1 Fib and Gcd commute

```
theory Fibonacci
  imports "HOL-Computational_Algebra.Primes"
begin
```

A few proofs of laws taken from Graham et al. [?].

1.1 Fibonacci numbers

```
fun fib :: "nat ⇒ nat" where
    "fib 0 = 0"
  | "fib (Suc 0) = 1"
  | "fib (Suc (Suc n)) = fib n + fib (Suc n)"

lemma fib_positive: "fib (Suc n) > 0"
  by (induction n rule: fib.induct) auto
```

1.2 Fib and gcd commute

```
lemma fib_add: "fib (Suc (n + k)) = fib (Suc k) * fib (Suc n) + fib k
* fib n"
  by (induction n rule: fib.induct) (auto simp: distrib_left)

lemma coprime_fib_Suc: "coprime (fib n) (fib (Suc n))"
proof (induction n rule: fib.induct)
  case (3 x)
  then show ?case
    by (metis coprime_iff_gcd_eq_1 fib.simps(3) gcd.commute gcd_add1)
qed auto

lemma gcd_fib_add: "gcd (fib m) (fib (n + m)) = gcd (fib m) (fib n)"
proof (cases m)
  case 0
  then show ?thesis by simp
next
  case (Suc k)
  then have "gcd (fib m) (fib (n + m))
          = gcd (fib k * fib n) (fib (Suc k))"
    by (metis add_Suc_right fib_add gcd.commute gcd_add_mult mult.commute)
  also have "... = gcd (fib n) (fib (Suc k))"
    using coprime_commute coprime_fib_Suc gcd_mult_left_left_cancel by
blast
  also have "... = gcd (fib m) (fib n)"
    using Suc by (simp add: ac_simps)
  finally show ?thesis .
qed

lemma gcd_fib_diff: "gcd (fib m) (fib (n - m)) = gcd (fib m) (fib n)"
if "m ≤ n"
```

```
proof -
  have "gcd (fib m) (fib (n - m)) = gcd (fib m) (fib (n - m + m))"
    by (simp add: gcd_fib_add)
  also from ‹m ≤ n› have "n - m + m = n"
    by simp
  finally show ?thesis .
qed

lemma gcd_fib_mod: "gcd (fib m) (fib (n mod m)) = gcd (fib m) (fib n)"
if "0 < m"
proof (induction n rule: less_induct)
  case (less n)
  show ?case
  proof -
    have "n mod m = (if n < m then n else (n - m) mod m)"
      by (rule mod_if)
    also have "gcd (fib m) (fib ...) = gcd (fib m) (fib n)"
      using gcd_fib_diff less.IH that by fastforce
    finally show ?thesis .
  qed
qed

theorem fib_gcd: "fib (gcd m n) = gcd (fib m) (fib n)"
proof (induction m n rule: gcd_nat_induct)
  case (step m n)
  then show ?case
    by (metis gcd.commute gcd_fib_mod gcd_red_nat)
qed auto

end
theory CauchySchwarz imports "HOL-Analysis.Analysis"
begin
```

Remark: the predicate `concave_on` is now to be found in the standard analysis library, as is indeed much of the material below.

```
lemma ln_concave: "concave_on {0<..} ln"
  unfolding concave_on_def
  by (rule f''_ge0_imp_convex derivative_eq_intros | simp)+

lemma powr_convex:
  assumes "p ≥ 1" shows "convex_on {0<..} (λx. x powr p)"
  using assms
  by (intro f''_ge0_imp_convex derivative_eq_intros | simp)+

lemma Youngs_inequality_0:
  fixes a::real
  assumes "0 ≤ α" "0 ≤ β" "α+β = 1" "a>0" "b>0"
  shows "a powr α * b powr β ≤ α*a + β*b"
proof -
```

```
  have "α * ln a + β * ln b ≤ ln (α * a + β * b)"
    using assms ln_concave by (simp add: concave_on_iff)
  moreover have "0 < α * a + β * b"
    using assms by (smt (verit) mult_pos_pos split_mult_pos_le)
  ultimately show ?thesis
    using assms by (simp add: powr_def mult_exp_exp flip: ln_ge_iff)
qed

lemma Youngs_inequality:
  fixes p::real
  assumes "p>1" "q>1" "1/p + 1/q = 1" "a≥0" "b≥0"
  shows "a * b ≤ a powr p / p + b powr q / q"
proof (cases "a=0 ∨ b=0")
  case False
  then show ?thesis
  using Youngs_inequality_0 [of "1/p" "1/q" "a powr p" "b powr q"] assms
  by (simp add: powr_powr)
qed (use assms in auto)

lemma Cauchy_Schwarz_ineq_sum:
  fixes a :: "'a ⇒ 'b::linordered_field"
  shows "(∑i∈I. a i * b i)^2 ≤ (∑i∈I. (a i)^2) * (∑i∈I. (b i)^2)"
proof (cases "(∑i∈I. (b i)^2) > 0")
  case False
  then consider "⋀i. i∈I ⟹ b i = 0" | "infinite I"
    by (metis (mono_tags, lifting) sum_pos2 zero_le_power2 zero_less_power2)
  thus ?thesis
    by fastforce
next
  case True
  define r where "r ≡ (∑i∈I. a i * b i) / (∑i∈I. (b i)^2)"
  have "0 ≤ (∑i∈I. (a i - r * b i)^2)"
    by (simp add: sum_nonneg)
  also have "... = (∑i∈I. (a i)^2) - 2 * r * (∑i∈I. a i * b i) + r^2
* (∑i∈I. (b i)^2)"
    by (simp add: algebra_simps power2_eq_square sum_distrib_left flip:
sum.distrib)
  also have "... = (∑i∈I. (a i)^2) - ((∑i∈I. a i * b i))^2 / (∑i∈I.
(b i)^2)"
    by (simp add: r_def power2_eq_square)
  finally have "0 ≤ (∑i∈I. (a i)^2) - ((∑i∈I. a i * b i))^2 / (∑i∈I.
(b i)^2)" .
  hence "((∑i∈I. a i * b i))^2 / (∑i∈I. (b i)^2) ≤ (∑i∈I. (a i)^2)"
    by (simp add: le_diff_eq)
  thus "((∑i∈I. a i * b i))^2 ≤ (∑i∈I. (a i)^2) * (∑i∈I. (b i)^2)"
    by (simp add: pos_divide_le_eq True)
qed

lemma "∃f'. ((λx. x^3 + x^2) has_real_derivative f' x) (at x) ∧ P (λx.
```

```
f' x)"
  apply (rule exI conjI derivative_eq_intros | simp)+
  oops

lemma "x > 0 ⟹ ∃f'. ((λx. (x² - 1) * ln x) has_real_derivative f'
x) (at x) ∧ P (λx. f' x)"
  apply (rule exI conjI derivative_eq_intros | simp)+
  oops

lemma "∃f'. ((λx. (sin x)² + (cos x)²) has_real_derivative f' x) (at
x) ∧ P (λx. f' x)"
  apply (rule exI conjI derivative_eq_intros | simp)+
  oops

end
theory Calculus
  imports Complex_Main
begin

theorem mvt:
  fixes φ :: "real ⇒ real"
  assumes "a < b"
    and contf: "continuous_on {a..b} φ"
    and derf: "⋀x. ⟦a < x; x < b⟧ ⟹ (φ has_derivative φ' x) (at x)"
  obtains ξ where "a < ξ" "ξ < b" "φ b - φ a = (φ' ξ) (b-a)"
proof -
  define f where "f ≡ λx. φ x - (φ b - φ a) / (b-a) * x"
  have "∃ξ. a < ξ ∧ ξ < b ∧ (λy. φ' ξ y - (φ b - φ a) / (b-a) * y) =
(λv. 0)"
  proof (intro Rolle_deriv[OF ‹a < b›])
    fix x
    assume x: "a < x" "x < b"
    show "(f has_derivative (λy. φ' x y - (φ b - φ a) / (b-a) * y)) (at
x)"
      unfolding f_def by (intro derivative_intros derf x)
  next
    show "f a = f b"
      using assms by (simp add: f_def field_simps)
  next
    show "continuous_on {a..b} f"
      unfolding f_def by (intro continuous_intros assms)
  qed
  then show ?thesis
    by (smt (verit, ccfv_SIG) pos_le_divide_eq pos_less_divide_eq that)
qed

end
```

2 Baby examples

```
theory Baby
  imports "HOL-Library.Sum_of_Squares"
          "HOL-Decision_Procs.Approximation"
          "HOL-Analysis.Analysis"

begin
```

a simplification rule for powers

```
thm power_Suc
```

Kevin Buzzard's examples

```
lemma
  fixes x::real
  shows "(x+y)*(x+2*y)*(x+3*y) = x^3 + 6*x^2*y + 11*x*y^2 + 6*y^3"
  by (simp add: algebra_simps eval_nat_numeral)

lemma "sqrt 2 + sqrt 3 < sqrt 10"
proof -
  have "(sqrt 2 + sqrt 3)^2 < (sqrt 10)^2"
  proof (simp add: algebra_simps eval_nat_numeral)
    have "(2 * (sqrt 2 * sqrt 3))^2 < 5 ^ 2"
      by (simp add: algebra_simps eval_nat_numeral)
    then show "2 * (sqrt 2 * sqrt 3) < 5"
      by (smt (verit, best) power_mono)
  qed
  then show ?thesis
    by (simp add: real_less_rsqrt)
qed

lemma "sqrt 2 + sqrt 3 < sqrt 10"
  by (approximation 10)

lemma "x ∈ {0.999..1.001} ⟹ |pi - 4 * arctan x| < 0.0021"
  by (approximation 20)

lemma "3.141592635389 < pi"
  by (approximation 30)

lemma
  fixes a::real
  shows "(a*b + b * c + c*a)^3 ≤ (a^2 + a * b + b^2) * (b^2 + b * c +
c^2) * (c^2 + c*a + a^2)"
  by sos

lemma "sqrt 2 ∉ ℚ"
proof
  assume "sqrt 2 ∈ ℚ"
  then obtain q::rat where "sqrt 2 = of_rat q"
```

```
    using Rats_cases by blast
  then have "q² = 2"
    by (metis abs_numeral of_rat_eq_iff of_rat_numeral_eq of_rat_power
real_sqrt_abs
        real_sqrt_power)
  then obtain m n where "coprime m n" "q = of_int m / of_int n"
    by (metis Fract_of_int_quotient Rat_cases)
  then have "(rat_of_int m)² / (rat_of_int n)² = 2"
    by (metis ‹q² = 2› power_divide)
  then have 2: "(rat_of_int m)² = 2 * (rat_of_int n)²"
    by (metis div_by_0 nonzero_mult_div_cancel_right times_divide_eq_left
zero_neq_numeral)
  then have "2 dvd m"
    by (metis (mono_tags, lifting) even_mult_iff even_numeral of_int_eq_iff
of_int_mult
              of_int_numeral power2_eq_square)
  then have "2² dvd m²"
    using dvd_power_same by blast
  then have "2 dvd n"
    by (metis "2" even_mult_iff of_int_eq_iff of_int_mult of_int_numeral
power2_eq_square
        zdvd_mono zero_neq_numeral)
  then show False
    using ‹coprime m n› ‹even m› by auto
qed
```

