( $\lambda x. q \ x = 1$ ) (at z)"
        using not_essential_frequently_0_imp_eventually_0[of " $\lambda z. q \ z - 1$ " z]
        by (auto simp: meromorphic_at_iff_frequently_def)
      hence "eventually ( $\lambda x. \text{qbracket } (q \ x) \ n = \text{of\_int } n$ ) (at z)"
        by eventually_elim auto
      hence "( $\lambda x. \text{qbracket } (q \ x) \ n$ ) meromorphic_on {z}  $\longleftrightarrow$  ( $\lambda \_. \text{of\_int}$ 
n) meromorphic_on {z}"
        by (intro meromorphic_on_cong) auto
      thus ?thesis
        by auto
    qed
  qed
  thus ?thesis
    using meromorphic_on_meromorphic_at by blast
qed

```

2.2 The q -factorial $[n]_q!$

Since the q -bracket gives us the q -analogue of an integer n , we can use this to recursively define the q -factorial $[n]_q!$. Again, letting $q \rightarrow 1$, we recover the “normal” factorial.

definition `qfact` :: "'a \Rightarrow int \Rightarrow 'a :: field" where

`"qfact q n = (if n < 0 then 0 else ($\prod_{k=1..n}$ qbracket q k))"`

lemma `qfact_1_of_nat [simp]`: `"qfact 1 (int n) = fact n"`

proof -

have `"qfact 1 (int n) = of_int ($\prod_{k=1..int\ n}$ k)"`

by `(simp add: qfact_def)`

also have `"($\prod_{k=1..int\ n}$ k) = ($\prod_{k=1..n}$ int k)"`

by `(intro prod.reindex_bij_witness[of _ int nat]) auto`

finally show `?thesis`

by `(simp add: fact_prod)`

qed

lemma `qfact_1_nonneg [simp]`: `"n \geq 0 \implies qfact 1 n = fact (nat n)"`

by `(subst qfact_1_of_nat [symmetric], subst int_nat_eq) auto`

lemma `qfact_neg [simp]`: `"n < 0 \implies qfact q n = 0"`

by `(simp add: qfact_def)`

lemma `qfact_0 [simp]`: `"qfact q 0 = 1"`

by `(simp add: qfact_def)`

lemma `qfact_1 [simp]`: `"qfact q 1 = 1"`

by `(simp add: qfact_def)`

lemma `qfact_2`: `"qfact q 2 = 1 + q"`

proof -

have `"{1..2::int} = {1, 2}"`

by `auto`

thus `?thesis`

by `(simp add: qfact_def)`

qed

lemma `qfact_of_real`: `"qfact (of_real q :: 'a :: real_field) n = of_real (qfact q n)"`

by `(simp add: qfact_def qbracket_of_real)`

lemma `qfact_plus1`: `"n \neq -1 \implies qfact q (n + 1) = qfact q n * qbracket q (n + 1)"`

unfolding `qfact_def` by `(simp add: add.commute atLeastAtMostPlus1_int_conv)`

lemma `qfact_rec`: `"n > 0 \implies qfact q n = qbracket q n * qfact q (n - 1)"`

using `qfact_plus1[of "n - 1" q]` by `auto`

```

lemma qfact_altdef: "q ≠ 1 ⇒ n ≥ 0 ⇒ qfact q n = (∏ k=1..n. 1 -
q powi k) * (1 - q) powi (-n)"
  by (auto simp: qfact_def qbracket_def prod_dividef power_int_def field_simps)

lemma qfact_int_def: "qfact q (int n) = (∏ k=1..n. qbracket q (int k))"
  unfolding qfact_def by (auto intro!: prod.reindex_bij_witness[of _ int
nat])

lemma qfact_eq_0_iff:
  fixes q :: "'a :: field_char_0"
  shows "qfact q n = 0 ⟷ n < 0 ∨ (q ≠ 1 ∧ (∃ k ∈ {1..nat n}. q ^ k
= 1))"
proof (cases "n < 0")
  case False
  have "qfact q (int m) = 0 ⟷ q ≠ 1 ∧ (∃ k ∈ {1..m}. q ^ k = 1)" for
m
  proof (cases "q = 1")
    case False
    show ?thesis
    proof (induction m)
      case (Suc m)
      have *: "int (Suc m) - 1 = int m"
        by simp
      have "(qfact q (int (Suc m)) = 0) ⟷ (q ^ Suc m = 1 ∨ (∃ k ∈ {1..m}.
q ^ k = 1))"
        using False by (simp add: qfact_rec Suc qbracket_eq_0_iff * del:
of_nat_Suc)
      also have "... ⟷ (∃ k ∈ {1..Suc m}. q ^ k = 1)"
        by (subst atLeastAtMostSuc_conv) auto
      finally show ?case using False by simp
    qed auto
  qed auto
  from this[of "nat n"] False show ?thesis
  by simp
qed auto

lemma qfact_eq_0_iff' [simp]:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q ≠ 1"
  shows "qfact q n = 0 ⟷ n < 0"
  using assms by (subst qfact_eq_0_iff) (auto dest: power_eq_1_iff)

lemma prod_neg_qbracket_conv_qfact:
  assumes [simp]: "q ≠ 0"
  shows "(∏ k=1..n. qbracket q (-int k)) = (-1)^n * qfact q n / q ^
((n+1) choose 2)"
proof (cases "q = 1")
  case [simp]: False
  have "(-1)^n * qfact q n / q ^ ((n+1) choose 2) =

```

```

      (∏ k=1..n. (1 - q ^ k) / (1 - q)) / ((-1) ^ n * q ^ (Suc n choose
2))"
    by (simp add: qbracket_def prod_dividef qfact_int_def power_int_minus
divide_simps)
    also have "(Suc n choose 2) = (∑ k=1..n. k)"
    by (induction n) (auto simp: choose_two)
    also have "(-1) ^ n * q ^ (∑ k=1..n. k) = (∏ k=1..n. -(q ^ k))"
    by (simp add: power_sum prod_uminus)
    also have "(∏ k=1..n. (1 - q ^ k) / (1 - q)) / (∏ k=1..n. -(q ^ k))
=
      (∏ k=1..n. (1 - q ^ k) / (1 - q) / (-(q ^ k)))"
    by (rule prod_dividef [symmetric])
    also have "... = (∏ k=1..n. qbracket q (-int k))"
    by (intro prod.cong refl) (auto simp: qbracket_def power_int_minus
divide_simps)
    finally show ?thesis ..
qed (auto simp: prod_uminus qfact_int_def)

```

```

lemma norm_qfact_le:
  fixes q :: "'a :: real_normed_field"
  assumes "n ≥ 0" "norm q ≤ 1"
  shows "norm (qfact q n) ≤ fact (nat n)"
proof -
  have "norm (qfact q n) = (∏ k=1..n. norm (qbracket q k))"
    using assms by (simp add: qfact_def prod_norm)
  also have "... ≤ (∏ k=1..n. real_of_int k)"
    using assms by (intro prod_mono norm_qbracket_le conjI) auto
  also have "... = of_nat (∏ k=1..nat n. k)"
    unfolding of_nat_prod by (intro prod.reindex_bij_witness[of _ int
nat]) auto
  also have "... = fact (nat n)"
    using assms by (simp add: fact_prod)
  finally show ?thesis .
qed

```

```

lemma continuous_on_qfact [continuous_intros]:
  fixes q :: "'a::topological_space ⇒ 'b :: real_normed_field"
  assumes [continuous_intros]: "continuous_on A q"
  shows "continuous_on A (λx. qfact (q x) n)"
proof (cases "n ≥ 0")
  case True
  thus ?thesis
    by (auto simp: qfact_def intro!: continuous_intros)
qed auto

```

```

lemma continuous_qfact [continuous_intros]:
  fixes q :: "'a::t2_space ⇒ 'b :: real_normed_field"
  assumes [continuous_intros]: "continuous F q"

```

```

    shows "continuous F ( $\lambda x. \text{qfact } (q \ x) \ n$ )"
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
      by (auto simp: qfact_def intro!: continuous_intros)
  qed auto

lemma tendsto_qfact [tendsto_intros]:
  fixes q :: "'a::topological_space  $\Rightarrow$  'b :: real_normed_field"
  assumes [tendsto_intros]: "(q  $\longrightarrow$  Q) F"
  shows " $((\lambda x. \text{qfact } (q \ x) \ n) \longrightarrow \text{qfact } Q \ n) F$ "
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
      by (auto simp: qfact_def intro!: tendsto_intros)
  qed auto

lemma holomorphic_on_qfact [holomorphic_intros]:
  assumes [holomorphic_intros]: "q holomorphic_on A"
  shows " $(\lambda x. \text{qfact } (q \ x) \ n)$  holomorphic_on A"
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
      by (auto simp: qfact_def intro!: holomorphic_intros)
  qed auto

lemma analytic_on_qfact [analytic_intros]:
  assumes [analytic_intros]: "q analytic_on A"
  shows " $(\lambda x. \text{qfact } (q \ x) \ n)$  analytic_on A"
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
      by (auto simp: qfact_def intro!: analytic_intros)
  qed auto

lemma meromorphic_on_qfact [meromorphic_intros]:
  assumes [meromorphic_intros]: "q meromorphic_on A"
  shows " $(\lambda x. \text{qfact } (q \ x) \ n)$  meromorphic_on A"
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
      by (auto simp: qfact_def intro!: meromorphic_intros)
  qed auto

```

2.3 q -binomial coefficients $\binom{n}{k}_q$

We can also define q -binomial coefficients in such a way that we will get

$$\binom{n}{k}_q = \frac{[n]_q!}{[k]_q! [n-k]_q!}$$

and therefore recover the “normal” binomial coefficients if we let $q \rightarrow 1$.

```

fun qbinomial :: "'a  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  'a :: field" where
  "qbinomial q n 0 = 1"
| "qbinomial q 0 (Suc k) = 0"
| "qbinomial q (Suc n) (Suc k) = q ^ Suc k * qbinomial q n (Suc k) + qbinomial
  q n k"

```

```

lemma qbinomial_induct [case_names zero_right zero_left step]:
  "( $\bigwedge$ n. P n 0)  $\implies$  ( $\bigwedge$ k. P 0 (Suc k))  $\implies$ 
  ( $\bigwedge$ n k. P n (Suc k)  $\implies$  P n k  $\implies$  P (Suc n) (Suc k))  $\implies$  P n k"
  by (induction_schema, pat_completeness, lexicographic_order)

```

```

lemma qbinomial_1_left [simp]: "qbinomial 1 n k = of_nat (binomial n
  k)"
  by (induction n k rule: qbinomial_induct) simp_all

```

```

lemma qbinomial_eq_0 [simp]: "k > n  $\implies$  qbinomial q n k = 0"
  by (induction q n k rule: qbinomial.induct) auto

```

```

lemma qbinomial_n_n [simp]: "qbinomial q n n = 1"
  by (induction n) simp_all

```

```

lemma qbinomial_0_left: "qbinomial 0 n k = (if k  $\leq$  n then 1 else 0)"
  by (induction n k rule: qbinomial_induct) auto

```

```

lemma qbinomial_0_left' [simp]: "k  $\leq$  n  $\implies$  qbinomial 0 n k = 1"
  by (simp add: qbinomial_0_left)

```

```

lemma qbinomial_0_middle: "qbinomial q 0 k = (if k = 0 then 1 else 0)"
  by (cases k) auto

```

```

lemma qbinomial_of_real: "qbinomial (of_real q :: 'a :: real_field) m
  n = of_real (qbinomial q m n)"
  by (induction m n rule: qbinomial_induct) simp_all

```

```

lemma qbinomial_qfact_lemma:
  assumes "k  $\leq$  n"
  shows "qfact q k * qfact q (int (n - k)) * qbinomial q n k = qfact
  q n"
  using assms
proof (induction q n k rule: qbinomial.induct)
  case (3 q n k)

```

```

show ?case
proof (cases "n = k")
  case False
  with "3.prem1" have "k < n"
  by auto
  hence "(qfact q (int (Suc k)) * qfact q (int (Suc n - Suc k)) * qbinomial
q (Suc n) (Suc k)) =
      qbracket q (int (n-k)) * q^(k+1) *
      (qfact q (Suc k) * qfact q (int (n-Suc k)) * qbinomial
q n (Suc k)) +
      (qbracket q (k+1) * (qfact q k * qfact q (int (n-k)) * qbinomial
q n k))"
  by (simp add: qfact_rec of_nat_diff algebra_simps)
  also have "qfact q (Suc k) * qfact q (int (n-Suc k)) * qbinomial q
n (Suc k) = qfact q (int n)"
  using <k < n> by (subst 3) auto
  also have "qbracket q (k+1) * (qfact q k * qfact q (int (n-k)) * qbinomial
q n k) =
      qbracket q (k+1) * qfact q (int n)"
  using <k < n> by (subst 3) auto
  also have "qbracket q (int (n - k)) * q^(k+1) * qfact q (int n) +
      qbracket q (int (k + 1)) * qfact q (int n) =
      (qbracket q (int (n - k)) * q^(k+1) + qbracket q (int
(k + 1))) * qfact q (int n)"
  by (simp add: algebra_simps)
  also have "qbracket q (int (n - k)) * q^(k+1) + qbracket q (int (k
+ 1)) =
      qbracket q (int n - int k) * q powi (int (k+1)) + qbracket
q (int (k+1))"
  using <k < n> by (simp add: power_int_add of_nat_diff)
  also have "... = qbracket q (int (k + 1) + (int n - int k))"
  by (rule qbracket_add [symmetric]) auto
  also have "... = qbracket q (int (Suc n))"
  by simp
  also have "... * qfact q (int n) = qfact q (int (Suc n))"
  by (simp add: qfact_rec)
  finally show ?thesis .
qed simp_all
qed simp_all

lemma qbinomial_qfact:
  fixes q :: "'a :: field_char_0"
  assumes "¬(∃k∈{1..n}. q ^ k = 1)"
  shows "qbinomial q n k = qfact q n / (qfact q k * qfact q (int n -
int k))"
proof (cases "k ≤ n")
  case True
  thus ?thesis using assms
  by (subst qbinomial_qfact_lemma[of k n q, symmetric])

```

```

      (auto simp add: qfact_eq_0_iff of_nat_diff divide_simps)
qed auto

lemma qbinomial_qfact':
  fixes q :: "'a :: real_normed_field"
  assumes "q = 1  $\vee$  norm q  $\neq$  1"
  shows "qbinomial q n k = qfact q n / (qfact q k * qfact q (int n -
int k))"
proof (cases "q = 1")
  case False
  thus ?thesis
    using assms by (subst qbinomial_qfact) (auto dest!: power_eq_1_iff)
next
  case True
  thus ?thesis
    by (cases "k  $\leq$  n") (auto simp: binomial_fact simp flip: of_nat_diff)
qed

lemma qbinomial_symmetric:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q  $\neq$  1" "k  $\leq$  n"
  shows "qbinomial q n (n - k) = qbinomial q n k"
  using assms by (subst (1 2) qbinomial_qfact') (auto simp: of_nat_diff)

lemma qbinomial_rec1:
  "n > 0  $\implies$  k > 0  $\implies$ 
  qbinomial q n k = q ^ k * qbinomial q (n - 1) k + qbinomial q (n
- 1) (k - 1)"
  by (cases n; cases k) auto

lemma qbinomial_rec2:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q  $\neq$  1" "n > 0" "k < n"
  shows "qbinomial q n k = (1 - q ^ n) / (1 - q ^ (n - k)) * qbinomial
q (n-1) k"
proof (cases "q = 0")
  case [simp]: False
  have *: "q ^ i = q ^ j  $\iff$  i = j" for i j
  proof
    assume "q ^ i = q ^ j"
    hence "norm (q ^ i) = norm (q ^ j)"
      by (rule arg_cong)
    hence "norm q ^ i = norm q ^ j"
      by (simp add: norm_power)
    thus "i = j"
      by (subst (asm) power_inject_exp') (use assms in auto)
  qed auto
show ?thesis using assms
  by (subst (1 2) qbinomial_qfact')

```

```

      (use assms
        in <simp_all add: divide_simps of_nat_diff power_int_diff qfact_rec
qbracket_eq_0_iff
          power_0_left qbracket_def power_diff Groups.diff_right_commute
*>)
qed (use assms in <auto simp: power_0_left>)

lemma qbinomial_rec3:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q ≠ 1" "k > 0" "k ≤ n"
  shows "qbinomial q n k = (1 - q ^ n) / (1 - q ^ k) * qbinomial q (n-1)
(k-1)"
  using assms
  by (subst (1 2) qbinomial_qfact')
    (auto simp: divide_simps of_nat_diff power_int_diff qfact_rec qbracket_eq_0_iff
      power_0_left qbracket_def power_diff dest: power_eq_1_iff)

lemma qbinomial_rec4:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q ≠ 1" "n > 0" "k > 0" "k ≤ n"
  shows "qbinomial q n k = (1 - q ^ (Suc n - k)) / (1 - q ^ k) * qbinomial
q n (k-1)"
proof (cases "q = 0")
  case False
  have "q ^ Suc n ≠ q ^ k"
  proof
    assume *: "q ^ Suc n = q ^ k"
    have "q ^ Suc n = q ^ (Suc n - k) * q ^ k"
      by (subst power_add [symmetric]) (use assms in simp)
    with * have "q ^ (Suc n - k) = 1"
      using assms False by (auto simp: power_0_left)
    thus False using assms by (auto dest: power_eq_1_iff)
  qed
  thus ?thesis
  using assms
  by (subst (1 2) qbinomial_qfact')
    (auto simp: divide_simps of_nat_diff power_int_diff qfact_rec qbracket_eq_0_iff
      power_0_left qbracket_def power_diff dest: power_eq_1_iff)
qed (use assms in <auto simp: power_0_left>)

lemmas qbinomial_Suc_Suc [simp del] = qbinomial.simps(3)

lemma qbinomial_Suc_Suc':
  fixes q :: "'a :: real_normed_field"
  assumes q: "norm q ≠ 1"
  shows "qbinomial q (Suc n) (Suc k) =
qbinomial q n (Suc k) + q^(n-k) * qbinomial q n k"
proof (cases "k < n")
  case True

```

```

have "qbinomial q (Suc n) (Suc k) = qbinomial q (Suc n) (Suc (n - Suc
k))"
  by (subst qbinomial_symmetric [symmetric]) (use True q in auto)
  also have "... = q ^ (n - k) * qbinomial q n (n - k) + qbinomial q n
(n - Suc k)"
  by (subst qbinomial_Suc_Suc) (use True in <simp_all del: power_Suc
add: Suc_diff_Suc>)
  also have "qbinomial q n (n - k) = qbinomial q n k"
  by (rule qbinomial_symmetric) (use q True in auto)
  also have "qbinomial q n (n - Suc k) = qbinomial q n (Suc k)"
  by (rule qbinomial_symmetric) (use q True in auto)
  finally show ?thesis by simp
qed (use assms in <auto simp: qbinomial_Suc_Suc>)

```

```

lemma continuous_on_qbinomial [continuous_intros]:
  fixes q :: "'a::topological_space ⇒ 'b :: real_normed_field"
  assumes [continuous_intros]: "continuous_on A q"
  shows "continuous_on A (λx. qbinomial (q x) m n)"
  by (induction m n rule: qbinomial_induct)
  (auto intro!: continuous_intros simp: qbinomial.simps)

```

```

lemma continuous_qbinomial [continuous_intros]:
  fixes q :: "'a::t2_space ⇒ 'b :: real_normed_field"
  assumes [continuous_intros]: "continuous F q"
  shows "continuous F (λx. qbinomial (q x) m n)"
  by (induction m n rule: qbinomial_induct)
  (auto intro!: continuous_intros simp: qbinomial.simps)

```

```

lemma tendsto_qbinomial [tendsto_intros]:
  fixes q :: "'a::topological_space ⇒ 'b :: real_normed_field"
  assumes [tendsto_intros]: "(q ⟶ Q) F"
  shows "((λx. qbinomial (q x) m n) ⟶ qbinomial Q m n) F"
  by (induction m n rule: qbinomial_induct)
  (auto intro!: tendsto_intros simp: qbinomial.simps)

```

```

lemma holomorphic_on_qbinomial [holomorphic_intros]:
  assumes [holomorphic_intros]: "q holomorphic_on A"
  shows "(λx. qbinomial (q x) m n) holomorphic_on A"
  by (induction m n rule: qbinomial_induct)
  (auto intro!: holomorphic_intros simp: qbinomial.simps)

```

```

lemma analytic_on_qbinomial [analytic_intros]:
  assumes [analytic_intros]: "q analytic_on A"
  shows "(λx. qbinomial (q x) m n) analytic_on A"
  by (induction m n rule: qbinomial_induct)
  (auto intro!: analytic_intros simp: qbinomial.simps)

```

```

lemma meromorphic_on_qbinomial [meromorphic_intros]:
  assumes [meromorphic_intros]: "q meromorphic_on A"
  shows "(λx. qbinomial (q x) m n) meromorphic_on A"
  by (induction m n rule: qbinomial_induct)
     (auto intro!: meromorphic_intros simp: qbinomial.simps)

```

2.4 The Gaussian polynomials

The q -binomial coefficient $\binom{n}{k}_q$ is a polynomial of degree $k(n-k)$ in q . These polynomials are often called the *Gaussian polynomials*.

