Real Exponents as the Limits of Sequences of Rational Exponents

Jacques D. Fleuriot
University of Edinburgh

June 17, 2024

Abstract

In this formalisation, we construct real exponents as the limits of sequences of rational exponents. In particular, if $a \geq 1$ and $x \in \mathbb{R}$, we choose an increasing rational sequence r_n such that $\lim_{n\to\infty} r_n = x$. Then the sequence a^{r_n} is increasing and if r is any rational number such that $r > x$, a^{r_n} is bounded above by a^r. By the convergence criterion for monotone sequences, a^{r_n} converges. We define $a^x = \lim_{n\to\infty} a^{r_n}$ and show that it has the expected properties (for $a \geq 0$).

This particular construction of real exponents is needed instead of the usual one using the natural logarithm and exponential functions (which already exists in Isabelle) to support our mechanical derivation of Euler's exponential series as an "infinite polynomial". Aside from helping us avoid circular reasoning, this is, as far as we are aware, the first time real exponents are mechanised in this way within a proof assistant.

Contents

theory *RatPower*
imports *HOL.NthRoot*
begin

1 Rational Exponents

A few lemmas about nth-root.

lemma *real_root_mult_exp_cancel:*
  "⟦ 0 < x; 0 < m; 0 < n ⟧
   *⟹ root (m * n) (x ^ (k * n)) = root m (x ^ k)"*
  by *(simp add: power_mult real_root_pos_unique)*

lemma *real_root_mult_exp_cancel1:*
  *"⟦ 0 < x; 0 < n ⟧ ⟹ root n (x ^ (k * n)) = x ^ k"*
by *(auto dest: real_root_mult_exp_cancel [of _ 1])*

lemma *real_root_mult_exp_cancel2:*
  "⟦ 0 < x; 0 < m; 0 < n ⟧
   *⟹ root (n * m) (x ^ (n * k)) = root m (x ^ k)"*
by *(simp add: mult.commute real_root_mult_exp_cancel)*

lemma *real_root_mult_exp_cancel3:*
  *"⟦ 0 < x; 0 < n ⟧ ⟹ root n (x ^ (n * k)) = x ^ k"*
by *(auto dest: real_root_mult_exp_cancel2 [of _ 1])*

Definition of rational exponents,

definition
  powrat :: "[real,rat] => real" (**infixr** *"pow$_{\mathbb{Q}}$" 80*) **where**
  "x pow$_{\mathbb{Q}}$ r = (if r > 0
              then root (nat (snd(quotient_of r)))
                        (x ^ (nat (fst(quotient_of r))))
              else root (nat (snd(quotient_of r)))
                        (1/x ^ (nat (- fst(quotient_of r)))))"

declare *quotient_of_denom_pos' [simp]*

lemma *powrat_one_eq_one [simp]: "1 pow$_{\mathbb{Q}}$ a = 1"*
  by *(simp add: powrat_def)*

lemma *powrat_zero_eq_one [simp]: "x pow$_{\mathbb{Q}}$ 0 = 1"*
by *(simp add: powrat_def)*

lemma *powrat_one [simp]: "x pow$_{\mathbb{Q}}$ 1 = x"*
by *(simp add: powrat_def)*

lemma *powrat_mult_base:*
      *"(x * y) pow$_{\mathbb{Q}}$ r = (x pow$_{\mathbb{Q}}$ r) * (y pow$_{\mathbb{Q}}$ r)"*
proof *(cases r)*
  case *(Fract a b)*
  then show *?thesis*
    using *powrat_def quotient_of_Fract real_root_mult [symmetric]*
          power_mult_distrib **by** *fastforce*
qed

lemma *powrat_divide:*

```
     "(x / y) pow_Q r = (x pow_Q r)/(y pow_Q r)"
proof (cases r)
  case (Fract a b)
  then show ?thesis
    using powrat_def quotient_of_Fract real_root_divide [symmetric]
          power_divide by fastforce
qed

lemma powrat_zero_base [simp]:
  assumes "r ≠ 0" shows "0 pow_Q r = 0"
proof (cases r)
  case (Fract a b)
  then show ?thesis
  proof (cases "a > 0")
    case True
    then show ?thesis
      using Fract powrat_def quotient_of_Fract zero_less_Fract_iff
      by simp
  next
    case False
    then  have "a ≠ 0"
    using Fract(1) assms rat_number_collapse(1) by blast
  then
    show ?thesis
      using Fract powrat_def quotient_of_Fract zero_less_Fract_iff
      by auto
  qed
qed



lemma powrat_inverse:
      "(inverse y) pow_Q r = inverse(y pow_Q r)"
proof (cases "r=0")
  case True
  then show ?thesis by simp
next
  case False
  then show ?thesis
    by (simp add: inverse_eq_divide powrat_divide)
qed

lemma powrat_minus:
   "x pow_Q (-r) = inverse (x pow_Q r)"
proof (cases r)
  case (Fract a b)
  then show ?thesis
    by (auto simp add: powrat_def divide_inverse real_root_inverse
                       quotient_of_Fract zero_less_Fract_iff)
```

qed

lemma *powrat_gt_zero:*
assumes *"x > 0"* **shows** *"x pow$_{\mathbb{Q}}$ r > 0"*
proof *(cases r)*
case *(Fract a b)*
then show *?thesis*
by *(simp add: assms powrat_def)*
qed

lemma *powrat_not_zero:*
assumes *"x ≠ 0"* **shows** *"x pow$_{\mathbb{Q}}$ r ≠ 0"*
proof *(cases r)*
case *(Fract a b)*
then show *?thesis*
by *(simp add: assms powrat_def)*
qed

lemma *gcd_add_mult_commute: "gcd (m::'a::semiring_gcd) (n + k * m) = gcd m n"*
by *(metis add.commute gcd_add_mult)*

lemma *coprime_add_mult_iff1 [simp]:*
*"coprime (n + k * m) (m::'a::semiring_gcd) = coprime n m"*
by *(simp add: coprime_iff_gcd_eq_1 gcd.commute gcd_add_mult_commute)*

lemma *coprime_add_mult_iff2 [simp]:*
*"coprime (k * m + n) (m::'a::semiring_gcd) = coprime n m"*
by *(simp add: add.commute)*

lemma *gcd_mult_div_cancel_left1 [simp]:*
*"gcd a b * (a div gcd a b) = (a::'a::semiring_gcd)"*
by *simp*

lemma *gcd_mult_div_cancel_left2 [simp]:*
*"gcd b a * (a div gcd b a) = (a::'a::semiring_gcd)"*
by *simp*

lemma *gcd_mult_div_cancel_right1 [simp]:*
*"(a div gcd a b) * gcd a b = (a::'a::semiring_gcd)"*
by *simp*