2.1 Material for a later post, about descriptions

```
lemma
  fixes 𝓑 :: "'a::metric_space set set" and L :: "nat list set"
  assumes "𝓢 ⊆ {ball x r | x r. r>0}" and "L ≠ {}"
  shows "P 𝓢 L"
proof -
  have "⋀B. B ∈ 𝓢 ⟹ ∃x. ∃r>0. B = ball x r"
    using assms by blast
  then obtain centre rad where rad: "⋀B. B ∈ 𝓢 ⟹ rad B > 0"
                         and centre: "⋀B. B ∈ 𝓢 ⟹ B = ball (centre
B) (rad B)"
    by metis
  define infrad where "infrad ≡ Inf (rad ` 𝓢)"
  have "infrad ≤ rad B" if "B ∈ 𝓢" for B
    by (smt (verit, best) bdd_below.I cINF_lower image_iff infrad_def
rad that)

  have "∃B ∈ 𝓢. rad B = infrad" if "finite 𝓢" "𝓢 ≠ {}"
    by (smt (verit) empty_is_image finite_imageI finite_less_Inf_iff imageE
infrad_def that)

  define minl where "minl = Inf (length ` L)"
```

```
  obtain l0 where  "l0 ∈ L" "length l0 = minl"
    by (metis Inf_nat_def1 empty_is_image imageE minl_def ‹L ≠ {}›)
  then have "length l0 ≤ length l" if "l ∈ L" for l
    by (simp add: cINF_lower minl_def that)

  show ?thesis
    sorry
qed




end
```

3 A Tail-Recursive, Stack-Based Ackermann's Function

Unlike the other Ackermann example, this termination proof uses the argument from Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. Communications of the ACM 22 (8) 1979, 465–476.

```
theory AckermannM imports "HOL-Library.Multiset_Order" "HOL-Library.Product_Lexorder"

begin
```

This theory investigates a stack-based implementation of Ackermann's function. Let's recall the traditional definition, as modified by Rózsa Péter and Raphael Robinson.

```
fun ack :: "[nat,nat] ⇒ nat" where
  "ack 0 n             = Suc n"
| "ack (Suc m) 0       = ack m 1"
| "ack (Suc m) (Suc n) = ack m (ack (Suc m) n)"
```

Setting up the termination proof for the stack-based version.

```
fun ack_mset :: "nat list ⇒ (nat×nat) multiset" where
  "ack_mset [] = {#}"
| "ack_mset [x] = {#}"
| "ack_mset (z#y#l) =  mset ((y,z) # map (λx. (Suc x, 0)) l)"

lemma case1: "ack_mset (Suc n # l) < add_mset (0, n) {#(Suc x, 0). x
∈# mset l#}"
proof (cases l)
  case (Cons m list)
  have "{#(m, Suc n)#} < {#(Suc m, 0)#}"
    by auto
  also have "... ≤ {#(Suc m, 0), (0,n)#}"
    by auto
  finally show ?thesis
```

```
    by (simp add: Cons)
qed auto
```

Here is the stack-based version, which uses lists.

```
function ackloop :: "nat list ⇒ nat" where
  "ackloop (n # 0 # l)         = ackloop (Suc n # l)"
| "ackloop (0 # Suc m # l)     = ackloop (1 # m # l)"
| "ackloop (Suc n # Suc m # l) = ackloop (n # Suc m # m # l)"
| "ackloop [m] = m"
| "ackloop [] =  0"
  by pat_completeness auto

termination
  by (relation "inv_image {(x,y). x<y} ack_mset") (auto simp: wf case1)
```

Unlike the other Ackermann theory, no extra function is needed to prove equivalence

```
lemma ackloop_ack: "ackloop (n # m # l) = ackloop (ack m n # l)"
  by (induction m n arbitrary: l rule: ack.induct) auto

theorem ack: "ack m n = ackloop [n,m]"
  by (simp add: ackloop_ack)

end
theory Sqrt2_Irrational imports
  Complex_Main

begin

proposition "sqrt 2 ∉ ℚ"
proof
  define R where "R ≡ {n. n > 0 ∧ real n * sqrt 2 ∈ ℕ}"
  define k where "k ≡ Inf R"
  assume "sqrt 2 ∈ ℚ"
  then obtain p q where "q≠0" "real p / real q = |sqrt 2|"
    by (metis Rats_abs_nat_div_natE)
  then have "R ≠ {}"
    by (simp add: R_def field_simps) (metis of_nat_in_Nats)
  then have "k ∈ R"
    by (simp add: Inf_nat_def1 k_def)
  then have kR: "real k * sqrt 2 ∈ ℕ" and "k > 0"
    by (auto simp add: R_def)
  define x where "x ≡ real k * (sqrt 2 - 1)"
  have "x ∈ ℕ"
    using ‹0 < k› by (simp add: kR right_diff_distrib' x_def)
  then obtain k' where k': "x = real k'"
    using Nats_cases by blast
  have "k' > 0"
    using ‹0 < k› k' of_nat_eq_0_iff x_def by fastforce
```

```
  have "real k' * sqrt 2 = 2 * k - k * sqrt 2"
    by (simp add: x_def algebra_simps flip: k')
  moreover have "real k' * sqrt 2 ≥ 0"
    by simp
  ultimately have "real k' * sqrt 2 ∈ ℕ"
    by (simp add: kR)
  with R_def ‹0 < k'› have "k' ∈ R"
    by blast
  have "x < real k"
    by (simp add: ‹0 < k› sqrt2_less_2 x_def)
  then have "k' < k"
    by (simp add: k')
  then show False
    using ‹k' ∈ R› k_def linorder_not_less wellorder_Inf_le1 by auto
qed

end
theory Cbrt23_Irrational
  imports Complex_Main

begin

lemma cuberoot_irrational:
  defines "x ≡ root 3 2 + root 3 3"
  shows "x ∉ ℚ"
proof
  assume "x ∈ ℚ"
  moreover
  have "root 3 8 = 2" "root 3 27 = 3"
    by auto
  then have xcubed: "x^3 = 5 + 3 * x * root 3 6"
    by (simp add: x_def algebra_simps eval_nat_numeral flip: real_root_mult)
  ultimately have Q: "5 + 3 * x * root 3 6 ∈ ℚ"
    by (metis Rats_power ‹x ∈ ℚ›)
  have "root 3 6 ∉ ℚ"
  proof
    assume "root 3 6 ∈ ℚ"
    then obtain a b where "a / b = root 3 6" and cop: "coprime a b" "b≠0"
      by (smt (verit, best) Rats_cases')
    then have "(a/b)^3 = 6"
      by auto
    then have eq: "a^3 = 2*3 * b^3"
      using of_int_eq_iff by (fastforce simp: divide_simps ‹b≠0›)
    then have p: "2 dvd a"
      by (metis coprime_left_2_iff_odd coprime_power_right_iff dvd_triv_left
mult.assoc)
    then obtain c where "a = 2*c"
      by blast
    with eq have "2 dvd b"
```

```
      by (simp add: eval_nat_numeral) (metis even_mult_iff even_numeral
odd_numeral)
    with p and cop show False
      by fastforce
  qed
  moreover have "3*x ∈ ℚ - {0}"
    using xcubed  by (force simp: ‹x ∈ ℚ›)
  ultimately have "3 * x * root 3 6 ∉ ℚ"
    using Rats_divide by force
  with Q show False
    by (metis Rats_diff Rats_number_of add.commute add_uminus_conv_diff
diff_add_cancel)
qed

end
theory Binary_Euclidean_Algorithm
  imports "HOL-Computational_Algebra.Primes"
begin
```

3.1 The binary GCD algorithm

```
inductive_set  bgcd :: "(nat × nat × nat) set" where
  bgcdZero: "(u, 0, u) ∈ bgcd"
| bgcdEven: "⟦ (u, v, g) ∈ bgcd ⟧ ⟹ (2*u, 2*v, 2*g) ∈ bgcd"
| bgcdOdd:  "⟦ (u, v, g) ∈ bgcd; ¬ 2 dvd v ⟧ ⟹ (2*u, v, g) ∈ bgcd"
| bgcdStep: "⟦ (u - v, v, g) ∈ bgcd; v ≤ u ⟧ ⟹ (u, v, g) ∈ bgcd"
| bgcdSwap: "⟦ (v, u, g) ∈ bgcd ⟧ ⟹ (u, v, g) ∈ bgcd"
```

3.2 Proving that the algorithm is correct

Show that the bgcd of x und y is really a divisor of both numbers

```
lemma bgcd_divides: "(x,y,g) ∈ bgcd ⟹ g dvd x ∧ g dvd y"
proof (induct rule: bgcd.induct)
  case (bgcdStep u v g)
  with dvd_diffD show ?case
    by blast
qed auto
```

The bgcd of x und y really is the greatest common divisor of both numbers, with respect to the divides relation.

```
lemma bgcd_greatest:
  "(x,y,g) ∈ bgcd ⟹ d dvd x ⟹ d dvd y ⟹ d dvd g"
proof (induct arbitrary: d rule: bgcd.induct)
  case (bgcdEven u v g d)
  show ?case
    proof (cases "2 dvd d")
      case True thus ?thesis using bgcdEven by (force simp add: dvd_def)

    next
```

```
      case False
      thus ?thesis using bgcdEven
        by (simp add: coprime_dvd_mult_right_iff)
    qed
next
  case (bgcdOdd u v g d)
  hence "coprime d 2"
    by fastforce
  thus ?case using bgcdOdd
    by (simp add: coprime_dvd_mult_right_iff)
qed auto
```

3.3 Proving uniqueness and existence

despite its apparent non-determinism, the relation bgcd is deterministic and therefore defines a function

```
lemma bgcd_unique:
  "(x,y,g) ∈ bgcd ⟹ (x,y,g') ∈ bgcd ⟹ g = g'"
  by (meson bgcd_divides bgcd_greatest gcd_nat.strict_iff_not)

lemma bgcd_defined_aux: "a+b ≤ n ⟹ ∃g. (a, b, g) ∈ bgcd"
proof (induction n arbitrary: a b rule: less_induct)
  case (less n a b)
  show ?case
  proof (cases b)
    case 0
    thus ?thesis by (metis bgcdZero)
  next
    case (Suc b')
    then have *: "a + b' < n"
      using Suc_le_eq add_Suc_right less.prems by presburger
    show ?thesis
    proof (cases "b ≤ a")
      case True
      thus ?thesis
        by (metis bgcd.simps le_add1 le_add_diff_inverse less.IH [OF *])
    next
      case False
      then show ?thesis
        by (metis less.IH [OF *] Suc Suc_leI bgcd.simps le_add_diff_inverse

            less_add_same_cancel2 nle_le zero_less_iff_neq_zero)
    qed
  qed
qed

theorem bgcd_defined: "∃!g. (a, b, g) ∈ bgcd"
  using bgcd_defined_aux bgcd_unique by auto
```