```

fun gauss_poly :: "nat ⇒ nat ⇒ 'a :: comm_semiring_1 poly" where
  "gauss_poly n 0 = 1"
| "gauss_poly 0 (Suc k) = 0"
| "gauss_poly (Suc n) (Suc k) = monom 1 (Suc k) * gauss_poly n (Suc k)
+ gauss_poly n k"

```

```

lemma poly_gauss_poly [simp]:
  "poly (gauss_poly n k) q = qbinomial q n k"
  by (induction q n k rule: qbinomial_induct) (auto simp: poly_monom qbinomial_Suc_Suc)

```

```

lemma of_nat_coeff_gauss_poly [simp]: "of_nat (coeff (gauss_poly n k)
i) = coeff (gauss_poly n k) i"
  by (induction n k arbitrary: i rule: gauss_poly.induct) (auto simp:
coeff_monom_mult)

```

```

lemma of_int_coeff_gauss_poly [simp]: "of_int (coeff (gauss_poly n k)
i) = coeff (gauss_poly n k) i"
  by (induction n k arbitrary: i rule: gauss_poly.induct) (auto simp:
coeff_monom_mult)

```

```

lemma norm_coeff_gauss_poly [simp]:
  "norm (coeff (gauss_poly n k) i :: 'a :: {real_normed_algebra_1, comm_semiring_1})
=
  coeff (gauss_poly n k) i"

```

```

proof -
  have "norm (coeff (gauss_poly n k) i :: 'a) = norm (of_nat (coeff (gauss_poly
n k) i) :: 'a)"
    by (subst of_nat_coeff_gauss_poly) auto
  also have "... = coeff (gauss_poly n k) i"
    by (subst norm_of_nat) auto
  finally show ?thesis .

```

qed

```

lemmas gauss_poly_Suc_Suc [simp del] = gauss_poly.simps(3)

```

```

lemma gauss_poly_eq_0 [simp]: "k > n ⇒ gauss_poly n k = 0"
  by (induction n k rule: gauss_poly.induct) (auto simp: gauss_poly_Suc_Suc)

```

```

lemma coeff_0_gauss_poly [simp]: "k ≤ n ⇒ coeff (gauss_poly n k) 0

```

```

= 1"
  by (induction n k rule: gauss_poly.induct) (auto simp: gauss_poly_Suc_Suc
coeff_monom_mult)

lemma gauss_poly_eq_0_iff [simp]: "gauss_poly n k = 0  $\longleftrightarrow$  k > n"
proof (cases "k  $\leq$  n")
  case True
  hence "coeff (gauss_poly n k) 0  $\neq$  coeff 0 0"
  by auto
  hence "gauss_poly n k  $\neq$  0"
  by metis
  thus ?thesis using True
  by simp
qed auto

lemma gauss_poly_n_n [simp]: "gauss_poly n n = 1"
  by (induction n) (auto simp: gauss_poly_Suc_Suc)

lemma coeff_gauss_poly_nonneg: "coeff (gauss_poly n k :: 'a :: linordered_semidom
poly) i  $\geq$  0"
  by (induction n k arbitrary: i rule: gauss_poly.induct)
    (auto simp: gauss_poly_Suc_Suc coeff_monom_mult)

lemma coeff_gauss_poly_le:
  "coeff (gauss_poly n k :: 'a :: linordered_semidom poly) i  $\leq$  of_nat
(n choose k)"
proof (induction n k arbitrary: i rule: gauss_poly.induct)
  case (3 n k)
  show ?case
  proof (cases "i  $\geq$  Suc k")
    case True
    hence "coeff (gauss_poly (Suc n) (Suc k) :: 'a poly) i =
      coeff (gauss_poly n (Suc k)) (i - Suc k) + coeff (gauss_poly
n k) i"
    by (auto simp: gauss_poly_Suc_Suc coeff_monom_mult not_less)
    also have "...  $\leq$  of_nat (n choose Suc k) + of_nat (n choose k)"
    by (intro add_mono "3.IH")
    finally show ?thesis
    by (simp add: add_ac)
  next
  case False
  hence "coeff (gauss_poly (Suc n) (Suc k) :: 'a poly) i = coeff (gauss_poly
n k) i + 0"
  by (auto simp: gauss_poly_Suc_Suc coeff_monom_mult)
  also have "...  $\leq$  of_nat (n choose k) + of_nat (n choose Suc k)"
  by (intro add_mono "3.IH") auto
  finally show ?thesis
  by (simp add: add_ac)
qed

```

```

qed auto

lemma degree_gauss_poly: "degree (gauss_poly n k :: 'a :: idom poly)
= k * (n - k)"
proof (cases "k ≤ n")
  case True
    have "int (degree (gauss_poly n k :: 'a poly)) = int k * (int n - int
k)"
      using True
    proof (induction n k rule: gauss_poly.induct)
      case (3 n k)
        note [simp] = "3.IH"
        have "int (degree (gauss_poly (Suc n) (Suc k) :: 'a poly)) =
          int (degree (monom 1 (Suc k) * gauss_poly n (Suc k) + gauss_poly
n k :: 'a poly))"
          by (auto simp: gauss_poly_Suc_Suc)
        also have "... = (int k + 1) * (int n - int k)"
        proof (cases "n = k")
          case True
            thus ?thesis using 3 by auto
          next
            case False
              have "int (degree (monom (1::'a) (Suc k) * gauss_poly n (Suc k)))
=
          int (Suc k + degree (gauss_poly n (Suc k) :: 'a poly))"
            using False "3.prem1" by (subst degree_mult_eq) (auto simp: degree_monom_eq)
          also have "... = (int k + 1) * (int n - int k)"
            using False "3.prem1" by (simp add: algebra_simps)
          finally have deg1: "int (degree (monom (1::'a) (Suc k) * gauss_poly
n (Suc k))) =
          (int k + 1) * (int n - int k)" .
          have "int (degree (gauss_poly n k :: 'a poly)) <
          int (degree (monom (1::'a) (Suc k) * gauss_poly n (Suc k)))"
            using False "3.prem1" by (subst deg1) (auto simp: degree_mult_eq)
          thus ?thesis
            by (subst degree_add_eq_left) (use deg1 in auto)
        qed
      finally show ?case
        by (simp add: algebra_simps)
    qed auto
  also have "... = int (k * (n - k))"
    using True by (simp add: algebra_simps of_nat_diff)
  finally show ?thesis
    by linarith
qed auto

lemma norm_qbinomial_le_binomial:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q < 1"

```

```

  shows "norm (qbinomial q n k) ≤ real (n choose k) * (1 - norm q ^
(k*(n-k)+1)) / (1 - norm q)"
proof (cases "k ≤ n")
  case True
  have "qbinomial q n k = poly (gauss_poly n k) q"
    by simp
  also have "... = (∑ i≤k*(n-k). coeff (gauss_poly n k) i * q ^ i)"
    unfolding poly_altdef using True by (simp add: degree_gauss_poly)
  also have "norm ... ≤ (∑ i≤k*(n-k). norm (coeff (gauss_poly n k) i
* q ^ i))"
    by (rule norm_sum)
  also have "... = (∑ i≤k * (n - k). coeff (gauss_poly n k) i * norm
q ^ i)"
    by (simp add: norm_mult norm_power)
  also have "... ≤ (∑ i≤k*(n-k). (n choose k) * norm q ^ i)"
    by (intro sum_mono mult_right_mono power_mono coeff_gauss_poly_le)
auto
  also have "... = (n choose k) * (∑ i≤k * (n - k). norm q ^ i)"
    by (simp add: sum_distrib_left)
  also have "... = real (n choose k) * (1 - norm q ^ (k * (n - k) + 1))
/ (1 - norm q)"
    by (subst sum_gp0) (use assms in auto)
  finally show ?thesis .
qed auto

lemma norm_qbinomial_le_binomial':
  fixes q :: "'a :: real_normed_field"
  assumes "norm q < 1"
  shows "norm (qbinomial q n k) ≤ real (n choose k) / (1 - norm q)"
proof -
  have "norm (qbinomial q n k) ≤ real (n choose k) * (1 - norm q ^ (k*(n-k)+1))
/ (1 - norm q)"
    by (rule norm_qbinomial_le_binomial) fact+
  also have "... ≤ real (n choose k) * (1 - 0) / (1 - norm q)"
    by (intro mult_left_mono divide_right_mono diff_left_mono) (use assms
in auto)
  finally show ?thesis
    by simp
qed

```

2.5 The finite Pochhammer symbol $(a; q)_n$

The definition of the q -Pochhammer symbol is a bit less obvious. Recall that the ordinary Pochhammer symbol is defined as

$$a^{\overline{n}} = a(a+1)\cdots(a+n-1).$$

The q -Pochhammer symbol is defined as

$$(a; q)_n = (1-a)(1-aq)(1-aq^2)\cdots(1-aq^{n-1})$$

for $n \geq 0$. We extend the definition to $n < 0$ such that the recurrences that hold for $n \geq 0$ carry over to the negative domain as well. Effectively, what we do is to define

$$(a; q)_{-n} = \frac{1}{(aq^{-n}; q)_n}$$

definition `qpochhammer` :: "int \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a :: field" where
`"qpochhammer n a q =`
 `(if n \geq 0 then ($\prod_{k < \text{nat } n} (1 - a * q ^ k)$) else ($\prod_{k=1..nat (-n)} (-n).$`
`1 / (1 - a / qk))"`

Seeing in which way it is an analogue of the “normal” Pochhammer symbol $a^{\bar{n}} = a(a+1)\cdots(a+n-1)$ is more involved than for the other analogues: if we simply let $q = 1$, we merely get $(1-a)^n$.

However, we do have:

$$\lim_{q \rightarrow 1} \frac{(q^a; q)_\infty}{(1-q)^n} = a^{\bar{n}}$$

lemma `qpochhammer_tendsto_pochhammer`:

`"($\lambda q :: \text{real}.$ qpochhammer (int n) (q powr a) q / (1 - q) ^ n) -1 \rightarrow pochhammer a n"`

proof (rule `Lim_transform_eventually`)

`have " $(\lambda q. \prod_{k < n} (1 - q \text{ powr } (a + \text{int } k)) / (1 - q)) -1 \rightarrow (\prod_{k < n} a + \text{real } k)$ "`

`by (rule tendsto_prod) real_asymp`

`also have " $(\prod_{k < n} a + \text{real } k) = \text{pochhammer a n}$ "`

`by (simp add: pochhammer_prod atLeast0LessThan)`

`finally show " $(\lambda q. \prod_{k < n} (1 - q \text{ powr } (a + \text{int } k)) / (1 - q)) -1 \rightarrow \text{pochhammer a n}$ " .`

next

`have "eventually ($\lambda q. q \in \{0 < ..\} - \{1\}$) (at (1 :: real))"`

`by (intro eventually_at_in_open) auto`

`thus "eventually ($\lambda q. (\prod_{k < n} (1 - q \text{ powr } (a + \text{int } k)) / (1 - q)) = \text{qpochhammer (int n) (q powr a) q / (1 - q) ^ n}$)`

`(at 1)"`

`by eventually_elim (simp add: qpochhammer_def powr_add powr_realpow prod_dividef)`

qed

lemma `qpochhammer_nonneg_def`: "`qpochhammer (int n) a q = ($\prod_{k < n} (1 - a * q ^ k)$)"`

`by (simp add: qpochhammer_def)`

lemma `qpochhammer_0` [`simp`]: "`qpochhammer 0 a q = 1`"

`by (simp add: qpochhammer_def)`

lemma `qpochhammer_1` [`simp`]: "`qpochhammer 1 a q = 1 - a`"

`by (simp add: qpochhammer_def)`

```

lemma qpochhammer_1_right [simp]: "qpochhammer n a 1 = (1 - a) powi n"
  by (simp add: qpochhammer_def power_int_def field_simps)

lemma qpochhammer_neg1 [simp]: "q ≠ 0 ⇒ q ≠ a ⇒ qpochhammer (-1)
a q = q / (q - a)"
  by (simp add: qpochhammer_def divide_simps)

lemma qpochhammer_0_middle [simp]: "qpochhammer n 0 q = 1"
  by (simp add: qpochhammer_def)

lemma qpochhammer_0_right: "qpochhammer n a 0 = (if n > 0 then 1 - a
else 1)"
proof (cases "n ≥ 0")
  case False
  thus ?thesis
    by (auto simp: qpochhammer_def power_0_left)
next
  case True
  hence "qpochhammer n a 0 = (∏ k < nat n. 1 - a * (if k = 0 then 1 else
0))"
    by (auto simp add: qpochhammer_def power_0_left)
  also have "... = (∏ k ∈ (if n = 0 then {} else {0::nat}). 1 - a)"
    using True by (intro prod.mono_neutral_cong_right) (auto split: if_splits)
  also have "... = (if n > 0 then 1 - a else 1)"
    using True by auto
  finally show ?thesis .
qed

lemma qpochhammer_0_right_pos [simp]: "n > 0 ⇒ qpochhammer n a 0 =
1 - a"
  and qpochhammer_0_right_nonpos [simp]: "n ≤ 0 ⇒ qpochhammer n a 0
= 1"
  by (simp_all add: qpochhammer_0_right)

lemma qpochhammer_nat_eq_0_iff:
  "qpochhammer (int n) a q = 0 ↔ (∃ k < n. a * q ^ k = 1)"
proof -
  have "qpochhammer (int n) a q = (∏ k < n. 1 - a * q ^ k)"
    unfolding qpochhammer_def by simp
  also have "... = 0 ↔ (∃ k < n. a * q ^ k = 1)"
    by (simp add: Bex_def)
  finally show ?thesis .
qed

lemma qpochhammer_of_real:
  "qpochhammer n (of_real a :: 'a :: real_field) (of_real q) = of_real
(qpochhammer n a q)"
  by (simp add: qpochhammer_def)

```

```

lemma qpochhammer_eq_0_iff:
  "qpochhammer n a q = 0  $\longleftrightarrow$  ( $\exists k \in \{\min n 0 .. \max n 0\}. a * q \text{ powi } k = 1$ )"
proof (cases "n  $\geq$  0")
  case True
  define m where "m = nat n"
  have n_eq: "n = int m"
    using True by (auto simp: m_def)
  have "qpochhammer n a q = 0  $\longleftrightarrow$  ( $\exists k \in \{.. < m\}. a * q ^ k = 1$ )"
    by (simp add: n_eq qpochhammer_nat_eq_0_iff Bex_def)
  also have "bij_betw int {k  $\in$  {.. < m}. a * q ^ k = 1} {k  $\in$  {0.. < int m}. a * q powi k = 1}"
    by (rule bij_betwI[of _ _ _ nat]) (auto simp: power_int_def)
  hence "( $\exists k \in \{.. < m\}. a * q ^ k = 1$ )  $\longleftrightarrow$  ( $\exists k \in \{0.. < \text{int } m\}. a * q \text{ powi } k = 1$ )"
    by (rule bij_betw_imp_Bex_iff)
  finally show ?thesis
    by (simp add: n_eq)
next
  case False
  define m where "m = nat (-n)"
  have n_eq: "n = -int m" and "m > 0"
    using False by (auto simp: m_def)
  have "qpochhammer n a q = ( $\prod_{k=1..m}. 1 / (1 - a / q ^ k)$ )"
    using <m > 0> by (simp add: qpochhammer_def n_eq)
  also have "... = 0  $\longleftrightarrow$  ( $\exists k \in \{1..m\}. 1 / (1 - a / q ^ k) = 0$ )"
    by simp
  also have "...  $\longleftrightarrow$  ( $\exists k \in \{1..m\}. a / q ^ k = 1$ )"
    by (intro bex_cong) (use <m > 0> in auto)
  also have "bij_betw ( $\lambda k. -\text{int } k$ ) {k  $\in$  {1..m}. a / q ^ k = 1} {k  $\in$  {-int m .. < 0}. a * q powi k = 1}"
    by (rule bij_betwI[of _ _ _ "\lambda k. nat (-k)"]) (auto simp: power_int_def field_simps)
  hence "( $\exists k \in \{1..m\}. a / q ^ k = 1$ )  $\longleftrightarrow$  ( $\exists k \in \{-\text{int } m .. < 0\}. a * q \text{ powi } k = 1$ )"
    by (rule bij_betw_imp_Bex_iff)
  finally show ?thesis
    using <m > 0> by (simp add: n_eq)
qed

lemma qpochhammer_rec:
  assumes " $\bigwedge k. \text{int } k \in \{0 .. -n\} \implies q ^ k \neq a$ "
  shows "qpochhammer (n + 1) a q = qpochhammer n a q * (1 - a * q powi n)"
proof -
  consider "n  $\geq$  0" | "n = -1" | "n < 0"
  by linarith
  thus ?thesis
  proof cases

```

```

    assume "n = -1"
    thus ?thesis using assms[of 1]
      by (auto simp: qpochhammer_def field_simps)
next
  assume "n ≥ 0"
  thus ?thesis
    by (auto simp: qpochhammer_def nat_add_distrib power_int_def)
next
  assume n: "n < 0"
  hence "qpochhammer n a q = (∏ k=1..nat (-n). 1 / (1 - a / q ^ k))"
    by (auto simp: qpochhammer_def)
  also have "{1..nat (-n)} = insert (nat (-n)) {1..nat (-n-1)}"
    using n by auto
  also have "(∏ k∈... 1 / (1 - a / q ^ k)) =
    (∏ k=1..nat (-n-1). 1 / (1 - a / q ^ k)) * (1 / (1 -
a / q ^ nat (-n)))"
    by (subst prod.insert) auto
  also have "(∏ k=1..nat (-n-1). 1 / (1 - a / q ^ k)) = qpochhammer
(n + 1) a q"
    using n by (simp add: qpochhammer_def)
  also have "a / q ^ nat (-n) = a * q powi n"
    using n by (simp add: power_int_def field_simps)
  finally show ?thesis
    using assms[of "nat (-n)"] n by (auto simp: power_int_def field_simps)
qed
qed

lemma qpochhammer_plus1:
  assumes "n ≥ 0 ∨ x * q powi n ≠ 1"
  shows "qpochhammer (n + 1) x q = qpochhammer n x q * (1 - x * q powi
n)"
proof (cases "q = 0")
  case True
  thus ?thesis by (auto simp: qpochhammer_def power_0_left power_int_def
nat_add_distrib)
next
  case [simp]: False
  consider "n < -1" | "n = -1" | "n ≥ 0"
  by linarith
  thus ?thesis
proof cases
  assume "n < -1"
  define m where "m = nat (-n-1)"
  have n_eq: "n = -int m-1" and "m > 0"
    using <n < -1> by (simp_all add: m_def)
  show ?thesis using <m > 0> assms
    by (simp add: n_eq qpochhammer_def power_int_diff power_int_minus

nat_add_distrib divide_simps mult_ac)

```

```

next
  assume [simp]: "n = -1"
  show ?thesis using assms
    by (simp add: qpochhammer_def divide_simps)
next
  assume "n ≥ 0"
  define m where "m = nat n"
  have n_eq: "n = int m"
    using <n ≥ 0> by (simp add: m_def)
  show ?thesis using assms
    by (simp add: n_eq qpochhammer_def nat_add_distrib)
qed
qed

lemma qpochhammer_minus1:
  assumes "x * q powi (n - 1) ≠ 1"
  shows "qpochhammer (n - 1) x q = qpochhammer n x q / (1 - x * q powi
(n - 1))"
  using qpochhammer_plus1[of "n - 1" x q] assms by simp

lemma qpochhammer_1plus:
  assumes "n ≥ 0 ∨ x * q powi n ≠ 1"
  shows "qpochhammer (1 + n) x q = qpochhammer n x q * (1 - x * q powi
n)"
  using qpochhammer_plus1[OF assms] by (simp add: add_ac)

lemma qpochhammer_nat_add:
  fixes m n :: nat
  shows "qpochhammer (int m + int n) x q = qpochhammer (int m) x q * qpochhammer
n (q ^ m * x) q"
proof -
  have "qpochhammer (int m + int n) x q = (∏ k<m+n. 1 - x * q ^ k)"
    by (simp add: qpochhammer_def nat_add_distrib)
  also have "... = (∏ k∈{..

```

```

lemma qpochhammer_minus:
  assumes "n < 0 → q ≠ 0"
  shows "qpochhammer (-n) a q = 1 / qpochhammer n (a / q powi n) q"
proof (cases "q = 0")
  case [simp]: True
  from assms have "n ≥ 0"
  by auto
  thus ?thesis
  by (simp add: power_int_0_left_if)
next
  case [simp]: False

  show ?thesis
  proof (cases n "0::int" rule: linorder_cases)
    case n: less
    define m where "m = nat (-n)"
    have n_eq: "n = -int m"
    using n by (simp add: m_def)
    have "1 / qpochhammer n (a / q powi n) q =
      (∏ k=1..m. 1 - a / (q ^ k / q ^ m))"
    by (simp add: qpochhammer_def prod_dividef n_eq power_int_minus
  inverse_eq_divide)
    also have "... = (∏ k<m. 1 - a * q ^ k)"
    by (rule prod.reindex_bij_witness[of _ "λi. m - i" "λi. m - i"])

    (auto simp: power_diff)
    also have "... = qpochhammer (-n) a q"
    by (simp add: qpochhammer_def n_eq)
    finally show ?thesis ..
  next
    case n: greater
    define m where "m = nat n"
    have n_eq: "n = int m" and "m > 0"
    using n by (simp_all add: m_def)
    have "qpochhammer (-n) a q = 1 / (∏ k=1..m. 1 - a / q ^ k)"
    using <m > 0> by (simp add: qpochhammer_def prod_dividef n_eq)
    also have "(∏ k=1..m. 1 - a / q ^ k) = (∏ k<m. 1 - a * q ^ k / q ^
  m)"
    by (rule prod.reindex_bij_witness[of _ "λi. m - i" "λi. m - i"])

    (auto simp: power_diff)
    also have "1 / ... = 1 / qpochhammer n (a / q powi n) q"
    by (simp add: qpochhammer_def n_eq)
    finally show ?thesis .
  qed auto
qed

lemma qpochhammer_add:
  assumes "∧k. k ∈ {m+min n 0..<m+max n 0} ⇒ x * q powi k ≠ 1" and

```