lemma *gcd_mult_div_cancel_right2 [simp]:*
*"(a div gcd b a) * gcd b a = (a::'a::semiring_gcd)"*
by *simp*

```
lemma real_root_normalize_cancel:
  assumes "0 < x" and "a ≠ 0" and "b > 0"
  shows "root (nat(snd(Rat.normalize(a,b))))
               (x ^ nat(fst(Rat.normalize(a,b)))) =
         root (nat b) (x ^ (nat a))"
proof -
  have "root (nat (b div gcd a b)) (x ^ nat (a div gcd a b)) =
        root (nat b) (x ^ nat a)"
  proof (cases "coprime a b")
    case True
      then show ?thesis by simp
  next
    case False
      have "0 < gcd a b"
        using assms(2) gcd_pos_int by blast
      then have "nat (gcd a b) > 0"
        by linarith
      moreover have "nat (b div gcd a b) > 0"
        using nonneg1_imp_zdiv_pos_iff assms(3) by auto
      moreover
       have "root (nat b) (x ^ nat a) =
             root (nat (gcd a b * (b div gcd a b)))
                   (x ^ nat (gcd a b * (a div gcd a b)))"
         by simp
       ultimately show ?thesis
         using assms(1) gcd_ge_0_int nat_mult_distrib real_root_mult_exp_cancel2

         by presburger
     qed
     then show ?thesis
       by (metis assms(3) fst_conv normalize_def snd_conv)
  qed

lemma powrat_add_pos:
  assumes "0 < x" and "0 < r" and "0 < s"
  shows "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
proof (cases r)
  case (Fract a b)
  assume b0: "b > 0" and rf: "r = Fract a b"
  then have a0: "a > 0"
    using Fract(1) assms(2) zero_less_Fract_iff by blast
  then show ?thesis
    proof (cases s)
      case (Fract c d)
      assume d0: "d > 0" and sf: "s = Fract c d"
      then have c0: "c > 0"
        using assms(3) zero_less_Fract_iff by blast
      then have bd0: "b * d > 0"
          using b0 zero_less_mult_iff Fract(2) by blast
```

```
          then show ?thesis
          proof (cases "a * d > 0 ∧ c * b > 0")
            case True
            assume abcd: "a * d > 0 ∧ c * b > 0"
            then have adcb0: "a * d + c * b > 0" by simp
            have "x ^ nat (a * d + c * b) =
                  ((x ^ (nat a)) ^ nat d) * (x ^ (nat c)) ^ nat b"
              using abcd nat_mult_distrib nat_add_distrib
                    zero_less_Fract_iff Fract(3) a0 c0
              by (simp add: power_mult power_add)
            then have "root (nat b)
                        (root (nat d)
                           (x ^ nat (a * d + c * b))) =
                       root (nat b)
                        (root (nat d)
                          (((x ^ (nat a)) ^ nat d) * (x ^ (nat c)) ^ nat
b))"
              by simp
            also have "... = root (nat b)
                              (root (nat d) ((x ^ (nat a)) ^ nat d) *

                               root (nat d) ((x ^ (nat c)) ^ nat b))"
              by (simp add: Fract(2) real_root_mult)
            also have "... = root (nat b)
                              (root (nat d) ((x ^ (nat a)) ^ nat d)) *
                             root (nat b)
                              (root (nat d) ((x ^ (nat c)) ^ nat b))"
              using real_root_mult by blast
            also have "... = root (nat b) (x ^ (nat a)) *
                             root (nat b)
                              (root (nat d) ((x ^ (nat c)) ^ nat b))"
              using real_root_power_cancel Fract(2) zero_less_nat_eq assms(1)

                    less_imp_le by simp
            also have "... =
                       root (nat b) (x ^ nat a) *  root (nat d) (x ^ nat
c)"
              using  real_root_power [of "nat d"] real_root_power_cancel

                     real_root_pos_pos_le zero_less_nat_eq
                     Fract(2) zero_less_nat_eq b0 assms(1) by auto
            finally have "root (nat b) (root (nat d) (x ^ nat (a * d +
c * b))) =
                          root (nat b) (x ^ nat a) *  root (nat d) (x
^ nat c)"
              by assumption
            then show ?thesis using a0 b0 c0 d0 bd0 abcd adcb0 assms(1)
sf rf
              by (auto simp add:  powrat_def quotient_of_Fract
```

```
                    real_root_mult_exp nat_mult_distrib
                    zero_less_Fract_iff real_root_normalize_cancel)
        next
          case False
          then show ?thesis
            by (simp add: a0 b0 c0 d0)
        qed
    qed
qed

lemma powrat_add_neg:
  assumes "0 < x" and "r < 0" and "s < 0"
  shows "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
proof -
  have "x pow_Q (- r + - s) = x pow_Q - r * x pow_Q - s"
    using assms powrat_add_pos neg_0_less_iff_less by blast
  then show ?thesis
    by (metis inverse_eq_imp_eq inverse_mult_distrib
        minus_add_distrib powrat_minus)
qed

lemma powrat_add_neg_pos:
    assumes pos_x: "0 < x" and
            neg_r: "r < 0" and
            pos_s: "0 < s"
    shows "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
proof (cases "r + s > 0")
  assume exp_pos: "r + s > 0"
  have "-r > 0" using neg_r by simp
  then have "x pow_Q (r + s) * (x pow_Q -r) =  (x pow_Q s)"
    using exp_pos pos_x powrat_add_pos by fastforce
  then have "x pow_Q (r + s) * inverse (x pow_Q r) =  (x pow_Q s)"
    by (simp add: powrat_minus)
  then show "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
    by (metis Groups.mult_ac(3) assms(1) mult.right_neutral
        order_less_irrefl powrat_gt_zero right_inverse)
next
  assume exp_pos: "¬ r + s > 0"
  then have "r + s = 0 ∨ r + s < 0" using neq_iff by blast
  then show "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
  proof
    assume exp_zero: "r + s = 0"
    then have "x pow_Q (r + s) = 1" by simp
    also have "... = (x pow_Q r) * inverse (x pow_Q r)"
      using pos_x powrat_not_zero by simp
    also have "... = (x pow_Q r) * (x pow_Q - r)"
      by (simp add: powrat_minus)
    finally show "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
      using exp_zero minus_unique by blast
```

```
    next
     assume "r + s < 0"
     have "-s < 0" using pos_s by simp
     then have "x pow_Q (r + s) * (x pow_Q -s) =  (x pow_Q r)"
       using ‹r + s < 0› pos_x powrat_add_neg by fastforce
     then have "x pow_Q (r + s) * inverse (x pow_Q s) =  (x pow_Q r)"
       by (simp add: powrat_minus)
     then show "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
       by (metis divide_eq_eq divide_real_def
             less_irrefl pos_x powrat_not_zero)
     qed
qed

lemma powrat_add_pos_neg:
 "⟦ 0 < x; 0 < r; s < 0 ⟧
  ⟹ x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
by (metis add.commute mult.commute powrat_add_neg_pos)

lemma powrat_add:
  assumes "0 < x"
  shows "x pow_Q (r + s) = (x pow_Q r) * (x pow_Q s)"
  proof (cases "(r > 0 ∨ r ≤ 0) ∧ (s > 0 ∨ s ≤ 0)")
    case True
    then show ?thesis using assms
      by (auto dest: powrat_add_pos powrat_add_neg powrat_add_neg_pos

           powrat_add_pos_neg simp add: le_less)
next
  case False
  then show ?thesis  by auto
qed

lemma powrat_diff:
     "0 < x ⟹  x pow_Q (a - b) = x pow_Q a / x pow_Q b"
by (metis add_uminus_conv_diff divide_inverse powrat_add powrat_minus)

lemma powrat_mult_pos:
  assumes "0 < x" and "0 < r" and "0 < s"
  shows "x pow_Q (r * s) = (x pow_Q r) pow_Q s"
proof (cases r)
  case (Fract a b)
  assume b0: "b > 0" and rf: "r = Fract a b" and coab: "coprime a b"
  have a0: "a > 0"
    using assms(2) b0 rf zero_less_Fract_iff by blast
  then show ?thesis
  proof (cases s)
    case (Fract c d)
    assume d0: "d > 0" and sf: "s = Fract c d" and coad: "coprime c
d"
```

```
      then have c0: "c > 0"
        using assms(3) zero_less_Fract_iff by blast
      then have  "b * d > 0"
        using b0 d0 by simp
      then show ?thesis using a0 c0 b0 d0 rf sf coab coad assms
        by (auto intro: mult_pos_pos simp add: powrat_def quotient_of_Fract
            zero_less_Fract_iff real_root_normalize_cancel real_root_power
            real_root_mult_exp [symmetric] nat_mult_distrib power_mult

            mult.commute)
  qed
qed

lemma powrat_mult_neg:
  assumes "0 < x" "r < 0" and "s < 0"
  shows "x pow_Q (r * s) = (x pow_Q r) pow_Q s"
proof -
  have " x pow_Q (- r * - s) = (x pow_Q - r) pow_Q - s"
    using powrat_mult_pos assms neg_0_less_iff_less by blast
  then show ?thesis
    by (simp add: powrat_inverse powrat_minus)
qed


lemma powrat_mult_neg_pos:
  assumes "0 < x" and "r < 0" and "0 < s"
  shows "x pow_Q (r * s) = (x pow_Q r) pow_Q s"
proof -
  have "x pow_Q (- r * s) = (x pow_Q - r) pow_Q s"
    using powrat_mult_pos assms neg_0_less_iff_less by blast
  then show ?thesis
    by (simp add: powrat_inverse powrat_minus)
qed

lemma powrat_mult_pos_neg:
  assumes "0 < x" and "0 < r" and "s < 0"
  shows "x pow_Q (r * s) = (x pow_Q r) pow_Q s"
proof -
  have "x pow_Q (r * - s) = (x pow_Q r) pow_Q - s"
    using powrat_mult_pos assms neg_0_less_iff_less by blast
  then show ?thesis
    by (simp add: powrat_minus)
qed

lemma powrat_mult:
  assumes "0 < x" shows "x pow_Q (r * s) = (x pow_Q r) pow_Q s"
proof -
  {fix q::rat
    assume "q = 0" then have "x pow_Q (q * s) = (x pow_Q q) pow_Q s"
```

```
    by simp
  }
  then show ?thesis
    by (metis assms linorder_neqE_linordered_idom mult_zero_right
        powrat_mult_neg powrat_mult_neg_pos powrat_mult_pos
        powrat_mult_pos_neg powrat_zero_eq_one)
qed

lemma powrat_less_mono:
  assumes "r < s" and "1 < x"
  shows "x  pow_Q r < x pow_Q s"
  proof (cases r)
    case (Fract a b)
    assume r_assms: "r = Fract a b" "0 < b" "coprime a b"
    then show ?thesis
  proof (cases s)
    case (Fract c d)
    assume s_assms: "s = Fract c d" "0 < d" "coprime c d"
    have adcb: "a * d < c * b"
      using assms r_assms s_assms by auto
    have b_ba: "0 < nat (b * d)"
      by (simp add: r_assms(2) s_assms(2))
    have root0: "0 ≤ root (nat d) (x ^ nat c)"
                "0 ≤ root (nat d) (1 / x ^ nat (- c))"
      using assms(2) real_root_pos_pos_le by auto
    then show ?thesis
    proof (auto simp add: powrat_def quotient_of_Fract zero_less_Fract_iff

                          s_assms r_assms)
      assume ac0: "0 < a" "0 < c"
      then have "(x ^ nat a) ^ nat d < (x ^ nat c) ^ nat b"
        using adcb assms(2) r_assms(2) less_imp_le mult_pos_pos
              nat_mono_iff nat_mult_distrib power_mult
              power_strict_increasing
        by metis
      then have "root (nat b) (x ^ nat a) ^ nat (b * d) <
                 root (nat d) (x ^ nat c) ^ nat (b * d)"
        using assms r_assms s_assms ac0 real_root_power
              [symmetric] nat_mult_distrib
        by (auto simp add:  power_mult)
      then show "root (nat b) (x ^ nat a) < root (nat d) (x ^ nat c)"
        using power_less_imp_less_base root0(1) by blast
    next
      assume "0 < a" "¬ 0 < c"
      then show "root (nat b) (x ^ nat a) <
                 root (nat d) (1 / x ^ nat (- c))"
        using assms(1) less_trans r_assms(1) r_assms(2) s_assms(1)
              s_assms(2) zero_less_Fract_iff by blast
    next
```

```
      assume ac0: "¬ 0 < a" "0 < c"
      then have "a = 0 ∨ a < 0" by auto
      then show "root (nat b) (1 / x ^ nat (- a)) <
                 root (nat d) (x ^ nat c)"
      proof
        assume "a = 0" then show ?thesis
          by (simp add: ac0(2) assms(2) r_assms(2) s_assms(2))
      next
        assume a0: "a < 0"
        have "(1 / x ^ nat (- a)) ^ nat d < 1"
          using a0 s_assms(2) assms(2) adcb power_mult [symmetric]
                power_one_over nat_mult_distrib [symmetric]
          by (metis (no_types) eq_divide_eq_1 inverse_eq_divide inverse_le_1_iff
              le_less neg_0_less_iff_less one_less_power power_one_over
              zero_less_nat_eq)
        moreover have "1 < (x ^ nat c) ^ nat b"
          by (simp add: ac0(2) assms(2) r_assms(2))
        ultimately have "(1 / x ^ nat (- a)) ^ nat d < (x ^ nat c) ^ nat
b"
          by linarith
        then have "root (nat b) (1 / x ^ nat (- a)) ^ nat (b * d)
                   < root (nat d) (x ^ nat c) ^ nat (b * d)"
        using  assms r_assms s_assms ac0 real_root_power [symmetric] nat_mult_distrib
        by (auto simp add:  power_mult)
        then show ?thesis
          using power_less_imp_less_base root0(1) by blast
      qed
    next
      assume ac0: "¬ 0 < a" "¬ 0 < c"
      then show "root (nat b) (1 / x ^ nat (- a)) <
                 root (nat d) (1 / x ^ nat (- c))"
      proof (cases "a = 0 ∨ c = 0")
        assume "a = 0 ∨ c = 0" then show ?thesis
          using assms r_assms s_assms adcb ac0
          by (auto simp add: not_less le_less)
      next
        assume "¬ (a = 0 ∨ c = 0)"
        then have ac00: "a < 0" "c < 0" using ac0 by auto
        then have "1 / x ^ nat (- (a * d)) <
                   1 / x ^ nat (- (c * b))"
          using ac00 r_assms s_assms assms
          by (simp add: divide_inverse mult_pos_neg2)
        then have "(1 / x ^ nat (- a)) ^ nat d <
                   (1 / x ^ nat (- c)) ^ nat b"
          using s_assms r_assms  ac00
          by (auto simp add: power_mult [symmetric]
                power_one_over nat_mult_distrib [symmetric])
        then have "root (nat b) (1 / x ^ nat (- a)) ^ nat (b * d)
                   < root (nat d) (1 / x ^ nat (- c)) ^ nat (b * d)"
```

```
          using assms r_assms s_assms ac0 real_root_power [symmetric]

                nat_mult_distrib
        by (auto simp add:  power_mult)
      then show "root (nat b) (1 / x ^ nat (- a)) <
                 root (nat d) (1 / x ^ nat (- c))"
          using power_less_imp_less_base root0(2) by blast
      qed
    qed
  qed
qed

lemma power_le_imp_le_base2:
  "⟦ (a::'a::linordered_semidom) ^ n ≤ b ^ n; 0 ≤ b; 0 < n ⟧
   ⟹ a ≤ b"
by (auto intro: power_le_imp_le_base [of _ "n - 1"])

lemma powrat_le_mono:
  assumes "r ≤ s" and "1 ≤ x"
  shows "x  pow_ℚ r ≤ x pow_ℚ s"
  by (metis (full_types) assms le_less powrat_less_mono powrat_one_eq_one)

lemma powrat_less_cancel:
  "⟦ x  pow_ℚ r < x  pow_ℚ s; 1 < x ⟧ ⟹ r < s"
  by (metis not_less_iff_gr_or_eq powrat_less_mono)


lemma powrat_less_cancel_iff [simp]:
  "1 < x ⟹ (x pow_ℚ r < x pow_ℚ s) = (r < s)"
by (blast intro: powrat_less_cancel powrat_less_mono)

lemma powrat_le_cancel_iff [simp]:
  "1 < x ⟹ (x  pow_ℚ r ≤ x  pow_ℚ s) = (r ≤ s)"
by (simp add: linorder_not_less [symmetric])


lemma power_inject_exp_less_one [simp]:
  "⟦0 < a; (a::'a::{linordered_field}) < 1 ⟧
   ⟹ a ^ m = a ^ n ⟷ m = n"
by (metis less_irrefl nat_neq_iff power_strict_decreasing)

lemma power_inject_exp_strong [simp]:
  "⟦0 < a; (a::'a::{linordered_field}) ≠ 1 ⟧
   ⟹ a ^ m = a ^ n ⟷ m = n"
by (case_tac "a < 1") (auto simp add: not_less)


lemma nat_eq_cancel: "0 < a ⟹ 0 < b ⟹ (nat a = nat b) = (a = b)"
by auto
```

```
lemma powrat_inject_exp [simp]:
  "1 < x  ⟹ (x pow_Q r = x pow_Q s) = (s = r)"
  by (metis neq_iff powrat_less_cancel_iff)

lemma powrat_inject_exp_less_one [simp]:
  assumes "0 < x" and "x < 1"
  shows "(x pow_Q r = x pow_Q s) = (s = r)"
proof -
  have "1 < inverse x"
    using assms one_less_inverse by blast
  then show ?thesis
    using powrat_inject_exp powrat_inverse by fastforce
qed

lemma powrat_inject_exp_strong [simp]:
   "⟦ 0 < x; x ≠ 1 ⟧  ⟹ (x pow_Q r = x pow_Q s) = (s = r)"
  using powrat_inject_exp_less_one by fastforce


lemma powrat_less_1_cancel_iff [simp]:
  assumes x0: "0 < x" and x1: "x < 1"
  shows "(x pow_Q r < x pow_Q s) = (s < r)"
proof
  assume xrs: "x pow_Q r < x pow_Q s"
  have invx: "1 < 1/x" using assms by simp
  have  "r < s ∨ r ≥ s" using leI by blast
  then show "s < r"
  proof
    assume "r < s"
    then have " inverse x pow_Q r < inverse x pow_Q s" using invx
      by (simp add: inverse_eq_divide)
    then have " x pow_Q s < x pow_Q r"
      by (simp add: powrat_gt_zero powrat_inverse x0)
    then show ?thesis using xrs by linarith
  next
    assume "r ≥ s"
    then show ?thesis
      using less_eq_rat_def xrs by blast
  qed
next
  assume sr: "s < r"
  have invx: "1 < 1/x" using assms by simp
  then have " inverse x pow_Q s < inverse x pow_Q r" using invx sr
    by (simp add: inverse_eq_divide)
  then  show "x pow_Q r < x pow_Q s"
    by (simp add: powrat_gt_zero powrat_inverse x0)
qed
```

```
lemma powrat_le_1_cancel_iff [simp]:
   "⟦0 < x; x < 1⟧ ⟹ (x pow_Q r ≤ x pow_Q s) = (s ≤ r)"
by (auto simp add: le_less)

lemma powrat_ge_one: "x ≥ 1 ⟹ r ≥ 0 ⟹ x pow_Q r ≥ 1"
by (metis powrat_le_mono powrat_zero_eq_one)

lemma isCont_powrat:
  assumes "0 < x" shows "isCont (λx. x pow_Q r) x"
proof (cases r)
  case (Fract a b)
    assume fract_assms: "r = Fract a b" "0 < b" "coprime a b"
    then show ?thesis
    proof (cases "0 < a")
      case True
      then show ?thesis
        using fract_assms isCont_o2 [OF isCont_power [OF continuous_ident]]
        by (auto intro: isCont_real_root simp add: powrat_def zero_less_Fract_iff)
      next
        case False
        then show ?thesis
          using fract_assms assms isCont_real_root  real_root_gt_zero
                continuous_at_within_inverse [intro!]
          by (auto intro!: isCont_o2 [OF isCont_power [OF continuous_ident]]
                   simp add: powrat_def zero_less_Fract_iff divide_inverse

                      real_root_inverse)
      qed
qed

lemma LIMSEQ_powrat_base:
  "⟦ X ⟶ a; a > 0 ⟧ ⟹ (λn. (X n) pow_Q q) ⟶ a pow_Q q"
by (metis isCont_tendsto_compose [where g="λx. x pow_Q q"] isCont_powrat)

lemma powrat_inverse_of_nat_ge_one [simp]:
      "a ≥ 1 ⟹ a pow_Q (inverse (of_nat n)) ≥ 1"
  by (simp add: powrat_ge_one)

lemma powrat_inverse_of_nat_le_self [simp]:
  assumes "1 ≤ a" shows "a pow_Q inverse (rat_of_nat n) ≤ a"
proof -
  have "inverse (rat_of_nat n) ≤ 1"
    by (auto simp add: inverse_le_1_iff)
  also have "a pow_Q 1 ≤ a" by simp
  ultimately show ?thesis
    using assms powrat_le_mono by fastforce
qed
```

```
lemma BseqI2': "∀n≥N. norm (X n) ≤ K ⟹ Bseq X"
  using BfunI eventually_sequentially by blast

lemma Bseq_powrat_inverse_of_nat_ge_one:
      "a ≥ 1 ⟹ Bseq (λn. a pow_ℚ (inverse (of_nat n)))"
by (auto intro: BseqI2' [of 1 _ a] simp add: less_imp_le powrat_gt_zero)

lemma decseq_powrat_inverse_of_nat_ge_one:
      "a ≥ 1 ⟹ decseq (λn. a pow_ℚ (inverse (of_nat (Suc n))))"
unfolding decseq_def by (auto intro: powrat_le_mono)

lemma convergent_powrat_inverse_Suc_of_nat_ge_one:
  assumes "a ≥ 1"
  shows "convergent (λn. a pow_ℚ (inverse (of_nat (Suc n))))"
proof -
  have "Bseq (λn. a pow_ℚ inverse (rat_of_nat n))"
    using Bseq_powrat_inverse_of_nat_ge_one assms by blast
  then have "Bseq (λn. a pow_ℚ inverse (rat_of_nat (Suc n)))"
    using Bseq_ignore_initial_segment [of _ 1] by fastforce
  also have "monoseq (λn. a pow_ℚ inverse (rat_of_nat (Suc n)))"
    using assms decseq_imp_monoseq decseq_powrat_inverse_of_nat_ge_one
by blast
  ultimately show ?thesis
    using Bseq_monoseq_convergent by blast
qed

lemma convergent_powrat_inverse_of_nat_ge_one:
  assumes "a ≥ 1" shows "convergent (λn. a pow_ℚ (inverse (of_nat n)))"
proof -
  have "convergent (λn. a pow_ℚ inverse (rat_of_nat (Suc n)))"
    using convergent_powrat_inverse_Suc_of_nat_ge_one assms by blast
  then obtain L where "(λn. a pow_ℚ inverse (1 + rat_of_nat n)) ⟶
L"
    using convergent_def by auto
  then have "(λn. a pow_ℚ inverse (rat_of_nat (n + 1))) ⟶ L"
    by simp
  then have "(λn. a pow_ℚ inverse (rat_of_nat n)) ⟶ L"
    by (rule LIMSEQ_offset  [of _ 1])
  then show ?thesis using convergent_def by auto
qed

lemma LIMSEQ_powrat_inverse_of_nat_ge_one:
  assumes "a ≥ 1" shows "(λn. a pow_ℚ (inverse (of_nat n))) ⟶ 1"
proof -
  have "convergent(λn. a pow_ℚ inverse (rat_of_nat n))"
    using convergent_powrat_inverse_of_nat_ge_one assms by blast
  then have "∃L. L ≠ 0 ∧ (λn. a pow_ℚ (inverse (of_nat n))) ⟶ L"
    using assms convergent_def powrat_inverse_of_nat_ge_one
          LIMSEQ_le_const not_one_le_zero
```

```
    by metis
  then obtain L
      where l0: "L ≠ 0"
      and liml: "(λn. a pow_Q (inverse (of_nat n))) ⟶ L"
    by blast
  then have "(λn. a pow_Q (inverse (of_nat n)) *
                  a pow_Q (inverse (of_nat n))) ⟶ L * L"
    by (simp add:  tendsto_mult)
  then have "(λn. a pow_Q (2 * inverse (of_nat n))) ⟶  L * L"
    using powrat_add [symmetric] assms by simp
  also have "(2::nat) > 0" by simp
  ultimately
  have "(λn. a pow_Q (2 * inverse (of_nat (n * 2)))) ⟶  L * L"
    using LIMSEQ_linear [of _ "L * L" 2] by blast
  then have "(λn. a pow_Q (inverse (of_nat n))) ⟶  L * L"
    by simp
  then show ?thesis using liml using LIMSEQ_unique l0
    by fastforce
qed

lemma LIMSEQ_powrat_inverse_of_nat_pos_less_one:
  assumes a0: "0 < a" and a1: "a < 1"
  shows "(λn. a pow_Q (inverse (of_nat n))) ⟶ 1"
proof -
  have "inverse a > 1" using a0 a1
    using one_less_inverse by blast
  then have "(λn. inverse a pow_Q inverse (rat_of_nat n)) ⟶ 1"
    using LIMSEQ_powrat_inverse_of_nat_ge_one by simp
  then have "(λn. inverse (a pow_Q inverse (rat_of_nat n))) ⟶ 1"
    using powrat_inverse by simp
  then have "(λx. 1 / inverse (a pow_Q inverse (rat_of_nat x))) ⟶
1/1"
    by (auto intro: tendsto_divide simp only:)
  then show ?thesis  by (auto simp add: divide_inverse)
qed

lemma LIMSEQ_powrat_inverse_of_nat:
   "a > 0 ⟹ (λn. a pow_Q (inverse (of_nat n))) ⟶ 1"
  by (metis LIMSEQ_powrat_inverse_of_nat_ge_one
      LIMSEQ_powrat_inverse_of_nat_pos_less_one leI)


lemma real_root_eq_powrat_inverse:
  assumes "n > 0" shows "root n x = x pow_Q (inverse (of_nat n))"
proof (cases n)
  case 0
  then show ?thesis
    using assms by blast
next
```

```
  case (Suc m)
  assume nSuc: "n = Suc m"
  then have "root (Suc m) x = root (nat (1 + int m)) x"
    by (metis nat_int of_nat_Suc)
  then show ?thesis
    by (auto simp add: nSuc powrat_def of_nat_rat
          zero_less_Fract_iff quotient_of_Fract)
qed

lemma powrat_power_eq:
   "0 < a ⟹ a pow_Q rat_of_nat n = a ^ n"
proof (induction n)
case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case using powrat_add by simp
qed

end
```