Alternative proof suggested by YawarRaza7349

```
lemma bgcd_defined_aux': "a+b = n ⟹ ∃g. (a, b, g) ∈ bgcd"
proof (induction n arbitrary: a b rule: less_induct)
  case (less n a b)
  then show ?case
  proof (cases "b ≤ a")
    case True
    with less obtain g where "(a-b, b, g) ∈ bgcd"
      by (metis add_cancel_right_right bgcd.simps le_add1 le_add_diff_inverse
nat_less_le)
    thus ?thesis
      using True bgcd.bgcdStep by blast
  next
    case False
    with less show ?thesis
      by (metis bgcd.simps le_add_diff_inverse less_add_same_cancel2 nle_le
zero_less_iff_neq_zero)
  qed
qed

end
theory Fun_Semantics
imports Main
begin

datatype exp = T | F | Zero | Succ exp | IF exp exp exp | EQ exp exp

inductive Eval :: "exp ⇒ exp ⇒ bool" (infix "⇛" 50) where
    IF_T:    "IF T x y ⇛ x"
  | IF_F:    "IF F x y ⇛ y"
  | IF_Eval: "p ⇛ q ⟹ IF p x y ⇛ IF q x y"
  | Succ_Eval: "x ⇛ y ⟹ Succ x ⇛ Succ y"
  | EQ_same: "EQ x x ⇛ T"
  | EQ_S0:   "EQ (Succ x) Zero ⇛ F"
  | EQ_0S:   "EQ Zero (Succ y) ⇛ F"
  | EQ_SS:   "EQ (Succ x) (Succ y) ⇛ EQ x y"
  | EQ_Eval1: "x ⇛ z ⟹ EQ x y ⇛ EQ z y"
  | EQ_Eval2: "y ⇛ z ⟹ EQ x y ⇛ EQ x z"

inductive_simps T_simp [simp]: "T ⇛ z"
inductive_simps F_simp [simp]: "F ⇛ z"
inductive_simps Zero_simp [simp]: "Zero ⇛ z"
inductive_simps Succ_simp [simp]: "Succ x ⇛ z"
inductive_simps IF_simp [simp]: "IF p x y ⇛  z"
inductive_simps EQ_simp [simp]: "EQ x y ⇛ z"

datatype tp = bool | num

inductive TP :: "exp ⇒ tp ⇒ bool" where
```

```
  T:    "TP T bool"
| F:    "TP F bool"
| Zero: "TP Zero num"
| IF:   "⟦TP p bool; TP x t; TP y t⟧ ⟹ TP (IF p x y) t"
| Succ: "TP x num ⟹ TP (Succ x) num"
| EQ:   "⟦TP x t; TP y t⟧ ⟹ TP (EQ x y) bool"

inductive_simps TP_IF [simp]: "TP (IF p x y) t"
inductive_simps TP_Succ [simp]: "TP (Succ x) t"
inductive_simps TP_EQ [simp]: "TP (EQ x y) t"

proposition type_preservation:
  assumes "x ⇛ y" "TP x t" shows "TP y t"
  using assms
  by (induction x y arbitrary: t rule: Eval.induct) (auto simp: TP.intros)

fun evl :: "exp ⇒ nat"
  where
    "evl T = 1"
  | "evl F = 0"
  | "evl Zero = 0"
  | "evl (Succ x) = evl x + 1"
  | "evl (IF x y z) = (if evl x = 1 then evl y else evl z)"
  | "evl (EQ x y) = (if evl x = evl y then 1 else 0)"

lemma
  assumes "TP x t" "t = bool" shows "evl x < 2"
  using assms by (induction x t; force)

proposition value_preservation:
  assumes "x ⇛ y" shows "evl x = evl y"
  using assms by (induction x y; force)
```

This doesn't hold

```
lemma
  assumes "x ⇛ y" "x ⇛ z" shows "∃u. y ⇛ u ∧ z ⇛ u"
  nitpick
  oops

inductive EvalStar :: "exp ⇒ exp ⇒ bool" (infix "⇛*" 50) where
    Id: "x ⇛* x"
  | Step: "x ⇛ y ⟹ y ⇛* z ⟹ x ⇛* z"

proposition type_preservation_Star:
  assumes "x ⇛* y" "TP x t" shows "TP y t"
  using assms by (induction x y) (auto simp: type_preservation)

lemma Succ_EvalStar:
  assumes "x ⇛* y" shows "Succ x ⇛* Succ y"
```

```
  using assms by induction (auto intro: Succ_Eval EvalStar.intros)

lemma IF_EvalStar:
  assumes "p ⇛* q" shows "IF p x y ⇛* IF q x y"
  using assms by induction (auto intro: IF_Eval EvalStar.intros)

lemma EQ_EvalStar1:
  assumes "x ⇛* z" shows "EQ x y ⇛* EQ z y"
  using assms by induction (auto intro: EQ_Eval1 EvalStar.intros)

lemma EQ_EvalStar2:
  assumes "y ⇛* z" shows "EQ x y ⇛* EQ x z "
  using assms by induction (auto intro: EQ_Eval2 EvalStar.intros)

proposition diamond:
  assumes "x ⇛ y" "x ⇛ z" shows "∃u. y ⇛* u ∧ z ⇛* u"
  using assms
proof (induction x y arbitrary: z)
  case (IF_Eval p q x y)
  then show ?case
    by (simp; meson F_simp IF_EvalStar T_simp)
next
  case (EQ_SS x y)
  then show ?case
    by (simp; meson Eval.intros EvalStar.intros)
next
  case (EQ_Eval1 x u y)
  then show ?case
    by (auto; meson EQ_EvalStar1 Eval.intros EvalStar.intros)+
next
    case (EQ_Eval2 y u x)
    then show ?case
    by (auto; meson EQ_EvalStar2 Eval.intros EvalStar.intros)+
qed (force intro: Succ_EvalStar Eval.intros EvalStar.intros)+

end
theory Binomial_Coeffs
imports Complex_Main "HOL-Number_Theory.Fib"
begin

lemma choose_row_sum: "(∑k≤n. n choose k) = 2^n"
  using binomial [of 1 1 n] by (simp add: numeral_2_eq_2)
```

sums of binomial coefficients.

```
lemma sum_choose_lower:
    "(∑k≤n. (r+k) choose k) = Suc (r+n) choose n"
  by (induction n) auto

lemma sum_choose_upper:
```

```
    "(∑k≤n. k choose m) = Suc n choose Suc m"
  by (induction n) auto

lemma sum_choose_diagonal:
  assumes "m≤n"
    shows "(∑k≤m. (n-k) choose (m-k)) = Suc n choose m"
proof -
  have "(∑k≤m. (n-k) choose (m-k)) = (∑k≤m. (n-m+k) choose k)"
    using sum.atLeastAtMost_rev [of "λk. (n - k) choose (m - k)" 0 m]
    by (simp add: atMost_atLeast0 ‹m≤n›)
  also have "... = Suc (n-m+m) choose m"
    by (rule sum_choose_lower)
  also have "... = Suc n choose m" using assms
    by simp
  finally show ?thesis .
qed

lemma choose_mult_lemma:
  "((m+r+k) choose (m+k)) * ((m+k) choose k) = ((m+r+k) choose k) * ((m+r)
choose m)"
  (is "?lhs = _")
proof -
  have "?lhs =
      fact(m+r+k) div (fact(m+k) * fact(m+r-m)) * (fact(m+k) div (fact
k * fact m))"
    by (simp add: binomial_altdef_nat)
  also have "... = fact(m+r+k) * fact(m+k) div
                 (fact(m+k) * fact(m+r-m) * (fact k * fact m))"
    by (metis add_implies_diff add_le_mono1 choose_dvd diff_cancel2 div_mult_div_if_dvd
        le_add1 le_add2)
  also have "... = fact(m+r+k) div (fact r * (fact k * fact m))"
    by (auto simp: algebra_simps fact_fact_dvd_fact)
  also have "... = (fact(m+r+k) * fact(m+r)) div (fact r * (fact k * fact
m) * fact(m+r))"
    by simp
  also have "... =
      (fact(m+r+k) div (fact k * fact(m+r)) * (fact(m+r) div (fact r *
fact m)))"
    by (smt (verit) fact_fact_dvd_fact div_mult_div_if_dvd mult.assoc
mult.commute)
  finally show ?thesis
    by (simp add: binomial_altdef_nat mult.commute)
qed
```

The "Subset of a Subset" identity.

```
lemma choose_mult:
  "k ≤ m ⟹ m ≤ n ⟹ (n choose m) * (m choose k) = (n choose k) * ((n
- k) choose (m - k))"
  using choose_mult_lemma [of "m-k" "n-m" k] by simp
```

Concrete Mathematics, 5.18: ”this formula is easily verified by induction on m”

```
lemma choose_row_sum_weighted:
  "(∑k≤m. (r choose k) * (r/2 - k)) = (Suc m)/2 * (r choose (Suc m))"
proof (induction m)
  case 0 show ?case by simp
next
  case (Suc m)
  have "(∑k≤Suc m. real (r choose k) * (r/2 - k))
      = ((r choose Suc m) * (r/2 - (Suc m))) + (Suc m) / 2 * (r choose
Suc m)"
    by (simp add: Suc)
  also have "... = (r choose Suc m) * (real r - (Suc m)) / 2"
    by (simp add: field_simps)
  also have "... = Suc (Suc m) / 2 * (r choose Suc (Suc m))"
  proof (cases "r ≥ Suc m")
    case True with binomial_absorb_comp[of r "Suc m"] show ?thesis
      by (metis binomial_absorption mult.commute of_nat_diff of_nat_mult
times_divide_eq_left)
  qed (simp add: binomial_eq_0)
  finally show ?case .
qed


lemma sum_drop_zero: "(∑k≤Suc n. if 0<k then (f (k - 1)) else 0) =
(∑j≤n. f j)"
  by (induction n) auto

lemma sum_choose_drop_zero:
  "(∑k≤Suc n. if k = 0 then 0 else (Suc n - k) choose (k - 1)) =
    (∑j≤n. (n-j) choose j)"
  by (rule trans [OF sum.cong sum_drop_zero]) auto

lemma ne_diagonal_fib:
   "(∑k≤n. (n-k) choose k) = fib (Suc n)"
proof (induction n rule: fib.induct)
  case 1 show ?case by simp
next
  case 2 show ?case by simp
next
  case (3 n)
  have "(∑k≤Suc n. Suc (Suc n) - k choose k) =
       (∑k≤Suc n. (Suc n - k choose k) + (if k=0 then 0 else (Suc n
- k choose (k - 1))))"
    by (rule sum.cong) (simp_all add: choose_reduce_nat)
  also have "... = (∑k≤Suc n. Suc n - k choose k) +
                   (∑k≤Suc n. if k=0 then 0 else (Suc n - k choose (k
- 1)))"
    by (simp add: sum.distrib)
```

```
  also have "... = (∑k≤Suc n. Suc n - k choose k) + (∑j≤n. n - j choose
j)"
    by (metis sum_choose_drop_zero)
  finally show ?case using 3
    by simp
qed

end
```

4 Tim Gowers' Example: A Characterisation of Bijections

```
theory Gowers_Bijection imports Complex_Main

begin
```

Isabelle's default syntax for set difference is nonstandard.

```
abbreviation set_difference :: "['a set,'a set] ⇒ 'a set" (infixl "\"
65)
  where "A \ B ≡ A-B"
```

The one-line proof found by sledgehammer

```
lemma "bij_betw f X Y ⟷ (∀A. A⊆X ⟶ f ` (X\A) = Y \ f`A)"
  by (metis Diff_empty Diff_eq_empty_iff Diff_subset bij_betw_def image_is_empty

            inj_on_image_set_diff subset_antisym subset_image_inj)
```

A more detailed proof. Note use of variables for the left and right sides

```
lemma "bij_betw f X Y ⟷ (∀A. A⊆X ⟶ f ` (X\A) = Y \ f`A)"  (is "?L=?R")
proof
  show "?L ⟹ ?R"
    by (metis Diff_subset bij_betw_def inj_on_image_set_diff)
  assume ?R
  then have "inj_on f X" "f ` X = Y"
    by (auto simp: inj_on_def)
  then show ?L
    by (simp add: bij_betw_def)
qed
```