```

[simp]: "q ≠ 0"
  shows "qpochhammer (m + n) x q = qpochhammer m x q * qpochhammer n
(q powi m * x) q"
proof -
  have *: "qpochhammer (m + int n) x q = qpochhammer m x q * qpochhammer
(int n) (q powi m * x) q"
  if "∀k<n. x * q powi (m + k) ≠ 1" for n :: nat and m :: int
  using that by (induction n) (auto simp: qpochhammer_1plus add_ac power_int_add)

show ?thesis
proof (cases "n ≥ 0")
  case True
  define n' where "n' = nat n"
  have n_eq: "n = int n'"
  using True by (simp add: n'_def)
  show ?thesis
  using *[of n' m] assms by (auto simp: n_eq)
next
  case False
  define n' where "n' = nat (-n)"
  have n_eq: "n = -int n'" and n': "n' > 0"
  using False by (simp_all add: n'_def)
  have "qpochhammer m x q = qpochhammer (m + n + int n') x q"
  by (simp add: n_eq)
  also have "... = qpochhammer (m + n) x q * qpochhammer (-n) (q powi
(m + n) * x) q"
  by (subst *) (use assms in <auto simp: n_eq>)
  also have "... = qpochhammer (m + n) x q / qpochhammer n (q powi m
* x) q"
  by (subst qpochhammer_minus) (use False in <auto simp: power_int_add>)
  finally have "qpochhammer m x q = qpochhammer (m + n) x q / qpochhammer
n (q powi m * x) q" .
  moreover have "qpochhammer n (q powi m * x) q ≠ 0"
  proof
    assume "qpochhammer n (q powi m * x) q = 0"
    then obtain k where k: "k ∈ {-int n'..<0}" "x * q powi (m + k)
= 1"
    using n' by (auto simp: n_eq qpochhammer_eq_0_iff power_int_add
mult_ac)
    moreover from k(1) have "m + k ∈ {m+min n 0..<m+max n 0}"
    using n' by (auto simp: n_eq)
    ultimately show False
    using k(2) assms by blast
  qed
  ultimately show ?thesis
  by (simp add: divide_simps power_int_add)
qed
qed

```

```

lemma qfact_conv_qpochhammer_aux:
  assumes "n < 0  $\longrightarrow$  q  $\neq$  0"
  shows "qpochhammer n q q = qfact q n * (1 - q) powi n"
proof (cases "q = 1")
  case q: False
  show ?thesis
  proof (cases "n  $\geq$  0")
    case True
    thus ?thesis
    proof (induction n rule: int_ge_induct)
      case base
      thus ?case by auto
    next
      case (step n)
      thus ?case using q
      by (subst qpochhammer_rec)
      (auto simp: qfact_plus1 power_int_diff qbracket_def power_int_add
      add_eq_0_iff2)
    qed
  qed (use assms in <auto simp: qpochhammer_def not_le intro: bexI[of
  _ 1]>)
qed (use assms in <auto simp: qpochhammer_def power_0_left qfact_def not_less>)

lemma qfact_conv_qpochhammer:
  assumes "if n  $\geq$  0 then q  $\neq$  1 else q  $\neq$  0"
  shows "qfact q n = qpochhammer n q q * (1 - q) powi (-n)"
  using qfact_conv_qpochhammer_aux[of n q] assms
  by (auto simp: power_int_minus divide_simps split: if_splits)

lemma qbinomial_conv_qpochhammer:
  fixes q :: "'a :: field_char_0"
  assumes "k  $\leq$  n"
  assumes " $\bigwedge$ k. 0 < k  $\implies$  k  $\leq$  n  $\implies$  q ^ k  $\neq$  1"
  shows "qbinomial q n k =
  qpochhammer (int n) q q / (qpochhammer (int k) q q * qpochhammer
  (int n - int k) q q)"
proof (cases "n = 0")
  case False
  with assms(2)[of 1] have [simp]: "q  $\neq$  1"
  by auto
  define P where "P = ( $\lambda$ n. qpochhammer (int n) q q)"
  have "qbinomial q n k = qfact q (int n) / (qfact q (int k) * qfact q
  (int n - int k))"
  using assms by (subst qbinomial_qfact) (use assms in auto)
  also have "... = P n / (P k * P (n - k))"
  by (subst (1 2 3) qfact_conv_qpochhammer)
  (use <k  $\leq$  n> in <auto simp: power_int_minus power_int_diff field_simps
  P_def of_nat_diff>)
  finally show ?thesis

```

```

    using assms(1) by (simp add: P_def of_nat_diff)
qed (use assms(1) in auto)

lemma norm_qpochhammer_nonneg_le:
  fixes a q :: "'a::{real_normed_field}"
  assumes "norm q ≤ 1"
  shows "norm (qpochhammer (int n) a q) ≤ (1 + norm a) ^ n"
proof -
  have "norm (qpochhammer (int n) a q) = (∏ x<n. norm (1 - a * q ^ x))"
    by (simp add: qpochhammer_nonneg_def flip: prod_norm)
  also have "... ≤ (∏ x<n. norm (1::'a) + norm (a * q ^ x))"
    by (intro prod_mono conjI norm_ge_zero) norm
  also have "... = (∏ k<n. norm (1::'a) + norm a * norm q ^ k)"
    by (simp add: norm_power norm_mult)
  also have "... ≤ (∏ k<n. norm (1::'a) + norm a * norm q ^ 0)"
    by (intro prod_mono add_mono mult_left_mono power_decreasing conjI)
  (use assms in auto)
  finally show ?thesis
    by simp
qed

lemma norm_qpochhammer_nonneg_ge:
  fixes a q :: "'a::{real_normed_field}"
  assumes "norm q ≤ 1" "norm a ≤ 1"
  shows "norm (qpochhammer (int n) a q) ≥ (1 - norm a) ^ n"
proof -
  have "(∏ k<n. norm (1::'a) - norm a * norm q ^ 0) ≤
        (∏ k<n. norm (1::'a) - norm a * norm q ^ k)"
    by (intro prod_mono diff_mono mult_left_mono power_decreasing conjI)
  (use assms in auto)
  also have "... ≤ (∏ k<n. norm (1::'a) - norm (a * q ^ k))"
    by (simp add: norm_power norm_mult)
  also have "... ≤ (∏ k<n. norm (1 - a * q ^ k))"
  proof (intro prod_mono conjI)
    fix i :: nat
    show "norm (1::'a) - norm (a * q ^ i) ≤ norm (1 - a * q ^ i)"
      by norm
    have "norm a * norm q ^ i ≤ 1 * 1 ^ i"
      using assms by (intro mult_mono power_mono) auto
    thus "norm (1::'a) - norm (a * q ^ i) ≥ 0"
      by (simp add: norm_power norm_mult)
  qed
  also have "... = norm (qpochhammer (int n) a q)"
    by (simp add: qpochhammer_nonneg_def flip: prod_norm)
  finally show ?thesis
    by simp
qed

lemma qpochhammer_nonneg_nonzero:

```

```

fixes q :: "'a :: real_normed_field"
assumes "norm q < 1" "norm a < 1"
shows "qpochhammer (int k) a q ≠ 0"
proof -
  have "0 < (1 - norm a) ^ k"
    using assms by simp
  also have "(1 - norm a) ^ k ≤ norm (qpochhammer (int k) a q)"
    by (rule norm_qpochhammer_nonneg_ge) (use assms in auto)
  finally show ?thesis
    by auto
qed

lemma qbinomial_conv_qpochhammer':
  fixes q :: "'a :: {real_normed_field}"
  assumes "norm q < 1" "k ≤ n"
  shows "qbinomial q n k = qpochhammer (int k) (q ^ (n + 1 - k)) q /
qpochhammer (int k) q q"
proof -
  have eq: "qpochhammer (int n) q q =
qpochhammer (int k) (q ^ Suc (n - k)) q * qpochhammer (int
(n - k)) q q"
    using qpochhammer_nat_add[of "n - k" k q] assms by (simp add: of_nat_diff
mult_ac)
  have [simp]: "q ^ k ≠ 1" if "k > 0" for k
    using assms power_eq_1_iff that by fastforce
  have "qbinomial q n k = (qpochhammer (int n) q q / qpochhammer (int
n - int k) q q) / (qpochhammer (int k) q q)"
    by (subst qbinomial_conv_qpochhammer) (use assms in <auto simp: field_simps>)
  also have "... = qpochhammer (int k) (q ^ (n + 1 - k)) q / qpochhammer
(int k) q q"
    unfolding eq using assms
    by (auto simp add: qpochhammer_nonneg_nonzero Suc_diff_le simp flip:
of_nat_diff)
  finally show ?thesis .
qed

lemma norm_qbinomial_le:
  fixes a q :: "'a :: {real_normed_field}"
  assumes "norm q < 1"
  shows "norm (qbinomial q n k) ≤ ((1 + norm q) / (1 - norm q)) ^ k"
proof (cases "k ≤ n")
  case True
  have [simp]: "q ^ k ≠ 1" if "k > 0" for k
    using assms(1) power_eq_1_iff that by force
  have "norm (qbinomial q n k) =
norm (qpochhammer (int k) (q ^ (Suc n - k)) q) / norm (qpochhammer
(int k) q q)"
    by (subst qbinomial_conv_qpochhammer')
    (use assms True in <auto simp: norm_divide norm_mult of_nat_diff>)

```

```

also have "... ≤ (1 + norm (q ^ (Suc n - k))) ^ k / (1 - norm q) ^ k"
  by (intro frac_le mult_mono norm_qpochhammer_nonneg_le
      norm_qpochhammer_nonneg_ge mult_pos_pos)
      (use assms in auto)
also have "... ≤ (1 + norm q ^ 1) ^ k / (1 - norm q) ^ k"
  unfolding norm_power
  by (intro divide_right_mono power_mono add_left_mono power_decreasing)
      (use assms True in auto)
also have "... = ((1 + norm q) / (1 - norm q)) ^ k"
  using assms by (simp add: power_divide True flip: power_add)
finally show ?thesis .
qed (use assms in auto)

lemma norm_qbinomial_ge:
  fixes a q :: "'a::{real_normed_field}"
  assumes "norm q < 1" "k ≤ n"
  shows "norm (qbinomial q n k) ≥ ((1 - norm q) / (1 + norm q)) ^ k"
proof -
  have not_one: "q ^ k ≠ 1" if "k > 0" for k
    using assms(1) power_eq_1_iff that by force
  have [simp]: "qpochhammer (int i) q q ≠ 0" for i
  proof
    assume "qpochhammer (int i) q q = 0"
    then obtain k where "q * q powi k = 1" "k ≥ 0"
      by (subst (asm) qpochhammer_eq_0_iff) auto
    hence "q ^ Suc (nat k) = 1"
      by (cases k) auto
    thus False
      using not_one[of "Suc (nat k)"] by simp
  qed
  have "((1 - norm q) / (1 + norm q)) ^ k = (1 - norm q ^ 1) ^ k / (1
+ norm q) ^ k"
    using assms by (simp add: power_divide flip: power_add)
  also have "... ≤ (1 - norm (q ^ (Suc n - k))) ^ k / (1 + norm q) ^ k"
    unfolding norm_power
    by (intro divide_right_mono diff_left_mono power_mono power_decreasing)
        (use assms in auto)
  also have "... ≤ norm (qpochhammer (int k) (q ^ (Suc n - k)) q) / norm
(qpochhammer (int k) q q)"
    by (intro frac_le mult_mono norm_qpochhammer_nonneg_le
        norm_qpochhammer_nonneg_ge mult_pos_pos)
        (use assms in <auto simp: norm_power power_le_one_iff>)
  also have "... = norm (qbinomial q n k)"
    by (subst qbinomial_conv_qpochhammer')
        (use assms in <auto simp: norm_divide norm_mult of_nat_diff not_one>)
  finally show ?thesis .
qed

```

```

lemma norm_qpochhammer_nonneg_le_qpochhammer:
  fixes q :: "'a :: real_normed_field"
  shows "norm (qpochhammer (int k) a q) ≤ qpochhammer (int k) (-norm
a) (norm q)"
proof -
  have "norm (qpochhammer (int k) a q) = (∏ i<k. norm (1 - a * q ^ i))"
    by (simp add: qpochhammer_nonneg_def prod_norm)
  also have "... ≤ (∏ i<k. norm (1::'a) + norm (a * q ^ i))"
    by (intro prod_mono conjI norm_ge_zero) norm
  also have "... = qpochhammer (int k) (-norm a) (norm q)"
    by (simp add: qpochhammer_nonneg_def norm_mult norm_power)
  finally show ?thesis .
qed

```

```

lemma norm_qpochhammer_nonneg_ge_qpochhammer:
  fixes q :: "'a :: real_normed_field"
  assumes "norm q ≤ 1" "norm a ≤ 1"
  shows "norm (qpochhammer (int k) a q) ≥ qpochhammer (int k) (norm
a) (norm q)"
proof -
  have "qpochhammer (int k) (norm a) (norm q) = (∏ i<k. norm (1::'a) -
norm (a * q ^ i))"
    by (simp add: qpochhammer_nonneg_def norm_mult norm_power)
  also have "... ≤ (∏ i<k. norm (1 - a * q ^ i))"
  proof (intro prod_mono conjI norm_ge_zero)
    fix i assume i: "i ∈ {..k}"
    have "norm a * norm q ^ i ≤ 1 * 1 ^ i"
      by (intro mult_mono power_mono) (use assms in auto)
    thus "0 ≤ norm (1::'a) - norm (a * q ^ i)"
      by (auto simp: norm_mult norm_power)
  qed norm
  also have "... = norm (qpochhammer (int k) a q)"
    by (simp add: qpochhammer_nonneg_def prod_norm)
  finally show ?thesis .
qed

```

```

lemma qpochhammer_nonneg:
  assumes "a ≤ 1" "0 ≤ q" "q ≤ 1"
  shows "qpochhammer (int n) a (q::real) ≥ 0"
proof -
  have "a * q ^ i ≤ 1" for i
  proof -
    have "a * q ^ i ≤ 1 * 1 ^ i"
      by (intro mult_mono power_mono) (use assms in auto)
    thus ?thesis
      by simp
  qed
  thus ?thesis
    unfolding qpochhammer_nonneg_def by (intro prod_nonneg) auto

```

qed

```
lemma qpochhammer_pos:
  assumes "a < 1" "0 ≤ q" "q ≤ 1"
  shows "qpochhammer (int n) a (q::real) > 0"
proof -
  have "a * q ^ i < 1" for i
  proof (cases "a ≥ 0")
    case True
    have "a * q ^ i ≤ a * 1 ^ i"
      by (intro mult_left_mono power_mono) (use assms True in auto)
    thus ?thesis
      using assms by auto
  next
    case False
    hence "a * q ^ i ≤ 0"
      by (intro mult_nonpos_nonneg) (use assms in auto)
    also have "... < 1"
      by simp
    finally show ?thesis
      by simp
  qed
thus ?thesis
  unfolding qpochhammer_nonneg_def by (intro prod_pos) auto
qed
```

```
lemma holomorphic_qpochhammer [holomorphic_intros]:
  fixes f g :: "complex ⇒ complex"
  assumes [holomorphic_intros]: "f holomorphic_on A" "g holomorphic_on
A"
  assumes "∧x k. x ∈ A ⇒ int k ∈ {0<..-n} ⇒ f x ^ k ≠ g x" "∧x.
x ∈ A ⇒ f x ≠ 0"
  shows "(λx. qpochhammer n (g x) (f x)) holomorphic_on A"
  unfolding qpochhammer_def using assms(3,4)
  by (cases "n ≥ 0")
    (force intro!: holomorphic_intros simp: Suc_le_eq not_le eq_commute[of
_ "g x" for x])+
```

```
lemma analytic_qpochhammer [analytic_intros]:
  fixes f g :: "complex ⇒ complex"
  assumes [analytic_intros]: "f analytic_on A" "g analytic_on A"
  assumes "∧x k. x ∈ A ⇒ int k ∈ {0<..-n} ⇒ f x ^ k ≠ g x" "∧x.
x ∈ A ⇒ f x ≠ 0"
  shows "(λx. qpochhammer n (g x) (f x)) analytic_on A"
  unfolding qpochhammer_def using assms(3,4)
  by (cases "n ≥ 0")
    (force intro!: analytic_intros simp: Suc_le_eq not_le eq_commute[of
_ "g x" for x])+
```

```

lemma meromorphic_qpochhammer [meromorphic_intros]:
  fixes f g :: "complex  $\Rightarrow$  complex"
  assumes [meromorphic_intros]: "f meromorphic_on A" "g meromorphic_on
A"
  shows "( $\lambda$ x. qpochhammer n (g x) (f x)) meromorphic_on A"
  unfolding qpochhammer_def by (cases "n  $\geq$  0") (auto intro!: meromorphic_intros)

lemma continuous_on_qpochhammer [continuous_intros]:
  fixes f g :: "'a :: topological_space  $\Rightarrow$  'b :: {real_normed_field}"
  assumes [continuous_intros]: "continuous_on A f" "continuous_on A g"
  assumes " $\bigwedge$ x k. x  $\in$  A  $\implies$  int k  $\in$  {0<.. $-n$ }  $\implies$  f x  $^k$   $\neq$  g x" " $\bigwedge$ x.
x  $\in$  A  $\implies$  f x  $\neq$  0"
  shows "continuous_on A ( $\lambda$ x. qpochhammer n (g x) (f x))"
  unfolding qpochhammer_def using assms(3,4)
  by (cases "n  $\geq$  0")
    (force intro!: continuous_intros simp: Suc_le_eq not_le eq_commute[of
_ "g x" for x])+

lemma continuous_qpochhammer [continuous_intros]:
  fixes f g :: "'a :: t2_space  $\Rightarrow$  'b :: {real_normed_field}"
  assumes [continuous_intros]: "continuous (at x within A) f" "continuous
(at x within A) g"
  assumes " $\bigwedge$ k. int k  $\in$  {0<.. $-n$ }  $\implies$  f x  $^k$   $\neq$  g x" "f x  $\neq$  0"
  shows "continuous (at x within A) ( $\lambda$ x. qpochhammer n (g x) (f x))"
  unfolding qpochhammer_def using assms(3,4)
  by (cases "n  $\geq$  0")
    (force intro!: continuous_intros simp: Suc_le_eq not_le eq_commute[of
_ "g x" for x])+

lemma tendsto_qpochhammer [tendsto_intros]:
  fixes f g :: "'a  $\Rightarrow$  'b :: {real_normed_field}"
  assumes [tendsto_intros]: "(f  $\longrightarrow$  q) F" "(g  $\longrightarrow$  a) F"
  assumes " $\bigwedge$ k. int k  $\in$  {0<.. $-n$ }  $\implies$  q  $^k$   $\neq$  a" "q  $\neq$  0"
  shows "(( $\lambda$ x. qpochhammer n (g x) (f x))  $\longrightarrow$  qpochhammer n a q) F"
proof (cases "n  $\geq$  0")
  case True
  have "(( $\lambda$ x.  $\prod$  k<nat n. 1 - g x * f x  $^k$ )  $\longrightarrow$  ( $\prod$  k<nat n. 1 - a *
q  $^k$ )) F"
  by (intro tendsto_intros)
  thus ?thesis
  using True by (simp add: qpochhammer_def [abs_def])
next
  case False
  have " (( $\lambda$ x.  $\prod$  k=1..nat (- n). 1 / (1 - g x / f x  $^k$ ))  $\longrightarrow$ 
( $\prod$  k=1..nat (- n). 1 / (1 - a / q  $^k$ ))) F"
  by (intro tendsto_intros; use assms False in <force simp: Suc_le_eq>)
  thus ?thesis
  using False by (simp add: qpochhammer_def [abs_def])

```

qed

end

3 Primitive roots in an integral domain

```
theory Primitive_Roots
imports
  Complex_Main
  "HOL-Number_Theory.Cong"
  "HOL-Computational_Algebra.Computational_Algebra"
  "HOL-Library.Real_Mod"
begin
```

We define the notion of a primitive root in an integral domain: an n -th root of unity that is not a k -th root of unity for any $k < n$ and therefore generates all n -th roots of unity.