2 Real Exponents via Limits

```
theory RealPower
imports RatPower
begin

instance rat :: ab_group_add
by intro_classes

instance rat :: field
by intro_classes

instance rat :: comm_ring_1
by intro_classes

instantiation rat :: dist
begin
definition rat_dist_def:
  "dist x y = of_rat (abs(y - x))"

instance ..
end

instantiation rat :: dist_norm
begin
definition rat_norm_def:
  "norm (q::rat) = of_rat |q|"
```

```
instance
by intro_classes (simp add: rat_dist_def rat_norm_def)
end

instantiation rat :: metric_space
begin

definition uniformity_rat_def [code del]:
  "(uniformity :: (rat × rat) filter) =
      (INF e∈{0 <..}. principal {((x::rat), y). dist x y < e})"

definition open_rat_def [code del]:
  "open (U :: rat set) ⟷
      (∀x∈U. eventually (λ(x', y). x' = x ⟶ y ∈ U) uniformity)"

instance
proof
  show "uniformity =
        (INF e∈{0<..}. principal {(x::rat, y). dist x y < e})"
    using uniformity_rat_def by blast
next
  fix U
  show "open U =
        (∀x∈U. ∀F (x', y::rat) in uniformity. x' = x ⟶ y ∈ U)"
    using open_rat_def by blast
next
  fix x y show "(dist x y = 0) = (x = (y::rat))"
    by (force simp add: rat_dist_def)
next
  fix x y z show "dist x y ≤ dist x z + dist y (z::rat)"
    by (force simp add: rat_dist_def of_rat_add [symmetric] of_rat_less_eq)
qed

end

instance rat :: topological_space
by intro_classes

lemma LIMSEQ_squeeze:
  assumes abc: "∀n. a n ≤ b n ∧ b n ≤ c n"
  and alim: "a ⟶ (L::real)"
  and clim: "c ⟶ L" shows "b ⟶ L"
proof -
  {fix r
  assume r0: "(r::real) > 0"
  then have "∃no. ∀n≥no. |a n - L| < r"
    by (metis LIMSEQ_def alim dist_real_def)
  then obtain p where 1: "∀n≥p. |a n - L| < r" by blast
  have "∃no. ∀n≥no. |c n - L| < r"
```

```
    by (metis LIMSEQ_def r0 clim dist_real_def)
  then obtain k where 2: "∀n≥k. |c n - L| < r" by blast
  have  "∃m. ∀n≥m. |b n - L| < r"
  proof -
    {fix n
      assume n_max: "max p k ≤ n"
      then have a_n: "|a n - L| < r" using 1 by simp
      have c_n: "|c n - L| < r" using n_max 2 by simp
      have "a n ≤ b n ∧ b n ≤ c n" using abc by simp
      then have "|b n - L| < r"
        using a_n c_n by linarith
    }
    then have "∀n≥max p k. |b n - L| < r"
      by blast
    then show ?thesis by blast
  qed
  }
  then have "∀r>0. ∃m. ∀n≥m. |b n - L| < r"
    by blast
  then show ?thesis
    by (simp add: dist_real_def lim_sequentially)
qed

primrec incratseq :: "real ⇒ nat ⇒ rat" where
  "incratseq x 0 = (ϵq. x - 1 < of_rat q ∧ of_rat q < x)"
| "incratseq x (Suc n) =
       (ϵq. max (of_rat(incratseq x n)) (x - 1/(n + 2)) < of_rat q ∧
            of_rat q < x)"

lemma incratseq_0_gt [simp]: "x - 1 < of_rat(incratseq x 0)"
proof -
  have "x - 1 < x" by simp
  then have "∃q. x - 1 < real_of_rat q ∧ real_of_rat q < x"
    using of_rat_dense
    by blast
  then obtain q where " x - 1 < real_of_rat q ∧ real_of_rat q < x"
    by force
  then show ?thesis
    by (auto intro: someI2)
qed

lemma incratseq_0_less [simp]: "of_rat(incratseq x 0) < x"
proof -
  have "x - 1 < x" by simp
  then have "∃q. x - 1 < real_of_rat q ∧ real_of_rat q < x"
    using of_rat_dense
    by blast
  then obtain q where " x - 1 < real_of_rat q ∧ real_of_rat q < x"
    by force
```

```
  then show ?thesis
    by (auto intro: someI2)
qed

declare incratseq.simps [simp del]

lemma incratseq_ub [simp]:
  "real_of_rat (incratseq x n) < x"
proof (induction n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case
  proof (cases "real_of_rat (incratseq x n) ≤ x - 1/(n + 2)")
    case True
    then have "x - 1/(Suc(Suc n)) < x" by simp
    then have "∃q. x - 1 / real (Suc (Suc n))
              < real_of_rat q ∧
              real_of_rat q < x"
      using of_rat_dense by blast

    then have "∃a. (if True then x - 1 / real (n + 2)
                  else real_of_rat
                        (incratseq x n))
                 < real_of_rat a ∧
                 real_of_rat a < x"
      by auto
    then have "real_of_rat
              (SOME q.
                  (if real_of_rat
                       (incratseq x n)
                      ≤ x - 1 / real (n + 2)
                   then x - 1 / real (n + 2)
                   else real_of_rat
                         (incratseq x n))
                  < real_of_rat q ∧
                  real_of_rat q < x)
             < x"
      using True by (fastforce intro: someI2_ex)
    then show ?thesis
      by (auto simp only: incratseq.simps max_def)
  next
    case False
    then have "∃a. real_of_rat (incratseq x n)
                 < real_of_rat a ∧
                 real_of_rat a < x"
      using Suc of_rat_dense by auto
    then have "real_of_rat
```

```
              (SOME q.
                  (if real_of_rat
                       (incratseq x n)
                      ≤ x - 1 / real (n + 2)
                   then x - 1 / real (n + 2)
                   else real_of_rat
                         (incratseq x n))
                  < real_of_rat q ∧
                  real_of_rat q < x)
             < x"
      using False by (fastforce intro: someI2_ex)
  then show ?thesis
    by (simp only: incratseq.simps max_def)
  qed
qed


lemma incratseq_incseq [simp]:
   "incratseq x n < incratseq x (Suc n)"
proof -
  have "max (real_of_rat (incratseq x n)) (x - 1 / real (n + 2)) < x"
    by simp
   then have "∃q. max (real_of_rat (incratseq x n))
          (x - 1 / real (n + 2))
         < real_of_rat q ∧
         real_of_rat q < x"
     using of_rat_dense by blast
   then have "incratseq x n
    < (SOME q.
          max (real_of_rat (incratseq x n))
           (x - 1 / real (n + 2))
          < real_of_rat q ∧
          real_of_rat q < x)"
     by (fastforce intro: someI2_ex simp add: of_rat_less)
   then show ?thesis
     by (simp only: incratseq.simps)
 qed

lemma incratseq_lb [simp]: "x - 1/(Suc n) < real_of_rat (incratseq x
n)"
proof (induction n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case
  proof (cases "real_of_rat (incratseq x n) ≤ x - 1/Suc(Suc n)")
    case True
    then have " 1 / Suc(Suc n) < 1 / Suc n"
```

```
      using Suc.IH by auto
     have "x - 1/Suc(Suc n) < x"
      by simp
    then have "∃a. x - 1 /Suc(Suc n) < real_of_rat a ∧ real_of_rat a
< x"
      using of_rat_dense by blast
    then have "x - 1 / real (Suc (Suc n))
             < real_of_rat
                (SOME q.
                    (if real_of_rat (incratseq x n) ≤ x - 1 / Suc(Suc
n)
                     then x - 1 / Suc(Suc n) else real_of_rat (incratseq
x n))
                    < real_of_rat q ∧
                    real_of_rat q < x)"
      using True by (auto intro: someI2_ex)
    then show ?thesis using True
      by (auto simp add: incratseq.simps(2) max_def)
  next
    case False
    then show ?thesis
      using incratseq_incseq
      by (meson less_trans not_less of_rat_less)
  qed
qed

lemma incseq_incratseq [simp]:
   "incseq (incratseq x)"
by (auto intro!: incseq_SucI less_imp_le)


lemma LIMSEQ_rat_real_ex:
    "∃r. incseq r ∧ (λn. of_rat (r n)) ⟶ (x::real)"
proof -
  have squeeze_left:
       "∀n. x - 1 / real (Suc n)
        ≤ real_of_rat (incratseq x n) ∧ real_of_rat (incratseq x n) ≤
x"
    using incratseq_lb less_imp_le
    by (simp add: less_imp_le)
  moreover have "(λn. x - 1 / real (Suc n)) ⟶ x"
    by (simp only: minus_real_def LIMSEQ_inverse_real_of_nat_add_minus

          inverse_eq_divide [symmetric])
  ultimately have "(λn. real_of_rat (incratseq x n)) ⟶ x"
    using  LIMSEQ_squeeze tendsto_const by fastforce
  then show ?thesis
    using incseq_incratseq by blast
qed
```

```
lemma incseq_incseq_powrat:
     "⟦ 1 ≤ a; incseq r ⟧ ⟹ incseq (λn. a pow_Q (r n))"
by (metis (lifting) incseq_def powrat_le_mono)

lemma ex_less_of_rat: "∃r. (x :: 'a :: archimedean_field) < of_rat r"
  using ex_less_of_int of_rat_of_int_eq by metis

lemma powrat_incseq_bounded:
     "⟦ 1 ≤ a; ∀n. r n < of_rat q; incseq r ⟧ ⟹  a pow_Q (r n) ≤ a pow_Q
q"
  by (simp add: less_imp_le powrat_le_mono)

lemma Bseq_powrat_incseq:
  assumes "1 ≤ a"
  and "incseq r"
  and "∀n. of_rat(r n) ≤ (x :: 'a :: archimedean_field)"
  shows "Bseq (λn. a pow_Q (r n))"
proof -
  from ex_less_of_rat
  obtain q where xq: "x < of_rat q" by blast
  then have "∀n. 0 ≤ a pow_Q r n ∧ a pow_Q r n ≤ a pow_Q of_rat q"
  proof -
    {fix m
      have " r m < q"
        using assms(3) le_less_trans of_rat_less xq by blast
      then have "a pow_Q r m ≤ a pow_Q of_rat q"
        by (simp add: assms(1) powrat_le_mono)}
    then show ?thesis using less_imp_le [OF powrat_gt_zero]
      using assms(1) by auto
  qed
  then show ?thesis
     by (metis BseqI2' abs_of_nonneg real_norm_def)
qed

lemma convergent_powrat_incseq:
   "⟦ 1 ≤ a; incseq r; ∀n. r n ≤ x ⟧ ⟹ convergent (λn. a pow_Q (r n))"
  by (auto intro!: Bseq_monoseq_convergent
       intro: incseq_imp_monoseq incseq_incseq_powrat Bseq_powrat_incseq
[of _ _ x])

definition
    incseq_of :: "real ⇒ (nat ⇒ rat)" where
    "incseq_of x = (SOME r. incseq r ∧  (λn. of_rat (r n)) ——→ x)"

lemma LIMSEQ_incseq_of [simp]:
     "(λn. of_rat (incseq_of x n)) ——→ x"
by (auto intro: someI2_ex [OF LIMSEQ_rat_real_ex] simp add: incseq_of_def)
```