Bonus: an example due to Terrence Tao

```
lemma
  fixes a :: "nat ⇒ real"
  assumes a: "decseq a" and D: "⋀k. D k ≥ 0" and aD: "⋀k. a k ≤ D
k - D(Suc k)"
  shows "a k ≤ D 0 / (Suc k)"
proof -
  have "a k = (∑i≤k. a k) / (Suc k)"
    by simp
```

```
  also have "... ≤ (∑i≤k. a i) / (Suc k)"
    using a sum_mono[of "{..k}" "λi. a k" a]
    by (simp add: monotone_def divide_simps mult.commute)
  also have "... ≤ (∑i≤k. D i - D(Suc i)) / (Suc k)"
    by (simp add: aD divide_right_mono sum_mono)
  also have "... ≤ D 0 / (Suc k)"
    by (simp add: sum_telescope D divide_right_mono)
  finally show ?thesis .
qed

end
```

5 Numerical experiments

```
theory Numeric imports
  "HOL-Decision_Procs.Approximation" "HOL-Computational_Algebra.Primes"

begin
```

Addition of polymorphic numerals actually works, though nobody should rely on this

```
lemma "2+2=4"
  by auto
```

Multiplication of polymorphic numerals does not work

```
lemma "2*3=6"
  oops
```

These do not work because the group identity law is not available.

```
lemma "0+2=2" "1*3=3"
  oops
```

Works because of the type constraint. And multiplcation is fast!

```
lemma "123456789 * (987654321::int) = 121932631112635269"
  by simp
```

The function `Suc` implies type `nat`

```
lemma "Suc (Suc 0) * n = n*2"
  by simp
```

We have to expand 5 into Suc-notation.

```
lemma "x^5 = x*x*x*x*(x::real)"
  by (simp add: eval_nat_numeral)
```

Decimal notation and arithmetic on complex numbers

```
lemma "(1 - 0.3*i) * (2.7 + 5*i) = 4.2 + 4.19*i"
  by (simp add: algebra_simps)
```

Applying a function to a numeral argument via eval. But only if this has been set up.

```
lemma "fact 20 < (2432902008176640001::nat)"
  by eval
```

Testing primality via eval, takes a couple of seconds. The type constraint is necessary!

```
lemma "prime (179424673::nat)"
  by eval
```

A simple demonstration of the approximation method

```
lemma "|pi - 355/113| < 1/10^6"
  by (approximation 25)
```

Ditto, the approximation method

```
lemma "|sqrt 2 - 1.4142135624| < 1/10^10"
  by (approximation 35)
```

The approximation method on a *closed* interval (SLOW). Must avoid zero!

```
lemma
  fixes x::real
  assumes "x ∈ {0.1 .. 1}"
  shows "x * ln(x) ≥ -0.368"
  using assms by (approximation 17 splitting: x=13)
```

A little more accuracy makes it MUCH slower (the exact answer is -1/e = 0.36787944117144233)

```
lemma
  fixes x::real
  assumes "x ∈ {0.1 .. 1}"
  shows "x * ln(x) ≥ -0.3679"
using assms
  by (approximation 18 splitting: x=16)

end
```

6 A tricky example of proving a lower bound

```
theory Ln_lower_bound imports
  "HOL-Analysis.Analysis" "HOL-Decision_Procs.Approximation"  "HOL-Real_Asymp.Real_Asymp"


begin
```

Thanks to Manuel Eberl

```
lemma continuous_at_0: "continuous (at_right 0) (λx::real. x * ln x)"
```

```
  unfolding continuous_within by real_asymp

lemma continuous_nonneg:
  fixes x::real
  assumes "x ≥ 0"
  shows "continuous (at x within {0..}) (λx. x * ln x)"
proof (cases "x = 0")
  case True with continuous_at_0 show ?thesis
    by (force simp add: continuous_within_topological less_eq_real_def)
qed (auto intro!: continuous_intros)

lemma continuous_on_x_ln: "continuous_on {0..} (λx::real. x * ln x)"
  unfolding continuous_on_eq_continuous_within
  using continuous_nonneg by blast

lemma xln_deriv:
  fixes x::real
  assumes "x > 0"
  shows "((λu. u * ln(u)) has_real_derivative ln x + 1) (at x)"
  by (rule derivative_eq_intros refl | use assms in force)+

theorem x_ln_lowerbound:
  fixes x::real
  assumes "x ≥ 0"
  shows "x * ln(x) ≥ -1 / exp 1"
proof -
  define xmin::real where "xmin ≡ 1 / exp 1"
  have "xmin > 0"
    by (auto simp: xmin_def)
  have "x * ln(x) > xmin * ln(xmin)" if "x < xmin"
  proof (intro DERIV_neg_imp_decreasing_open [OF that] exI conjI)
    fix u :: real
    assume "x < u" and "u < xmin"
    then have "ln u + 1 < ln 1"
      unfolding xmin_def
      by (smt (verit, del_insts) assms exp_diff exp_less_cancel_iff exp_ln_iff)
    then show "ln u + 1 < 0"
      by simp
  next
    show "continuous_on {x..xmin} (λu. u * ln u)"
      using continuous_on_x_ln continuous_on_subset assms by fastforce
  qed (use assms xln_deriv in auto)
  moreover
  have "x * ln(x) > xmin * ln(xmin)" if "x > xmin"
  proof (intro DERIV_pos_imp_increasing_open [OF that] exI conjI)
    fix u
    assume "x > u" and "u > xmin"
    then show "ln u + 1 > 0"
      by (smt (verit, del_insts) ‹0 < xmin› exp_minus inverse_eq_divide
```

```
        ln_less_cancel_iff ln_unique xmin_def)
  next
    show "continuous_on {xmin..x} (λu. u * ln u)"
      using continuous_on_x_ln continuous_on_subset xmin_def by fastforce
  qed (use ‹0 < xmin› xln_deriv in auto)
  moreover have "xmin * ln(xmin) = -1 / exp 1"
    using assms by (simp add: xmin_def ln_div)
  ultimately show ?thesis
    by force
qed


corollary
  fixes x::real
  assumes "x ≥ 0"
  shows "x * ln(x) ≥ -0.36787944117144233"  (is "_ ≥ ?rhs")
proof -
  have "(-1::real) / exp 1 ≥ ?rhs"
    by (approximation 60)
  with x_ln_lowerbound show ?thesis
    using assms by force
qed
```

The same proof works for a quite different function

```
lemma continuous_at_0_sin: "continuous (at_right 0) (λx::real. x * sin(1/x))"
  unfolding continuous_within by real_asymp

end
theory XsinX_lower_bounds
  imports
    "HOL-Analysis.Analysis"
    "HOL-Decision_Procs.Approximation"
    "HOL-Real_Asymp.Real_Asymp"

begin
```

7 Preliminaries

7.1 Splitting reals

```
lemma real_splits:
  fixes x d ::real
  assumes "d > 0"
  shows "x - d * floor (x/d) ≥ 0"
    and "x - d * floor (x/d) < d"
proof-
  let ?res = "x - d * real_of_int ⌊x / d⌋"
  show "0 ≤ ?res" by simp (metis assms floor_divide_lower mult.commute)
```

```
  show "?res < d"
  proof-
    have "?res < d ⟷ (x / d) < (1+floor(x/d))"
      using assms by (simp add:field_split_simps)
    also have "... ⟷ True" using floor_less_iff by fastforce
    finally show "?res < d" by simp
  qed
qed

lemma real_div_split_coi:
  fixes x offset :: real
  assumes "d > 0"
  obtains k :: int and res ::real where
    "x = res + d*real_of_int k" "res ∈ {offset..<offset+d}"
proof
  let ?k = "floor ((x-offset)/d)"
  let ?res = "x - d*?k"
  show "x = ?res + d*?k" by auto
  show "?res ∈ {offset..<offset+d}"
    using real_splits[OF assms, of "x-offset"] by simp
qed

lemma real_div_split_oci:
  fixes x offset :: real
  assumes "d > 0"
  obtains k :: int and res ::real where
    "x = res + d*real_of_int k" "res ∈ {offset<..offset+d}"
proof
  let ?kraw = "floor ((x-offset)/d)"
  let ?resraw = "x - d*?kraw"
  let ?k = "if ?resraw = offset then ?kraw - 1 else ?kraw"
  and ?res = "if ?resraw = offset then ?resraw+d else ?resraw"
  show "x = ?res + d*real_of_int ?k"
    by (simp add: algebra_simps)
  show "?res ∈ {offset<..offset + d}"
    using real_splits[OF assms, of "x-offset"] by simp
qed

lemma real_div_split0:
  fixes x :: real
  assumes "d > 0"
  obtains k :: int and res ::real where "x = res + d*real_of_int k" "0
≤ res" "res < d"
  using real_div_split_coi[where ?d = d and ?offset = 0 and ?x = x and
?thesis = thesis, OF assms]
  by simp
```