```
locale primroot =
  fixes n :: nat and w :: "'a :: idom"
  assumes pos_order: "n > 0"
  assumes root_of_unity: "w ^ n = 1"
  assumes primitive: "\k. 0 < k ==> k < n ==> w ^ k ≠ 1"
begin
```

```
lemma nonzero: "w ≠ 0"
  using root_of_unity pos_order by (auto simp: power_0_left)
```

```
lemma power_eq_iff: "w ^ k = w ^ l ↔ [k = l] (mod n)"
proof (induction k l rule: linorder_wlog)
```

```
  case (le k l)
  define m where "m = l - k"
  have l: "l = k + m"
    using le by (simp add: m_def)
```

```
  define a where "a = m div n"
  define b where "b = m mod n"
  have m: "m = n * a + b"
    by (simp add: a_def b_def)
  have b: "b < n"
    using pos_order by (simp add: b_def)
```

```
  have "w ^ k = w ^ l ↔ w ^ k = w ^ k * w ^ m"
    by (simp add: l power_add)
  also have "... ↔ w ^ m = 1"
    using nonzero by simp
  also have "... ↔ w ^ b = 1"
    by (simp add: m power_add power_mult root_of_unity)
  also have "... ↔ b = 0"
```

```

    using primitive[of b] b by (cases "b = 0") auto
  also have "...  $\longleftrightarrow$   $[k = 1] \pmod{n}$ "
    by (simp add: l cong_altdef_nat' b_def mod_eq_0_iff_dvd)
  finally show ?case .
qed (simp add: cong_sym_eq eq_commute)

lemma inj_on_power: "inj_on ( $\lambda k. w \wedge k$ )  $\{..<n\}$ "
  by (auto intro!: inj_onI simp: power_eq_iff cong_def)

end

locale primroot_field = primroot n w for n and w :: "'a :: field"
begin

lemma power_int_eq_iff: "w powi k = w powi l  $\longleftrightarrow$   $[k = l] \pmod{(\text{int } n)}$ "
proof -
  define a where "a = min k l"
  define k' where "k' = nat (k - a)"
  define l' where "l' = nat (l - a)"
  have k: "k = a + int k'" and l: "l = a + int l'"
    by (simp_all add: k'_def l'_def a_def)
  have "w powi k = w powi l  $\longleftrightarrow$   $[k' = l'] \pmod{n}$ "
    using nonzero by (simp add: k l power_int_add power_eq_iff)
  also have "...  $\longleftrightarrow$   $[k = l] \pmod{(\text{int } n)}$ "
    by (simp add: k l cong_add_lcancel cong_int_iff)
  finally show ?thesis .
qed

end

interpretation primroot_1: primroot 1 1
  by standard auto

interpretation primroot_neg1: primroot 2 "-1 :: 'a :: {idom, ring_char_0}"
  by standard (auto simp: numeral_2_eq_2 less_Suc_eq)

locale primroot_cis =
  fixes n :: nat and k :: int
  assumes coprime: "coprime k n"
  assumes pos_order': "n > 0"
begin

sublocale primroot n "cis (2 * pi * k / n)"
proof
  show "n > 0"
    using pos_order' by simp
next

```

```

    show "cis (2 * pi * real_of_int k / real n) ^ n = 1"
      using pos_order' by (simp add: Complex.DeMoivre)
next
  fix j :: nat assume j: "j > 0" "j < n"
  show "cis (2 * pi * real_of_int k / real n) ^ j ≠ 1"
  proof
    assume "cis (2 * pi * real_of_int k / real n) ^ j = 1"
    then obtain l where "real j * (2 * pi * real_of_int k / real n) =
real_of_int l * (2 * pi)"
      unfolding Complex.DeMoivre cis_eq_1_iff by blast
    hence "real_of_int (k * j) = real_of_int (l * n)"
      unfolding of_int_mult using pos_order' by (simp add: field_simps)
    hence eq: "k * j = l * n"
      by linarith
    have "n dvd k * j"
      by (simp add: eq)
    with coprime have "n dvd j"
      by (simp add: coprime_commute coprime_dvd_mult_right_iff)
    with j show False
      by force
  qed
qed
end

interpretation primroot_ii: primroot 4 i
proof -
  interpret primroot_cis 4 1
  by standard auto
  show "primroot 4 i"
    using primroot_axioms by simp
qed

lemma (in primroot) cyclotomic_poly_conv_prod_unity_root:
  "Polynomial.monom 1 n - 1 = (∏ k<n. [:- (w ^ k), 1:])" (is "?lhs = ?rhs")
proof (rule ccontr)
  assume neq: "?lhs ≠ ?rhs"
  have "?lhs = Polynomial.monom 1 n + (-1)"
    by simp
  also have "degree ... = n"
    by (subst degree_add_eq_left) (use pos_order in <auto simp: degree_monom_eq>)
  finally have deg1: "degree ?lhs = n" .

  have "poly.coeff (?lhs - ?rhs) n = 0"
  proof -
    have "poly.coeff ?rhs n = lead_coeff ?rhs"
      by (simp add: degree_prod_sum_eq)
    also have "... = 1"
      by (subst lead_coeff_prod) auto
  qed

```

```

    finally show ?thesis
      using pos_order by simp
qed
moreover have "?lhs - ?rhs ≠ 0"
  using neq by simp
ultimately have "degree (?lhs - ?rhs) ≠ n"
  by (metis leading_coeff_0_iff)
moreover have "degree (?lhs - ?rhs) ≤ n"
  by (rule degree_diff_le) (use deg1 in <auto simp: degree_prod_sum_eq>)
ultimately have "degree (?lhs - ?rhs) < n"
  by linarith

have root1: "poly ?lhs (w ^ k) = 0" for k
proof -
  have "poly ?lhs (w ^ k) = w ^ (n * k) - 1"
    by (simp add: poly_monom mult_ac flip: power_mult)
  also have "... = 0"
    by (simp add: power_mult root_of_unity)
  finally show ?thesis .
qed
have root2: "poly ?rhs (w ^ k) = 0" if "k < n" for k
  using that by (auto simp: poly_prod)

have "inj_on (λk. w ^ k) {..<n}"
  by (rule inj_on_power)
hence "card {..<n} = card ((λk. w ^ k) ‘ {..<n})"
  by (subst card_image) auto
also have "card ((λk. w ^ k) ‘ {..<n}) ≤ card {z. poly (?lhs - ?rhs)
z = 0}"
  by (intro card_mono poly_roots_finite) (use neq root1 root2 in auto)
also have "card {z. poly (?lhs - ?rhs) z = 0} ≤ degree (?lhs - ?rhs)"
  by (rule card_poly_roots_bound) (use neq in auto)
also have "... < n"
  by fact
finally show False
  by simp
qed

lemma (in primroot) cyclotomic_poly_conv_prod_unity_root':
  "1 - Polynomial.monom 1 n = (∏ k<n. [:1, -(w ^ k):])" (is "?lhs = ?rhs")
proof -
  define A where "A = insert 0 ((λk. w ^ k) ‘ {..<n})"
  have card_A: "card A = Suc n"
  proof -
    have "card A = Suc (card ((λk. w ^ k) ‘ {..<n}))"
      unfolding A_def using nonzero by (subst card.insert) auto
    also have "card ((λk. w ^ k) ‘ {..<n}) = n"
      by (subst card_image) (use pos_order inj_on_power in auto)
    finally show ?thesis .
  
```

```

qed

show ?thesis
proof (rule poly_eqI_degree)
  have "degree (1 - Polynomial.monom 1 n :: 'a poly) ≤ n"
    by (intro degree_diff_le) (auto simp: degree_monom_eq)
  thus "degree (1 - Polynomial.monom 1 n :: 'a poly) < card A"
    by (simp add: card_A)
next
  have "degree (∏ k<n. [:1, -(w ^ k):]) ≤ n"
    by (rule order.trans[OF degree_prod_sum_le]) (auto simp: nonzero)
  thus "degree (∏ k<n. [:1, -(w ^ k):]) < card A"
    by (simp add: card_A)
next
  fix x assume x: "x ∈ A"
  have "poly (1 - Polynomial.monom 1 n) (w ^ k) = 0" for k
  proof -
    have "poly (1 - Polynomial.monom 1 n) (w ^ k) = 1 - w ^ (n * k)"
      by (simp add: poly_monom mult_ac flip: power_mult)
    also have "... = 0"
      by (simp add: power_mult root_of_unity)
    finally show ?thesis .
  qed
  moreover have "poly (1 - Polynomial.monom 1 n :: 'a poly) 0 = 1"
    using pos_order by (simp add: poly_monom power_0_left)
  moreover have "poly (∏ k<n. [:1, -(w ^ k):]) (w ^ k) = 0" if k: "k
< n" for k
  proof -
    define k' where "k' = (if k = 0 then 0 else n - k)"
    have "k' < n"
      using pos_order k by (auto simp: k'_def)
    have "w ^ k * w ^ k' = w ^ (k + k)"
      by (simp add: power_add)
    also have "... = w ^ 0"
      by (subst power_eq_iff) (use k in <auto simp: k'_def cong_def>)
    finally show ?thesis
      using that <k' < n> by (auto simp: poly_prod)
  qed
  moreover have "poly (∏ k<n. [:1, -(w ^ k):]) 0 = 1"
    by (simp add: poly_prod)
  ultimately show "poly (1 - Polynomial.monom 1 n) x = poly (∏ k<n.
[:1, -(w ^ k):]) x"
    using x by (auto simp: A_def)
  qed
qed
end

```

4 The infinite q -Pochhammer symbol $(a; q)_\infty$

```
theory Q_Pochhammer_Infinite
imports
  More_Infinite_Products
  Q_Analogues
  Primitive_Roots
begin
```

4.1 Definition and basic properties

```
definition qepochhammer_inf :: "'a :: {real_normed_field, banach, heine_borel}
⇒ 'a ⇒ 'a" where
  "qepochhammer_inf a q = prodinf (λk. 1 - a * q ^ k)"
```

```
bundle qepochhammer_inf_notation
begin
notation qepochhammer_inf ("'(_ ; _)∞")
end
```

```
bundle no_qepochhammer_inf_notation
begin
no_notation qepochhammer_inf ("'(_ ; _)∞")
end
```

```
lemma qepochhammer_inf_0_left [simp]: "qepochhammer_inf 0 q = 1"
  by (simp add: qepochhammer_inf_def)
```

```
lemma qepochhammer_inf_0_right [simp]: "qepochhammer_inf a 0 = 1 - a"
proof -
  have "qepochhammer_inf a 0 = (∏ k ≤ 0. 1 - a * 0 ^ k)"
    unfolding qepochhammer_inf_def by (rule prodinf_finite) auto
  also have "... = 1 - a"
    by simp
  finally show ?thesis .
qed
```

```
lemma abs_convergent_qepochhammer_inf:
  fixes a q :: "'a :: {real_normed_div_algebra, banach}"
  assumes "norm q < 1"
  shows "abs_convergent_prod (λn. 1 - a * q ^ n)"
proof (rule summable_imp_abs_convergent_prod)
  show "summable (λn. norm (1 - a * q ^ n - 1))"
    using assms by (auto simp: norm_power norm_mult)
qed
```

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

```

shows "convergent_prod ( $\lambda n. 1 - a * q ^ n$ )"
using abs_convergent_qpochhammer_inf[OF assms] abs_convergent_prod_imp_convergent_prod
by blast

```

```

lemma has_prod_qpochhammer_inf:
  "norm q < 1  $\implies$  ( $\lambda n. 1 - a * q ^ n$ ) has_prod qpochhammer_inf a q"
using convergent_qpochhammer_inf unfolding qpochhammer_inf_def
by (intro convergent_prod_has_prod)

```

We now also see that the infinite q -Pochhammer symbol $(a; q)_\infty$ really is the limit of $(a; q)_n$ for $n \rightarrow \infty$:

```

lemma qpochhammer_tendsto_qpochhammer_inf:
  assumes q: "norm q < 1"
  shows " $(\lambda n. qpochhammer (int n) t q) \longrightarrow qpochhammer_inf t q$ "
  using has_prod_imp_tendsto'[OF has_prod_qpochhammer_inf[OF q, of t]]
  by (simp add: qpochhammer_def)

```

```

lemma qpochhammer_inf_of_real:
  assumes "|q| < 1"
  shows "qpochhammer_inf (of_real a) (of_real q) = of_real (qpochhammer_inf a q)"
proof -
  have " $(\lambda n. of\_real (1 - a * q ^ n) :: 'a)$  has_prod of_real (qpochhammer_inf a q)"
  unfolding has_prod_of_real_iff by (rule has_prod_qpochhammer_inf)
  (use assms in auto)
  also have " $(\lambda n. of\_real (1 - a * q ^ n) :: 'a) = (\lambda n. 1 - of\_real a * of\_real q ^ n)$ "
  by simp
  finally have "... has_prod of_real (qpochhammer_inf a q)" .
  moreover have " $(\lambda n. 1 - of\_real a * of\_real q ^ n :: 'a)$  has_prod qpochhammer_inf (of_real a) (of_real q)"
  by (rule has_prod_qpochhammer_inf) (use assms in auto)
  ultimately show ?thesis
  using has_prod_unique2 by blast
qed

```

```

lemma qpochhammer_inf_zero_iff:
  assumes q: "norm q < 1"
  shows "qpochhammer_inf a q = 0  $\iff$  ( $\exists n. a * q ^ n = 1$ )"
proof -
  have " $(\lambda n. 1 - a * q ^ n)$  has_prod qpochhammer_inf a q"
  using has_prod_qpochhammer_inf[OF q] by simp
  hence "qpochhammer_inf a q = 0  $\iff$  ( $\exists n. a * q ^ n = 1$ )"
  by (subst has_prod_eq_0_iff) auto
  thus ?thesis .
qed

```

```

lemma qpochhammer_inf_nonzero:

```

```

    assumes "norm q < 1" "norm a < 1"
    shows "qpochhammer_inf a q ≠ 0"
  proof
    assume "qpochhammer_inf a q = 0"
    then obtain n where n: "a * q ^ n = 1"
      using assms by (subst (asm) qpochhammer_inf_zero_iff) auto
    have "norm (q ^ n) * norm a ≤ 1 * norm a"
      unfolding norm_power using assms by (intro mult_right_mono power_le_one)
  auto
    also have "... < 1"
      using assms by simp
    finally have "norm (a * q ^ n) < 1"
      by (simp add: norm_mult mult.commute)
    with n show False
      by auto
  qed

```

```

lemma qpochhammer_inf_pos:
  assumes "|q| < 1" "|a| < (1::real)"
  shows "qpochhammer_inf a q > 0"
  using has_prod_qpochhammer_inf
  proof (rule has_prod_pos)
    fix n :: nat
    have "|a * q ^ n| = |a| * |q| ^ n"
      by (simp add: abs_mult power_abs)
    also have "|a| * |q| ^ n ≤ |a| * 1 ^ n"
      by (intro mult_left_mono power_mono) (use assms in auto)
    also have "... < 1"
      using assms by simp
    finally show "0 < 1 - a * q ^ n"
      by simp
  qed (use assms in auto)

```

```

lemma qpochhammer_inf_nonneg:
  assumes "|q| < 1" "|a| ≤ (1::real)"
  shows "qpochhammer_inf a q ≥ 0"
  using has_prod_qpochhammer_inf
  proof (rule has_prod_nonneg)
    fix n :: nat
    have "|a * q ^ n| = |a| * |q| ^ n"
      by (simp add: abs_mult power_abs)
    also have "|a| * |q| ^ n ≤ |a| * 1 ^ n"
      by (intro mult_left_mono power_mono) (use assms in auto)
    also have "... ≤ 1"
      using assms by simp
    finally show "0 ≤ 1 - a * q ^ n"
      by simp
  qed (use assms in auto)

```

4.2 Uniform convergence and its consequences

```

context
  fixes P :: "nat  $\Rightarrow$  'a :: {real_normed_field, banach, heine_borel}  $\Rightarrow$ 
'a  $\Rightarrow$  'a"
  defines "P  $\equiv$  ( $\lambda N$  a q.  $\prod n < N. 1 - a * q ^ n$ )"
begin

lemma uniformly_convergent_qpochhammer_inf_aux:
  assumes r: "0  $\leq$  ra" "0  $\leq$  rq" "rq < 1"
  shows "uniformly_convergent_on (cball 0 ra  $\times$  cball 0 rq) ( $\lambda n$  (a,q).
P n a q)"
  unfolding P_def case_prod_unfold
proof (rule uniformly_convergent_on_prod')
  show "uniformly_convergent_on (cball 0 ra  $\times$  cball 0 rq)
( $\lambda n$  aq.  $\sum n < N. \text{norm } (1 - \text{fst } aq * \text{snd } aq ^ n - 1 :: 'a)$ )"
  proof (intro Weierstrass_m_test'_ev always_eventually allI ballI)
    show "summable ( $\lambda n. ra * rq ^ n$ )" using r
      by (intro summable_mult summable_geometric) auto
    next
      fix n :: nat and aq :: "'a  $\times$  'a"
      assume "aq  $\in$  cball 0 ra  $\times$  cball 0 rq"
      then obtain a q where [simp]: "aq = (a, q)" and aq: "norm a  $\leq$  ra"
      "norm q  $\leq$  rq"
      by (cases aq) auto
      have "norm (norm (1 - a * q ^ n - 1)) = norm a * norm q ^ n"
        by (simp add: norm_mult norm_power)
      also have "...  $\leq$  ra * rq ^ n"
        using aq r by (intro mult_mono power_mono) auto
      finally show "norm (norm (1 - fst aq * snd aq ^ n - 1))  $\leq$  ra * rq
^ n"
        by simp
    qed
  qed (auto intro!: continuous_intros compact_Times)

lemma uniformly_convergent_qpochhammer_inf:
  assumes "compact A" "A  $\subseteq$  UNIV  $\times$  ball 0 1"
  shows "uniformly_convergent_on A ( $\lambda n$  (a,q). P n a q)"
proof (cases "A = {}")
  case False
  obtain rq where rq: "rq  $\geq$  0" "rq < 1" " $\wedge$  a q. (a, q)  $\in$  A  $\implies$  norm
q  $\leq$  rq"
  proof -
    from <compact A> have "compact (norm ' snd ' A)"
      by (intro compact_continuous_image continuous_intros)
    hence "Sup (norm ' snd ' A)  $\in$  norm ' snd ' A"
      by (intro closed_contains_Sup bounded_imp_bdd_above compact_imp_bounded
compact_imp_closed)
      (use <A  $\neq$  {}> in auto)
    then obtain aq0 where aq0: "aq0  $\in$  A" "norm (snd aq0) = Sup (norm

```

```

' snd ' A)"
  by auto
show ?thesis
proof (rule that[of "norm (snd aq0)"])
  show "norm (snd aq0) ≥ 0" and "norm (snd aq0) < 1"
    using assms(2) aq0(1) by auto
next
  fix a q assume "(a, q) ∈ A"
  hence "norm q ≤ Sup (norm ' snd ' A)"
    by (intro cSup_upper bounded_imp_bdd_above compact_imp_bounded
assms
      compact_continuous_image continuous_intros) force
  with aq0 show "norm q ≤ norm (snd aq0)"
    by simp
qed
qed

obtain ra where ra: "ra ≥ 0" "∧ a q. (a, q) ∈ A ⇒ norm a ≤ ra"
proof -
  have "bounded (fst ' A)"
    by (intro compact_imp_bounded compact_continuous_image continuous_intros
assms)
  then obtain ra where ra: "norm a ≤ ra" if "a ∈ fst ' A" for a
    unfolding bounded_iff by blast
  from <A ≠ {}> obtain aq0 where "aq0 ∈ A"
    by blast
  have "0 ≤ norm (fst aq0)"
    by simp
  also have "fst aq0 ∈ fst ' A"
    using <aq0 ∈ A> by blast
  with ra[of "fst aq0"] and <A ≠ {}> have "norm (fst aq0) ≤ ra"
    by simp
  finally show ?thesis
    using that[of ra] ra by fastforce
qed

have "uniformly_convergent_on (cball 0 ra × cball 0 rq) (λn (a,q).
P n a q)"
  by (intro uniformly_convergent_qpochhammer_inf_aux) (use ra rq in
auto)
  thus ?thesis
    by (rule uniformly_convergent_on_subset) (use ra rq in auto)
qed auto

lemma uniform_limit_qpochhammer_inf:
  assumes "compact A" "A ⊆ UNIV × ball 0 1"
  shows "uniform_limit A (λn (a,q). P n a q) (λ(a,q). qpochhammer_inf
a q) at_top"
proof -

```

```

obtain g where g: "uniform_limit A (λn (a,q). P n a q) g at_top"
  using uniformly_convergent_qpochhammer_inf[OF assms(1,2)]
  by (auto simp: uniformly_convergent_on_def)
also have "?this ↔ uniform_limit A (λn (a,q). P n a q) (λ(a,q). qpochhammer_inf
a q) at_top"
proof (intro uniform_limit_cong)
  fix aq :: "'a × 'a"
  assume "aq ∈ A"
  then obtain a q where [simp]: "aq = (a, q)" and aq: "(a, q) ∈ A"
    by (cases aq) auto
  from aq and assms have q: "norm q < 1"
    by auto
  have "(λn. case aq of (a, q) ⇒ P n a q) → g aq"
    by (rule tendsto_uniform_limitI[OF g]) fact
  hence "(λn. case aq of (a, q) ⇒ P (Suc n) a q) → g aq"
    by (rule filterlim_compose) (rule filterlim_Suc)
  moreover have "(λn. case aq of (a, q) ⇒ P (Suc n) a q) → qpochhammer_inf
a q"
    using convergent_prod_LIMSEQ[OF convergent_qpochhammer_inf[of q
a]] aq q
    unfolding P_def lessThan_Suc_atMost
    by (simp add: qpochhammer_inf_def)
  ultimately show "g aq = (case aq of (a, q) ⇒ qpochhammer_inf a q)"
    using tendsto_unique by force
qed auto
finally show ?thesis .
qed

lemma continuous_on_qpochhammer_inf [continuous_intros]:
  fixes a q :: "'b :: topological_space ⇒ 'a"
  assumes [continuous_intros]: "continuous_on A a" "continuous_on A q"
  assumes "∧x. x ∈ A ⇒ norm (q x) < 1"
  shows "continuous_on A (λx. qpochhammer_inf (a x) (q x))"
proof -
  have *: "continuous_on (cball 0 ra × cball 0 rq) (λ(a,q). qpochhammer_inf
a q :: 'a)"
    if r: "0 ≤ ra" "0 ≤ rq" "rq < 1" for ra rq :: real
  proof (rule uniform_limit_theorem)
    show "uniform_limit (cball 0 ra × cball 0 rq) (λn (a,q). P n a q)
      (λ(a,q). qpochhammer_inf a q) at_top"
      by (rule uniform_limit_qpochhammer_inf) (use r in <auto simp: compact_Times>)
  qed (auto intro!: always_eventually intro!: continuous_intros simp:
P_def case_prod_unfold)

  have **: "isCont (λ(a,q). qpochhammer_inf a q) (a, q)" if q: "norm q
< 1" for a q :: 'a
  proof -
    obtain R where R: "norm q < R" "R < 1"
      using dense q by blast

```

```

with norm_ge_zero[of q] have "R ≥ 0"
  by linarith
have "continuous_on (cball 0 (norm a + 1) × cball 0 R) (λ(a,q). qpochhammer_inf
a q :: 'a)"
  by (rule *) (use R <R ≥ 0> in auto)
hence "continuous_on (ball 0 (norm a + 1) × ball 0 R) (λ(a,q). qpochhammer_inf
a q :: 'a)"
  by (rule continuous_on_subset) auto
moreover have "(a, q) ∈ ball 0 (norm a + 1) × ball 0 R"
  using q R by auto
ultimately show ?thesis
  by (subst (asm) continuous_on_eq_continuous_at) (auto simp: open_Times)
qed
hence ***: "continuous_on ((λx. (a x, q x)) ' A) (λ(a,q). qpochhammer_inf
a q)"
  using assms(3) by (intro continuous_at_imp_continuous_on) auto
have "continuous_on A ((λ(a,q). qpochhammer_inf a q) ∘ (λx. (a x, q
x)))"
  by (rule continuous_on_compose[OF _ ***]) (intro continuous_intros)
thus ?thesis
  by (simp add: o_def)
qed

```