lemma *incseq_incseq_of [simp]:*
"incseq (incseq_of x)"
by *(auto intro: someI2_ex [OF LIMSEQ_rat_real_ex] simp add: incseq_of_def)*

lemma *incseq_of_Suc [simp]:*
"incseq_of x n ≤ incseq_of x (Suc n)"
by *(metis incseq_def incseq_incseq_of le_SucI le_refl)*

lemma *incseq_of_rat_incseq_of:*
"incseq (λn. of_rat(incseq_of x n) :: 'a::linordered_field)"
by *(meson incseq_def incseq_incseq_of of_rat_less_eq)*

lemma *incseq_of_rat:*
"incseq s ⟹ incseq (λn. of_rat(s n) :: 'a :: linordered_field)"
by *(auto simp add: incseq_def of_rat_less_eq)*

lemma *incseq_rat_le_real:*
"⟦ incseq s; (λn. of_rat (s n)) ⟶ x ⟧ ⟹ of_rat (s n) ≤ (x::real)"
by *(auto intro: incseq_le incseq_of_rat)*

lemma *incseq_of_le_self: "of_rat(incseq_of x n) ≤ x"*
by *(auto intro!: incseq_rat_le_real LIMSEQ_incseq_of)*

lemma *powrat_incseq_of_bounded:*
"⟦ 1 ≤ a; x < of_rat r ⟧ ⟹ a pow$_{\mathbb{Q}}$ (incseq_of x n) ≤ a pow$_{\mathbb{Q}}$ r"
by *(meson incseq_of_le_self le_less le_less_trans of_rat_less powrat_le_mono)*

lemma *incseq_powrat_insec_of:*
"1 ≤ a ⟹ incseq (λn. a pow$_{\mathbb{Q}}$ (incseq_of x n))"
by *(auto intro: incseq_incseq_powrat)*

lemma *Bseq_powrat_incseq_of: "1 ≤ a ⟹ Bseq (λn. a pow$_{\mathbb{Q}}$ (incseq_of x n))"*
by *(auto intro!: Bseq_powrat_incseq incseq_of_le_self)*

lemma *convergent_powrat_incseq_of: "1 ≤ a ⟹ convergent (λn. a pow$_{\mathbb{Q}}$ (incseq_of x n))"*
by *(blast intro: Bseq_monoseq_convergent Bseq_powrat_incseq_of incseq_imp_monoseq incseq_powrat_insec_of)*

We're now ready to define real exponentation.

definition
powa :: "[real,real] ⇒ real" (**infixr** *"powa" 80*) **where**
"a powa x = (THE y. (λn. a pow$_{\mathbb{Q}}$ (incseq_of x n)) ⟶ y)"

real exponents.