7.2 Characterizations on the arguments of bounded cosine

```
lemma cos_greater_iff:
  assumes "t ∈ {-1..1}"
  shows "cos x > (t::real) ⟷ (∃k::int. x ∈ {2*pi*k - arccos t<..<2*pi*k
+ arccos t})"
    (is "_ = ?r")
proof(intro iffI)
  obtain res k where k: "x = res + 2*pi*real_of_int k" and res: "res
∈ {-pi<..pi}"
    using real_div_split_oci[of "2*pi" x "-pi" thesis] by (auto simp:
algebra_simps)
  hence kbounds: "2*pi*k - pi < x" "x ≤ 2*pi*k + pi" by auto
  define pres where "pres = abs res"
  hence presbd: "0 ≤ pres" "pres ≤ pi" using res by auto
  have "cos x > t ⟷ cos res > t" using k cos_periodic_int by (simp
add: mult.commute add.commute)
  also have "... ⟷ cos pres > t" unfolding pres_def by simp
  also have "... ⟷ pres < arccos t" using presbd arccos_less_arccos[of
t "cos pres"] arccos_cos[of pres]
    assms cos_monotone_0_pi[of pres "arccos t"] arccos_ubound[of t] by
auto
  also have "... ⟷ res ∈ {- arccos t<..<arccos t}" by (auto simp: pres_def)
  finally have equiv: "cos x > t ⟷ res ∈ {-arccos t<..<arccos t}" .
  {
    then show "t < cos x ⟹ ?r"
      using k by (auto intro: exI[of _ k])
  }
  {
    assume "?r"
    then obtain k'::int where k': "2*pi*k' - arccos t < x" "x < 2*pi*k'
+ arccos t" by auto
    hence k'bds:  "2*pi*k' - pi < x" "x < 2*pi*k' + pi" using arccos_lbound
arccos_ubound assms by fastforce+
    have "k' = k"
    proof(cases k' k rule: linorder_cases)
      case less
      have "2 * pi * real_of_int k - pi < 2 * pi * real_of_int k' + pi"
using kbounds(1) k'bds(2) by linarith
      hence "2 * pi * k < 2 * pi * (k' + 1)" by (auto simp: algebra_simps)
      with less show ?thesis by simp
    next
      case greater
      have "2 * pi * real_of_int k' - pi < 2 * pi * real_of_int k + pi"
using kbounds(2) k'bds(1) by linarith
      hence "2 * pi * k' < 2 * pi * (k + 1)" by (auto simp: algebra_simps)
      with greater show ?thesis by simp
    qed
    then show "t < cos x" by (intro equiv[THEN iffD2]) (use res k k'
in auto)
```

```
  }
qed

lemma cos_geq_iff:
  assumes "t ∈ {-1..1}"
  shows "cos x ≥ (t::real) ⟷ (∃k::int. x ∈ {2*pi*k - arccos t..2*pi*k
+ arccos t})"
  (is "_ = ?r")
proof(cases "cos x = t"; intro iffI)
  case True
  from cos_eq_arccos_Ex[THEN iffD1, OF True] obtain k where
    k: "x = arccos t + 2 * real_of_int k * pi ∨ x = - arccos t + 2 * real_of_int
k * pi" by auto
  then show "∃xa. x ∈ {2 * pi * real_of_int xa - arccos t..2 * pi *
real_of_int xa + arccos t}"
    using assms by (auto intro!: exI[of _ k] arccos_lbound)
  show "t ≤ cos x" using True by simp
next
  case False
  {
    have [intro]: "∃v. u ∈ {a v<..<b v} ⟹ ∃v . u∈{a v .. b v}" for
u :: "_::preorder" and a b
      using less_le_not_le by auto
    assume "t ≤ cos x"
    with False have "t < cos x" by simp
    with cos_greater_iff[OF assms, of x, THEN iffD1]
    show ?r by (auto simp del: greaterThanLessThan_iff atLeastAtMost_iff)
  }
  assume ?r
  then obtain k where "2 * pi * real_of_int k - arccos t ≤ x" "x ≤ 2
* pi * real_of_int k + arccos t"
    by auto
  with False have "2 * pi * real_of_int k - arccos t < x"
    apply (cases "2 * pi * real_of_int k - arccos t = x")
    using cos_periodic_int[of "-arccos t" k] assms by (auto simp: algebra_simps)
  moreover have "x < 2 * pi * real_of_int k + arccos t"
    apply (cases "x = 2 * pi * real_of_int k + arccos t")
    using ‹x ≤ 2 * pi * real_of_int k + arccos t›
      False cos_periodic_int[of "arccos t" k] assms by (auto simp: algebra_simps)
  ultimately have "t < cos x" using cos_greater_iff[of t x, THEN iffD2,
OF assms]
    by auto
  then show "t ≤ cos x" by simp
qed


lemma cos_less_iff:
  assumes "t ∈ {-1..1}"
  shows "cos x < (t::real) ⟷ (∃k::int. x ∈ {pi * (2*k) + arccos t<..<pi*(2*(k+1))
```

```
- arccos t})"
  (is "_ = ?r")
proof-
  have "(¬ (cos x < t)) = (t ≤ cos x)" by auto
  also have "... =  (∃k::int. x ∈ {2*pi*k - arccos t..2*pi*k + arccos
t})"
    using assms by (fact cos_geq_iff)
  also have "... = (¬ ( ?r))"
  proof (safe)
    fix k l assume asm: "x ∈ {2 * pi * real_of_int k - arccos t..2 * pi
* real_of_int k + arccos t}"
      "x ∈ {pi * (2 * real_of_int l) + arccos t<..<pi * (2 * (real_of_int
l + 1)) - arccos t}"
    hence "2 * pi * real_of_int k - arccos t < pi * (2 * (real_of_int
l + 1)) - arccos t" by (auto simp: algebra_simps)
    hence "k ≤ l" by auto
    have "pi * (2 * real_of_int l) + arccos t < x" using asm by auto
    also have "... ≤ 2 * pi * real_of_int k + arccos t" using asm by simp
    finally have "l < k" by simp
    with ‹k ≤ l› show False by simp
  next
    assume asm: "∄xa. x ∈ {pi * (2 * real_of_int xa) + arccos t<..<pi
* (2 * (real_of_int xa + 1)) - arccos t}"
    moreover obtain k res where res: "0 ≤ res" "res < 2*pi" and x:
"x = pi* 2* real_of_int k + res"
      using real_div_split_coi[of "2*pi" x 0 thesis] by fastforce
    have "res ≤ arccos t | 2*pi - arccos t ≤ res" using asm x
      apply (subst (asm) not_ex)
      apply (drule spec[of _ "k"])
      by (simp add: algebra_simps | safe)+
    then show "∃xa. x ∈ {2 * pi * real_of_int xa - arccos t..2 * pi
* real_of_int xa + arccos t}"
    proof(elim disjE; intro exI)
      assume "res ≤ arccos t"
      then show "x ∈ {2 * pi * real_of_int k - arccos t..2 * pi * real_of_int
k + arccos t}"
        using x res by auto
    next
      assume "2 * pi - arccos t ≤ res"
      then show "x ∈ {2 * pi * real_of_int (k+1) - arccos t..2 * pi *
real_of_int (k+1) + arccos t}"
        using x res by (auto simp: algebra_simps)
    qed
  qed
  finally show ?thesis by auto
qed

lemma cos_leq_iff:
  assumes "t ∈ {-1..1}"
```

```
  shows "cos x ≤ (t::real) ⟷ (∃k::int. x ∈ {pi * (2*k) + arccos t..pi*(2*(k+1))
- arccos t})"
    (is "_ = ?r")
proof-
  have "(¬ (cos x ≤ t)) = (t < cos x)" by auto
  also have "... =  (∃k::int. x ∈ {2*pi*k - arccos t<..<2*pi*k + arccos
t})"
    using assms by (fact cos_greater_iff)
  also have "... = (¬ ( ?r))"
  proof (safe)
    fix k l assume asm: "x ∈ {2 * pi * real_of_int k - arccos t<..<2 *
pi * real_of_int k + arccos t}"
      "x ∈ {pi * (2 * real_of_int l) + arccos t..pi * (2 * (real_of_int
l + 1)) - arccos t}"
    hence "2 * pi * real_of_int k - arccos t < pi * (2 * (real_of_int
l + 1)) - arccos t" by (auto simp: algebra_simps)
    hence "k ≤ l" by auto
    have "pi * (2 * real_of_int l) + arccos t ≤ x" using asm by auto
    also have "... < 2 * pi * real_of_int k + arccos t" using asm by simp
    finally have "l < k" by simp
    with ‹k≤l› show False by simp
  next
    assume asm: "∄xa. x ∈ {pi * (2 * real_of_int xa) + arccos t..pi *
(2 * (real_of_int xa + 1)) - arccos t}"
    moreover obtain k res where res: "0 ≤ res" "res < 2*pi" and x:
"x = pi* 2* real_of_int k + res"
      using real_div_split_coi[of "2*pi" x 0 thesis] by fastforce
    have "res < arccos t | 2*pi - arccos t < res" using asm x
      apply (subst (asm) not_ex)
      apply (drule spec[of _ "k"])
      by (simp add: algebra_simps | safe)+
    then show "∃xa. x ∈ {2 * pi * real_of_int xa - arccos t<..<2 * pi
* real_of_int xa + arccos t}"
    proof(elim disjE; intro exI)
      assume "res < arccos t"
      then show "x ∈ {2 * pi * real_of_int k - arccos t<..<2 * pi * real_of_int
k + arccos t}"
        using x res by auto
    next
      assume "2 * pi - arccos t < res"
      then show "x ∈ {2 * pi * real_of_int (k+1) - arccos t<..<2 * pi
* real_of_int (k+1) + arccos t}"
        using x res by (auto simp: algebra_simps)
    qed
  qed
  finally show ?thesis by auto
qed
```

7.3 Nonnegative derivatives with finitely many zeroes are still increasing

```
lemma DERIV_pos_imp_increasing_open_fin_zeroes:
  fixes a b :: real
    and f :: "real ⇒ real"
  assumes "finite N"
  assumes "a < b"
    and "⋀x. a < x ⟹ x < b ⟹ (∃y. DERIV f x :> y ∧ y ≥ 0 ∧ (y =
0 ⟶ x ∈ N))"
    and con: "continuous_on {a..b} f"
  shows "f a < f b"
  using assms
proof(induction N arbitrary: a b set: finite)
  case empty
  have "(0 ≤ y ∧ (y = 0 ⟶ x ∈ {})) = (0 < y)" for x y :: real by auto
  then show ?case using empty DERIV_pos_imp_increasing_open by presburger
next
  case (insert x F)
  show ?case
  proof(cases "x∈{a<..<b}")
    case True
    have "f a < f x"
    proof (rule insert(3))
      show "a < t ⟹ t < x ⟹ ∃y. (f has_real_derivative y) (at t)
∧ 0 ≤ y ∧ (y = 0 ⟶ t ∈ F)" for t
        using insert(5)[of t] True by auto
      show "a < x" using True by simp
      show "continuous_on {a..x} f" using insert(6) True continuous_on_subset
by fastforce
    qed
    also have "... < f b"
    proof (rule insert(3))
      show "x < t ⟹ t < b ⟹ ∃y. (f has_real_derivative y) (at t)
∧ 0 ≤ y ∧ (y = 0 ⟶ t ∈ F)" for t
        using insert(5)[of t] True by auto
      show "x < b" using True by simp
      show "continuous_on {x..b} f" using insert(6) True continuous_on_subset
by fastforce
    qed
    finally show ?thesis .
  next
    case False
    then show ?thesis using insert by auto
  qed
qed

lemma DERIV_pos_imp_increasing_fin_zeros:
  fixes a b :: real and f :: "real ⇒ real"
  assumes fin: "finite N"
```

```
  assumes "a < b"
    and der: "⋀x. ⟦a ≤ x; x ≤ b⟧ ⟹ ∃y. DERIV f x :> y ∧ y ≥ 0 ∧ (y
= 0 ⟶ x ∈ N)"
  shows "f a < f b"
  by (metis less_le_not_le DERIV_atLeastAtMost_imp_continuous_on
      DERIV_pos_imp_increasing_open_fin_zeroes [OF ‹finite N› ‹a < b›]
der)
```

7.4 Specializations of the IVT for (strict) monotone real functions

```
lemma real_mono_IVT'_set:
  fixes f :: "real ⇒ real"
  assumes y: "f a ≤ y" "y ≤ f b" "a ≤ b"
  assumes cont: "continuous_on {a..b} f"
  assumes mono: "mono_on {a..b} f"
  shows "∃u v. {x . a ≤ x ∧ x ≤ b ∧ f x = y } = {u..v} ∧ a ≤ u ∧ u
≤ v ∧ v ≤ b"
proof-
  let ?P = "λx. a ≤ x ∧ x ≤ b ∧ f x = y"
  let ?S = "{x. ?P x}"
  from IVT'[OF y cont] obtain x where x: "f x = y" "a ≤ x" "x ≤ b" by
auto
  have "closed ?S"
    using continuous_closed_preimage_constant[OF cont closed_atLeastAtMost,
of y] by simp
  have "connected ?S"

  proof (rule connectedI_interval)
    fix r s t assume asm: "r ∈ ?S" "t∈?S" "r ≤ s" "s ≤ t"
    then have "a ≤ s" "s ≤ b" by simp_all
    moreover with asm mono[THEN mono_onD, of r s] mono[THEN mono_onD,
of s t]
    have "f s = y" by auto
    ultimately show "s∈?S" by simp
  qed
  have "?S ∩ {a..b} = ?S" by auto
  then have "compact ?S"
    using closed_Int_compact[OF ‹closed ?S› compact_Icc, of a b] by (simp
only:)
  with connected_compact_interval_1[of ?S] ‹connected ?S›
  obtain u v where "?S = {u..v}" by auto
  then show ?thesis
  proof(intro exI conjI, assumption)
    assume asm: "{x. a ≤ x ∧ x ≤ b ∧ f x = y} = {u..v}"
    with x show "u ≤ v" using disjoint_iff by fastforce
    then show "a ≤ u" "v ≤ b" using asm by fastforce+
  qed
qed
```

```
lemma real_smono_IVT'_set:
  fixes f :: "real ⇒ real"
  assumes y: "f a ≤ y" "y ≤ f b" "a ≤ b"
  assumes cont: "continuous_on {a..b} f"
  assumes smono: "strict_mono_on {a..b} f"
  shows "∃!ξ. a ≤ ξ ∧ ξ ≤ b ∧ f ξ = y"
proof-
  have mono: "mono_on {a..b} f" using smono by (fact strict_mono_on_imp_mono_on)
  from real_mono_IVT'_set[OF y cont mono]
  obtain u v where uv: "{x. a ≤ x ∧ x ≤ b ∧ f x = y} = {u..v}" "a ≤
u" "u ≤ v" "v ≤ b"
    by auto
  have "u ∈ {x. a ≤ x ∧ x ≤ b ∧ f x = y}" "v ∈ {x. a ≤ x ∧ x ≤ b ∧
f x = y}"
    using uv by simp_all
  hence "f u = y" "f v = y" by simp_all
  then have False if "u < v"
    using uv(2-4) strict_mono_onD[OF smono _ _ that] by auto
  with uv(3) le_less have "u = v" by auto
  show ?thesis
    by (intro ex1I[of _ u] conjI ‹f u = y› uv(2) order.trans[OF uv(3,4)])
      (use ‹u = v› uv(1) in auto)
qed
```