```

lemma continuous_qpochhammer_inf [continuous_intros]:
  fixes a q :: "'b :: t2_space ⇒ 'a"
  assumes "continuous (at x within A) a" "continuous (at x within A)
q" "norm (q x) < 1"
  shows "continuous (at x within A) (λx. qpochhammer_inf (a x) (q x))"
proof -
  have "continuous_on (UNIV × ball 0 1) (λx. qpochhammer_inf (fst x)
(snd x) :: 'a)"
    by (intro continuous_intros) auto
  moreover have "(a x, q x) ∈ UNIV × ball 0 1"
    using assms(3) by auto
  ultimately have "isCont (λx. qpochhammer_inf (fst x) (snd x)) (a x,
q x)"
    by (simp add: continuous_on_eq_continuous_at open_Times)
  hence "continuous (at (a x, q x) within (λx. (a x, q x)) ' A)
(λx. qpochhammer_inf (fst x) (snd x))"
    using continuous_at_imp_continuous_at_within by blast
  hence "continuous (at x within A) ((λx. qpochhammer_inf (fst x) (snd
x)) ∘ (λx. (a x, q x)))"
    by (intro continuous_intros assms)
  thus ?thesis
    by (simp add: o_def)
qed

```

```

lemma tendsto_qpochhammer_inf [tendsto_intros]:
  fixes a q :: "'b ⇒ 'a"

```

```

    assumes "(a  $\longrightarrow$  a0) F" "(q  $\longrightarrow$  q0) F" "norm q0 < 1"
    shows "(( $\lambda$ x. qpochhammer_inf (a x) (q x))  $\longrightarrow$  qpochhammer_inf a0
q0) F"
  proof -
    define f where "f = ( $\lambda$ x. qpochhammer_inf (fst x) (snd x) :: 'a)"
    have "(( $\lambda$ x. f (a x, q x))  $\longrightarrow$  f (a0, q0)) F"
    proof (rule isCont_tendsto_compose[of _ f])
      show "isCont f (a0, q0)"
        using assms(3) by (auto simp: f_def intro!: continuous_intros)
      show "(( $\lambda$ x. (a x, q x))  $\longrightarrow$  (a0, q0)) F "
        by (intro tendsto_intros assms)
    qed
    thus ?thesis
      by (simp add: f_def)
  qed

end

context
  fixes P :: "nat  $\Rightarrow$  complex  $\Rightarrow$  complex  $\Rightarrow$  complex"
  defines "P  $\equiv$  ( $\lambda$ N a q.  $\prod_{n < N}$ . 1 - a * q ^ n)"
begin

lemma holomorphic_qpochhammer_inf [holomorphic_intros]:
  assumes [holomorphic_intros]: "a holomorphic_on A" "q holomorphic_on
A"
  assumes " $\bigwedge$ x. x  $\in$  A  $\implies$  norm (q x) < 1" "open A"
  shows "(( $\lambda$ x. qpochhammer_inf (a x) (q x)) holomorphic_on A"
  proof (rule holomorphic_uniform_sequence)
    fix x assume x: "x  $\in$  A"
    then obtain r where r: "r > 0" "cball x r  $\subseteq$  A"
      using <open A> unfolding open_contains_cball by blast
    have *: "compact (( $\lambda$ x. (a x, q x)) ' cball x r)" using r
      by (intro compact_continuous_image continuous_intros)
      (auto intro!: holomorphic_on_imp_continuous_on[OF holomorphic_on_subset]
holomorphic_intros)
    have "uniform_limit (( $\lambda$ x. (a x, q x)) ' cball x r) ( $\lambda$ n (a,q). P n a
q) ( $\lambda$ (a,q). qpochhammer_inf a q) at_top"
      unfolding P_def
      by (rule uniform_limit_qpochhammer_inf[OF *]) (use r assms(3) in <auto
simp: compact_Times>)
    hence "uniform_limit (cball x r) ( $\lambda$ n x. case (a x, q x) of (a, q)  $\Rightarrow$ 
P n a q)
      ( $\lambda$ x. case (a x, q x) of (a, q)  $\Rightarrow$  qpochhammer_inf a q) at_top"
      by (rule uniform_limit_compose') auto
    thus " $\exists d > 0$ . cball x d  $\subseteq$  A  $\wedge$  uniform_limit (cball x d)
      ( $\lambda$ n x. case (a x, q x) of (a, q)  $\Rightarrow$  P n a q)
      ( $\lambda$ x. qpochhammer_inf (a x) (q x)) sequentially"
      using r by fast
  qed

```

```

qed (use <open A> in <auto intro!: holomorphic_intros simp: P_def>)

lemma analytic_qpochhammer_inf [analytic_intros]:
  assumes [analytic_intros]: "a analytic_on A" "q analytic_on A"
  assumes "\x. x \in A \implies norm (q x) < 1"
  shows "(λx. qpochhammer_inf (a x) (q x)) analytic_on A"
proof -
  from assms(1) obtain A1 where A1: "open A1" "A \subseteq A1" "a holomorphic_on A1"
  by (auto simp: analytic_on_holomorphic)
  from assms(2) obtain A2 where A2: "open A2" "A \subseteq A2" "q holomorphic_on A2"
  by (auto simp: analytic_on_holomorphic)
  have "continuous_on A2 q"
  by (rule holomorphic_on_imp_continuous_on) fact
  hence "open (q -` ball 0 1 \cap A2)"
  using A2 by (subst (asm) continuous_on_open_vimage) auto
  define A' where "A' = (q -` ball 0 1 \cap A2) \cap A1"
  have "open A'"
  unfolding A'_def by (rule open_Int) fact+

  note [holomorphic_intros] = holomorphic_on_subset[OF A1(3)] holomorphic_on_subset[OF A2(3)]
  have "(λx. qpochhammer_inf (a x) (q x)) holomorphic_on A'"
  using <open A'> by (intro holomorphic_intros) (auto simp: A'_def)
  moreover have "A \subseteq A'"
  using A1(2) A2(2) assms(3) unfolding A'_def by auto
  ultimately show ?thesis
  using analytic_on_holomorphic <open A'> by blast
qed

lemma meromorphic_qpochhammer_inf [meromorphic_intros]:
  assumes [analytic_intros]: "a analytic_on A" "q analytic_on A"
  assumes "\x. x \in A \implies norm (q x) < 1"
  shows "(λx. qpochhammer_inf (a x) (q x)) meromorphic_on A"
  by (rule analytic_on_imp_meromorphic_on) (use assms(3) in <auto intro!: analytic_intros>)

```

end

4.3 Bounds for $(a; q)_n$ and $\binom{n}{k}_q$ in terms of $(a; q)_\infty$

```

lemma qpochhammer_le_qpochhammer_inf:
  assumes "q \ge 0" "q < 1" "a \le 0"
  shows "qpochhammer (int k) a q \le qpochhammer_inf a (q::real)"
  unfolding qpochhammer_nonneg_def qpochhammer_inf_def
proof (rule prod_le_producing)
  show "(λk. 1 - a * q ^ k) has_prod qpochhammer_inf a q"
  by (rule has_prod_qpochhammer_inf) (use assms in auto)

```

```

next
  fix i :: nat
  have *: "a * q ^ i ≤ 0"
    by (rule mult_nonpos_nonneg) (use assms in auto)
  show "1 - a * q ^ i ≥ 0" "1 ≤ 1 - a * q ^ i"
    using * by simp_all
qed

lemma qpochhammer_ge_qpochhammer_inf:
  assumes "q ≥ 0" "q < 1" "a ≥ 0" "a ≤ 1"
  shows "qpochhammer (int k) a q ≥ qpochhammer_inf a (q::real)"
  unfolding qpochhammer_nonneg_def qpochhammer_inf_def
proof (rule prod_ge_producing)
  show "(λk. 1 - a * q ^ k) has_prod qpochhammer_inf a q"
    by (rule has_prod_qpochhammer_inf) (use assms in auto)
next
  fix i :: nat
  have "a * q ^ i ≤ 1 * 1 ^ i"
    using assms by (intro mult_mono power_mono) auto
  thus "1 - a * q ^ i ≥ 0"
    by auto
  show "1 - a * q ^ i ≤ 1"
    using assms by auto
qed

lemma norm_qbinomial_le_qpochhammer_inf_strong:
  fixes q :: "'a :: {real_normed_field}"
  assumes q: "norm q < 1"
  shows "norm (qbinomial q n k) ≤
    qpochhammer_inf (-(norm q ^ (n + 1 - k))) (norm q) /
    qpochhammer_inf (norm q) (norm q)"
proof (cases "k ≤ n")
  case k: True
  have "norm (qbinomial q n k) =
    norm (qpochhammer (int k) (q ^ (n + 1 - k)) q) /
    norm (qpochhammer (int k) q q)"
    using q k by (subst qbinomial_conv_qpochhammer') (simp_all add: norm_divide)
  also have "... ≤ qpochhammer (int k) (-(norm (q ^ (n + 1 - k))) (norm
q) /
    qpochhammer (int k) (norm q) (norm q))"
    by (intro frac_le norm_qpochhammer_nonneg_le_qpochhammer norm_qpochhammer_nonneg_ge_qpochhammer_nonneg_qpochhammer_pos)
    (use assms in <auto intro: order.trans[OF _ norm_ge_zero]>)
  also have "... ≤ qpochhammer_inf (-(norm (q ^ (n+1-k)))) (norm q) /
qpochhammer_inf (norm q) (norm q)"
    by (intro frac_le qpochhammer_le_qpochhammer_inf qpochhammer_ge_qpochhammer_inf
qpochhammer_inf_pos qpochhammer_inf_nonneg)
    (use assms in <auto simp: norm_power power_le_one_iff simp del:
power_Suc>)

```

```

    finally show ?thesis
      by (simp_all add: norm_power)
qed (use q in <auto intro!: divide_nonneg_nonneg qpochhammer_inf_nonneg>)

lemma norm_qbinomial_le_qpochhammer_inf:
  fixes q :: "'a :: {real_normed_field}"
  assumes q: "norm q < 1"
  shows "norm (qbinomial q n k) ≤
        qpochhammer_inf (-norm q) (norm q) / qpochhammer_inf (norm
q) (norm q)"
proof (cases "k ≤ n")
  case True
  have "norm (qbinomial q n k) ≤
        qpochhammer_inf (-(norm q ^ (n + 1 - k))) (norm q) /
        qpochhammer_inf (norm q) (norm q)"
    by (rule norm_qbinomial_le_qpochhammer_inf_strong) (use q in auto)
  also have "... ≤ qpochhammer_inf (-norm q) (norm q) / qpochhammer_inf
(norm q) (norm q)"
  proof (rule divide_right_mono)
    show "qpochhammer_inf (- (norm q ^ (n + 1 - k))) (norm q) ≤ qpochhammer_inf
(- norm q) (norm q)"
  proof (intro has_prod_le[OF has_prod_qpochhammer_inf has_prod_qpochhammer_inf]
conjI)
    fix i :: nat
    have "norm q ^ (n + 1 - k + i) ≤ norm q ^ (Suc i)"
      by (intro power_decreasing) (use assms True in simp_all)
    thus "1 - - (norm q ^ (n + 1 - k)) * norm q ^ i ≤ 1 - - norm q
* norm q ^ i"
      by (simp_all add: power_add)
  qed (use assms in auto)
  qed (use assms in <auto intro!: qpochhammer_inf_nonneg>)
  finally show ?thesis .
qed (use q in <auto intro!: divide_nonneg_nonneg qpochhammer_inf_nonneg>)

```

4.4 Limits of the q -binomial coefficients

The following limit is Fact 7.7 in Andrews & Eriksson [2].

```

lemma tendsto_qbinomial1:
  fixes q :: "'a :: {real_normed_field, banach, heine_borel}"
  assumes q: "norm q < 1"
  shows "(λn. qbinomial q n m) → 1 / qpochhammer m q q"
proof -
  have not_one: "q ^ k ≠ 1" if "k > 0" for k :: nat
    using power_eq_1_iff[of q k] that q by force
  have [simp]: "q ≠ 1"
    using q by auto

  define P where "P = (λn. qpochhammer (int n) q q)"
  have [simp]: "qpochhammer_inf q q ≠ 0"

```

```

    using q by (auto simp: qpochhammer_inf_zero_iff not_one simp flip:
power_Suc)
    have [simp]: "P m ≠ 0"
    proof
      assume "P m = 0"
      then obtain k where k: "q * q powi k = 1" "k ∈ {0..<int m}"
        by (auto simp: P_def qpochhammer_eq_0_iff power_int_add)
      show False
        by (use k not_one[of "Suc (nat k)"] in <auto simp: power_int_add
power_int_def>)
    qed

    have [tendsto_intros]: "(λn. P (h n)) ⟶ qpochhammer_inf q q"
      if h: "filterlim h at_top at_top" for h :: "nat ⇒ nat"
      unfolding P_def using filterlim_compose[OF qpochhammer_tendsto_qpochhammer_inf[OF
q] h, of q] .
    have "(λn. P n / (P m * P (n - m))) ⟶ 1 / P m"
      by (auto intro!: tendsto_eq_intros filterlim_ident filterlim_minus_const_nat_at_top)
    also have "(∀F n in at_top. P n / (P m * P (n - m)) = qbinomial q n
m)"
      using eventually_ge_at_top[of m]
      by eventually_elim (auto simp: qbinomial_conv_qpochhammer P_def not_one
of_nat_diff)
    hence "(λn. P n / (P m * P (n - m))) ⟶ 1 / P m ⟷
(λn. qbinomial q n m) ⟶ 1 / P m"
      by (intro filterlim_cong) auto
    finally show "(λn. qbinomial q n m) ⟶ 1 / qpochhammer m q q"
      unfolding P_def .
  qed

```

The following limit is a slightly stronger version of Fact 7.8 in Andrews & Eriksson [2]. Their version has $f(n) = rn + c_1$ and $g(n) = sn + c_2$ with $r > s$.

```

lemma tendsto_qbinomial2:
  fixes q :: "'a :: {real_normed_field, banach, heine_borel}"
  assumes q: "norm q < 1"
  assumes lim_fg: "filterlim (λn. f n - g n) at_top F"
  assumes lim_g: "filterlim g at_top F"
  shows "(λn. qbinomial q (f n) (g n)) ⟶ 1 / qpochhammer_inf q
q) F"
proof -
  have not_one: "q ^ k ≠ 1" if "k > 0" for k :: nat
    using power_eq_1_iff[of q k] that q by force
  have [simp]: "q ≠ 1"
    using q by auto

  define P where "P = (λn. qpochhammer (int n) q q)"
  have [simp]: "qpochhammer_inf q q ≠ 0"
    using q by (auto simp: qpochhammer_inf_zero_iff not_one simp flip:

```

```

power_Suc)
  have lim_f: "filterlim f at_top F"
    using lim_fg by (rule filterlim_at_top_mono) auto

  have fg: "eventually ( $\lambda n. f\ n \geq g\ n$ ) F"
  proof -
    have "eventually ( $\lambda n. f\ n - g\ n > 0$ ) F"
      using lim_fg by (metis eventually_gt_at_top filterlim_iff)
    thus ?thesis
      by eventually_elim auto
  qed
  from lim_g and fg have lim_f: "filterlim f at_top F"
    using filterlim_at_top_mono by blast

  have [tendsto_intros]: " $((\lambda n. P\ (h\ n)) \longrightarrow \text{qpochhammer\_inf}\ q\ q)\ F"$ 
    if h: "filterlim h at_top F" for h
    unfolding P_def using filterlim_compose[OF qpochhammer_tendsto_qpochhammer_inf[OF
q] h, of q] .
  have " $((\lambda n. P\ (f\ n) / (P\ (g\ n) * P\ (f\ n - g\ n))) \longrightarrow 1 / \text{qpochhammer\_inf}\ q\ q)\ F"$ "
    by (auto intro!: tendsto_eq_intros lim_f lim_g lim_fg)
  also from fg have " $(\forall_F\ n\ \text{in}\ F. P\ (f\ n) / (P\ (g\ n) * P\ (f\ n - g\ n))) = \text{qbinomial}\ q\ (f\ n)\ (g\ n)"$ "
    = qbinomial q (f n) (g n)"
    by eventually_elim
    (auto simp: qbinomial_qfact not_one of_nat_diff qfact_conv_qpochhammer
power_int_minus power_int_diff P_def field_simps)
  hence " $((\lambda n. P\ (f\ n) / (P\ (g\ n) * P\ (f\ n - g\ n))) \longrightarrow 1 / \text{qpochhammer\_inf}\ q\ q)\ F \longleftrightarrow$ 
 $((\lambda n. \text{qbinomial}\ q\ (f\ n)\ (g\ n)) \longrightarrow 1 / \text{qpochhammer\_inf}\ q\ q)\ F"$ "
    by (intro filterlim_cong) auto
  finally show " $((\lambda n. \text{qbinomial}\ q\ (f\ n)\ (g\ n)) \longrightarrow 1 / \text{qpochhammer\_inf}\ q\ q)\ F"$ " .
  qed

```

4.5 Useful identities

The following lemmas give a recurrence for the infinite q -Pochhammer symbol similar to the one for the “normal” Pochhammer symbol.

```

lemma qpochhammer_inf_mult_power_q:
  assumes "norm q < 1"
  shows "qpochhammer_inf a q = qpochhammer (int n) a q * qpochhammer_inf
(a * q ^ n) q"
proof -
  have " $(\lambda n. 1 - a * q ^ n)$  has_prod qpochhammer_inf a q"
    by (rule has_prod_qpochhammer_inf) (use assms in auto)
  hence "convergent_prod ( $\lambda n. 1 - a * q ^ n$ )"
    by (simp add: has_prod_iff)
  hence " $(\lambda n. 1 - a * q ^ n)$  has_prod

```

```

      (( $\prod_{k < n}. 1 - a * q ^ k$ ) * ( $\prod_{k}. 1 - a * q ^ (k + n)$ ))"
    by (intro has_prod_ignore_initial_segment')
    also have " $(\prod_{k}. 1 - a * q ^ (k + n)) = (\prod_{k}. 1 - (a * q ^ n) * q ^ k)$ "
    by (simp add: power_add mult_ac)
    also have " $(\lambda k. 1 - (a * q ^ n) * q ^ k)$  has_prod qepochhammer_inf (a * q ^ n) q"
    by (rule has_prod_qepochhammer_inf) (use assms in auto)
    hence " $(\prod_{k}. 1 - (a * q ^ n) * q ^ k) = \text{qepochhammer\_inf } (a * q ^ n) q$ "
    by (simp add: qepochhammer_inf_def)
    finally show ?thesis
    by (simp add: qepochhammer_inf_def has_prod_iff qepochhammer_nonneg_def)
qed

```

One can express the finite q -Pochhammer symbol in terms of the infinite one:

$$(a; q)_n = \frac{(a; q)_\infty}{(a; q^n)_\infty}$$

```

lemma qepochhammer_conv_qepochhammer_inf_nonneg:
  assumes "norm q < 1" " $\bigwedge m. m \geq n \implies a * q ^ m \neq 1$ "
  shows "qepochhammer (int n) a q = qepochhammer_inf a q / qepochhammer_inf (a * q ^ n) q"
proof (cases "qepochhammer_inf (a * q ^ n) q = 0")
  case False
  thus ?thesis
    by (subst qepochhammer_inf_mult_power_q[OF assms(1), of _ n])
      (auto simp: qepochhammer_inf_zero_iff)
next
  case True
  with assms obtain k where "a * q ^ (n + k) = 1"
  by (auto simp: qepochhammer_inf_zero_iff power_add mult_ac)
  moreover have "n + k  $\geq$  n"
  by auto
  ultimately have " $\exists m \geq n+k. a * q ^ m = 1$ "
  by blast
  with assms have False
  by auto
  thus ?thesis ..
qed

```

```

lemma qepochhammer_conv_qepochhammer_inf:
  fixes q a :: "'a :: {real_normed_field, banach, heine_borel}"
  assumes q: "norm q < 1" "n < 0  $\implies$  q  $\neq$  0"
  assumes not_one: " $\bigwedge k. \text{int } k \geq n \implies a * q ^ k \neq 1$ "
  shows "qepochhammer n a q = qepochhammer_inf a q / qepochhammer_inf (a * q powi n) q"
proof (cases "n  $\geq$  0")
  case n: True

```

```

define m where "m = nat n"
have n_eq: "n = int m"
  using n by (auto simp: m_def)
show ?thesis unfolding n_eq
  by (subst qpochhammer_conv_qpochhammer_inf_nonneg) (use q not_one
in <auto simp: n_eq>)
next
case n: False
define m where "m = nat (-n)"
have n_eq: "n = -int m" and m: "m > 0"
  using n by (auto simp: m_def)
have nz: "qpochhammer_inf a q ≠ 0"
  using q not_one n by (auto simp: qpochhammer_inf_zero_iff)
have "qpochhammer n a q = 1 / qpochhammer (int m) (a / q ^ m) q"
  using <m > 0> by (simp add: n_eq qpochhammer_minus)
also have "... = qpochhammer_inf a q / qpochhammer_inf (a / q ^ m) q"
  using qpochhammer_inf_mult_power_q[OF q(1), of "a / q ^ m" m] nz q
n
  by (auto simp: divide_simps)
also have "a / q ^ m = a * q powi n"
  by (simp add: n_eq power_int_minus field_simps)
finally show "qpochhammer n a q = qpochhammer_inf a q / qpochhammer_inf
(a * q powi n) q" .
qed

lemma qpochhammer_inf_divide_power_q:
  assumes "norm q < 1" and [simp]: "q ≠ 0"
  shows "qpochhammer_inf (a / q ^ n) q = (∏ k = 1..n. 1 - a / q ^ k)
* qpochhammer_inf a q"
proof -
  have "qpochhammer_inf (a / q ^ n) q =
qpochhammer (int n) (a / q ^ n) q * qpochhammer_inf (a / q ^ n
* q ^ n) q"
  using assms(1) by (rule qpochhammer_inf_mult_power_q)
  also have "qpochhammer (int n) (a / q ^ n) q = (∏ k < n. 1 - a / q ^ (n
- k))"
  unfolding qpochhammer_nonneg_def by (intro prod.cong) (auto simp:
power_diff)
  also have "... = (∏ k = 1..n. 1 - a / q ^ k)"
  by (intro prod.reindex_bij_witness[of _ "λk. n - k" "λk. n - k"])
auto
  finally show ?thesis
  by simp
qed

lemma qpochhammer_inf_mult_q:
  assumes "norm q < 1"
  shows "qpochhammer_inf a q = (1 - a) * qpochhammer_inf (a * q) q"
  using qpochhammer_inf_mult_power_q[OF assms, of a 1] by simp

```