definition
powreal :: "[real,real] ⇒ real" (**infixr** *"pow$_{\mathbb{R}}$" 80*) **where**

```
    "a pow_ℝ x = (if 0 < a ∧ a < 1 then (inverse a) powa (-x)
                 else if a ≥ 1 then a powa x else 0)"

lemma powreal_eq_powa:
    "a ≥ 1 ⟹ a pow_ℝ x = a powa x"
by (simp add: powreal_def)

lemma LIMSEQ_powrat_incseq_of_ex1:
   "1 ≤ a ⟹ ∃!y. (λn. a pow_ℚ (incseq_of x n))⟶ y"
  by (metis LIMSEQ_unique convergentD convergent_powrat_incseq_of)

lemma LIMSEQ_powa:
   "1 ≤ a ⟹ (λn. a pow_ℚ (incseq_of x n)) ⟶ a powa x"
unfolding powa_def by (erule theI' [OF LIMSEQ_powrat_incseq_of_ex1])

lemma lemma_incseq_incseq_diff_inverse:
   "incseq s ⟹ incseq (λn. (s n :: rat) - 1/of_nat(Suc n))"
  by (auto intro: diff_mono simp add: divide_inverse incseq_def)

lemma lemma_incseq_diff_inverse_ub:
  assumes "incseq s"
  and "(λn. of_rat (s n)) ⟶ x"
shows "of_rat(s n - 1/of_nat(Suc n)) < (x::real)"
proof -
  have "real_of_rat (s n - 1 / rat_of_nat (Suc n)) < real_of_rat (s n)"
    by (simp add: of_rat_diff)
  then show ?thesis
    by (meson assms incseq_rat_le_real linorder_not_less order_trans)

qed

lemma lemma_LIMSEQ_incseq_diff_inverse:
  assumes "(λn. of_rat (s n)) ⟶ x"
  shows " (λn. of_rat(s n - 1/of_nat(Suc n))) ⟶ (x::real)"
proof -
  have "(λx. real_of_rat (s x) - inverse (real (Suc x))) ⟶ x"
    using assms tendsto_diff [OF _ LIMSEQ_inverse_real_of_nat]
    by simp
  then show ?thesis
    by (simp add: divide_inverse  of_rat_diff of_rat_inverse of_rat_add)
qed

lemma lemma_LIMSEQ_powrat_diff_inverse:
  assumes "1 ≤ a"
  and "(λn. a pow_ℚ (s n))⟶ y"
  shows "(λn. a pow_ℚ (s n - 1/of_nat(Suc n))) ⟶ y"
proof -
  have "(λx. a pow_ℚ (1 / rat_of_nat (Suc x))) ⟶ 1"
    using assms(1) LIMSEQ_powrat_inverse_of_nat
```

```
    by (auto intro!: LIMSEQ_Suc simp only: divide_inverse mult_1_left)
  then have " (λn. a pow_Q s n / a pow_Q (1 / rat_of_nat (Suc n))) ⟶
y"
    using assms(2) tendsto_divide by fastforce
  then show ?thesis using powrat_diff assms(1) by auto
qed

lemma lemma_LIMSEQ_powrat_diff_inverse2:
  assumes "1 ≤ a"
  and "(λn. a pow_Q (s n - 1/of_nat(Suc n)))⟶ y"
  shows "(λn. a pow_Q (s n)) ⟶ y"
proof -
  have "(λx. a pow_Q (1 / rat_of_nat (Suc x))) ⟶ 1"
    using assms(1) LIMSEQ_powrat_inverse_of_nat
    by (auto intro!: LIMSEQ_Suc simp only: divide_inverse mult_1_left)
  then have "(λx. a pow_Q inverse (rat_of_nat (Suc x)) *
                a pow_Q (s x - inverse(rat_of_nat (Suc x))))
              ⟶ 1 * y"
    using assms(2) by (auto intro!: tendsto_mult simp only: inverse_eq_divide)
  then show ?thesis using assms(1)
    by (auto simp add: powrat_add [symmetric]  )
qed

lemma lemma_seq_point_gt_ex:
  "⟦ (λn. of_rat (r n)) ⟶ (x::real); y < x ⟧
   ⟹ ∃(m::nat).  y < of_rat(r m)"
  by (metis convergent_def limI lim_le not_less)

lemma lemma_seq_point_gt_ex2:
   "⟦ (λn. of_rat (r n)) ⟶ (x::real); of_rat y < x ⟧
    ⟹ (∃m. y < r m)"
  by (force dest: lemma_seq_point_gt_ex simp add: of_rat_less)

primrec interlaced_index :: "(nat ⇒ rat) ⇒ (nat ⇒ rat) ⇒ nat ⇒ nat"
where
  "interlaced_index r s 0 = 0"
| "interlaced_index r s (Suc n) =
         (LEAST m. if odd n then r (interlaced_index r s n) < s m
                   else s (interlaced_index r s n) < r m)"


definition interlaced_seq :: "(nat ⇒ rat) ⇒ (nat ⇒ rat) ⇒ nat ⇒ rat"
where
  "interlaced_seq r s n = (if odd n then r (interlaced_index r s n)
                           else s (interlaced_index r s n))"

lemma incseq_interlaced_seq:
  assumes "(λn. of_rat (r n)) ⟶ (x::real)"
  and "(λn. of_rat (s n)) ⟶ (x::real)"
```

```
  and "∀n. of_rat (r n) < x"
  and "∀n. of_rat (s n) < x"
  shows "incseq (interlaced_seq r s)"
proof -
  {fix n
  have "interlaced_seq r s n ≤ interlaced_seq r s (Suc n)"
    using assms
    by (auto intro: LeastI2_ex [OF lemma_seq_point_gt_ex2]
          simp add: interlaced_seq_def)}
  then show ?thesis
    by (simp add: incseq_SucI)
qed

lemma incseq_of_rat_interlaced_seq:
   "⟦ (λn. of_rat (r n)) ⟶ (x::real);
      (λn. of_rat (s n)) ⟶ (x::real);
      ∀n. of_rat (r n) < x; ∀n. of_rat (s n) < x ⟧
    ⟹ incseq (λn. real_of_rat (interlaced_seq r s n))"
  using incseq_interlaced_seq incseq_of_rat by blast

lemma interlaced_seq_bounded:
   "⟦ ∀n. of_rat (r n) < x; ∀n. of_rat (s n) < x⟧
    ⟹ of_rat (interlaced_seq r s n) < x"
unfolding interlaced_seq_def by auto


lemma convergent_interlaced_seq:
  assumes "(λn. of_rat (r n)) ⟶ (x::real)"
  and "(λn. of_rat (s n)) ⟶ (x::real)"
  and "∀n. of_rat (r n) < x"
  and "∀n. of_rat (s n) < x"
  shows "convergent (λn. real_of_rat (interlaced_seq r s n))"
proof -
  have mono: "monoseq (λn. real_of_rat (interlaced_seq r s n))"
    using assms incseq_interlaced_seq incseq_of_rat monoseq_iff by blast

  {fix n
     have "interlaced_seq r s 0 ≤ interlaced_seq r s n"
     proof (induction n)
       case 0
       then show ?case by simp
     next
       case (Suc n)
       then show ?case
         using assms incseq_def incseq_interlaced_seq by blast
     qed}
  moreover
  {fix n
   have "real_of_rat (interlaced_seq r s n) ≤ x"
```

```
    by (simp add: assms(3,4) interlaced_seq_bounded le_less)}
  ultimately have "Bseq (λn. real_of_rat (interlaced_seq r s n))"
    by (metis decseq_bounded incseq_bounded mono monoseq_iff of_rat_less_eq)
  then show ?thesis using mono
    by (simp add: Bseq_monoseq_convergent)
qed

lemma convergent_powrat_interlaced_seq:
   "⟦ 1 ≤ a; (λn. of_rat (r n)) ⟶ (x::real);
      (λn. of_rat (s n)) ⟶ (x::real);
      ∀n. of_rat (r n) < x; ∀n. of_rat (s n) < x ⟧
    ⟹ convergent (λn. a pow_Q (interlaced_seq r s n))"
by (auto intro: Bseq_monoseq_convergent incseq_interlaced_seq
           interlaced_seq_bounded less_imp_le incseq_imp_monoseq
           incseq_incseq_powrat Bseq_powrat_incseq [of _ _ x])

lemma LIMSEQ_even_odd_subseq_LIMSEQ:
  assumes "(λn. (X (2 *n))) ⟶ a" "(λn. (X (Suc(2 *n)))) ⟶ a"
  shows "X ⟶ (a :: 'a::real_normed_vector)"
proof (auto simp add: LIMSEQ_iff)
  fix r::real
  assume e0: "r > 0"
  obtain p where evenx: "∀n≥p. norm (X (2 * n) - a) < r"
    using  e0 assms(1) by (metis  LIMSEQ_iff)
  obtain q where oddx: "∀n≥q. norm (X (Suc (2 * n)) - a) < r"
    using  e0 assms(2) by (metis  LIMSEQ_iff)
  {fix n
    assume max: "max (2 * p) (2 * q) ≤ n"
    have "norm (X n - a) < r"
    proof (cases "even n")
      case True
      then show ?thesis
        using dvd_def evenx max
        by auto
    next
      case False
      then show ?thesis using oddx max
        by (auto elim: oddE)
    qed}
  then show "∃no. ∀n≥no. norm (X n - a) < r"
    by blast
qed


lemma incseqD2: "incseq r ⟹ r m < r n ⟹ m < n"
  by (metis le_less mono_def not_le order.asym)

lemma subseq_interlaced_index_even:
  assumes "incseq r"
```

```
  and "incseq s"
  and "(λn. of_rat (r n)) ⟶ (x::real)"
  and "(λn. of_rat (s n)) ⟶ (x::real)"
  and "∀n. of_rat (r n) < x"
  and "∀n. of_rat (s n) < x"
  shows "strict_mono (λn. interlaced_index r s (2 * n))"
proof -
 {fix n
  have "interlaced_index r s (2 * n)
        < (LEAST m. r (LEAST m. s (interlaced_index r s (2 * n)) < r
m) < s m)"
  proof (rule LeastI2_ex [of "λ m. s (interlaced_index r s (2 * n)) <
r m"])
    show "∃a. s (interlaced_index r s (2 * n)) < r a"
      using assms(3) assms(6) lemma_seq_point_gt_ex2 by blast
  next
    fix m
    assume inter_s: "s (interlaced_index r s (2 * n)) < r m"
    show "interlaced_index r s (2 * n) < (LEAST ma. r m < s ma) "
    proof (rule LeastI2_ex)
      show "∃a. r m < s a"
        using assms(4) assms(5) lemma_seq_point_gt_ex2 by blast
    next
      fix k
      assume "r m < s k"
      then show "interlaced_index r s (2 * n) < k"
        using inter_s assms(2) incseqD2 less_trans by blast
    qed
  qed}
  then show ?thesis
    by (simp add: strict_mono_Suc_iff)
qed

lemma subseq_interlaced_index_odd:
  assumes "incseq r"
  and "incseq s"
  and "(λn. of_rat (r n)) ⟶ (x::real)"
  and "(λn. of_rat (s n)) ⟶ (x::real)"
  and "∀n. of_rat (r n) < x"
  and "∀n. of_rat (s n) < x"
shows "strict_mono (λn. interlaced_index r s (Suc (2 * n)))"
proof -
  {fix n
    have "(LEAST m. s (interlaced_index r s (2 * n)) < r m)
            < (LEAST m.
                 s (LEAST m. r (LEAST m. s (interlaced_index r s (2 *
n)) < r m) < s m)
                  < r m)"
    proof (rule LeastI2_ex [of "(λ m. s (interlaced_index r s (2 * n))
```

```
< r m)"])
      show "∃a. s (interlaced_index r s (2 * n)) < r a"
        using assms(3) assms(6) lemma_seq_point_gt_ex2 by blast
    next
      fix m
      assume inter_s: "s (interlaced_index r s (2 * n)) < r m"
      show "m < (LEAST ma. s (LEAST ma. r m < s ma) < r ma)"
      proof (rule LeastI2_ex [of "λma. r m < s ma"])
        show "∃a. r m < s a"
          using assms(4) assms(5) lemma_seq_point_gt_ex2 by blast
      next
        fix k
        assume rs: "r m < s k"
        show "m < (LEAST ma. s k < r ma)"
        proof (rule LeastI2_ex)
          show "∃a. s k < r a"
            using assms(3) assms(6) lemma_seq_point_gt_ex2 by blast
        next
          fix l
          assume "s k < r l"
          then show "m < l" using rs
            using assms(1) incseqD2 less_trans by blast
        qed
      qed
    qed}
  then show ?thesis
    by (simp add: strict_mono_Suc_iff)
qed

lemma interlaced_seq_even:
      "interlaced_seq r s (2*n) = s (interlaced_index r s (2*n))"
  by (simp add: interlaced_seq_def)


lemma interlaced_seq_odd:
      "interlaced_seq r s (Suc (2*n)) = r (interlaced_index r s (Suc (2*n)))"
  by (simp add: interlaced_seq_def)

lemma powa_indep_incseq_of:
  assumes "1 ≤ a"
  and "incseq r"
  and "incseq s"
  and "(λn. real_of_rat (r n)) ⟶ x"
  and "(λn. real_of_rat (s n)) ⟶ x"
  and "(λn. a pow_ℚ (r n)) ⟶ y"
  and "(λn. a pow_ℚ (s n)) ⟶ z"
shows "y = z"
proof -
  have rx: "(λn. real_of_rat (r n - 1 / rat_of_nat (Suc n))) ⟶ x"
```

```
    using assms(4) lemma_LIMSEQ_incseq_diff_inverse by blast
  have sx: "(λn. real_of_rat (s n - 1 / rat_of_nat (Suc n))) ⟶ x"
    using assms(5) lemma_LIMSEQ_incseq_diff_inverse by blast
  have "convergent
      (λn. a pow_ℚ
           interlaced_seq (λn. r n - 1 / rat_of_nat (Suc n))
            (λn. s n - 1 / rat_of_nat (Suc n)) n)"
  using convergent_powrat_interlaced_seq
            [of _ "(λn. r n - 1 / rat_of_nat (Suc n))"
                _ "(λn. s n - 1 / rat_of_nat (Suc n))"]
              assms(1,2,3,4,5) lemma_incseq_diff_inverse_ub
              lemma_LIMSEQ_incseq_diff_inverse by blast
  then obtain L
    where converge: "(λn. a pow_ℚ
                interlaced_seq (λn. r n - 1 / rat_of_nat (Suc n))
                (λn. s n - 1 / rat_of_nat (Suc n)) n) ⟶ L"
    using convergent_def by force
  then have "((λn. a pow_ℚ
                interlaced_seq (λn. r n - 1 / rat_of_nat (Suc n))
                 (λn. s n - 1 / rat_of_nat (Suc n)) n) ∘ (λn. 2 * n))
⟶ L"
    by (simp add: LIMSEQ_subseq_LIMSEQ strict_mono_Suc_iff)
  moreover have "((λn. a pow_ℚ
                interlaced_seq (λn. r n - 1 / rat_of_nat (Suc n))
                 (λn. s n - 1 / rat_of_nat (Suc n)) n) ∘ (λn. Suc (2 *
n))) ⟶ L"
    using converge by (simp add: LIMSEQ_subseq_LIMSEQ strict_mono_Suc_iff)

  moreover have "((λn. a pow_ℚ (r n - 1 / rat_of_nat (Suc n))) ∘
          (λn. interlaced_index (λn. r n - 1 / rat_of_nat (Suc n))
                 (λn. s n - 1 / rat_of_nat (Suc n)) (Suc (2 * n)))) ⟶
y"
  proof -
    from rx sx
    have "strict_mono
            (λn. interlaced_index (λn. r n - 1 / rat_of_nat (Suc n))
              (λn. s n - 1 / rat_of_nat (Suc n)) (Suc (2 * n)))"
      using subseq_interlaced_index_odd lemma_incseq_incseq_diff_inverse
                  lemma_incseq_diff_inverse_ub assms by blast
    moreover  have "(λn. a pow_ℚ (r n - 1 / rat_of_nat (Suc n))) ⟶
y"
      using assms(1) assms(6) lemma_LIMSEQ_powrat_diff_inverse by blast
    ultimately show ?thesis
      using LIMSEQ_subseq_LIMSEQ by blast
  qed
  moreover have "((λn. a pow_ℚ (s n - 1 / rat_of_nat (Suc n))) ∘
          (λn. interlaced_index (λn. r n - 1 / rat_of_nat (Suc n))
                 (λn. s n - 1 / rat_of_nat (Suc n)) (2 * n)))
         ⟶ z"
```

```
  proof -
    from rx sx
    have "strict_mono
     (λn. interlaced_index (λn. r n - 1 / rat_of_nat (Suc n))
           (λn. s n - 1 / rat_of_nat (Suc n)) (2 * n))"
      using subseq_interlaced_index_even lemma_incseq_incseq_diff_inverse
                  lemma_incseq_diff_inverse_ub assms by blast
    moreover have "(λn. a pow_Q (s n - 1 / rat_of_nat (Suc n))) ⟶
z"
      using assms(1) assms(7) lemma_LIMSEQ_powrat_diff_inverse by blast
    ultimately show ?thesis
      using LIMSEQ_subseq_LIMSEQ by blast
  qed
  ultimately show ?thesis
    by (force dest: LIMSEQ_unique simp only: o_def interlaced_seq_even
interlaced_seq_odd)
qed

lemma powa_indep_incseq_of':
      "⟦ 1 ≤ a; incseq r;
         (λn. real_of_rat (r n)) ⟶ x;
         (λn. a pow_Q (r n)) ⟶ y ⟧
       ⟹ (λn. a pow_Q (incseq_of x n)) ⟶ y"
  using powa_indep_incseq_of [of a r "incseq_of x"] LIMSEQ_powa
  by fastforce


lemma lemma_incseq_incseq_of_diff_inverse:
   "incseq (λn. incseq_of x n - 1/of_nat(Suc n))"
by (blast intro: lemma_incseq_incseq_diff_inverse incseq_incseq_of)

lemma lemma_incseq_of_diff_inverse_ub:
   "of_rat(incseq_of x n - 1/of_nat(Suc n)) < x"
by (blast intro: lemma_incseq_diff_inverse_ub incseq_incseq_of LIMSEQ_incseq_of)

 lemma lemma_LIMSEQ_incseq_of_diff_inverse:
    "(λn. of_rat(incseq_of x n - 1/of_nat(Suc n))) ⟶ x"
by (blast intro: lemma_LIMSEQ_incseq_diff_inverse incseq_incseq_of LIMSEQ_incseq_of)

lemma powa_add:
  assumes "1 ≤ a"
  shows "a powa (x + y) = a powa x * a powa y"
proof -
  obtain k where 1: "(λn. a pow_Q incseq_of y n) ⟶ k"
    using LIMSEQ_powrat_incseq_of_ex1 assms by blast
  moreover obtain l where 2: "(λn. a pow_Q incseq_of x n) ⟶ l"
    using LIMSEQ_powrat_incseq_of_ex1 assms by blast
  ultimately have "(λn. a pow_Q (incseq_of x n + incseq_of y n)) ⟶
l * k"
```

```
    using assms by (auto intro:  tendsto_mult simp add: powrat_add )
  moreover
  have "incseq (λn. incseq_of x n + incseq_of y n)"
    by  (force intro: incseq_SucI add_mono)
  moreover have "(λn. real_of_rat (incseq_of x n + incseq_of y n)) ⟶
x + y"
    by (auto simp add: of_rat_add intro: tendsto_add LIMSEQ_incseq_of)
  ultimately have "(λn. a pow_Q incseq_of (x + y) n) ⟶ l * k"
    using assms powa_indep_incseq_of' by blast
  then show ?thesis using powa_def 1 2
    by (metis Lim_def limI)
qed

lemma real_inverse_ge_one_lemma:
      "⟦ 0 < (a::real); a < 1 ⟧ ⟹ inverse a ≥ 1"
by (metis less_eq_real_def one_le_inverse_iff)

lemma real_inverse_gt_one_lemma:
      "⟦ 0 < (a::real); a < 1 ⟧ ⟹ inverse a > 1"
by (metis one_less_inverse_iff)

lemma real_inverse_bet_one_one_lemma:
      "1 < (a::real) ⟹ 0 < inverse a ∧ inverse a < 1"
  by (metis inverse_less_1_iff inverse_positive_iff_positive
        le_less_trans zero_le_one)

lemma powreal_add:
   "a pow_R (x + y) = a pow_R x * a pow_R y"
  by (metis minus_add_distrib mult_zero_right powa_add
       powreal_def real_inverse_ge_one_lemma)

lemma powa_one_eq_one [simp]: "1 powa a = 1"
proof -
  have "(λn. 1 pow_Q incseq_of a n) ⟶ 1"
    by simp
  then show ?thesis
    by (metis Lim_def limI powa_def)
qed

lemma powreal_one_eq_one [simp]: "1 pow_R a = 1"
by (simp add: powreal_def)

lemma powa_zero_eq_one [simp]: "a ≥ 1 ⟹ a powa 0 = 1"
by (auto intro!: the1_equality LIMSEQ_powrat_incseq_of_ex1
           intro: powa_indep_incseq_of' [of a "λn. 0"] incseq_SucI
           simp add: powa_def)

lemma powreal_zero_eq_one [simp]: "a > 0 ⟹ a pow_R 0 = 1"
by (auto dest: real_inverse_ge_one_lemma simp add: powreal_def)
```

```
lemma powr_zero_eq_one_iff [simp]: "x pow_ℝ 0 = (if x ≤ 0 then 0 else
1)"
  using powreal_def powreal_zero_eq_one by force

lemma powa_one_gt_zero [simp]: "1 ≤ a ⟹ a powa 1 = a"
by (auto intro!: LIMSEQ_powrat_incseq_of_ex1 the1_equality
    powa_indep_incseq_of' [of a "λn. 1"]  incseq_SucI simp add: powa_def)

lemma powa_minus_one:
  assumes "1 ≤ a" shows "a powa -1 = inverse a"
proof -
  have "(λn. a pow_ℚ - 1) ⟶ inverse a" using assms
    by (simp add: powrat_minus)
  then have "(λn. a pow_ℚ incseq_of (- 1) n) ⟶ inverse a"
    using powa_indep_incseq_of' [of a "λn. -1"] assms
    by simp
  then show ?thesis using powa_def
    by (metis Lim_def limI)
qed

lemma powreal_minus_one: "0 ≤ a ⟹ a pow_ℝ -1 = inverse a"
  by (simp add: powa_minus_one powreal_def real_inverse_ge_one_lemma)

lemma powreal_one [simp]: "a ≥ 0 ⟹ a pow_ℝ 1 = a"
  by (simp add: powa_minus_one powreal_def real_inverse_ge_one_lemma)

lemma powa_gt_zero:
  assumes "a ≥ 1"
  shows "a powa x > 0"
proof -
  obtain y where 1: "(λn. a pow_ℚ incseq_of x n) ⟶ y"
    using LIMSEQ_powrat_incseq_of_ex1 assms by blast
  moreover have "incseq (λn. a pow_ℚ incseq_of x n)"
    by (simp add: assms incseq_powrat_insec_of)
  ultimately have "y > 0"
    using assms incseq_le powrat_gt_zero
    by (metis less_le_trans zero_less_one)
  then show ?thesis using powa_def 1
    by (metis Lim_def limI)
qed

lemma powreal_gt_zero: "a > 0 ⟹ a pow_ℝ x > 0"
  by (auto dest: powa_gt_zero real_inverse_ge_one_lemma
        simp add: powreal_def not_less)

lemma powreal_not_zero: "a > 0 ⟹ a pow_ℝ x ≠ 0"
by (metis order_less_imp_not_eq powreal_gt_zero)
```

```
lemma powreal_minus:
    "a powℝ -x = inverse (a powℝ x)"
proof (cases "a < 0")
  case True
  then show ?thesis
    using powreal_def by force
next
  case False
  then have "a powℝ (x + -x) = a powℝ x * a powℝ -x"
    using powreal_add by presburger
  then show ?thesis
    using inverse_unique powreal_def powreal_zero_eq_one
    by fastforce
qed

lemma powreal_minus_base_ge_one:
   "a powℝ (-x) = (inverse a) powℝ x"
using one_le_inverse_iff powreal_def  by auto

lemma powreal_inverse:
     "inverse (a powℝ x) = (inverse a) powℝ x"
  using powreal_minus powreal_minus_base_ge_one
  by presburger

lemma powa_minus:"a ≥ 1 ⟹ a powa (-x) = inverse (a powa x)"
  by (metis powreal_eq_powa powreal_minus)

lemma powa_mult_base:
  assumes "1 ≤ a" and "1 ≤ b"
  shows "(a * b) powa x = (a powa x) * (b powa x)"
proof -
  obtain x' where lim_a: "(λn. a powℚ incseq_of x n) ⟶ x'"
    using LIMSEQ_powrat_incseq_of_ex1 assms(1) by blast
   then have lim_a_eq: "(THE y. (λn. a powℚ incseq_of x n) ⟶ y)
= x'"
     by (metis Lim_def limI)
   obtain y' where lim_b: "(λn. b powℚ incseq_of x n) ⟶ y'"
     using LIMSEQ_powrat_incseq_of_ex1 assms(2) by blast
   then have lim_b_eq: "(THE y. (λn. b powℚ incseq_of x n) ⟶ y)
= y'"
     by (metis Lim_def limI)
   have lim_ab: "(λn. (a * b) powℚ incseq_of x n) ⟶ x' * y'"
     using lim_a lim_b by (auto dest: tendsto_mult simp add: powrat_mult_base)
   then have "(THE y. (λn. (a * b) powℚ incseq_of x n) ⟶ y) = x'
* y'"
     by (metis Lim_def limI)
   then show ?thesis
     by (simp add: lim_a_eq lim_b_eq powa_def)
 qed
```

```
lemma powreal_mult_base_lemma1:
  "⟦ 1 ≤ a; 1 ≤ b ⟧
   ⟹ (a * b) powℝ x = (a powℝ x) * (b powℝ x)"
by (metis mult.left_neutral mult_mono' powa_mult_base powreal_eq_powa
zero_le_one)

lemma powreal_mult_base_lemma2:
  assumes "1 ≤ a"
  and "0 < b"
  and "b < 1"
shows "(a * b) powℝ x = (a powℝ x) * (b powℝ x)"
proof (cases "a * b < 1")
  case True
  assume ab1: "a * b < 1"
  have "a * b > 0"
    using assms(1) assms(2) by auto
  then have inv_ab1: "1 ≤ inverse (a * b)"
    using ab1 real_inverse_ge_one_lemma by blast
  then have "(a * (inverse a * inverse b)) powa - x =
             a powa - x * (inverse a * inverse b) powa - x"
    by (simp add: assms(1) powa_mult_base)
  then have "(inverse b) powa - x =
             a powa - x * (inverse a * inverse b) powa - x"
    using assms(1) by (simp add: mult.assoc [symmetric])
  then have "(inverse a * inverse b) powa - x = a powa x * inverse b
powa - x"
    by (simp add:  mult.assoc [symmetric]  powa_add [symmetric] assms(1))
  then show ?thesis
    using ab1 assms powreal_def by auto
next
  case False
  have inv_b: "inverse b ≥ 1"
    by (simp add: assms real_inverse_ge_one_lemma)
  assume "¬ a * b < 1"
  then have "a * b ≥ 1" by simp
  then have "(a * b * inverse b) powa x = (a * b) powa x * inverse b
powa x"
    by (simp add: assms(2) assms(3) powa_mult_base real_inverse_ge_one_lemma)
  then have "a powa x = (a * b) powa x * inverse b powa x"
  using assms(2) by (auto simp add: mult.assoc)
  then have "(a * b) powa x = a powa x * inverse b powa - x"
    by (simp add: mult.assoc powa_add [symmetric] assms inv_b)
  then show ?thesis
    using False assms powreal_def by auto
qed

lemma powreal_mult_base_lemma3:
```

```
  assumes "0 < a"
  and "a < 1"
  and "0 < b"
  and "b < 1"
shows "(a * b) powℝ x = (a powℝ x) * (b powℝ x)"
proof -
  have "a * b < 1" using assms
    by (metis less_trans mult.left_neutral mult_less_cancel_right_disj)

  moreover have "(inverse a * inverse b) powa - x =
                  inverse a powa - x * inverse b powa - x"
    by (simp add: assms powa_mult_base real_inverse_ge_one_lemma)
  ultimately show ?thesis using powreal_def assms by simp
qed

lemma powreal_mult_base:
  assumes "0 ≤ a" and "0 ≤ b"
  shows "(a * b) powℝ x = (a powℝ x) * (b powℝ x)"
proof (cases "a ≥ 1 ∧ b ≥ 1")
  case True
  then show ?thesis
    by (simp add: powreal_mult_base_lemma1)
next
  case False
  then show ?thesis
    using powreal_mult_base_lemma3 powreal_mult_base_lemma2 assms
    by (smt (verit) mult.commute mult_minus_left powreal_def)
qed

lemma incseq_le_all: "incseq X ⟹ X ⟶ L ⟹ ∀n. X n ≤ (L::real)"
by (metis incseq_le)

lemma powa_powrat_eq:
  assumes "a ≥ 1" shows "a powa (of_rat q) = a powℚ q"
proof -
  have "(λn. a powℚ incseq_of (real_of_rat q) n) ⟶ a powℚ q"
    using powa_indep_incseq_of' assms by fastforce
  then show ?thesis using powa_def
    by (metis Lim_def limI)
qed


lemma realpow_powrat_eq: "a > 0 ⟹ a powℝ (of_rat q) = a powℚ q"
proof -
  assume a1: "0 < a"
  then have "¬ a < 1 ∨ 1 ≤ inverse a"
    using real_inverse_ge_one_lemma by blast
  then show ?thesis
    using a1
```

```
    by (metis inverse_inverse_eq not_le powa_powrat_eq
        powrat_inverse powreal_eq_powa powreal_inverse)
qed


lemma LIMSEQ_real_root:
   "⟦ X ⟶ a; m > 0 ⟧ ⟹ (λn. root m (X n)) ⟶ (root m a)"
by (metis isCont_tendsto_compose [where g="λx. root m x"] isCont_real_root)

lemma powa_powrat_lemma1:
  assumes "a ≥ 1" and "p ≥ 0"
  shows "(a powa x) pow_ℚ p = (a powa (x * of_rat p))"
proof -
  obtain y where lim_a: "(λn. a pow_ℚ incseq_of x n) ⟶ y"
    using LIMSEQ_powrat_incseq_of_ex1 assms(1) by blast
   then have the_lim: "(THE y. (λn. a pow_ℚ incseq_of x n) ⟶ y) =
y"
     by (metis Lim_def limI)
   then have  "(λn. (a pow_ℚ incseq_of x n) pow_ℚ p) ⟶ y pow_ℚ p"
     using LIMSEQ_powrat_base assms(1) lim_a powa_def powa_gt_zero by
auto
   then have lim_ap: "(λn. a pow_ℚ (incseq_of x n * p)) ⟶ y pow_ℚ
p"
     using assms(1) powrat_mult by auto
   then have "(λn. a pow_ℚ incseq_of (x * real_of_rat p) n) ⟶ y pow_ℚ
p"
   proof -
     have "incseq (λn. incseq_of x n * p)"
       by (simp add: assms(2) incseq_SucI mult_right_mono)
     moreover have "(λn. real_of_rat (incseq_of x n * p)) ⟶ x *
real_of_rat p"
       by (auto intro!: tendsto_mult simp add: of_rat_mult)
     ultimately show ?thesis
       using assms(1) lim_ap powa_indep_incseq_of' by blast
   qed
   then show ?thesis using powa_def
     by (metis Lim_def limI the_lim)
 qed

lemma powa_powrat_lemma2:
  assumes "a ≥ 1" and "p < 0"
  shows "(a powa x) pow_ℚ p = (a powa (x * of_rat p))"
proof -
  have "(a powa x) pow_ℚ - p = a powa (x * real_of_rat (- p))"
    by (simp add: assms(1) assms(2) less_imp_le powa_powrat_lemma1)
  then have  "inverse ((a powa x) pow_ℚ p) = inverse (a powa (x * real_of_rat
p))"
    by (simp add: assms(1) of_rat_minus powa_minus powrat_minus)
  then show ?thesis by simp
```

