The Theorem of Three Circles

Fox Thomson, Wenda Li

April 18, 2024

Abstract

The Descartes test based on Bernstein coefficients and Descartes' rule of signs effectively (over-)approximates the number of real roots of a univariate polynomial over an interval. In this entry we formalise the theorem of three circles (Theorem 10.50 in [1]), which gives sufficient conditions for when the Descartes test returns 0 or 1. This is the first step for efficient root isolation.

Contents

1 Misc results about polynomials **2**
- 1.1 Misc . . . 2
- 1.2 Misc results about polynomials . . . 3
- 1.3 The reciprocal polynomial . . . 11
- 1.4 More about *proots-count* . . . 17
- 1.5 More about *changes* . . . 18

2 Bernstein Polynomials over the interval [0, 1] **24**
- 2.1 Definition and basic results . . . 24
- 2.2 *Bernstein-Poly-01* and *reciprocal-poly* . . . 26
- 2.3 Bernstein coefficients and changes . . . 30
- 2.4 Expression as a Bernstein sum . . . 35

3 Bernstein Polynomials over any finite interval **37**
- 3.1 Definition and relation to Bernstein Polynomials over [0, 1] . . . 37
- 3.2 Bernstein coefficients and changes over any interval . . . 40
- 3.3 The control polygon of a polynomial . . . 43

4 Normal Polynomials **45**

5 Proof of the theorem of three circles **70**
- 5.1 No sign changes case . . . 71
- 5.2 One sign change case . . . 77
- 5.3 The theorem of three circles . . . 86

6 Acknowledgements 86

1 Misc results about polynomials

theory *RRI-Misc* **imports**
HOL−Computational-Algebra.Computational-Algebra
Budan-Fourier.BF-Misc
Polynomial-Interpolation.Ring-Hom-Poly
begin

1.1 Misc

declare *pcompose-pCons*[*simp del*]

lemma *Setcompr-subset*: $\bigwedge f\ P\ S.\ \{f\ x \mid x.\ P\ x\} \subseteq S = (\forall\ x.\ P\ x \longrightarrow f\ x \in S)$
by *blast*

lemma *map-cong'*:
assumes $xs = map\ h\ ys$ **and** $\bigwedge y.\ y \in set\ ys \Longrightarrow f\ (h\ y) = g\ y$
shows $map\ f\ xs = map\ g\ ys$
using *assms map-replicate-trivial* **by** *simp*

lemma *nth-default-replicate-eq*:
$nth\text{-}default\ dflt\ (replicate\ n\ x)\ i = (if\ i < n\ then\ x\ else\ dflt)$
by (*auto simp*: *nth-default-def*)

lemma *square-bounded-less*:
fixes $a\ b::'a$ :: *linordered-ring-strict*
shows $-a < b \wedge b < a \Longrightarrow b*b < a*a$
by (*metis* (*no-types*, *lifting*) *leD leI minus-less-iff minus-mult-minus mult-strict-mono'*
neg-less-eq-nonneg neg-less-pos verit-minus-simplify(*4*) *zero-le-mult-iff zero-le-square*)

lemma *square-bounded-le*:
fixes $a\ b::'a$ :: *linordered-ring-strict*
shows $-a \leq b \wedge b \leq a \Longrightarrow b*b \leq a*a$
by (*metis le-less minus-mult-minus square-bounded-less*)

context *vector-space*
begin

lemma *card-le-dim-spanning*:
assumes BV: $B \subseteq V$
and VB: $V \subseteq span\ B$
and fB: *finite B*
and dVB: $dim\ V \geq card\ B$
shows *independent B*
proof −
{
fix a

```
    assume a: a ∈ B a ∈ span (B − {a})
    from a fB have c0: card B ≠ 0
      by auto
    from a fB have cb: card (B − {a}) = card B − 1
      by auto
    {
      fix x
      assume x: x ∈ V
      from a have eq: insert a (B − {a}) = B
        by blast
      from x VB have x': x ∈ span B
        by blast
      from span-trans[OF a(2), unfolded eq, OF x']
      have x ∈ span (B − {a}) .
    }
    then have th1: V ⊆ span (B − {a})
      by blast
    have th2: finite (B − {a})
      using fB by auto
    from dim-le-card[OF th1 th2]
    have c: dim V ≤ card (B − {a}) .
    from c c0 dVB cb have False by simp
  }
  then show ?thesis
    unfolding dependent-def by blast
qed

end
```