```

lemma qpochhammer_inf_divide_q:
  assumes "norm q < 1" "q ≠ 0"
  shows "qpochhammer_inf (a / q) q = (1 - a / q) * qpochhammer_inf
a q"
  using qpochhammer_inf_divide_power_q[of q a 1] assms by simp

```

The following lemma allows combining a product of several q -Pochhammer symbols into one by grouping factors:

$$(a; q^m)_\infty (aq; q^m)_\infty \cdots (aq^{m-1}; q^m)_\infty = (a; q)_\infty$$

```

lemma prod_qpochhammer_inf_group:
  assumes "norm q < 1" and "m > 0"
  shows "(∏ i < m. qpochhammer_inf (a * q^i) (q^m)) = qpochhammer_inf
a q"
proof (rule has_prod_unique2)
  show "(λ n. (∏ i < m. 1 - a * q^i * (q^m) ^ n)) has_prod (∏ i < m. qpochhammer_inf
(a * q^i) (q^m))"
    by (intro has_prod_prod has_prod_qpochhammer_inf)
    (use assms in <auto simp: norm_power power_less_one_iff>)
next
  have "(λ n. 1 - a * q ^ n) has_prod qpochhammer_inf a q"
    by (rule has_prod_qpochhammer_inf) (use assms in auto)
  hence "(λ n. ∏ i = n * m .. < n * m + m. 1 - a * q^i) has_prod qpochhammer_inf
a q"
    by (rule has_prod_group) (use assms in auto)
  also have "(λ n. ∏ i = n * m .. < n * m + m. 1 - a * q^i) = (λ n. ∏ i < m. 1 - a *
q ^ i * (q ^ m) ^ n)"
  proof
    fix n :: nat
    have "(∏ i = n * m .. < n * m + m. 1 - a * q^i) = (∏ i < m. 1 - a * q^(n * m + i))"
      by (intro prod.reindex_bij_witness[of _ "λ i. i + n * m" "λ i. i
- n * m"]) auto
    thus "(∏ i = n * m .. < n * m + m. 1 - a * q^i) = (∏ i < m. 1 - a * q ^ i * (q
^ m) ^ n)"
      by (simp add: power_add mult_ac flip: power_mult)
  qed
  finally show "(λ n. (∏ i < m. 1 - a * q^i * (q^m) ^ n)) has_prod qpochhammer_inf
a q" .
qed

```

In a similar fashion, let w be a primitive n -th root of unity. Then:

$$(a; q)_\infty (aw; q)_\infty (aw^2; q)_\infty \cdots (aw^{m-1}; q)_\infty = (a^m; q^m)_\infty$$

```

lemma prod_qpochhammer_group_primroot:
  assumes "norm q < 1"
  assumes "primroot m w"

```

```

    shows "( $\prod_{k < m}. \text{qpochhammer\_inf } (w \wedge k * a) q) = \text{qpochhammer\_inf } (a \wedge m) (q \wedge m)"
  proof (rule has_prod_unique2)
    interpret primroot m w
    by fact
    show "( $\lambda i. 1 - a \wedge m * (q \wedge m) \wedge i) \text{ has\_prod qpochhammer\_inf } (a \wedge m) (q \wedge m)"
    by (rule has_prod_qpochhammer_inf)
      (use assms pos_order in <auto simp: norm_power power_less_one_iff>)
  next
    interpret primroot m w
    by fact
    have "( $\lambda i. \prod_{k < m}. 1 - (w \wedge k * a) * q \wedge i) \text{ has\_prod } (\prod_{k < m}. \text{qpochhammer\_inf } (w \wedge k * a) q)"
    by (intro has_prod_prod has_prod_qpochhammer_inf) (use assms in auto)
    also have "( $\lambda i. \prod_{k < m}. 1 - (w \wedge k * a) * q \wedge i) = (\lambda i. 1 - a \wedge m * (q \wedge m) \wedge i)"
    proof
      fix i :: nat
      have "( $\prod_{k < m}. 1 - (w \wedge k * a) * q \wedge i) = \text{poly } (\prod_{k < m}. [:1, -(w \wedge k):]) (a * q \wedge i)"
      by (simp add: poly_prod mult_ac)
      also have "( $\prod_{k < m}. [:1, -(w \wedge k):]) = 1 - \text{Polynomial.monom } 1 m"$ 
      by (rule cyclotomic_poly_conv_prod_unity_root' [symmetric])
      also have "poly ... (a * q \wedge i) = 1 - a \wedge m * (q \wedge m) \wedge i"
      by (simp add: poly_monom power_mult_distrib mult_ac flip: power_mult)
      finally show "( $\prod_{k < m}. 1 - (w \wedge k * a) * q \wedge i) = 1 - a \wedge m * (q \wedge m) \wedge i"$ 
    .
    qed
    finally show "( $\lambda i. 1 - a \wedge m * (q \wedge m) \wedge i) \text{ has\_prod } (\prod_{k < m}. \text{qpochhammer\_inf } (w \wedge k * a) q)"
    by simp
  qed$$$$$$ 
```

lemma (in primroot_cis) prod_qpochhammer_group_cis:
 assumes "norm q < 1"
 defines "w \equiv ($\lambda j. \text{cis } (2 * \pi * j * k / n)$)"
 shows "($\prod_{j < n}. \text{qpochhammer_inf } (w j * a) q) = \text{qpochhammer_inf } (a \wedge n) (q \wedge n)"$
 using prod_qpochhammer_group_primroot[OF assms(1) primroot_axioms, of a]
 by (simp add: w_def mult_ac Complex.DeMoivre)

The particular instance of the above for $m = 2$ is the following: A product of two q -Pochhammer symbols $(\pm a; q)_{\infty}$ can be combined into a single q -Pochhammer symbol:

```

lemma qpochhammer_inf_square:
  assumes q: "norm q < 1"
  shows "qpochhammer_inf a q * qpochhammer_inf (-a) q = qpochhammer_inf (a \wedge 2) (q \wedge 2)"

```

```

      (is "?lhs = ?rhs")
proof -
  have *: "primroot 2 (-1 :: 'a)" ..
  show ?thesis
    using prod_qpochhammer_group_primroot[OF assms *] by (simp add: numeral_2_eq_2)
qed

```

4.6 Two series expansions by Euler

The following two theorems and their proofs are taken from Bellman [3][§40]. He credits them, in their original form, to Euler. One could also deduce these relatively easily from the infinite version of the q -binomial theorem (which we will prove later), but the proves given by Bellman are so nice that I do not want to omit them from here.

The first theorem states that for any complex x, t with $|x| < 1$, we have:

$$(t; x)_\infty = \prod_{k=0}^{\infty} (1 - tx^k) = \sum_{n=0}^{\infty} \frac{x^{n(n-1)/2} t^n}{(x-1) \cdots (x^n - 1)}$$

This tells us the power series expansion for $f_x(t) = (t; x)_\infty$.

lemma

```

  fixes x :: complex
  assumes x: "norm x < 1"
  shows sums_qpochhammer_inf_complex:
    "(λn. x^(n*(n-1) div 2) * t^n / (∏ k=1..n. x^k - 1)) sums qpochhammer_inf
  t x"
  and has_fps_expansion_qpochhammer_inf_complex:
    "(λt. qpochhammer_inf t x) has_fps_expansion
    Abs_fps (λn. x^(n*(n-1) div 2) / (∏ k=1..n. x^k - 1))"

```

proof -

For a fixed x , we define $f(t) = (t; x)_\infty$ and note that f satisfies the functional equation $f(t) = (1 - t)f(tx)$.

```

  define f where "f = (λt. qpochhammer_inf t x)"
  have f_eq: "f t = (1 - t) * f (t * x)" for t
    unfolding f_def using qpochhammer_inf_mult_q[of x t] x by simp
  define F where "F = fps_expansion f 0"
  define a where "a = fps_nth F"
  have ana: "f analytic_on UNIV"
    unfolding f_def by (intro analytic_intros) (use x in auto)

```

We note that f is entire and therefore has a Maclaurin expansion, say $f(t) = \sum_{n=0}^{\infty} a_n x^n$.

```

  have F: "f has_fps_expansion F"
    unfolding F_def by (intro analytic_at_imp_has_fps_expansion_0 analytic_on_subset[OF ana]) auto
  have "fps_conv_radius F ≥ ∞"

```

```

    unfolding F_def by (rule conv_radius_fps_expansion) (auto intro!:
analytic_imp_holomorphic ana)
    hence [simp]: "fps_conv_radius F = ∞"
    by simp
    have F_sums: "(λn. fps_nth F n * t ^ n) sums f t" for t
    proof -
    have "(λn. fps_nth F n * t ^ n) sums eval_fps F t"
    using sums_eval_fps[of t F] by simp
    also have "eval_fps F t = f t"
    by (rule has_fps_expansion_imp_eval_fps_eq[OF F, of _ "norm t +
1"])
    (auto intro!: analytic_imp_holomorphic analytic_on_subset[OF
ana])
    finally show ?thesis .
    qed

```

```

have F_eq: "F = (1 - fps_X) * (F oo (fps_const x * fps_X))"
proof -
have "(λt. (1 - t) * (f o (λt. t * x)) t) has_fps_expansion
(fps_const 1 - fps_X) * (F oo (fps_X * fps_const x))"
by (intro fps_expansion_intros F) auto
also have "... = (1 - fps_X) * (F oo (fps_const x * fps_X))"
by (simp add: mult_ac)
also have "(λt. (1 - t) * (f o (λt. t * x)) t) = f"
unfolding o_def by (intro ext f_eq [symmetric])
finally show "F = (1 - fps_X) * (F oo (fps_const x * fps_X))"
using F fps_expansion_unique_complex by blast
qed

```

```

have a_0 [simp]: "a 0 = 1"
using has_fps_expansion_imp_0_eq_fps_nth_0[OF F] by (simp add: a_def
f_def)

```

Applying the functional equation to the Maclaurin series, we obtain a recurrence for the coefficients a_n , namely $a_{n+1} = \frac{a_n x^n}{x^{n+1} - 1}$.

```

have a_rec: "(x ^ Suc n - 1) * a (Suc n) = x ^ n * a n" for n
proof -
have "a (Suc n) = fps_nth F (Suc n)"
by (simp add: a_def)
also have "F = (F oo (fps_const x * fps_X)) - fps_X * (F oo (fps_const
x * fps_X))"
by (subst F_eq) (simp_all add: algebra_simps)
also have "fps_nth ... (Suc n) = x ^ Suc n * a (Suc n) - x ^ n * a
n"
by (simp add: fps_compose_linear a_def)
finally show "(x ^ Suc n - 1) * a (Suc n) = x ^ n * a n"
by (simp add: algebra_simps)
qed

```

```

define tri where "tri = (λn::nat. n * (n-1) div 2)"
have not_one: "x ^ k ≠ 1" if k: "k > 0" for k :: nat
proof -
  have "norm (x ^ k) < 1"
    using x k by (simp add: norm_power power_less_one_iff)
  thus ?thesis
    by auto
qed

```

The recurrence is easily solved and we get $a_n = x^{n(n-1)/2}(x-1)(x^2-1)\cdots(x^n-1)$.

```

have a_sol: "(∏k=1..n. (x^k - 1)) * a n = x ^ tri n" for n
proof (induction n)
  case 0
  thus ?case
    by (simp add: tri_def)
next
  case (Suc n)
  have "(∏k=1..Suc n. (x^k - 1)) * a (Suc n) =
    (∏k=1..n. x ^ k - 1) * ((x ^ Suc n - 1) * a (Suc n))"
    by (simp add: a_rec mult_ac)
  also have "... = (∏k = 1..n. x ^ k - 1) * a n * x ^ n"
    by (subst a_rec) simp_all
  also have "(∏k=1..n. x ^ k - 1) * a n = x ^ tri n"
    by (subst Suc.IH) auto
  also have "x ^ tri n * x ^ n = x ^ (tri n + (2*n) div 2)"
    by (simp add: power_add)
  also have "tri n + (2*n) div 2 = tri (Suc n)"
    unfolding tri_def
    by (subst div_plus_div_distrib_dvd_left [symmetric]) (auto simp:
algebra_simps)
  finally show ?case .
qed

```

```

have a_sol': "a n = x ^ tri n / (∏k=1..n. (x ^ k - 1))" for n
  using not_one a_sol[of n] by (simp add: divide_simps mult_ac)

```

```

show "(λn. x ^ tri n * t ^ n / (∏k=1..n. x ^ k - 1)) sums f t"
  using F_sums[of t] a_sol' by (simp add: a_def)

```

```

have "F = Abs_fps (λn. x^(n*(n-1) div 2) / (∏k=1..n. x^k - 1))"
  by (rule fps_ext) (simp add: a_sol'[unfolded a_def] tri_def)
thus "f has_fps_expansion Abs_fps (λn. x^(n*(n-1) div 2) / (∏k=1..n.
x^k - 1))"
  using F by simp
qed

```

```

lemma sums_qpochhammer_inf_real:
  assumes "|x| < (1 :: real)"
  shows "(λn. x^(n*(n-1) div 2) * t^n / (∏k=1..n. x^k - 1)) sums qpochhammer_inf

```

```

t x"
proof -
  have "(λn. complex_of_real x ^ (n*(n-1) div 2) * of_real t ^ n / (∏k=1..n.
of_real x ^ k - 1))
    sums qepochhammer_inf (of_real t) (of_real x)" (is "?f sums ?S")
  by (intro sums_qepochhammer_inf_complex) (use assms in auto)
  also have "?f = (λn. complex_of_real (x ^ (n*(n-1) div 2) * t ^ n /
(∏k=1..n. x ^ k - 1)))"
  by simp
  also have "qepochhammer_inf (of_real t) (of_real x) = complex_of_real
(qepochhammer_inf t x)"
  by (rule qepochhammer_inf_of_real) fact
  finally show ?thesis
  by (subst (asm) sums_of_real_iff)
qed

```

```

lemma norm_summable_qepochhammer_inf:
  fixes x t :: "'a :: {real_normed_field}"
  assumes "norm x < 1"
  shows "summable (λn. norm (x^(n*(n-1) div 2) * t ^ n / (∏k=1..n.
x^k - 1)))"
proof -
  have "norm x < 1"
  using assms by simp
  then obtain r where "norm x < r" "r < 1"
  using dense by blast
  hence r: "0 < r" "norm x < r" "r < 1"
  using le_less_trans[of 0 "norm x" r] by auto
  define R where "R = Max {2, norm t, r + 1}"
  have R: "r < R" "norm t ≤ R" "R > 1"
  unfolding R_def by auto

  show ?thesis
  proof (rule summable_comparison_test_bigo)
    show "summable (λn. norm ((1/2::real) ^ n))"
    unfolding norm_power norm_divide by (rule summable_geometric) (use
r in auto)
  next
    have "(λn. norm (x ^ (n * (n - 1) div 2) * t ^ n / (∏k = 1..n. x
^ k - 1))) ∈
      O(λn. r^(n*(n-1) div 2) * R ^ n / (1 - r) ^ n)"
    proof (rule bigoI[of _ 1], intro always_eventually_allI)
      fix n :: nat
      have "norm (norm (x^(n*(n-1) div 2) * t ^ n / (∏k=1..n. x^k - 1)))
=
      norm x ^ (n * (n - 1) div 2) * norm t ^ n / (∏k=1..n. norm
(1 - x ^ k))"
      by (simp add: norm_power norm_mult norm_divide norm_minus_commute
abs_prod flip: prod_norm)
    qed
  qed

```

```

    also have "... ≤ norm x ^ (n * (n - 1) div 2) * norm t ^ n / (∏k=1..n.
1 - norm x)"
    proof (intro divide_left_mono mult_pos_pos prod_pos prod_mono conjI
mult_nonneg_nonneg)
      fix k :: nat assume k: "k ∈ {1..n}"
      have "norm x ^ k ≤ norm x ^ 1"
        by (intro power_decreasing) (use assms k in auto)
      hence "1 - norm x ≤ norm (1::'a) - norm (x ^ k)"
        by (simp add: norm_power)
      also have "... ≤ norm (1 - x ^ k)"
        by norm
      finally show "1 - norm x ≤ norm (1 - x ^ k)" .
      have "0 < 1 - norm x"
        using assms by simp
      also have "... ≤ norm (1 - x ^ k)"
        by fact
      finally show "norm (1 - x ^ k) > 0" .
    qed (use assms in auto)
    also have "(∏k=1..n. 1 - norm x) = (1 - norm x) ^ n"
      by simp
    also have "norm x ^ (n*(n-1) div 2) * norm t ^ n / (1 - norm x)
^ n ≤
      r ^ (n*(n-1) div 2) * R ^ n / (1 - r) ^ n"
      by (intro frac_le mult_mono power_mono) (use r R in auto)
    also have "... ≤ abs (r^(n*(n-1) div 2) * R ^ n / (1 - r) ^ n)"
      by linarith
    finally show "norm (norm (x ^ (n * (n - 1) div 2) * t ^ n / (∏k
= 1..n. x ^ k - 1)))
      ≤ 1 * norm (r ^ (n * (n - 1) div 2) * R ^ n / (1
- r) ^ n)"
      by simp
    qed
    also have "(λn. r ^ (n*(n-1) div 2) * R ^ n / (1 - r) ^ n) ∈ O(λn.
(1/2) ^ n)"
      using r R by real_asymp
    finally show "(λn. norm (x ^ (n * (n - 1) div 2) * t ^ n / (∏k =
1..n. x ^ k - 1))) ∈
      O(λn. (1/2) ^ n)" .
  qed
qed

```

The second theorem states that for any complex x, t with $|x| < 1, |t| < 1$, we have:

$$\frac{1}{(t; x)_\infty} = \prod_{k=0}^{\infty} \frac{1}{1 - tx^k} = \sum_{n=0}^{\infty} \frac{t^n}{(1-x) \cdots (1-x^n)}$$

This gives us the multiplicative inverse of the power series from the previous theorem.

lemma

```

fixes x :: complex
assumes x: "norm x < 1" and t: "norm t < 1"
shows sums_inverse_qpochhammer_inf_complex:
  "(λn. t^n / (∏k=1..n. 1 - x^k)) sums inverse (qpochhammer_inf
t x)"
  and has_fps_expansion_inverse_qpochhammer_inf_complex:
    "(λt. inverse (qpochhammer_inf t x)) has_fps_expansion
Abs_fps (λn. 1 / (∏k=1..n. 1 - x^k))"

```

proof -

The proof is very similar to the one before, except that our function is now $g(x) = 1/(t;x)_\infty$ with the functional equation is $g(tx) = (1-t)g(t)$.

```

define f where "f = (λt. qpochhammer_inf t x)"
define g where "g = (λt. inverse (f t))"
have f_nz: "f t ≠ 0" if t: "norm t < 1" for t
proof
  assume "f t = 0"
  then obtain n where "t * x ^ n = 1"
    using x by (auto simp: qpochhammer_inf_zero_iff f_def mult_ac)
  have "norm (t * x ^ n) = norm t * norm (x ^ n)"
    by (simp add: norm_mult)
  also have "... ≤ norm t * 1"
    using x by (intro mult_left_mono) (auto simp: norm_power power_le_one_iff)
  also have "norm t < 1"
    using t by simp
  finally show False
    using <t * x ^ n = 1> by simp
qed

```

```

have mult_less_1: "a * b < 1" if "0 ≤ a" "a < 1" "b ≤ 1" for a b ::
real

```

```

proof -
  have "a * b ≤ a * 1"
    by (rule mult_left_mono) (use that in auto)
  also have "a < 1"
    by fact
  finally show ?thesis
    by simp
qed

```

```

have g_eq: "g (t * x) = (1 - t) * g(t)" if t: "norm t < 1" for t
proof -

```

```

  have "f t = (1 - t) * f (t * x)"
    using qpochhammer_inf_mult_q[of x t] x
    by (simp add: algebra_simps power2_eq_square f_def)
  moreover have "norm (t * x) < 1"
    using t x by (simp add: norm_mult mult_less_1)
  ultimately show ?thesis
    using t by (simp add: g_def field_simps f_nz)

```

qed

```
define G where "G = fps_expansion g 0"
define a where "a = fps_nth G"
have [analytic_intros]: "f analytic_on A" for A
  unfolding f_def by (intro analytic_intros) (use x in auto)
have ana: "g analytic_on ball 0 1" unfolding g_def
  by (intro analytic_intros)
  (use x in <auto simp: qepochhammer_inf_zero_iff f_nz>)
have G: "g has_fps_expansion G" unfolding G_def
  by (intro analytic_at_imp_has_fps_expansion_0 analytic_on_subset[OF
ana]) auto
have "fps_conv_radius G ≥ 1"
  unfolding G_def
  by (rule conv_radius_fps_expansion)
  (auto intro!: analytic_imp_holomorphic ana analytic_on_subset[OF
ana])

have G_sums: "(λn. fps_nth G n * t ^ n) sums g t" if t: "norm t < 1"
for t
proof -
  have "ereal (norm t) < 1"
    using t by simp
  also have "... ≤ fps_conv_radius G"
    by fact
  finally have "(λn. fps_nth G n * t ^ n) sums eval_fps G t"
    using sums_eval_fps[of t G] by simp
  also have "eval_fps G t = g t"
    by (rule has_fps_expansion_imp_eval_fps_eq[OF G, of _ 1])
    (auto intro!: analytic_imp_holomorphic analytic_on_subset[OF
ana] t)
  finally show ?thesis .
qed

have G_eq: "(G oo (fps_const x * fps_X)) - (1 - fps_X) * G = 0"
proof -
  define G' where "G' = (G oo (fps_const x * fps_X)) - (1 - fps_X) *
G"
  have "(λt. (g o (λt. t * x)) t - (1 - t) * g t) has_fps_expansion
G'"
    unfolding G'_def by (subst mult.commute, intro fps_expansion_intros
G) auto
  also have "eventually (λt. t ∈ ball 0 1) (nhds (0::complex))"
    by (intro eventually_nhds_in_open) auto
  hence "eventually (λt. (g o (λt. t * x)) t - (1 - t) * g t = 0) (nhds
0)"
    unfolding o_def by eventually_elim (subst g_eq, auto)
  hence "(λt. (g o (λt. t * x)) t - (1 - t) * g t) has_fps_expansion
```