qed

lemma *powa_powrat_lemma:*
*"a ≥ 1 ⟹ (a powa x) pow$_{\mathbb{Q}}$ p = (a powa (x * of_rat p))"*
by *(metis linorder_not_less powa_powrat_lemma1 powa_powrat_lemma2)*

lemma *LIMSEQ_iff2:*
fixes *L :: "'a::metric_space"*
shows *"(X ⟶ L) = (∀m::nat>0. ∃no. ∀n≥no. dist (X n) L < inverse m)"*
proof
assume *"X ⟶ L"*
then show *"∀m>0. ∃no. ∀n≥no. dist (X n) L < inverse (real m)"*
by *(auto simp add: LIMSEQ_def)*
next
assume *assm: "∀m>0. ∃no. ∀n≥no. dist (X n) L < inverse (real m)"*
{fix *r*
assume *"(r::real) > 0"*
then have *" ∃no. ∀n≥no. dist (X n) L < r"*
using *assm*
by *(metis ex_inverse_of_nat_less less_trans)*
}
then show *"X ⟶ L"*
by *(simp add: metric_LIMSEQ_I)*
qed

lemma *LIM_def2:*
"f −a→ L = (∀m::nat>0. ∃s>0. ∀x. x ≠ a ∧ dist x a < s ⟶ dist (f x) L < inverse m)"
for *a :: "'a::metric_space"* **and** *L :: "'b::metric_space"*
proof
assume *"f −a→ L"*
then show *"∀m::nat>0. ∃s>0. ∀x. x ≠ a ∧ dist x a < s ⟶ dist (f x) L < inverse m"*
by *(auto simp add: LIM_def)*
next
assume *assm: "∀m::nat>0. ∃s>0. ∀x. x ≠ a ∧ dist x a < s ⟶ dist (f x) L < inverse m"*
{fix *r*
assume *r0: "(r::real) > 0"*
then obtain *n* **where** *"inverse (real (Suc n)) < r"*
using *reals_Archimedean* **by** *blast*
then have *"∃s>0. ∀x. x ≠ a ∧ dist x a < s ⟶ dist (f x) L < r"*