7.5 Monotonicity rules for mono_on

```
lemma strict_mono_on_less:
  "strict_mono_on S (f:: _ ::linorder ⇒ _ ::preorder) ⟹ x∈S ⟹ y∈S
⟹ (f x < f y) = (x < y)"
proof (intro iffI)
  assume asm: "strict_mono_on S f" "x ∈ S" "y ∈ S" "f x < f y"
  show "x < y"
  proof(rule ccontr)
    assume "¬ (x < y)"
    hence "y ≤ x" by simp
    then have "f y ≤ f x" using asm strict_mono_on_imp_mono_on[of S f]
mono_onD by blast

    with asm(4) show False using less_le_not_le by blast
  qed
qed (rule monotone_onD)

lemma sin_antimono_on_pi_halves_3_pi_halves:
"⟦pi/2 ≤ x; x ≤ y; y ≤ 3*pi / 2⟧ ⟹ sin y ≤ sin x"

  apply (rule DERIV_nonpos_imp_nonincreasing[of x y sin])
  apply (assumption)
  apply (intro exI conjI derivative_eq_intros)
```

```
  apply rule+
proof(subst mult_1_right)
  fix u assume "pi / 2 ≤ x" "x ≤ y" "y ≤ 3 * pi / 2" "x ≤ u" "u ≤ y"
  hence "pi / 2 ≤ u" "u ≤ 3* pi / 2" by simp_all
  then show "cos u ≤ 0"
    by (intro cos_leq_iff[THEN iffD2] exI[of _ 0]) simp_all
qed
```

8 Main part

```
abbreviation "xsix ≡ λx::real. x * sin (1/x)"

lemma xsininvx_sym:
  shows "xsix (- x) = xsix x" "xsix 0 = 0" by simp+
```