```

G'  $\longleftrightarrow$ 
  (\lambda t. 0) has_fps_expansion G'"
  by (intro has_fps_expansion_cong refl)
  finally have "G' = 0"
  by (rule fps_expansion_unique_complex) auto
  thus ?thesis
  unfolding G'_def .
qed

have not_one: "x ^ k  $\neq$  1" if k: "k > 0" for k :: nat
proof -
  have "norm (x ^ k) < 1"
  using x k by (simp add: norm_power power_less_one_iff)
  thus ?thesis
  by auto
qed

have a_rec: "a (Suc m) = a m / (1 - x ^ Suc m)" for m
proof -
  have "0 = fps_nth ((G oo (fps_const x * fps_X)) - (1 - fps_X) * G)
(Suc m)"
  by (subst G_eq) simp_all
  also have "... = (x ^ Suc m - 1) * a (Suc m) + a m"
  by (simp add: ring_distrib fps_compose_linear a_def)
  finally show ?thesis
  using not_one[of "Suc m"] by (simp add: field_simps)
qed

have a_0: "a 0 = 1"
  using has_fps_expansion_imp_0_eq_fps_nth_0[OF G] by (simp add: a_def
f_def g_def)
have a_sol: "a n = 1 / ( $\prod_{k=1..n} (1 - x^k)$ )" for n
  by (induction n) (simp_all add: a_0 a_rec)

show "( $\lambda n. t^n / (\prod_{k=1..n} (1 - x^k))$ ) sums (inverse (qepochhammer_inf
t x))"
  using G_sums[of t] t by (simp add: a_sol[unfolded a_def] f_def g_def)

have "G = Abs_fps ( $\lambda n. 1 / (\prod_{k=1..n} (1 - x^k)$ )"
  by (rule fps_ext) (simp add: a_sol[unfolded a_def])
thus "g has_fps_expansion Abs_fps ( $\lambda n. 1 / (\prod_{k=1..n} (1 - x^k)$ )"
  using G by simp
qed

lemma sums_inverse_qepochhammer_inf_real:
  assumes "|x| < (1 :: real)" "|t| < 1"
  shows "( $\lambda n. t^n / (\prod_{k=1..n} (1 - x^k))$ ) sums inverse (qepochhammer_inf
t x)"
proof -

```

```

have "(λn. complex_of_real t ^ n / (∏k=1..n. 1 - of_real x ^ k))
      sums inverse (qepochhammer_inf (of_real t) (of_real x))" (is "?f
sums ?S")
  by (intro sums_inverse_qepochhammer_inf_complex) (use assms in auto)
also have "?f = (λn. complex_of_real (t ^ n / (∏k=1..n. 1 - x ^ k)))"
  by simp
also have "inverse (qepochhammer_inf (of_real t) (of_real x)) =
          complex_of_real (inverse (qepochhammer_inf t x))"
  by (subst qepochhammer_inf_of_real) (use assms in auto)
finally show ?thesis
  by (subst (asm) sums_of_real_iff)
qed

```

```

lemma norm_summable_inverse_qepochhammer_inf:
  fixes x t :: "'a :: {real_normed_field}"
  assumes "norm x < 1" "norm t < 1"
  shows "summable (λn. norm (t ^ n / (∏k=1..n. 1 - x^k)))"
proof (rule summable_comparison_test)
  show "summable (λn. norm t ^ n / (∏k=1..n. 1 - norm x ^ k))"
    by (rule sums_summable, rule sums_inverse_qepochhammer_inf_real) (use
assms in auto)
next
  show "∃N. ∀n≥N. norm (norm (t ^ n / (∏k = 1..n. 1 - x ^ k))) ≤
        norm t ^ n / (∏k = 1..n. 1 - norm x ^ k)"
  proof (intro exI[of _ 0] allI impI)
    fix n :: nat
    have "norm (norm (t ^ n / (∏k=1..n. 1 - x ^ k))) = norm t ^ n / (∏k=1..n.
norm (1 - x ^ k))"
      by (simp add: norm_mult norm_power norm_divide abs_prod flip:prod_norm)
    also have "... ≤ norm t ^ n / (∏k=1..n. 1 - norm x ^ k)"
  proof (intro divide_left_mono mult_pos_pos prod_pos prod_mono)
    fix k assume k: "k ∈ {1..n}"
    have *: "0 < norm (1::'a) - norm (x ^ k)"
      using assms k by (simp add: norm_power power_less_one_iff)
    also have "... ≤ norm (1 - x ^ k)"
      by norm
    finally show "norm (1 - x ^ k) > 0" .
    from * show "1 - norm x ^ k > 0"
      by (simp add: norm_power)
    have "norm (1::'a) - norm (x ^ k) ≤ norm (1 - x ^ k)"
      by norm
    thus "0 ≤ 1 - norm x ^ k ∧ 1 - norm x ^ k ≤ norm (1 - x ^ k)"
      using assms by (auto simp: norm_power power_le_one_iff)
  qed auto
  finally show "norm (norm (t ^ n / (∏k = 1..n. 1 - x ^ k)))
        ≤ norm t ^ n / (∏k = 1..n. 1 - norm x ^ k)" .
qed
qed
qed

```

end

5 q -binomial identities

```
theory Q_Binomial_Identities
  imports Q_Pochhammer_Infinite
begin
```

5.1 The q -binomial theorem

Recall the binomial theorem:

$$(1 + t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

The q -binomial numbers satisfy an analogous theorem:

$$\prod_{k=0}^{n-1} (1 + tq^k) = \sum_{k=0}^n q^{k(k-1)/2} \binom{n}{k}_q t^k$$

It can be seen easily that letting $q \rightarrow 1$ would give us the “normal” binomial theorem.

theorem *qbinomial_theorem*:

```
"qpochhammer (int n) (-t) q = (∑ k ≤ n. qbinomial q n k * q ^ (k choose 2) * t ^ k)"
```

proof (induction n arbitrary: t)

```
case (Suc n)
```

```
have *: "{..Suc n} = insert 0 {1..Suc n}"
```

```
by auto
```

```
have "(∑ k ≤ Suc n. qbinomial q (Suc n) k * q ^ (k choose 2) * t ^ k)
```

```
=
```

```
1 + (∑ k = 1..Suc n. qbinomial q (Suc n) k * q ^ (k choose 2) * t ^ k)"
```

```
unfolding * by (subst sum.insert) (auto simp: binomial_eq_0)
```

```
also have "(∑ k = 1..Suc n. qbinomial q (Suc n) k * q ^ (k choose 2) * t ^ k) =
```

```
(∑ k ≤ n. q ^ (Suc k choose 2) * qbinomial q (Suc n) (Suc k) * t ^ Suc k)"
```

```
by (intro sum.reindex_bij_witness[of _ "Suc" "λk. k - 1"]) auto
```

```
also have "... = (∑ k ≤ n. q ^ (Suc (Suc k) choose 2) * qbinomial q n (Suc k) * t ^ Suc k) +
```

```
(∑ k ≤ n. q ^ (Suc k choose 2) * qbinomial q n k * t ^ Suc k)"
```

```
by (simp add: qbinomial_Suc_Suc ring_distrib sum.distrib power_add mult_ac numeral_2_eq_2)
```

```
also have "(∑ k ≤ n. q ^ (Suc (Suc k) choose 2) * qbinomial q n (Suc k) * t ^ Suc k) =
```

```

      (∑ k=1..Suc n. q ^ (Suc k choose 2) * qbinomial q n k * t
^ k)"
    by (intro sum.reindex_bij_witness[of _ "λk. k - 1" "Suc"]) auto
    also have "... = (∑ k∈insert 0 {1..Suc n}. q ^ (Suc k choose 2) * qbinomial
q n k * t ^ k) - 1"
    by (subst sum.insert) (auto simp: numeral_2_eq_2)
    also have "(∑ k∈insert 0 {1..Suc n}. q ^ (Suc k choose 2) * qbinomial
q n k * t ^ k)
      = (∑ k≤n. q ^ (Suc k choose 2) * qbinomial q n k * t ^ k)"
    by (intro sum.mono_neutral_right) auto
    also have "1 + ((∑ k≤n. q ^ (Suc k choose 2) * qbinomial q n k * t
^ k) -
      1 + (∑ k≤n. q ^ (Suc k choose 2) * qbinomial q n k * t
^ Suc k)) =
      (∑ k≤n. q ^ (Suc k choose 2) * qbinomial q n k * (t ^ Suc
k + t ^ k))"
    unfolding ring_distrib sum.distrib by simp
    also have "... = (∑ k≤n. qbinomial q n k * q ^ (k choose 2) * (q *
t)^k) * (1 + t)"
    by (simp add: sum_distrib_left sum_distrib_right algebra_simps numeral_2_eq_2
power_add)
    also have "... = qpochhammer (int n) (-q * t) q * (1 + t)"
    by (subst Suc.IH [symmetric]) (simp_all add: algebra_simps)
    also have "qpochhammer (int n) (-q * t) q = (∏ k<n. 1 + t * q ^ Suc
k)"
    by (simp add: qpochhammer_def mult_ac)
    also have "... = (∏ k=1..<Suc n. 1 + t * q ^ k)"
    by (intro prod.reindex_bij_witness[of _ "λk. k - 1" "Suc"]) auto
    also have "... * (1 + t) = (∏ k∈insert 0 {1..<Suc n}. 1 + t * q ^ k)"
    by (subst prod.insert) auto
    also have "insert 0 {1..<Suc n} = {..<Suc n}"
    by auto
    also have "(∏ k<Suc n. 1 + t * q ^ k) = qpochhammer (int (Suc n)) (-
t) q"
    unfolding qpochhammer_def by (subst nat_int) auto
    finally show ?case ..
qed (auto simp: binomial_eq_0)

lemma qbinomial_theorem':
  "qpochhammer (int n) t q = (∑ k≤n. qbinomial q n k * q ^ (k choose
2) * (-t) ^ k)"
  using qbinomial_theorem[of n "-t" q] by simp

```

5.2 The infinite q -binomial theorem

Taking the limit $n \rightarrow \infty$ in the q -binomial theorem and interchanging the limits with Tannery's Theorem, we obtain, for any q with $|q| < 1$:

$$\sum_{k=0}^{\infty} \frac{t^k q^{k(k-1)/2}}{[k]_q!(1-q)^k} = \prod_{k=0}^{\infty} (1 + tq^k) = (-t; q)_{\infty}$$

```

theorem qbinomial_theorem_inf:
  fixes q t :: "'a :: {real_normed_field, banach, heine_borel}"
  assumes q: "q ∈ ball 0 1"
  defines "S ≡ (λk. (q ^ (k choose 2) * t ^ k) / (qfact q (int k) * (1
- q) ^ k))"
  shows "summable (λk. norm (S k))" and "(∑ k. S k) = qpochhammer_inf
(-t) q"
proof -
  have q': "norm q < 1"
    using q by auto
  from q have [simp]: "q ≠ 1"
    by auto
  have "(λn. qpochhammer (int n) (-t) q) ⟶ qpochhammer_inf (-t) q"
    by (rule qpochhammer_tendsto_qpochhammer_inf) (use q in auto)
  also have "(λn. qpochhammer (int n) (-t) q) = (λn. (∑ k ≤ n. qbinomial
q n k * q ^ (k choose 2) * t ^ k))"
    by (simp only: qbinomial_theorem)
  finally have "(λn. ∑ k ≤ n. q ^ (k choose 2) * qbinomial q n k * t ^ k)
    ⟶ qpochhammer_inf (-t) q" by (simp only: mult_ac)
  also have "(λn. ∑ k ≤ n. q ^ (k choose 2) * qbinomial q n k * t ^ k)
    =
      (λn. ∑ k ≤ n. qfact q n / qfact q (n - k) * (q ^ (k choose
2) * t ^ k / qfact q k))"
    by (intro ext sum.cong refl, subst qbinomial_qfact') (use q in <auto
simp: field_simps>)
  also have "... = (λn. ∑ k ≤ n. (∏ i < k. qbracket q (n - int i)) * (q ^
(k choose 2) * t ^ k / qfact q k))"
    proof (intro ext sum.cong refl, goal_cases)
      case (1 n k)
      have "(∏ i < k. qbracket q (n - int i)) = (∏ i ∈ {n-k+1..n}. qbracket
q (int i))"
        by (rule prod.reindex_bij_witness[of _ "λi. n - i" "λi. n - i"])
        (use 1 in <auto simp: of_nat_diff>)
      also have "... = (∏ i ∈ {1..n}-{1..n-k}. qbracket q (int i))"
        by (intro prod.cong refl) auto
      also have "... = qfact q n / qfact q (n - k)"
        using q by (subst prod_diff) (auto simp: qbracket_def qfact_int_def
dest: power_eq_1_iff)
      finally show ?case
        using 1 by (simp add: of_nat_diff)
    qed
  qed

```

```

also have "... = (λn. ∑ k ≤ n. (∏ i < k. 1 - q ^ (n - i)) * S k)"
  by (simp add: qbracket_def prod_dividef mult_ac S_def flip: of_nat_diff)
finally have lim1: "(λn. ∑ k ≤ n. (∏ i < k. 1 - q ^ (n - i)) * S k) →
qepochhammer_inf (- t) q" .

define g where "g = (λk. 2 ^ k * (norm q ^ (k choose 2) * norm t ^
k / (1 - norm q) ^ k))"
have g_altdef: "g k = 2 ^ k * norm q powr (k * (k - 1) / 2) * norm t
^ k / (1 - norm q) ^ k"
  if [simp]: "q ≠ 0" for k
proof -
  have "norm q ^ (k choose 2) = norm q powr real (k choose 2)"
    by (auto simp: powr_realpow)
  also have "real (k choose 2) = real k * (real k - 1) / 2"
    unfolding choose_two by (subst real_of_nat_div) (auto simp: )
  finally show ?thesis
    by (simp add: g_def)
qed

have lim2: "eventually (λn. summable (λk. norm ((∏ i < k. 1 - q ^ (n
- i)) * S k)) at_top ∧
  summable (λn. norm (S n)) ∧
  (λn. ∑ k. (∏ i < k. 1 - q ^ (n - i)) * S k) → suminf
S"
proof (rule tannerys_theorem)
  show "(λn. (∏ i < k. 1 - q ^ (n - i)) * S k) → S k" for k
    by (rule tendsto_eq_intros tendsto_power_zero filterlim_minus_const_nat_at_top
refl q')+ simp
  next
    show "∀ F (k, n) in at_top × F at_top. norm ((∏ i < k. 1 - q ^ (n -
i)) * S k) ≤ g k"
      proof (intro always_eventually, safe)
        fix k n :: nat
        have "norm ((∏ i < k. 1 - q ^ (n - i)) * S k) = (∏ i < k. norm (1 -
q ^ (n - i))) * norm (S k)"
          by (simp add: norm_mult flip: prod_norm)
        also have "... ≤ 2 ^ k * (norm q ^ (k choose 2) * norm t ^ k / (1
- norm q) ^ k)"
          proof (rule mult_mono)
            have "(∏ i < k. norm (1 - q ^ (n - i))) ≤ (∏ i < k. 2)"
              proof (intro prod_mono conjI)
                fix i :: nat assume i: "i ∈ {..<k}"
                have "norm (1 - q ^ (n - i)) ≤ norm (1 :: 'a) + norm (q ^ (n
- i))"
                  by norm
                also have "norm (q ^ (n - i)) ≤ norm (q ^ 0)"
                  using q i unfolding norm_power by (intro power_decreasing)
              auto
            finally show "norm (1 - q ^ (n - i)) ≤ 2"

```

```

        by simp
      qed auto
      thus "( $\prod_{i < k}. \text{norm } (1 - q ^ (n - i))$ )  $\leq 2 ^ k$ "
        by simp
    next
      have "norm (S k) = norm q ^ (k choose 2) * norm t ^ k / (norm
(qfact q (int k) * (1 - q) ^ k))"
        by (simp add: S_def norm_divide norm_mult norm_power)
      also have "qfact q (int k) * (1 - q) ^ k = ( $\prod_{k = 1..int k}. 1
- q \text{ powi } k$ )"
        by (simp add: qfact_altdef power_int_minus field_simps)
      also have "... = ( $\prod_{k = 1..k}. 1 - q ^ k$ )"
        by (intro prod.reindex_bij_witness[of _ int nat]) (auto simp:
power_int_def)
      also have "norm ... = ( $\prod_{k=1..k}. \text{norm } (1 - q ^ k)$ )"
        by (simp add: prod_norm)
      also have "1 - norm q  $\leq$  norm (1 - q ^ i)" if "i > 0" for i
      proof -
        have "norm (1 - q ^ i)  $\geq$  norm (1 - q) - norm (q ^ i)"
          by norm
        moreover have "norm q ^ i  $\leq$  norm q ^ 1"
          using q that by (intro power_decreasing) auto
        ultimately show ?thesis
          by (simp add: norm_power)
      qed
      hence "norm q ^ (k choose 2) * norm t ^ k / ( $\prod_{k = 1..k}. \text{norm}
(1 - q ^ k)$ )  $\leq$ 
        norm q ^ (k choose 2) * norm t ^ k / ( $\prod_{i = 1..k}. 1 - \text{norm}
q$ )"
        using q
        by (intro divide_left_mono prod_mono mult_pos_pos prod_pos)
        (auto intro: power_le_one simp: power_less_one_iff dest:
power_eq_1_iff)
      finally show "norm (S k)  $\leq$  norm q ^ (k choose 2) * norm t ^ k
/ (1 - norm q) ^ k"
        by simp
      qed auto
      also have "... = g k"
        by (simp add: g_def)
      finally show "norm (( $\prod_{i < k}. 1 - q ^ (n - i)$ ) * S k)  $\leq$  g k" .
    qed
  next
    show "summable g"
    proof (rule summable_comparison_test_bigo)
      show "g  $\in O(\lambda k. (1/2) ^ k)$ "
      proof (cases "q = 0  $\vee$  t = 0")
        case True
        have "eventually ( $\lambda k. g k = 0$ ) at_top"
          using eventually_gt_at_top[of 2] by eventually_elim (use True

```

```

in <auto simp: g_def>
  from landau_o.big.in_cong[OF this] show ?thesis
    by simp
next
  case False
  hence "q ≠ 0"
    by auto
  have 1: "1 + norm q > 0"
    using q by (auto intro: add_pos_nonneg)
  have 2: "ln (norm q) / 2 < 0"
    using 1 False q by (auto simp: field_simps)
  show ?thesis
    unfolding g_altdf[OF <q ≠ 0>] using False 1 2 by real_asymp
qed
next
  show "summable (λn. norm ((1 / 2) ^ n :: real))"
    by (simp add: norm_power)
qed
qed auto

from lim2 show "summable (λk. norm (S k))"
  by blast

note lim2
also have "(λn. ∑ k. (∏ i<k. 1 - q ^ (n - i)) * S k) = (λn. ∑ k≤n.
(∏ i<k. 1 - q ^ (n - i)) * S k)"
proof (intro ext suminf_finite)
  fix n k :: nat assume k: "k ∉ {..n}"
  hence "n ∈ {..<k}" "q ^ (n - n) = 1"
    by auto
  hence "∃ a∈{..<k}. q ^ (n - a) = 1"
    by blast
  thus "(∏ i<k. 1 - q ^ (n - i)) * S k = 0"
    by auto
qed auto
finally have "(λn. ∑ k≤n. (∏ i<k. 1 - q ^ (n - i)) * S k) → (∑ a.
S a)"
  by blast
with lim1 show "(∑ a. S a) = qepochhammer_inf (-t) q"
  using LIMSEQ_unique by blast
qed

```

5.3 The q -Vandermonde identity

The following is the q -analog of Vandermonde's identity

$$\binom{m+n}{r} = \sum_{i=0}^r \binom{m}{i} \binom{n}{r-i},$$

namely:

$$\binom{m+n}{r}_q = \sum_{i=0}^r \binom{m}{i}_q \binom{n}{r-i}_q q^{(m-i)(r-i)}$$

theorem *qvandermonde*:

```

fixes m n :: nat and q :: "'a :: real_normed_field"
assumes "norm q ≠ 1"
shows "qbinomial q (m + n) r =
      (∑ i ≤ r. qbinomial q m i * qbinomial q n (r - i) * q ^ ((m
- i) * (r - i)))"
proof (cases "q = 0")
  case [simp]: False
  define Q where "Q = fls_const q"
  define X where "X = (fls_X :: 'a fls)"
  have [simp]: "qbinomial (fls_const q) n k = fls_const (qbinomial q n
k)" for n k
    by (induction q n k rule: qbinomial.induct)
      (simp_all add: qbinomial_Suc_Suc fls_plus_const fls_const_mult_const
flip: fls_const_power)
  define F where
    "F = Abs_fps (λk. if k ≤ m + n then qbinomial q (m + n) k * q ^ (k
choose 2) else 0)"
  define G where
    "G = Abs_fps (λk. if k ≤ m then qbinomial q m k * q ^ (k choose 2)
else 0)"
  define H where
    "H = Abs_fps (λk. if k ≤ n then qbinomial q n k * q ^ (k choose 2)
* q ^ (m * k) else 0)"
  have two_times_choose_two: "2 * int (n choose 2) = n * (n - 1)" for
n
proof -
  have "2 * int (n choose 2) = int (2 * (n choose 2))"
    by simp
  also have "2 * (n choose 2) = n * (n - 1)"
    unfolding choose_two by (simp add: algebra_simps)
  finally show ?thesis
    by simp
qed

have *: "(∑ k ∈ A. if x = int k then f k else 0) = (if x ≥ 0 ∧ nat x
∈ A then f (nat x) else 0)"
  if "finite A" for A :: "nat set" and f :: "nat ⇒ 'a" and x
proof -
  have "(∑ k ∈ A. if x = int k then f k else 0) =
      (∑ k ∈ (if x ≥ 0 ∧ nat x ∈ A then {nat x} else {})). if x
= int k then f k else 0"
    using that by (intro sum.mono_neutral_right) auto
  thus ?thesis
    by auto

```