using *assm r0* **by** *(metis order_less_trans zero_less_Suc)*
}
then show *"f −a→ L"*
by *(simp add: metric_LIM_I)*

```
qed

lemma powa_ge_one:
  assumes "a ≥ 1"
  and "x ≥ 0"
  shows "a powa x ≥ 1"
proof -
  obtain y where lima: "(λn. a pow_ℚ incseq_of x n) ⟶ y"
    using LIMSEQ_powrat_incseq_of_ex1 assms(1) by blast
  moreover have "incseq (λn. a pow_ℚ incseq_of x n)"
    using assms(1) incseq_powrat_insec_of by blast
  moreover have "∀n. a pow_ℚ incseq_of x n ≤ y"
    using incseq_le_all using calculation by blast
  ultimately have "1 ≤ y"
    using assms LIMSEQ_incseq_of LIMSEQ_powa LIMSEQ_powrat_incseq_of_ex1

          lemma_seq_point_gt_ex2 powa_zero_eq_one powrat_ge_one
    by (metis less_le  of_rat_0  less_eq_real_def xt1(6))
  then show ?thesis
    using LIMSEQ_powa LIMSEQ_unique assms(1) lima by blast
qed

lemma powreal_ge_one: "a ≥ 1 ⟹ x ≥ 0 ⟹ a pow_ℝ x ≥ 1"
by (simp add: powreal_def powa_ge_one)

lemma powreal_ge_one2:
   "⟦ 0 < a; a < 1; x ≤ 0 ⟧ ⟹ a pow_ℝ x ≥ 1"
  by (simp add: powa_ge_one powreal_def real_inverse_ge_one_lemma)

lemma inverse_of_real_nat_of_rat_of_nat:
      "inverse (real_of_nat n) =  of_rat(inverse (of_nat n))"
by (metis Ratreal_def of_rat_of_nat_eq real_inverse_code)

lemma LIMSEQ_powa_inverse_of_nat:
   "a ≥ 1 ⟹ (λn. a powa inverse (real_of_nat n)) ⟶ 1"
  by (simp add:  inverse_of_real_nat_of_rat_of_nat powa_powrat_eq
     LIMSEQ_powrat_inverse_of_nat)

lemma incseq_of_le_mono:
  assumes "r ≤ s"
  shows "∃N. ∀n≥N. incseq_of r n ≤  incseq_of s n"
proof -
  have "r = s ∨ r < s" using assms by force
  then show ?thesis
  proof
    assume "r = s" then show ?thesis by simp
  next
    assume rs: "r < s"
    then obtain m where less_incseq: "r < real_of_rat (incseq_of s m)"
```

```
    using LIMSEQ_incseq_of lemma_seq_point_gt_ex by blast
  moreover have "∀n. real_of_rat (incseq_of r n) ≤ r"
    using incseq_of_le_self by simp
  ultimately have "∀n. real_of_rat (incseq_of r n) < real_of_rat (incseq_of
s m)"
    using le_less_trans by blast
  then have incrs: "∀n. incseq_of r n ≤ incseq_of s m"
    by (simp add: less_imp_le of_rat_less)
  then show ?thesis
    by (meson incseq_def incseq_incseq_of order_trans)
 qed
qed

lemma powa_le_mono:
 assumes "r ≤ s"
 and "a ≥ 1"
 shows "a powa r ≤ a powa s"
proof -
 obtain y where "(λn. a pow_ℚ incseq_of s n) ⟶ y"
   using LIMSEQ_powrat_incseq_of_ex1 assms(2) by blast
 moreover obtain x where "(λn. a pow_ℚ incseq_of r n) ⟶ x"
   using LIMSEQ_powrat_incseq_of_ex1 assms(2) by blast
 moreover have "∃N. ∀n≥N. a pow_ℚ incseq_of r n ≤ a pow_ℚ incseq_of
s n"
   by (meson assms(1) assms(2) incseq_of_le_mono powrat_le_mono)
 moreover have "x ≤ y"
   using LIMSEQ_le calculation by blast
 ultimately show ?thesis
   by (metis Lim_def limI powa_def)
qed

lemma powreal_le_mono:
  "⟦ r ≤ s; a ≥ 1 ⟧ ⟹ a pow_ℝ r ≤ a pow_ℝ s"
by (metis powa_le_mono powreal_eq_powa)

lemma powreal_le_anti_mono:
  "⟦ r ≤ s; 0 < a; a < 1 ⟧ ⟹ a pow_ℝ r ≥ a pow_ℝ s"
 by (simp add: powa_le_mono powreal_def real_inverse_ge_one_lemma)

lemma powreal_less_cancel:
  "⟦ a pow_ℝ r < a pow_ℝ s; a ≥ 1 ⟧ ⟹ r < s"
by (metis less_le_not_le linorder_not_less powreal_eq_powa powreal_le_mono)

lemma powa_less_mono:
 assumes "r < s" and "a > 1"
 shows "a powa r < a powa s"
proof -
 obtain q where "r < real_of_rat q" "real_of_rat q < s"
   using assms(1) of_rat_dense by blast
```

```
  moreover obtain qa where "real_of_rat q < real_of_rat qa" "real_of_rat
qa < s"
    using of_rat_dense calculation(2) by blast
  ultimately show ?thesis using assms
    by (metis (full_types) antisym less_eq_real_def less_imp_not_eq2 powa_le_mono

        powa_powrat_eq powrat_inject_exp)
  qed

lemma powreal_less_anti_mono:
  assumes "r < s"
  and "0 < a"
  and "a < 1"
shows "a pow_ℝ r > a pow_ℝ s"
proof -
  have "inverse a > 1"
    by (simp add: assms(2, 3) one_less_inverse_iff)
  moreover have "inverse a powa r < inverse a powa s"
    using assms(1) powa_less_mono
    using calculation by blast
  ultimately show ?thesis
    using powreal_eq_powa powreal_inverse powreal_le_anti_mono powreal_less_cancel
    by (metis assms(2,3) le_less less_irrefl)
qed

lemma powreal_less_mono:
   "⟦ r < s; a > 1 ⟧ ⟹ a pow_ℝ r < a pow_ℝ s"
by (simp add: powreal_def powa_less_mono)

lemma powa_le_cancel:
   "⟦ a powa r ≤ a powa s; a > 1 ⟧ ⟹ r ≤ s"
by (metis not_le powa_less_mono)

lemma powreal_le_cancel:
   "⟦ a pow_ℝ r ≤ a pow_ℝ s; a > 1 ⟧ ⟹ r ≤ s"
by (metis not_le powreal_less_mono)

lemma powreal_less_cancel_iff [simp]:
  "1 < a ⟹ (a pow_ℝ r < a pow_ℝ s) = (r < s)"
by (metis less_imp_le powreal_less_cancel powreal_less_mono)

lemma powreal_le_cancel_iff [simp]:
  "1 < a ⟹ (a pow_ℝ r ≤ a pow_ℝ s) = (r ≤ s)"
by (simp add: linorder_not_less [symmetric])

lemma powreal_inject_exp1 [simp]:
  "1 < a  ⟹ (a pow_ℝ r = a pow_ℝ s) = (s = r)"
by (metis antisym_conv3 less_irrefl powreal_less_mono)
```

```
lemma powreal_eq_one_iff [simp]:
  "a pow_ℝ x = 1 ⟷ x = 0" if "a > 1"
  using powreal_inject_exp1 that by fastforce

lemma powreal_inject_base_less_one [simp]:
  "0 < a ⟹ a < 1 ⟹ (a pow_ℝ r = a pow_ℝ s) = (s = r)"
by (metis linorder_neq_iff order_less_imp_not_eq2 powreal_less_anti_mono)

lemma powreal_inject [simp]:
  "0 < a ⟹ a ≠ 1 ⟹ (a pow_ℝ r = a pow_ℝ s) = (s = r)"
  using powreal_inject_base_less_one by fastforce

lemma powreal_gt_one: "a > 1 ⟹ x > 0 ⟹ a pow_ℝ x > 1"
by (metis less_eq_real_def powa_less_mono powa_zero_eq_one powreal_eq_powa)

lemma isCont_powa_exponent_at_zero:
  assumes "a > 1" shows "isCont (λx. a powa x) 0"
proof -
  {fix m
  assume m0: "(m::nat) > 0"
  have lim1: "(λn. a powa inverse (real n)) ⟶ 1"
    by (simp add: LIMSEQ_powa_inverse_of_nat assms less_imp_le)
  then have "∃no. ∀n≥no. |a powa inverse (real n) - 1| < inverse (real
m)"
    using lim1 LIMSEQ_iff2  dist_real_def m0 by metis
  then obtain "no"
    where pow1: "∀n≥no. |a powa inverse (real n) - 1| < inverse (real
m)"
    by blast

  have lim2: "(λx. inverse (a powa inverse (real x))) ⟶ 1"
    using tendsto_inverse lim1 by fastforce
  then have "∃k. ∀n≥k. |inverse (a powa inverse (real n)) - 1| < inverse
(real m)"
    using LIMSEQ_iff2 dist_real_def m0 by metis
  then obtain "k"
    where pow2: "∀n≥k. |inverse (a powa inverse (real n)) - 1| < inverse
(real m)"
    by blast
  have "∃s>0. ∀x. x ≠ 0 ∧ |x| < s ⟶  |(a powa x) - 1| < inverse (real
m)"
  proof -
    let ?d = "min (inverse (Suc no)) (inverse (Suc k))"
    {fix x
    assume "abs x < ?d"
    then have x_bounds: "- inverse (of_nat(Suc k)) < x" "x < inverse
(of_nat(Suc no))"
      by linarith+
    then have "a powa - inverse (of_nat (Suc k)) < a powa x"
```

```
      using assms powa_less_mono by blast
    moreover have "- inverse (real m) < a powa - inverse (real (Suc k))
- 1"
      using assms pow2 powa_minus
      by (metis minus_diff_eq neg_less_iff_less abs_less_iff lessI less_imp_le)

    ultimately have lo: "- inverse(real m) < a powa x - 1"
      by linarith
    have "a powa x < a powa inverse (of_nat(Suc no))"
      using assms powa_less_mono x_bounds(2) by blast
    moreover have "a powa inverse (real (Suc no)) - 1 < inverse (real
m)"
      using assms pow1 by (metis less_imp_le abs_less_iff lessI)
    ultimately have "a powa x - 1 < inverse(real m)"
      by linarith
    then have "abs (a powa x - 1) < inverse(real m)"
      using lo by linarith}
    then show ?thesis
      by (metis inverse_positive_iff_positive min_less_iff_conj of_nat_0_less_iff
zero_less_Suc)
  qed}
  then have "∀m>0. ∃s>0. ∀x. x ≠ 0 ∧ |x| < s ⟶ |a powa x - 1| < inverse
(real m)"
    by blast
  then show ?thesis
    by (auto simp add: assms isCont_def LIM_def2 dist_real_def less_imp_le)
qed

lemma LIM_powa_exponent_at_zero: "1 < a ⟹ (λh. a powa h) −0→ 1"
by (metis isCont_def isCont_powa_exponent_at_zero less_eq_real_def powa_zero_eq_one)

lemma isCont_powa_exponent_gt_one:
  assumes "a > 1"
  shows "isCont (λx. a powa x) x"
proof -
  have "(λh. a powa x * a powa h) −0→ a powa x"
    using LIM_powa_exponent_at_zero assms tendsto_mult_left by fastforce
  then have "(λh. a powa (x + h)) −0→ a powa x"
    using assms powa_add by auto
  then have "(powa) a −x→ a powa x"
    using LIM_offset_zero_cancel by blast
  then show ?thesis
    using isCont_def by blast
qed

lemma isCont_powreal_exponent_gt_one:
   "a > 1 ⟹ isCont (λx. a pow_ℝ x) x"
by (metis ext isCont_powa_exponent_gt_one less_eq_real_def powreal_eq_powa)
```

```
lemma isCont_powreal_exponent_less_one:
  assumes "0 < a"
  and "a < 1"
  shows "isCont (λx. a pow_ℝ x) x"
proof -
  have "1 < inverse a"
    by (simp add: assms one_less_inverse)
  then have "isCont ((pow_ℝ) (inverse a)) x"
    by (simp add: isCont_powreal_exponent_gt_one)
  then have "isCont (λx. inverse (inverse a pow_ℝ x)) x"
    using assms(1) continuous_at_within_inverse powreal_not_zero by fastforce
  then show ?thesis
    using assms(1) powreal_inverse by auto
qed

lemma isCont_powreal_exponent:
  assumes a_gt_0: "0 < a" shows "isCont (λx. a pow_ℝ x) x"
proof cases
  assume "a > 1" then show ?thesis
    using isCont_powreal_exponent_gt_one by blast
  next
  assume "¬ a > 1"
    then have "a < 1 ∨ a = 1" by auto
    then show ?thesis
    proof
      assume "a < 1"  then show ?thesis
        using a_gt_0 isCont_powreal_exponent_less_one by blast
    next
     assume "a =1" then show ?thesis by simp
    qed
qed


lemma real_of_rat_abs:
    "real_of_rat(abs a) = abs(of_rat a)"
by (simp add: abs_if of_rat_minus)

lemma isCont_powrat_exponent:
  assumes "0 < a" shows "isCont (λx. a pow_ℚ x) x"
proof -
  have isCont_of_rat: "isCont real_of_rat x"  using isCont_def LIM_def
dist_real_def
    by (metis dist_commute of_rat_diff rat_dist_def real_of_rat_abs)
  have "isCont ((pow_ℝ) a) (real_of_rat x)" using assms
    by (simp add: isCont_powreal_exponent)
  then have "isCont (λx. a pow_ℝ real_of_rat x) x" using isCont_of_rat
    using isCont_o2 by blast
  then show ?thesis using realpow_powrat_eq assms by simp
qed
```

```
lemma LIMSEQ_powrat_exponent:
  "⟦ X ⟶ x; a > 0 ⟧ ⟹ (λn. a pow_ℚ (X n)) ⟶ a pow_ℚ x"
by (metis isCont_tendsto_compose isCont_powrat_exponent)


lemma powa_mult:
  assumes "1 ≤ a" and "0 ≤ x"
  shows "(a powa x) powa y = a powa (x * y)"
proof (cases)
  assume "a ≠ 1"
  then have a_gt_1: "a > 1" using assms(1) by simp
  have "a powa x ≥ 1" using assms powa_ge_one by blast
  then have lim1: "(λn. a powa (x * real_of_rat (incseq_of y n))) ⟶
(a powa x) powa y"
    using LIMSEQ_powa [of "a powa x" y] powa_powrat_lemma assms(1) by
simp
  have "isCont ((powa) a) (x * y)" using isCont_powa_exponent_gt_one a_gt_1
by blast
  moreover have "(λn. x * real_of_rat (incseq_of y n)) ⟶ x * y"
    by (simp add: tendsto_mult_left)
  ultimately have "(λn. a powa (x * real_of_rat (incseq_of y n))) ⟶
a powa (x * y)"
    using isCont_tendsto_compose by blast
  then show ?thesis
    using LIMSEQ_unique lim1 by blast
next
  assume "¬ a ≠ 1"
  then show ?thesis
    by auto
qed

lemma powreal_mult1:
      "⟦ 1 ≤ a; 0 ≤ x ⟧ ⟹ (a pow_ℝ x) pow_ℝ y = a pow_ℝ (x * y)"
by (metis powa_mult powreal_eq_powa powreal_ge_one)

lemma powreal_mult2:
  assumes "0 < a" and "a < 1" and "0 ≤ x"
  shows "(a pow_ℝ x) pow_ℝ y = a pow_ℝ (x * y)"
proof -
  have "1 ≤ inverse a" using assms using real_inverse_ge_one_lemma by
simp
  then have "(inverse a pow_ℝ x) pow_ℝ y = inverse a pow_ℝ (x * y)"
    using powreal_mult1 assms(3) by blast
  then have "inverse((inverse a pow_ℝ x) pow_ℝ y) =  a pow_ℝ (x * y)"
    by (simp add: assms(1) powreal_inverse)
  then show ?thesis
    using assms(1) powreal_gt_zero powreal_inverse by auto
```

qed

lemma *powreal_mult3:*
"⟦ 0 < a; 0 ≤ x ⟧ ⟹ (a pow$_{\mathbb{R}}$ x) pow$_{\mathbb{R}}$ y = a pow$_{\mathbb{R}}$ (x * y)"
by *(metis linorder_not_less powreal_mult1 powreal_mult2)*

lemma *powreal_mult4:*
assumes *a0: "0 < a"* **and** *x0: "x ≤ 0"*
shows *"(a pow$_{\mathbb{R}}$ x) pow$_{\mathbb{R}}$ y = a pow$_{\mathbb{R}}$ (x * y)"*
proof -
have *minux0: "-x ≥ 0"* **using** *x0* **by** *simp*
then have *"(a pow$_{\mathbb{R}}$ -x) pow$_{\mathbb{R}}$ y = a pow$_{\mathbb{R}}$ (-x * y)"*
using *powreal_mult3 a0* **by** *simp*
then have *"inverse (a pow$_{\mathbb{R}}$ x) pow$_{\mathbb{R}}$ y = inverse (a pow$_{\mathbb{R}}$ (x * y))"*
by *(simp add: powreal_minus)*
then have *"inverse ((a pow$_{\mathbb{R}}$ x) pow$_{\mathbb{R}}$ y) = inverse (a pow$_{\mathbb{R}}$ (x * y))"*
by *(simp add: a0 powreal_gt_zero powreal_inverse)*
then show *?thesis*
using *inverse_eq_iff_eq* **by** *blast*
qed