— Note that Isabelle, due to being a logic of total functions, considers the second goal to be a trivial consequence of (semi-)ring axioms. It's very convenient that limit and Isabelle's function evaluation are equal here, such that we do not need a case distinction in our function definition.

```
lemma continuous_at_0_xsininvx:
  "continuous (at_right 0) (λx::real. x * sin(1/x))"
  "continuous (at_left 0) (λx::real. x * sin(1/x))"
  unfolding continuous_within by real_asymp+

lemma cont_nonzero_xsininvx: "x ≠ 0 ⟹ continuous (at x) (λx::real.
x * sin(1/x))"
  by (intro continuous_intros)

lemma f_cont: "continuous_on UNIV (λx::real. x * sin(1/x))"
  unfolding continuous_on_eq_continuous_within
  by (metis cont_nonzero_xsininvx continuous_at_0_xsininvx continuous_at_split)

lemma xsix_deriv:
  fixes x::real
  assumes "x ≠ 0"
  shows "(xsix has_real_derivative (- cos (1/x) / x + sin(1/x))) (at x)"
  using assms by (fastforce intro: derivative_eq_intros)

lemma xsix_extrema_equiv:
  fixes x::real
  defines "u ≡ 1 / x"
  shows "(- cos (1/x) / x + sin(1/x)) = 0 ⟷ tan u = u"
proof(cases "x = 0")
  case False
  have "sin u = 0 ⟹ cos u = 0 ⟹ False"
    by (metis abs_zero sin_zero_abs_cos_one zero_neq_one)
```

```
  with False show ?thesis
    by (auto simp: field_simps tan_def u_def)
qed (auto simp: u_def)
```

```
lemma xsix_sym_bounded: "(⋀y. y≥0 ⟹ xsix y ≥ b) ⟹ xsix x ≥ b"
  by (cases "x ≥ 0") (smt (verit, best) xsininvx_sym(1))+

lemma bd_below_linear_xsininvx: "- abs x ≤ xsix x"
  and bd_above_linear_xsininvx: "xsix x ≤ abs x"
  by (smt (verit) sin_le_one sin_ge_minus_one minus_mult_minus mult_left_le
mult_minus_right)+
```

```
lemma xsix_bounded_critical_ivl: "∀x∈{2/(3*pi) <..<1/pi}. xsix x ≥ b
⟹ xsix x ≥ b"
proof(rule xsix_sym_bounded)
  fix y::real assume asm: "0 ≤ y" "∀x∈{2/(3*pi) <..<1/pi}. xsix x ≥
b"
  have bd: "b ≤ -0.215"
  proof-
    let ?x = "0.22255"
    have "?x∈{2/(3*pi) <..<1/pi}" by (simp, safe) (approximation 6)+
    moreover have "xsix ?x ≤ -0.215" by (approximation 10)
    ultimately show ?thesis using asm(2) by fastforce
  qed
  consider "0 ≤ y" "y ≤ 2/(3*pi)" | "y > 2/(3*pi)" "y < 1/pi" | "y ≥
1/pi"
    using ‹0 ≤ y› by linarith
  thus "b ≤ xsix y"
  proof(cases)
    case 1
    have "b ≤ -0.215" by (fact bd)
    also have "... ≤ -2 / (3 * pi)" by (approximation 8)
    also have "... ≤ - abs y" using 1 by auto
    also have "... ≤ xsix y" by (fact bd_below_linear_xsininvx)
    finally show ?thesis .
  next
    case 2
    then show ?thesis using asm(2) by simp
  next
    case 3
    have "-0.215 ≤ xsix (1/pi)" by (approximation 4)
    also have "... ≤ xsix y"
    proof(intro DERIV_nonneg_imp_nondecreasing[OF 3, of xsix] exI conjI,
rule xsix_deriv)
      fix x assume "1/pi ≤ x"
```

```
      moreover have "0 < 1/pi" by simp
      ultimately have "x > 0" by linarith
      then show "x ≠ 0" by simp
      have *: "0 ≤ - u * cos u + sin u" if "0 ≤ u" "u ≤ pi" for u
      proof-
        let ?f = "λu. -u*cos u + sin u"
        have "0 = ?f 0" by simp
        also have "... ≤ ?f u"
        proof(intro DERIV_nonneg_imp_nondecreasing[OF that(1)] exI conjI)
          fix a assume asm: ‹0 ≤ a› "a ≤ u"
          show "((λa. - a * cos a + sin a) has_real_derivative sin a
* a) (at a)"
            by (fastforce intro: derivative_eq_intros)
          show "0 ≤ sin a * a"
          proof (intro mult_nonneg_nonneg sin_ge_zero asm(1))
            from asm(2) that(2) show "a ≤ pi" by linarith
          qed
        qed
        finally show "0 ≤ - u * cos u + sin u" .
      qed
      let ?u = "1/x"
      have "0 ≤ - ?u * cos ?u + sin ?u"
      proof(rule *)
        show "?u ≥ 0"
          using ‹1/pi ≤ x› ‹0 < 1 / pi› zero_le_divide_1_iff by fastforce
        show "?u ≤ pi"
          using ‹0 < 1/pi› ‹1/pi ≤ x› ‹x > 0› by (simp add: field_split_simps)
      qed
      then show "0 ≤ - cos (1 / x)/x + sin (1 / x) "
        by simp
    qed
    finally show ?thesis using bd by linarith
  qed
qed

lemma tan_minus_id_strict_mono:
  "strict_mono_on {-pi/2+ real_of_int k*pi<..<pi/2 + k*pi} (λu. tan u
- u)"
  (is "strict_mono_on ?S ?f")
proof (intro strict_mono_onI)
  fix r s assume "r∈?S" "s∈?S" "r < s"
  show "?f r < ?f s"

  proof(intro DERIV_pos_imp_increasing_fin_zeros[where ?N = "{k*pi}"
and ?f = ?f, OF _ ‹r < s›] conjI exI)
    show "finite {real_of_int k * pi}" by simp
    fix x assume xbounds: "r ≤ x" "x ≤ s"
    have "cos x ≠ 0"
    proof (subst cos_zero_iff_int2, safe)
```

```
      fix m assume x: "x = real_of_int m * pi + pi / 2"
      then have "m < k"
        using order.strict_trans1[OF ‹x ≤ s›, of "pi / 2 + real_of_int
k * pi"] ‹s ∈ ?S› by simp
      moreover have "k ≤ m"
      proof-
        have "r ≤ pi * real_of_int m + pi / 2" "pi * real_of_int k <
r + pi / 2 "
          using ‹r ∈ ?S› ‹r ≤ x› by (simp_all add: x algebra_simps)
        hence "pi * real_of_int k < pi + pi * (real_of_int m)"
          by simp
        hence "pi * real_of_int k < pi * (real_of_int (1+m))"
          by (simp only: of_int_add of_int_1 distrib_left)
        thus "k ≤ m" by simp
      qed
      ultimately show False by simp
    qed
    then show "((λu. tan u - u) has_real_derivative 1/(cos x)² - 1) (at
x)"
      by (intro derivative_eq_intros; blast?) (simp add: field_simps)
    show "0 ≤ 1 / (cos x)² - 1"
    proof-
      have "(cos x)² ≤ 1"
      proof(cases "cos x ≥ 0")
        case True
        have "cos x * cos x ≤ 1 * 1" by (intro mult_mono' True cos_le_one)
        then show ?thesis by (simp add: power2_eq_square)
      next
        case False
        have "cos x * cos x ≤ (-1)*(-1)" by (rule mult_mono_nonpos_nonpos)
(use False in simp_all)
        then show ?thesis by (simp add: power2_eq_square)
      qed
      thus ?thesis by (simp add: field_split_simps ‹cos x ≠ 0›)
    qed
    show "1 / (cos x)² - 1 = 0 ⟶ x ∈ {real_of_int k * pi}"
    proof(simp add: field_simps power2_eq_square, safe)
      have eql: "l = k" if "x = real_of_int l * pi" for l
      proof(cases l k rule: linorder_cases)
        case less
        have "r ≤ pi * real_of_int l" using ‹r ≤ x› that by (auto simp:
mult.commute)
        also have "... ≤ pi* real_of_int (k-1)" using less by simp
        also have "... < r - pi + pi / 2" using ‹r∈?S› by (simp add: algebra_simps)
        also have "... < r" by simp
        finally show ?thesis .
      next
        case greater
        then have "pi*real_of_int (k+1) ≤ pi* real_of_int l" by auto
```

```
        also have "... < pi * real_of_int k + pi / 2"
          using ‹x ≤ s› ‹s ∈ ?S› that by (auto simp: algebra_simps)
        finally show ?thesis by (simp add: algebra_simps)
      qed
      assume "cos x * cos x = 1"
      hence "cos x = 1 | cos x = -1" by algebra
      then show "x = pi * real_of_int k"
      proof
        assume "cos x = - 1"
        then obtain l where l: "x = (2 * real_of_int l + 1) *pi" by (auto
simp add: cos_eq_minus1)
        let ?l = "2* l +1"
        have "x = real_of_int ?l * pi" using l by simp
        then show "x = pi * real_of_int k" using eql mult.commute by
blast
      next
        assume "cos x = 1"
        then obtain l where "x = pi*(2*real_of_int l)" by (auto simp:
cos_one_2pi_int)
        with eql[of "(2*l)"] show "x = pi * real_of_int k" by simp
      qed
    qed
  qed
qed

lemma xsix_interval_narrowing:
  fixes lb ub :: real
  defines "dvxsix ≡ (λx::real. - cos (1/x) / x + sin(1/x))"
    and "dvsgn ≡ λu::real. tan u - u"
  assumes bds: "2/(3*pi) < lb" "lb ≤ ub" "ub < 1/pi"
    and dvsgn: "dvsgn (1/lb) ≥ 0" "dvsgn (1/ub) ≤ 0"
  shows "xsix x ≥ sin (1/lb) * ub"
proof(rule xsix_bounded_critical_ivl, safe)
  fix x assume "x ∈ {2 / (3 * pi)<..<1 / pi}"
  hence xbds: "2 /(3*pi) < x" "x < 1/pi" "0 < x" by simp_all

  from bds have bdssgn[simp]: "0 < ub" "0 < lb" "0 < 1/ub" by auto
  have "continuous_on {lb..ub} dvxsix"
    unfolding dvxsix_def by (intro continuous_intros) (use bds in simp_all)

  have "dvxsix y = ((λu. cos u * (tan u - u)) o (λx. 1/x)) y" if "y≠0"
"cos (1/y) ≠ 0" for y
    using that by (simp add: dvxsix_def tan_def o_def field_split_simps
split: if_splits)

  have cosnz: "cos (1/y) < 0" if "2/(3*pi) < y" "y < 1/pi" for y
  proof (rule cos_less_iff[THEN iffD2])
    show "∃x. 1 / y ∈ {pi * (2 * real_of_int x) + arccos 0<..<pi * (2
* (real_of_int x + 1)) - arccos 0}"
```

```
    proof (rule exI[where ?x = 0])
      have "pi < 2 / y" "2 / y < 3 * pi"
        using that apply (simp_all add: field_split_simps)
         apply safe
        using that by simp+
      then show "1 / y ∈ {pi * (2 * real_of_int 0) + arccos 0<..<pi *
(2 * (real_of_int 0 + 1)) - arccos 0}"
        by simp
    qed
  qed simp

  have *: "dvsgn (1/y) ≥ 0 ⟷ dvxsix y ≤ 0" if "2/(3*pi) < y" "y < 1/pi"
for y
    unfolding dvsgn_def dvxsix_def tan_def
    apply (simp only: diff_ge_0_iff_ge diff_le_0_iff_le uminus_add_conv_diff
divide_minus_left)
    apply (rule neg_le_divide_eq[of "cos (1/y)" "1/y" "sin (1/y)", simplified])
    using that by (fact cosnz)

  have invbds: "u < 3 * pi / 2" "pi < u" if "u ∈ {1 / ub..1 / lb}" for
u
  proof-
    have "u ≤ 1/lb" using that by simp
    also have "... < (3 * pi) / 2"
      using bds by (auto simp: field_split_simps)
    finally show "u < 3 * pi / 2" .
    have "pi < 1/ub" using bds by (use bds in simp | safe | simp add:
field_split_simps)+
    also have "1/ub ≤ u" using that by auto
    finally show "pi < u" .
  qed

  have "1 / ub ≤ 1 / lb"
    using bds by (simp | safe | simp add: field_split_simps)+

  have "continuous_on {pi<..<3* pi / 2} dvsgn" unfolding dvsgn_def
  proof (safe intro!: continuous_intros)
    fix z assume asm: "z ∈ {pi<..<3* pi / 2}" "cos z = 0"
    have "cos z < 0"
    proof(rule cos_less_iff[of 0 z, THEN iffD2])
      show "∃y. z ∈ {pi * (2 * real_of_int y) + arccos 0<..<pi * (2 *
(real_of_int y + 1)) - arccos 0}"
        using asm by (auto intro: exI[of _ 0])
    qed simp
    with asm(2) show False by simp
  qed
  then have cont: "continuous_on {1 / ub..1 / lb} dvsgn"
    using invbds by (auto intro: continuous_on_subset)
```

```
  moreover have smono: "strict_mono_on {pi / 2<..<3 * pi / 2} dvsgn"
    unfolding dvsgn_def by (fact tan_minus_id_strict_mono[of 1, simplified])
  then have "strict_mono_on {1 / ub..1 / lb} dvsgn"
    by (rule monotone_on_subset) (use invbds in fastforce)

  with real_smono_IVT'_set[of dvsgn "1/ub" 0 "1/lb" , OF dvsgn(2,1) ‹1/ub
≤ 1/lb› cont]
  obtain ξ where xi: "1 / ub ≤ ξ" "ξ ≤ 1 / lb" "dvsgn ξ = 0"
    by fast
  hence "0 < ξ" using ‹0 < 1/ub› by linarith
  define xmin where "xmin ≡ 1/ξ"
  hence "0 < xmin" using ‹0 < ξ› by (simp_all add: field_split_simps)

  have "lb ≤ xmin" unfolding xmin_def
    using xi(2) ‹0 < ξ› by (simp add: field_split_simps)
  with bds(1) have "2 / (3 * pi) < xmin" by simp

  have "ξ * 2 < 3 * pi"
    using ‹2 / (3 * pi) < xmin› xmin_def ‹0 < ξ› by (auto simp: field_split_simps)

  have "pi < ξ * 2"
    using xi(1) invbds(2)[of "1/ub", simplified] ‹1 / ub ≤ 1 / lb› bdssgn(3)
by linarith

  have "1 / ξ ≤ ub"
  proof-
    have "0 < ξ / ub" by (rule divide_pos_pos) fact+
    with divide_left_mono[of "1/ ub" ξ 1, OF ‹1/ub ≤ ξ›]
    show ?thesis by simp
  qed

  have "xsix x > xsix xmin" if "x < xmin"
  proof (intro DERIV_neg_imp_decreasing_open [OF that] exI conjI)
    show "⟦x < u; u < xmin⟧ ⟹ (xsix has_real_derivative dvxsix u) (at
u)" for u
      unfolding dvxsix_def by (rule xsix_deriv) (use ‹0 < x› in simp)
    show "continuous_on {x..xmin} xsix" using continuous_on_subset[OF
f_cont subset_UNIV] .
    {
      fix u assume "x<u" "u<xmin"
      hence "2 / (3 * pi) < u" "u ≠ 0" "u > 0" using xbds by (auto simp:
xmin_def)

      have "u < 1 / pi" using ‹u < xmin› ‹1 / ξ ≤ ub› bds(2,3) by (simp
add: xmin_def)

      have "dvxsix u < 0 ⟷ dvsgn (1/u) > 0"
        using *[of u, symmetric, OF ‹2/(3*pi) < u› ‹u < 1/pi›] xsix_extrema_equiv[of
u]
```

```
        by (auto simp: dvxsix_def dvsgn_def)
      moreover have "dvsgn (1/u) > 0"
      proof(rule smono[THEN strict_mono_onD, of ξ "1/u", simplified, simplified
xi(3)])
        show "pi < ξ * 2" by fact
        show "2 / u < 3 * pi" using ‹u > 0› ‹2 / (3 * pi) < u› by (simp
add: field_split_simps)
        show "ξ < 1 / u"
          using ‹u > 0› ‹u < xmin› ‹0 < ξ› by (simp add: field_split_simps
xmin_def)
      qed
      ultimately show "dvxsix u < 0"
        by (simp only:)
    }
  qed
  moreover have "xsix x > xsix xmin" if "x > xmin"
  proof (intro DERIV_pos_imp_increasing_open [OF that] exI conjI)
    show "⟦xmin < u⟧ ⟹ (xsix has_real_derivative dvxsix u) (at u)" for
u
      unfolding dvxsix_def by (rule xsix_deriv) (use ‹0 < xmin› in simp)
    show "continuous_on {xmin..x} xsix" using continuous_on_subset[OF
f_cont subset_UNIV] .
    {
      fix u assume "xmin < u" "u < x"
      hence u: "2 / (3 * pi) < u" "u < 1 / pi" "u≠ 0" "u > 0"
        using ‹xmin > 0› ‹2 / (3 * pi) < xmin› xbds(2) by auto

      have "dvxsix u > 0 ⟷ dvsgn (1/u) < 0"
        using *[of u, symmetric, OF ‹2 / (3 * pi) < u› ‹u < 1 / pi›] xsix_extrema_equiv[of
u]
        by (auto simp: dvxsix_def dvsgn_def)
      moreover have "dvsgn (1/u) < 0"
      proof(rule smono[THEN strict_mono_onD, of "1/u" ξ, simplified, simplified
xi(3)])
        show "pi < 2 / u" using u by (auto simp: field_split_simps)
        show "ξ * 2 < 3 * pi" by fact
        show "1 / u < ξ" using xmin_def u(4) apply (auto simp: field_split_simps)
          by (metis ‹0 < ξ› ‹xmin < u› divide_less_eq mult.commute)
      qed
      ultimately show "0 < dvxsix u"
        by simp
    }
  qed
  ultimately have "xsix x ≥ xsix xmin" by fastforce
  also have "xsix xmin ≥ sin (1/lb) * ub"
  proof (subst mult.commute, rule mult_mono_nonneg_nonpos)
    show "sin (1 / lb) ≤ sin (1 / xmin)"
    proof(rule sin_antimono_on_pi_halves_3_pi_halves)
      show "pi / 2 ≤ 1 / xmin" "1 / xmin ≤ 1 / lb" "1 / lb ≤ 3 * pi
```

```
/ 2"
        using xi(2) ‹pi < ξ * 2› ‹0 <lb › bds(1) by (simp_all add: xmin_def
field_split_simps)
    qed
    show "sin (1 / xmin) ≤ 0"
    proof (rule sin_le_zero)
      show "1 / xmin < 2 * pi" using xmin_def ‹2 / (3 * pi) < xmin› ‹0
< ξ›
        by (auto simp: field_split_simps)
      show "pi ≤ 1/xmin" unfolding xmin_def
        using strict_mono_on_less[OF smono, of pi ξ, simplified,
            simplified ‹ξ * 2 < 3 * pi› ‹dvsgn ξ = 0›] ‹pi < ξ * 2› by
(simp add: dvsgn_def)
    qed
    show "0 ≤ xmin" using ‹0 < xmin› by simp
    show "xmin ≤ ub" unfolding xmin_def by fact
  qed
  finally show "sin (1 / lb) * ub ≤ xsix x" .
qed

lemma xsix_lowerbound_78:
  "xsix x ≥ -0.217233628211221657408279325562470734223044915435587482365449027714505343589О
  (is "?u ≤ _")
proof-
— Because of approximation, we need this awkward format
  let ?lb = "1 / (1.43029665312420275777219844545854836892203153170119392995362943759339439б
* pi )"
  and ?ub = "1 / (1.43029665312420275777219844545854836892203153170119392995362943759339439
* pi )"

  let ?uclaim= "-0.2172336282112216574082793255624707342230449154355874823654490277145053435
  and ?ureal = "-0.217233628211221657408279325562470734223044915435587482365449027714505343

  have "?u ≤ sin (1 / ?lb) * ?ub" by (approximation 267)
  also have "... ≤ xsix x"
  proof(rule xsix_interval_narrowing)
    show "2 / (3 * pi) < ?lb" by (approximation 11)
    show "?ub < 1/pi" by (approximation 7)
    show "0 ≤ tan (1/?lb) - 1/?lb" by (approximation 264)
    show "?lb ≤ ?ub" by (approximation 262)
    show "tan (1/?ub) - 1/?ub ≤ 0" by (approximation 263)
  qed
  finally show ?thesis .
qed
thm_oracles xsix_lowerbound_78
```