```

qed

have "0 = qpochhammer (m + n) (-X) Q - qpochhammer m (-X) Q * qpochhammer
n (Q ^ m * (-X)) Q"
  unfolding of_nat_add by (subst qpochhammer_nat_add) auto
  also have "... = (∑ k ≤ m + n. qbinomial Q (m + n) k * Q ^ (k choose
2) * X ^ k) -
      (∑ k ≤ m. qbinomial Q m k * Q ^ (k choose 2) * X ^ k)
*
      (∑ k ≤ n. qbinomial Q n k * Q ^ (k choose 2) * Q ^ (m
* k) * X ^ k)"
  by (subst (1 2 3) qbinomial_theorem') (simp add: power_mult_distrib
mult_ac flip: power_mult)
  also have "(∑ k ≤ m + n. qbinomial Q (m + n) k * Q ^ (k choose 2) * X
^ k) = fps_to_fls F"
  by (rule fls_eqI)
  (auto simp: F_def Q_def X_def fls_nth_sum fls_X_power_times_conv_shift
*
  simp flip: fls_const_power)
  also have "(∑ k ≤ m. qbinomial Q m k * Q ^ (k choose 2) * X ^ k) = fps_to_fls
G"
  by (rule fls_eqI)
  (auto simp: G_def Q_def X_def fls_nth_sum fls_X_power_times_conv_shift
*
  simp flip: fls_const_power)
  also have "(∑ k ≤ n. qbinomial Q n k * Q ^ (k choose 2) * Q ^ (m * k)
* X ^ k) = fps_to_fls H"
  by (rule fls_eqI)
  (auto simp: H_def Q_def X_def fls_nth_sum fls_X_power_times_conv_shift
*
  simp flip: fls_const_power)
  also have "fls_nth (fps_to_fls F - fps_to_fls G * fps_to_fls H) (int
r) =
      fps_nth F r - fps_nth (G * H) r"
  by (simp flip: fls_times_fps_to_fls)
  finally have eq: "fps_nth F r = fps_nth (G * H) r"
  by simp

show "qbinomial q (m + n) r =
      (∑ i ≤ r. qbinomial q m i * qbinomial q n (r - i) * q ^ ((m
- i) * (r - i)))"
  proof (cases "r ≤ m + n")
  case True
  have "qbinomial q (m + n) r * q ^ (r choose 2) =
      (∑ i ≤ r. qbinomial q m i * q ^ (i choose 2) * qbinomial q
n (r - i) *
          q ^ ((r - i) choose 2) * q ^ (m * (r - i)))"
  using eq True
  by (auto simp: F_def G_def H_def fps_mult_nth atLeast0AtMost intro!)

```

```

sum.cong)
  also have "... = ( $\sum_{i \leq r}. \text{qbinomial } q \ m \ i \ * \ \text{qbinomial } q \ n \ (r - i)$ )
* q ^
      ((i choose 2) + ((r - i) choose 2) + m *
(r - i)))"
  by (subst power_add)+ (simp add: mult_ac)
  also have "... = ( $\sum_{i \leq r}. \text{qbinomial } q \ m \ i \ * \ \text{qbinomial } q \ n \ (r - i)$ )
*
      q ^ ((r choose 2) + (m - i) * (r - i)))"
proof (intro sum.cong refl, goal_cases)
  case (1 k)
  have eq: "(k choose 2) + (r - k choose 2) + m * (r - k) = (r choose
2) + (m - k) * (r - k)"
  if "k ≤ m" "k ≤ r"
  proof -
  have "2 * (int (k choose 2) + int (r - k choose 2) + m * (int
r - int k)) =
      2 * ((r choose 2) + (int m - int k) * (int r - int k))"
  unfolding ring_distrib two_times_choose_two using that
  apply (cases "k = 0"; cases "r = 0"; cases "r = k")
  apply (simp_all add: of_nat_diff)
  apply (simp_all add: algebra_simps)?
  done
  hence "nat (2 * (int (k choose 2) + int (r - k choose 2) + m *
(int r - int k))) =
      nat (2 * ((r choose 2) + (int m - int k) * (int r - int
k)))" by simp
  hence "2 * ((k choose 2) + (r - k choose 2) + m * (r - k)) =
      2 * ((r choose 2) + (m - k) * (r - k))"
  using that by (simp add: nat_plus_as_int of_nat_diff)
  thus ?thesis
  by simp
qed
show ?case
proof (cases "k ≤ m")
  case True
  thus ?thesis using 1
  by (subst eq) auto
next
  case False
  thus ?thesis using True
  by (auto simp: not_le choose_two)
qed
qed
also have "... = ( $\sum_{i \leq r}. \text{qbinomial } q \ m \ i \ * \ \text{qbinomial } q \ n \ (r - i)$ )
*
      q ^ ((m - i) * (r - i)) * q ^ (r choose 2)"
  by (simp add: sum_distrib_right sum_distrib_left power_add mult_ac)
finally show ?thesis

```

```

    by simp
  next
    case False
    hence "i > m ∨ r - i > n" if "i ≤ r" for i
      using that by linarith
    have "(∑ i ≤ r. qbinomial q m i * qbinomial q n (r - i) * q ^ ((m - i) * (r - i))) = 0"
    proof (intro sum.neutral ballI, goal_cases)
      case (1 i)
      hence "i ≤ r"
        by simp
      hence "i > m ∨ r - i > n"
        using False by linarith
      thus ?case
        by auto
    qed
    thus ?thesis using False
      by simp
  qed
next
case [simp]: True
have "(∑ i ≤ r. qbinomial q m i * qbinomial q n (r - i) * q ^ ((m - i) * (r - i))) =
  (∑ i ∈ (if r ≤ m + n then {min m r} else {}). 1)"
  using True by (intro sum.mono_neutral_cong_right)
  (auto simp: qbinomial_0_left min_def split: if_splits)
also have "... = qbinomial q (m + n) r"
  by auto
finally show ?thesis ..
qed

```

We therefore also get the following identity for the central q -binomial coefficient:

```

corollary qbinomial_square_sum:
  fixes q :: "'a :: real_normed_field"
  assumes q: "norm q ≠ 1"
  shows "(∑ k ≤ n. qbinomial q n k ^ 2 * q ^ (k ^ 2)) = qbinomial q
  (2 * n) n"
proof -
  have "qbinomial q (2 * n) n = (∑ k ≤ n. qbinomial q n k ^ 2 * q ^ ((n - k) ^ 2))"
    using qvandermonde[of q n n n] q
    by (auto simp: power2_eq_square qbinomial_symmetric simp flip: mult_2
  intro!: sum.cong)
  also have "... = (∑ k ≤ n. qbinomial q n k ^ 2 * q ^ (k ^ 2))"
    using q
    by (intro sum.reindex_bij_witness[of _ "λk. n - k" "λk. n - k"])
    (auto simp: qbinomial_symmetric)
  finally show ?thesis ..

```

qed

end

6 Euler's function

```
theory Euler_Function
  imports Q_Pochhammer_Infinite "Lambert_Series.Lambert_Series"
begin
```

6.1 Definition and basic properties

Euler's ϕ function is closely related to the Dedekind η function and the Jacobi ϑ nullwert functions. The q -Pochhammer symbol gives us a simple and convenient way to define it.

```
definition euler_phi :: "'a :: {real_normed_field, banach, heine_borel}
 $\Rightarrow$  'a" where
  "euler_phi q = qpochhammer_inf q q"
```

```
lemma euler_phi_0 [simp]: "euler_phi 0 = 1"
  by (simp add: euler_phi_def)
```

```
lemma euler_phi_of_real: " $|x| < 1 \implies$  euler_phi (of_real x) = of_real
(euler_phi x)"
  unfolding euler_phi_def by (simp add: qpochhammer_inf_of_real)
```

```
lemma abs_convergent_euler_phi:
  assumes "(q :: 'a :: real_normed_div_algebra)  $\in$  ball 0 1"
  shows "abs_convergent_prod ( $\lambda n. 1 - q ^ \text{Suc } n$ )"
proof (rule summable_imp_abs_convergent_prod)
  show "summable ( $\lambda n. \text{norm } (1 - q ^ \text{Suc } n - 1)$ )"
    using assms by (subst summable_Suc_iff) (auto simp: norm_power)
qed
```

```
lemma convergent_euler_phi:
  assumes "(q :: 'a :: {real_normed_field, banach})  $\in$  ball 0 1"
  shows "convergent_prod ( $\lambda n. 1 - q ^ \text{Suc } n$ )"
  using abs_convergent_euler_phi[OF assms] abs_convergent_prod_imp_convergent_prod
  by blast
```

```
lemma has_prod_euler_phi:
  " $\text{norm } q < 1 \implies (\lambda n. 1 - q ^ \text{Suc } n)$  has_prod euler_phi q"
  using has_prod_qpochhammer_inf[of q q] by (simp add: euler_phi_def)
```

```
lemma euler_phi_nonzero [simp]:
  assumes x: " $x \in \text{ball } 0 1$ "
  shows "euler_phi x  $\neq$  0"
  using assms by (simp add: euler_phi_def qpochhammer_inf_nonzero)
```

```

lemma euler_phi_pos_real:
  assumes x: "|x::real| < 1"
  shows "euler_phi x > 0"
proof (rule has_prod_pos[OF has_prod_euler_phi[of x]])
  fix n :: nat
  have "|x ^ Suc n| = |x| ^ Suc n"
    by (simp only: power_abs)
  also have "... < 1 ^ Suc n"
    by (intro power_strict_mono) (use x in auto)
  finally show "1 - x ^ Suc n > 0"
    unfolding power_one by linarith
qed (use x in auto)

lemma holomorphic_euler_phi [holomorphic_intros]:
  assumes [holomorphic_intros]: "f holomorphic_on A"
  assumes " $\bigwedge z. z \in A \implies \text{norm } (f z) < 1$ "
  shows " $(\lambda z. \text{euler\_phi } (f z)) \text{ holomorphic\_on } A$ "
proof -
  have *: "euler_phi holomorphic_on ball 0 1"
    unfolding euler_phi_def by (intro holomorphic_intros) auto
  show ?thesis
    by (rule holomorphic_on_compose_gen[OF assms(1) *, unfolded o_def])
  (use assms(2) in auto)
qed

lemma analytic_euler_phi [analytic_intros]:
  assumes [analytic_intros]: "f analytic_on A"
  assumes " $\bigwedge z. z \in A \implies \text{norm } (f z) < 1$ "
  shows " $(\lambda z. \text{euler\_phi } (f z)) \text{ analytic\_on } A$ "
  using assms(2) by (auto intro!: analytic_intros simp: euler_phi_def)

lemma meromorphic_on_euler_phi [meromorphic_intros]:
  "f analytic_on A  $\implies (\bigwedge z. z \in A \implies \text{norm } (f z) < 1) \implies (\lambda z. \text{euler\_phi } (f z)) \text{ meromorphic\_on } A$ "
  unfolding euler_phi_def by (intro meromorphic_intros)

lemma continuous_on_euler_phi [continuous_intros]:
  assumes "continuous_on A f" " $\bigwedge z. z \in A \implies \text{norm } (f z) < 1$ "
  shows "continuous_on A  $(\lambda z. \text{euler\_phi } (f z))$ "
  using assms unfolding euler_phi_def by (intro continuous_intros) auto

lemma continuous_euler_phi [continuous_intros]:
  fixes a q :: "'b :: t2_space  $\Rightarrow$  'a :: {real_normed_field, banach, heine_borel}"
  assumes "continuous (at x within A) f" "norm (f x) < 1"
  shows "continuous (at x within A)  $(\lambda x. \text{euler\_phi } (f x))$ "
  unfolding euler_phi_def by (intro continuous_intros assms)

lemma tendsto_euler_phi [tendsto_intros]:

```

```

assumes [tendsto_intros]: "(f ⟶ c) F" and "norm c < 1"
shows   "((λx. euler_phi (f x)) ⟶ euler_phi c) F"
unfolding euler_phi_def using assms by (auto intro!: tendsto_intros)

lemma uniform_limit_euler_phi:
  fixes A :: "complex set"
  assumes A: "compact A" "A ⊆ ball 0 1"
  shows "uniform_limit A (λn q. ∏ k<n. 1 - q ^ Suc k) euler_phi sequentially"
proof -
  have "uniform_limit (A × A)
        (λn (a, q). ∏ k<n. 1 - a * q ^ k) (λ(a, q). qpochhammer_inf
a q) sequentially"
    by (rule uniform_limit_qpochhammer_inf) (use A in <auto intro!: compact_Times>)
  hence "uniform_limit A (λn q. case (q, q) of (a, q) ⇒ ∏ k<n. 1 - a
* q ^ k)
        (λq. case (q,q) of (a,q) ⇒ qpochhammer_inf a q) sequentially"
    by (rule uniform_limit_compose) auto
  thus ?thesis
    by (simp add: euler_phi_def [abs_def])
qed

```

6.2 Connection to Lambert series

The Lambert series $\sum_{k=1}^{\infty} \frac{1}{k} \frac{q^k}{1-q^k}$ is a logarithm of $\frac{1}{\phi(q)}$:

```

lemma euler_phi_conv_lambert_complex':
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "euler_phi q = exp (-lambert (λn. 1 / of_nat n) q)"
proof (rule has_prod_unique2)
  show "(λn. 1 - q ^ Suc n) has_prod euler_phi q"
    by (rule has_prod_euler_phi) (use q in auto)
  show "(λn. 1 - q ^ Suc n) has_prod exp (-lambert (λn. 1 / of_nat n)
q)"
    by (rule euler_phi_conv_lambert_complex) (use q in auto)
qed

```

```

lemma euler_phi_conv_lambert_real':
  fixes q :: real
  assumes q: "|q| < 1"
  shows "euler_phi q = exp (-lambert (λn. 1 / real n) q)"
proof (rule has_prod_unique2)
  show "(λn. 1 - q ^ Suc n) has_prod euler_phi q"
    by (rule has_prod_euler_phi) (use q in auto)
  show "(λn. 1 - q ^ Suc n) has_prod exp (-lambert (λn. 1 / real n) q)"
    by (rule euler_phi_conv_lambert_real) (use q in auto)
qed

```

As an application of this, we obtain the useful inequality $\phi(q) \geq -\frac{\pi^2 q}{6(1-q)}$ for real $q \in (0, 1)$.

```

theorem ln_euler_phi_ge:
  fixes x :: real
  assumes x: "x ∈ {0 <..<1}"
  shows "ln (euler_phi x) ≥ -pi2 / 6 * x / (1 - x)"
proof -
  have lt1: "x ^ Suc n < 1" for n
    by (subst power_less_one_iff) (use x in auto)
  define t where "t = (1 - x) / x"

  have "-ln (euler_phi x) = lambert (λn. 1 / real n) x"
    by (subst euler_phi_conv_lambert_real') (use x in auto)
  also have "lambert (λn. 1 / real n) x ≤ pi2 / (6 * t)"
  proof (rule sums_le)
    show "(λn. 1 / real (Suc n) * x ^ Suc n / (1 - x ^ Suc n)) sums lambert
      (λn. 1 / real n) x"
      by (rule sums_lambert)
        (use lambert_conv_radius_power_int[of "-1", where ?'a = real]
x
        in <auto simp: field_simps>)
  next
    have "(λn. (x / (1 - x)) * (1 / real (Suc n) ^ 2)) sums ((x / (1 -
x)) * (pi2 / 6))"
      by (intro sums_mult) (use inverse_squares_sums in auto)
    thus "(λn. 1 / real (Suc n) ^ 2 * x / (1 - x)) sums (pi2 / (6 *
t))"
      by (simp add: t_def field_simps)
  next
    fix n :: nat
    define f where "f = (λa. a * x powr a / (1 - x powr a))"
    have *: "f (Suc n) ≤ f 1"
    proof (rule DERIV_nonpos_imp_nonincreasing[of _ _ f])
      show "real (Suc n) ≥ 1"
        by simp
    next
      fix a assume "1 ≤ a" "a ≤ real (Suc n)"
      define f' where "f' = x powr a * (1 - x powr a + ln x * a) / (1
- x powr a) ^ 2"
      have lt1: "x powr a < 1"
        by (subst power01_less_one) (use x <a ≥ 1> in auto)
      have "(f has_real_derivative f') (at a)"
        unfolding f_def f'_def using lt1
        by (auto intro!: derivative_eq_intros simp: field_simps power2_eq_square)
      moreover have "1 + ln x * a ≤ x powr a"
        using exp_ge_add_one_self[of "ln x * a"] x by (simp add: powr_def
mult_ac)
      hence "f' ≤ 0"
        unfolding f'_def using x lt1
        by (intro divide_nonpos_pos zero_less_power mult_nonneg_nonpos)
    auto

```

```

ultimately show "∃y. (f has_real_derivative y) (at a) ∧ y ≤ 0"
  by blast
qed
have "1 / real (Suc n) * x ^ Suc n / (1 - x ^ Suc n) = 1 / real (Suc
n) ^ 2 * f (real (Suc n))"
  using x by (simp add: f_def power2_eq_square powr_realpow del: of_nat_Suc
power_Suc)
  also have "... ≤ 1 / real (Suc n) ^ 2 * f 1"
    by (intro mult_left_mono *) (auto simp del: of_nat_Suc)
  also have "... = 1 / real (Suc n) ^ 2 * x / (1 - x)"
    using x by (simp add: f_def)
  finally show "1 / real (Suc n) * x ^ Suc n / (1 - x ^ Suc n) ≤
1 / real (Suc n) ^ 2 * x / (1 - x)" .
qed
also have "pi^2 / (6 * t) = pi^2 / 6 * x / (1 - x)"
  unfolding t_def using x by (simp add: divide_simps)
finally show ?thesis
  using x by (simp add: t_def divide_simps)
qed

```

6.3 Logarithmic derivative

The logarithmic derivative of ϕ is given by the following Lambert-style series:

$$\frac{\phi'(q)}{\phi(q)} = \sum_{k=0}^{\infty} (k+1) \frac{q^k}{q^{k+1} - 1}$$

```

lemma sums_logderiv_euler_phi:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "(λk. of_nat (Suc k) * q ^ k / (q ^ Suc k - 1)) sums (deriv
euler_phi q / euler_phi q)"
proof -
  from q obtain r where r: "norm q < r" "r < 1"
    using dense by blast

  have "(λk. deriv (λq. 1 - q ^ Suc k) q / (1 - q ^ Suc k)) sums (deriv
euler_phi q / euler_phi q)"
  proof (rule logderiv_producing_complex_uniform_limit)
    show "open (ball 0 r :: complex set)" "q ∈ ball 0 r"
      using q r by auto
    using q r by auto
  next
    have "uniform_limit (cball 0 r :: complex set) (λn x. ∏[k<n. 1 - x
^ Suc k] euler_phi sequentially"
      by (rule uniform_limit_euler_phi) (use r in auto)
    thus "uniform_limit (ball 0 r :: complex set) (λn x. ∏[k<n. 1 - x
^ Suc k] euler_phi sequentially"
      by (rule uniform_limit_on_subset) auto
  qed (use q in <auto intro!: holomorphic_intros>)

```

```

also have "(λk. deriv (λq. 1 - q ^ Suc k) q / (1 - q ^ Suc k)) =
           (λk. of_nat (Suc k) * q ^ k / (q ^ Suc k - 1))"
proof
  fix k :: nat
  have "deriv (λq. 1 - q ^ Suc k) q = -(of_nat (k+1) * q ^ k)"
    by (auto intro!: DERIV_imp_deriv derivative_eq_intros simp del:
power_Suc)
  thus "deriv (λq. 1 - q ^ Suc k) q / (1 - q ^ Suc k) = of_nat (Suc
k) * q ^ k / (q ^ Suc k - 1)"
    by (simp add: divide_simps del: power_Suc) (auto simp: algebra_simps)?
  qed
  finally show ?thesis .
qed

lemma deriv_euler_phi_aux:
  fixes q :: complex
  assumes q: "norm q < 1"
  shows "q * deriv euler_phi q = -lambert of_nat q * euler_phi q"
proof -
  have "(λk. of_nat (Suc k) * q ^ Suc k / (1 - q ^ (Suc k))) sums lambert
of_nat q"
    by (rule sums_lambert)
    (use q lambert_conv_radius_power_of_nat[of 1, where ?'a = complex]
in auto)
  hence "(λk. -(of_nat (Suc k) * q ^ Suc k / (1 - q ^ (Suc k)))) sums
          (-lambert of_nat q)"
    by (intro sums_minus)
  also have "(λk. -(of_nat (Suc k) * q ^ Suc k / (1 - q ^ (Suc k)))) =
            (λk. q * (of_nat (Suc k) * q ^ k / (q ^ (Suc k) - 1)))"
    using q by (auto simp: fun_eq_iff divide_simps) (auto simp: algebra_simps)?
  finally have "(λk. q * (of_nat (Suc k) * q ^ k / (q ^ (Suc k) - 1)))
sums (-lambert of_nat q)" .

  moreover have "(λk. q * (of_nat (Suc k) * q ^ k / (q ^ Suc k - 1)))
sums
                (q * (deriv euler_phi q / euler_phi q))"
    by (intro sums_mult sums_logderiv_euler_phi) (fact q)
  ultimately have "-lambert of_nat q = q * (deriv euler_phi q / euler_phi
q)"
    using sums_unique2 by blast
  thus ?thesis
    using q by (simp add: field_simps)
qed

theorem deriv_euler_phi:
  fixes q :: complex
  assumes q: "norm q < 1" "q ≠ 0"
  shows "deriv euler_phi q = -lambert of_nat q * euler_phi q / q"

```

```
using deriv_euler_phi_aux[of q] assms by (simp add: field_simps)
end
```

References

- [1] G. Andrews, R. Askey, and R. Roy. *Special Functions*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1999.
- [2] G. Andrews and K. Eriksson. *Integer Partitions*. Cambridge University Press, 2004.
- [3] R. Bellman. *A Brief Introduction to Theta Functions*. Athena series. Holt, Rinehart and Winston, 1961.