lemma *powreal_mult:*
*"(a pow$_{\mathbb{R}}$ x) pow$_{\mathbb{R}}$ y = a pow$_{\mathbb{R}}$ (x * y)"*
by *(smt (verit, best) powreal_def powreal_mult3 powreal_mult4)*

lemma *powreal_divide:*
"⟦ 0 ≤ a; 0 ≤ b ⟧ ⟹ (a/b) pow$_{\mathbb{R}}$ x = (a pow$_{\mathbb{R}}$ x) / (b pow$_{\mathbb{R}}$ x)"
by *(simp add: divide_inverse powreal_inverse powreal_mult_base)*

lemma *powreal_divide2:*
"0 ≤ a ⟹ a pow$_{\mathbb{R}}$ (x - y) = (a pow$_{\mathbb{R}}$ x) / (a pow$_{\mathbb{R}}$ y)"
proof -
assume *a1: "0 ≤ a"*
have *f2: "∀x0. - (x0::real) = - 1 * x0"*
by *auto*
have *"x - y = x + - 1 * y"*
by *auto*
then show *?thesis*
using *f2 a1* **by** *(metis field_class.field_divide_inverse powreal_add powreal_minus)*
qed

lemma *powreal_less_mono_base:*
assumes *r0: "r > 0"* **and** *a0: "0 < a"* **and** *ab: "a < b"*
shows *"a pow$_{\mathbb{R}}$ r < b pow$_{\mathbb{R}}$ r"*
proof -
have *"b/a > 1"* **using** *ab a0* **by** *simp*
then have *"(b/a) pow$_{\mathbb{R}}$ r > 1"*

```
    using powreal_gt_one r0 by simp
  then have "b pow_ℝ r / a pow_ℝ r > 1"
    using ab a0 powreal_divide by simp
  also have "a pow_ℝ r > 0"
    by (simp add: a0 powreal_gt_zero)
  ultimately show ?thesis
    using less_divide_eq_1_pos by blast
qed

lemma powreal_less_antimono_base:
  assumes "r < 0" and "0 < a" and "a < b"
  shows "a pow_ℝ r > b pow_ℝ r"
proof -
  have "0 < -r" using assms(1) by simp
  then have "a pow_ℝ - r < b pow_ℝ - r"
    using assms(2) assms(3) powreal_less_mono_base by blast
  then show ?thesis
    using assms(2) assms(3) powreal_gt_zero powreal_minus by auto
qed

lemma powa_power_eq:
  assumes "a ≥ 1" shows "a powa (of_nat n) = a ^ n"
proof -
  have "incseq (λm. rat_of_nat n)" by simp
  moreover have "(λna. real_of_rat (rat_of_nat n)) ⟶ real n" by
simp
  moreover have "(λna. a pow_ℚ rat_of_nat n) ⟶ a ^ n"
    using powrat_power_eq assms by auto
  ultimately have "(λna. a pow_ℚ incseq_of (real n) na) ⟶ a ^ n"
    using powa_indep_incseq_of' assms by blast
  then show ?thesis
    by (metis Lim_def limI powa_def)
qed

lemma powreal_power_eq:
  "a > 0 ⟹ a pow_ℝ (of_nat n) = a ^ n"
unfolding powreal_def
by (simp add: powa_minus powa_power_eq power_inverse real_inverse_ge_one_lemma)

lemma powreal_power_eq2:
   "0 ≤ a ⟹ 0 < n ⟹ a ^ n = (if a = 0 then 0 else a pow_ℝ (real n))"
  by (auto simp add:  powreal_power_eq)

lemma powreal_mult_power: "a > 0 ⟹ a pow_ℝ (n * x) = (a pow_ℝ x) ^ n"
  by (metis mult.commute powreal_gt_zero powreal_mult powreal_power_eq)

lemma powreal_int:
  assumes "x > 0"
  shows "x pow_ℝ i = (if i ≥ 0 then x ^ nat i else 1 / x ^ nat (-i))"
```

proof *cases*
  assume *"i < 0"*
  have *r: "x pow$_{\mathbb{R}}$ i = 1 / x pow$_{\mathbb{R}}$ (-i)"* **by** *(simp add: assms powreal_minus divide_inverse)*
  show *?thesis* **using** ‹*i < 0*› ‹*x > 0*› **by** *(simp add: r field_simps powreal_power_eq [symmetric])*
qed *(simp add: assms powreal_power_eq[symmetric])*

lemma *powreal_numeral: "0 ≤ x ⟹ x pow$_{\mathbb{R}}$ numeral n = x^numeral n"*
  using *powreal_power_eq [of x "numeral n"] powreal_def*
  by *fastforce*

lemma *root_powreal_inverse:*
  assumes *"0 < n"* **and** *"0 ≤ x"*
  shows *"root n x = x pow$_{\mathbb{R}}$ (1/n)"*
proof -
  have *"root n x = x pow$_{\mathbb{R}}$ real_of_rat (inverse (rat_of_nat n))"*
    using *assms real_root_eq_powrat_inverse realpow_powrat_eq [symmetric] powreal_def*
    by *simp*
  then show *?thesis*
    by *(metis inverse_eq_divide of_rat_inverse of_rat_of_nat_eq)*
qed

lemma *powreal_inverse_of_nat_gt_one:*
  "⟦ 1 < a; n ≠ 0⟧ ⟹ a pow$_{\mathbb{R}}$ (inverse (of_nat n)) > 1"
by *(metis inverse_positive_iff_positive neq0_conv of_nat_0_less_iff powreal_gt_one)*

end

3 Real Logarithms (Redefined)

theory *Log*
imports *RealPower*
begin

We can now directly define real logarithm of *x* to base *a*.

definition
    Log :: "[real,real] ⇒ real" **where**
   "Log a x = (THE y. a pow$_{\mathbb{R}}$ y = x)"

lemma *IVT_simple:*
  "⟦f (a::real) ≤ (y::real); y ≤ f b; a ≤ b;
    ∀x. a ≤ x ∧ x ≤ b ⟶ isCont f x⟧
   ⟹ ∃x. f x = y"
by *(frule IVT [of f]) auto*

lemma *inj_on_powreal:*
   "0 < a ⟹ a ≠ 1 ⟹ inj_on (λx. a pow$_{\mathbb{R}}$ x) UNIV"

```
by (auto simp add: inj_on_def)

lemma LIMSEQ_powreal_minus_nat:
  "a > 1 ⟹ (λn. a powℝ (-real n)) ⟶ 0"
by (simp add: powreal_minus powreal_power_eq
        LIMSEQ_inverse_realpow_zero)

lemma LIMSEQ_less_Ex:
   "⟦ X ⟶ (x::real); x < y ⟧ ⟹ ∃n. X n < y"
  by (meson LIMSEQ_le_const not_less)

lemma powreal_IVT_upper_lemma:
  assumes "a > (1::real)" and "x > 0"
  shows "∃n::nat. a powℝ (-real n) < x"
proof -
  have "(λn. a powℝ - real n) ⟶ 0"
    by (simp add: LIMSEQ_powreal_minus_nat assms(1))
  then show ?thesis
    using LIMSEQ_less_Ex assms(2) by blast
qed

lemma powreal_IVT_lower_lemma:
  assumes "a > (1::real)"
  and "x > 0"
  shows "∃n::nat. x < a powℝ (real n)"
proof -
  have invx0: "0 < inverse x"
    by (simp add: assms(2))
  then have "∃n. a powℝ - real n < inverse x"
    using assms(1) powreal_IVT_upper_lemma by blast
  then show ?thesis
    using assms(1)
    by (auto dest: inverse_less_imp_less
         simp add: powreal_minus powreal_gt_zero )
qed

lemma powreal_surj:
  assumes "a > 1"
  and "x > 0"
  shows "∃y. a powℝ y = x"
proof -
  obtain n where "a powℝ - real n < x"
    using assms powreal_IVT_upper_lemma by blast
  moreover obtain na where "x < a powℝ real na"
    using assms powreal_IVT_lower_lemma by blast
  moreover have "∀x. - real n ≤ x ∧ x ≤ real na ⟶ isCont ((powℝ)
a) x"
    using assms(1) isCont_powreal_exponent_gt_one by blast
  ultimately show ?thesis
```

```
    using IVT_simple [of _ "-real n" _ "real na"] by force
qed

lemma powreal_surj2:
    "⟦ 0 < a; a < 1; x > 0 ⟧ ⟹ ∃y. a pow_ℝ y = x"
  using powreal_minus_base_ge_one powreal_surj real_inverse_gt_one_lemma

  by blast

lemma powreal_ex1_eq:
  assumes "a > 0"
  and "a ≠ 1"
  and "x > 0"
  shows "∃! y. a pow_ℝ y = x"
proof (cases "a < 1")
  case True
  then show ?thesis
    using assms powreal_inject powreal_surj2 by blast
next
  case False
  then show ?thesis
    using assms(2) assms(3) powreal_surj by auto
qed

lemma powreal_Log_cancel [simp]:
   "⟦ a > 0; a ≠ 1; x > 0 ⟧ ⟹ a pow_ℝ (Log a x) = x"
by (auto intro: the1I2 [OF powreal_ex1_eq] simp add: Log_def)

lemma Log_powreal_cancel [simp]:
  "⟦ 0 < a; a ≠ 1 ⟧ ⟹ Log a (a pow_ℝ y) = y"
by (metis powreal_ex1_eq powreal_gt_zero powreal_Log_cancel)

lemma Log_mult:
     "⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧
      ⟹ Log a (x * y) = Log a x + Log a y"
  by (metis Log_powreal_cancel powreal_Log_cancel powreal_add)

lemma Log_one [simp]: "⟦ 0 < a; a ≠ 1 ⟧ ⟹ Log a 1 = 0"
by (metis Log_powreal_cancel powreal_zero_eq_one)

lemma Log_eq_one [simp]: "⟦ 0 < a; a ≠ 1 ⟧ ⟹ Log a a = 1"
  using powreal_inject by fastforce

lemma Log_inverse:
  "⟦ a > 0; a ≠ 1; x > 0 ⟧ ⟹ Log a (inverse x) = - Log a x"
by (metis Log_powreal_cancel powreal_Log_cancel powreal_minus)

lemma Log_divide:
  "⟦ 0 < a; a ≠ 1; 0 < x; 0 < y ⟧
```

```
   ⟹ Log a (x/y) = Log a x - Log a y"
  by (metis Log_inverse Log_mult divide_real_def
       inverse_positive_iff_positive minus_real_def)

lemma Log_less_cancel_iff [simp]:
  assumes "1 < a"
  and "0 < x"
  and "0 < y"
shows "(Log a x < Log a y) = (x < y)"
proof
  assume "Log a x < Log a y"
  then show "x < y" using powreal_Log_cancel assms powreal_less_cancel_iff

    by (metis less_irrefl real_inverse_bet_one_one_lemma
          inverse_positive_iff_positive)
next
  assume "x < y"
  then show "Log a x < Log a y"
    using assms(1) assms(2) powreal_less_cancel_iff by fastforce
qed

lemma Log_inj: assumes "1 < b" shows "inj_on (Log b) {0 <..}"
proof (rule inj_onI, simp)
  fix x y assume pos: "0 < x" "0 < y" and *: "Log b x = Log b y"
  show "x = y"
  proof (cases rule: linorder_cases)
    assume "x < y" hence "Log b x < Log b y"
      using Log_less_cancel_iff[OF ‹1 < b›] pos by simp
    thus ?thesis using * by simp
  next
    assume "y < x" hence "Log b y < Log b x"
      using Log_less_cancel_iff[OF ‹1 < b›] pos by simp
    thus ?thesis using * by simp
  qed simp
qed

lemma Log_le_cancel_iff [simp]:
     "⟦ 1 < a; 0 < x; 0 < y ⟧ ⟹ (Log a x ≤ Log a y) = (x ≤ y)"
by (simp add: linorder_not_less [symmetric])

lemma zero_less_Log_cancel_iff [simp]:
  "1 < a ⟹ 0 < x ⟹ 0 < Log a x ⟷ 1 < x"
  using Log_less_cancel_iff[of a 1 x] by simp

lemma zero_le_Log_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ 0 ≤ Log a x ⟷ 1 ≤ x"
  using Log_le_cancel_iff[of a 1 x] by simp

lemma Log_less_zero_cancel_iff[simp]:
```

```
  "1 < a ⟹ 0 < x ⟹ Log a x < 0 ⟷ x < 1"
  using Log_less_cancel_iff[of a x 1] by simp

lemma Log_le_zero_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ Log a x ≤ 0 ⟷ x ≤ 1"
  using Log_le_cancel_iff[of a x 1] by simp

lemma one_less_Log_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ 1 < Log a x ⟷ a < x"
  using Log_less_cancel_iff[of a a x] by simp

lemma one_le_Log_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ 1 ≤ Log a x ⟷ a ≤ x"
  using Log_le_cancel_iff[of a a x] by simp

lemma Log_less_one_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ Log a x < 1 ⟷ x < a"
  using Log_less_cancel_iff[of a x a] by simp

lemma Log_le_one_cancel_iff[simp]:
  "1 < a ⟹ 0 < x ⟹ Log a x ≤ 1 ⟷ x ≤ a"
  using Log_le_cancel_iff[of a x a] by simp

lemma Log_powreal:
  assumes "0 < x"
  and "1 < b"
  and "b ≠ 1"
shows "Log b (x pow_ℝ y) = y * Log b x"
proof -
  have "b pow_ℝ (Log b x * y) = x pow_ℝ y"
    using assms powreal_mult [symmetric] by simp
  moreover have "0 < x pow_ℝ y"
    by (simp add: assms(1) powreal_gt_zero)
  ultimately have "b pow_ℝ (y * Log b x) = b pow_ℝ Log b (x pow_ℝ y)"
    using powreal_Log_cancel assms powreal_Log_cancel
    by (simp add: mult.commute)
  then show ?thesis
    using assms(2) powreal_inject_exp1 by blast
qed

lemma Log_nat_power:
  assumes "0 < x"
  and "1 < b" and "b ≠ 1"
  shows " Log b (x ^ n) = real n * Log b x"
proof -
  have "Log b (x pow_ℝ real n) = real n * Log b x"
    by (simp add: Log_powreal assms)
  then show ?thesis
    by (simp add: assms(1) powreal_power_eq)
```

qed

end