end

9 A Simple Probabilistic Proof by Paul Erdős.

Actual source is the webpage here: https://www.cut-the-knot.org/Probability/ProbabilisticMethod.shtml. Could not find the Erdős article and he wrote a dozen in 1963, in four languages.

```
theory Probabilistic_Example_Erdos
  imports "HOL-Library.Ramsey" "HOL-Probability.Probability"

begin

theorem Erdos_1963:
  assumes X: "𝓕 ⊆ nsets X n" "finite X"
    and "card 𝓕 = m" and m: "m < 2^(n-1)" and n: "0 < n" "n ≤ card X"
  obtains f::"'a⇒nat" where "f ∈ X →E {..<2}" "⋀F c. ⟦F ∈ 𝓕; c<2⟧
⟹ ¬ f ` F ⊆ {c}"
proof -
  have "finite 𝓕"
    using X finite_imp_finite_nsets finite_subset by blast
  let ?two = "{..<2::nat}"
  define Ω where "Ω ≡ X →E ?two"
  define M where "M ≡ uniform_count_measure Ω"
  have space_eq: "space M = Ω"
    by (simp add: M_def space_uniform_count_measure)
  have sets_eq: "sets M = Pow Ω"
    by (simp add: M_def sets_uniform_count_measure)
  have cardΩ: "card Ω = 2 ^ card X"
    using ‹finite X› by (simp add: Ω_def card_funcsetE)
  have Ω: "finite Ω" "Ω ≠ {}"
    using cardΩ less_irrefl by fastforce+
  interpret P: prob_space M
    unfolding M_def by (intro prob_space_uniform_count_measure Ω)
  define mchrome where "mchrome ≡ λc F. {f ∈ Ω. f ` F ⊆ {c}}"
      — the event to avoid: monochromatic sets
  have mchrome: "mchrome c F ∈ P.events" "mchrome c F ⊆ Ω" for F c
    by (auto simp: sets_eq mchrome_def Ω_def)
  have card_mchrome: "card (mchrome c F) = 2 ^ (card X - n)" if "F ∈ 𝓕"
"c<2" for F c
  proof -
    have F: "finite F" "card F = n" "F ⊆ X"
      using assms that by (auto simp: nsets_def)
    with ‹finite X› have "card (X-F →E ?two) = 2 ^ (card X - n)"
      by (simp add: card_funcsetE card_Diff_subset)
    moreover
    have "bij_betw (λf. restrict f (X-F)) (mchrome c F) (X-F →E ?two)"
```

```
    proof (intro bij_betwI)
      show "(λg x. if x∈F then c else g x) ∈ (X-F →E ?two) → mchrome
c F"
        using that ‹F ⊆ X› by (auto simp: mchrome_def Ω_def)
    qed (fastforce simp: mchrome_def Ω_def)+
    ultimately show ?thesis
      by (metis bij_betw_same_card)
  qed
  have prob_mchrome: "P.prob (mchrome c F) = 1 / 2^n"
    if "F ∈ ℱ" "c<2" for F c
  proof -
    have emeasure_eq: "emeasure M U = (if U⊆Ω then ennreal(card U / card
Ω) else 0)" for U
      by (simp add: M_def emeasure_uniform_count_measure_if ‹finite Ω›)
    have "emeasure M (mchrome c F) = ennreal (2 ^ (card X - n) / card
Ω)"
      using that mchrome by (simp add: emeasure_eq card_mchrome)
    also have "... = ennreal (1 / 2^n)"
      by (simp add: n power_diff cardΩ)
    finally show ?thesis
      by (simp add: P.emeasure_eq_measure)
  qed

  have "(⋃F∈ℱ. ⋃c<2. mchrome c F) ⊆ Ω"
    by (auto simp: mchrome_def Ω_def)
  moreover have "(⋃F∈ℱ. ⋃c<2. mchrome c F) ≠ Ω"
  proof -
    have "P.prob (⋃F∈ℱ. ⋃c<2. mchrome c F) ≤ (∑F∈ℱ. P.prob (⋃c<2.
mchrome c F))"
      by (intro measure_UNION_le) (auto simp: countable_Un_Int mchrome
‹finite ℱ›)
    also have "... ≤ (∑F∈ℱ. ∑c<2. P.prob (mchrome c F))"
      by (intro sum_mono measure_UNION_le) (auto simp: mchrome)
    also have "... = m * 2 * (1 / 2^n)"
      by (simp add: prob_mchrome ‹card ℱ = m›)
    also have "... < 1"
    proof -
      have "real (m * 2) < 2 ^ n"
        using mult_strict_right_mono [OF m, of 2] ‹n>0›
        by (metis of_nat_less_numeral_power_cancel_iff pos2 power_minus_mult)

      then show ?thesis
        by (simp add: divide_simps)
    qed
    finally have "P.prob (⋃F∈ℱ. ⋃c<2. mchrome c F) < 1" .
    then show ?thesis
      using P.prob_space space_eq by force
  qed
  ultimately obtain f where f: "f ∈ Ω - (⋃F∈ℱ. ⋃c<2. mchrome c F)"
```

```
    by blast
  with that show ?thesis
    by (fastforce simp: mchrome_def Ω_def)
qed

end
```

10 Holiday exercises: the greatest prime power divisor

```
theory Index_ex
  imports "HOL-Computational_Algebra.Primes"

begin

lemma primepow_divides_prod:
  fixes p::nat
  assumes "prime p" "(p^k) dvd (m * n)"
  shows "∃i j. (p^i) dvd m ∧ (p^j) dvd n ∧ k = i + j"
  sorry

lemma maximum_index:
  fixes n::nat
  assumes "n > 0" "2 ≤ p"
  shows "∃k. p^k dvd n ∧ (∀l>k. ¬ p^l dvd n)"
  sorry

definition index :: "nat ⇒ nat ⇒ nat" where
  "index p n
    ≡ if p ≤ 1 ∨ n = 0 then 0 else card {j. 1 ≤ j ∧ p^j dvd n}"

proposition pow_divides_index:
  "p^k dvd n ⟷ n = 0 ∨ p = 1 ∨ k ≤ index p n"
  sorry

corollary le_index:
  "k ≤ index p n ⟷ (n = 0 ∨ p = 1 ⟶ k = 0) ∧ p^k dvd n"
  sorry

corollary exp_index_divides: "p^(index p n) dvd n"
  sorry

lemma index_trivial_bound: "index p n ≤ n"
  sorry

proposition index_mult:
  assumes "prime p" "m > 0" "n > 0"
  shows "index p (m * n) = index p m + index p n"
```

```
  sorry

corollary index_exp:
  assumes "prime p"
  shows "index p (n^k) = k * index p n"
  sorry

end
```

11 Iterative Fibonacci and Code Generation

```
theory Fib_Iter
  imports Complex_Main  — allows implicit coercions
begin
```

The type of integers banishes the successor function

```
fun itfib :: "[int,int,int] ⇒ int" where
  "itfib n j k = (if n≤1 then k else itfib (n-1) k (j+k))"
```

Simplification alone can compute this colossal number in a fraction of a second

```
lemma "itfib 200 0 1 = 280571172992510140037611932413038677189525"
  by simp

fun fib :: "nat ⇒ int" where
    "fib 0 = 0"
  | "fib (Suc 0) = 1"
  | "fib (Suc (Suc n)) = fib n + fib (Suc n)"
```

The two functions are closely related

```
lemma itfib_fib: "n > 0 ⟹ itfib n (fib k) (fib(k+1)) = fib (k+n)"
proof (induction n arbitrary: k)
  case 0
  then show ?case
    by auto
next
  case (Suc n)
  have "n > 0 ⟹ itfib n (fib (k+1)) (fib k + fib (k+1)) = fib (k+n+1)"
    by (metis Suc.IH Suc_eq_plus1 add.commute add_Suc_right fib.simps(3))
  with Suc show ?case
    by simp
qed
```

Direct recursive evaluation is exponential and slow

```
value "fib 44"

declare fib.simps [code del]
```

New code equation for `fib`

```
lemma fib_eq_itfib[code]: "fib n = (if n=0 then 0 else itfib (int n)
0 1)"
  using itfib_fib [of n 0] by simp
```

FAST

```
value "fib 200"

end
theory Dirichlet_approx_thm

imports "Complex_Main" "HOL-Library.FuncSet"

begin

theorem Dirichlet_approx:
  fixes ϑ::real and N::nat
  assumes "N > 0"
  obtains h k where "0 < k" "k ≤ int N" "|of_int k * ϑ - of_int h| < 1/N"
proof -
  define X where "X ≡ (λk. frac (k*ϑ)) ` {..N}"
  define Y where "Y ≡ (λk::nat. {k/N..< Suc k/N}) ` {..<N}"
  have False
    if non: "¬ (∃b≤N. ∃a<b. |frac (real b * ϑ) - frac (real a * ϑ)| <
1/N)"
  proof -
    have "inj_on (λk::nat. frac (k*ϑ)) {..N}"
      using non by (intro linorder_inj_onI; force)
    then have caX: "card X = Suc N"
      by (simp add: X_def card_image)
    have caY: "card Y ≤ N" "finite Y"
      unfolding Y_def using card_image_le by force+
    define f where "f ≡ λx::real. let k = nat ⌊x * N⌋ in {k/N ..< Suc
k/N}"
    have "nat ⌊x * N⌋ < N" if "0 ≤ x" "x < 1" for x::real
      using that assms floor_less_iff nat_less_iff by fastforce
    then have "f ∈ X → Y"
      by (force simp: f_def Let_def X_def Y_def frac_lt_1)
    then have "¬ inj_on f X"
      using ‹finite Y› caX caY card_inj by fastforce
    then obtain x x' where "x≠x'" "x ∈ X" "x' ∈ X" and eq: "f x = f
x'"
      by (auto simp: inj_on_def)
    then obtain c d::nat where c: "c ≠ d" "c ≤ N" "d ≤ N"
          and xeq: "x = frac (c*ϑ)" and xeq': "x' = frac (d*ϑ)"
      by (metis (no_types, lifting) X_def atMost_iff image_iff)
    define i where "i ≡ nat ⌊x * N⌋"
    then have i: "x ∈ {i/N ..< Suc i/N}"
      using assms by (auto simp: divide_simps xeq) linarith
    have i': "x' ∈ {i/N ..< Suc i/N}"
```

```
      using eq assms unfolding f_def Let_def xeq' i_def
      by (simp add: divide_simps) linarith
    with assms i have "|frac (d*ϑ) - frac (c*ϑ)| < 1 / N"
      by (simp add: xeq xeq' abs_if add_divide_distrib)
    with c show False
      by (metis abs_minus_commute nat_neq_iff non)
  qed
  then obtain a b::nat where *: "a<b" "b≤N" "|frac (b*ϑ) - frac (a*ϑ)|
< 1/N"
    by metis
  let ?k = "b-a" and ?h = "⌊b*ϑ⌋ - ⌊a*ϑ⌋"
  show ?thesis
  proof
    have "frac (b*ϑ) - frac (a*ϑ) = ?k*ϑ - ?h"
      using ‹a < b› by (simp add: frac_def left_diff_distrib of_nat_diff)
    with * show "|of_int ?k * ϑ - ?h| < 1/N"
      by (metis of_int_of_nat_eq)
  qed (use * in auto)
qed

end
```