1.2 Misc results about polynomials

```
lemma smult-power: smult (x^n) (p^n) = smult x p ^ n
  apply (induction n)
  subgoal by fastforce
  by (metis smult-power)

lemma reflect-poly-monom: reflect-poly (monom n i) = monom n 0
  apply (induction i)
  by (auto simp: coeffs-eq-iff coeffs-monom coeffs-reflect-poly)

lemma poly-eq-by-eval:
  fixes P Q :: 'a::{comm-ring-1,ring-no-zero-divisors,ring-char-0} poly
  assumes h: ⋀x. poly P x = poly Q x shows P = Q
proof −
  have poly P = poly Q using h by fast
  thus ?thesis by (auto simp: poly-eq-poly-eq-iff)
qed

lemma poly-binomial:
```

```
  [:(1::'a::comm-ring-1), 1:]^n = (∑k≤n. monom (of-nat (n choose k)) k)
proof −
  have [:(1::'a::comm-ring-1), 1:]^n = (monom 1 1 + 1)^n
    by (metis (no-types, lifting) add.left-neutral add.right-neutral add-pCons
        monom-altdef pCons-one power-one-right smult-1-left)
  also have ... = (∑k≤n. of-nat (n choose k) * monom 1 1 ^ k)
    apply (subst binomial-ring)
    by force
  also have ... = (∑k≤n. monom (of-nat (n choose k)) k)
    by (auto simp: monom-altdef of-nat-poly)
  finally show ?thesis .
qed

lemma degree-0-iff: degree P = 0 ⟷ (∃ a. P = [:a:])
  by (meson degree-eq-zeroE degree-pCons-0)

interpretation poly-vs: vector-space smult
  by (simp add: vector-space-def smult-add-right smult-add-left)

lemma degree-subspace: poly-vs.subspace {x. degree x ≤ n}
  by (auto simp: poly-vs.subspace-def degree-add-le)

lemma monom-span:
  poly-vs.span {monom 1 x | x. x ≤ p} = {(x::'a::field poly). degree x ≤ p}
(is ?L = ?R)
proof
  show ?L ⊆ ?R
  proof
    fix x assume x ∈ ?L
    moreover have hfin: finite {P. ∃ x ∈ {..p}. P = monom 1 x}
      by auto
    ultimately have
      x ∈ range (λu. ∑ v∈{monom 1 x | x. x ∈ {..p}}. smult (u v) v)
      by (simp add: poly-vs.span-finite)
    hence ∃ u. x = (∑ v∈{monom 1 x | x. x ∈ {..p}}. smult (u v) v)
      by (auto simp: image-iff)
    then obtain u
      where p': x = (∑ v∈{monom 1 x | x. x ∈ {..p}}. smult (u v) v)
      by blast
    have ⋀v. v ∈ {monom 1 x | x. x ∈ {..p}} ⟹ degree (smult (u v) v) ≤ p
      by (auto simp add: degree-monom-eq)
    hence degree x ≤ p using hfin
      apply (subst p')
      apply (rule degree-sum-le)
      by auto
    thus x ∈ {x. degree x ≤ p} by force
  qed
next
  show ?R ⊆ ?L
```

proof
fix x **assume** $x \in ?R$
hence $degree\ x \leq p$ **by** *force*
hence $x = (\sum i{\leq}p.\ monom\ (coeff\ x\ i)\ i)$
by (*simp add*: *poly-as-sum-of-monoms'*)
also have
$... = (\sum i{\leq}p.\ smult\ (coeff\ x\ (degree\ (monom\ (1{::}'a)\ i)))\ (monom\ 1\ i))$
by (*auto simp add*: *smult-monom degree-monom-eq*)
also have
$... = (\sum v{\in}\{monom\ 1\ x \mid x.\ x \in \{..p\}\}.\ smult\ ((\lambda v.\ coeff\ x\ (degree\ v))\ v)\ v)$
proof (*rule sum.reindex-cong*)
show *inj-on degree* $\{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
proof
fix x
assume $x \in \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
hence $\exists\ a.\ x = monom\ 1\ a$ **by** *blast*
then obtain a **where** *hx*: $x = monom\ 1\ a$ **by** *blast*
fix y
assume $y \in \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
hence $\exists\ b.\ y = monom\ 1\ b$ **by** *blast*
then obtain b **where** *hy*: $y = monom\ 1\ b$ **by** *blast*
assume $degree\ x = degree\ y$
thus $x = y$ **using** *hx hy* **by** (*simp add*: *degree-monom-eq*)
qed
show $\{..p\} = degree\ `\ \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
proof
show $\{..p\} \subseteq degree\ `\ \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
proof
fix x **assume** $x \in \{..p\}$
hence $monom\ (1{::}'a)\ x \in \{monom\ 1\ x \mid x.\ x \in \{..p\}\}$ **by** *force*
moreover have $degree\ (monom\ (1{::}'a)\ x) = x$
by (*simp add*: *degree-monom-eq*)
ultimately show $x \in degree\ `\ \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$ **by** *auto*
qed
show $degree\ `\ \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\} \subseteq \{..p\}$
by (*auto simp add*: *degree-monom-eq*)
qed
next
fix y **assume** $y \in \{monom\ (1{::}'a)\ x \mid x.\ x \in \{..p\}\}$
hence $\exists z \in \{..p\}.\ y = monom\ (1{::}'a)\ z$ **by** *blast*
then obtain z **where** $y = monom\ (1{::}'a)\ z$ **by** *blast*
thus
$smult\ (coeff\ x\ (degree\ (monom\ (1{::}'a)\ (degree\ y))))\ (monom\ (1{::}'a)\ (degree\ y)) =$
$smult\ (coeff\ x\ (degree\ y))\ y$
by (*simp add*: *smult-monom degree-monom-eq*)
qed
finally have $x = (\sum v{\in}\{monom\ 1\ x \mid x.\ x \in \{..p\}\}.$
$smult\ ((\lambda v.\ coeff\ x\ (degree\ v))\ v)\ v)$ **.**

```
    thus x ∈ ?L by (auto simp add: poly-vs.span-finite)
  qed
qed

lemma monom-independent:
  poly-vs.independent {monom (1::'a::field) x | x. x ≤ p}
proof (rule poly-vs.independent-if-scalars-zero)
  fix f::'a poly ⇒ 'a
  assume h: (∑ x∈{monom 1 x |x. x ≤ p}. smult (f x) x) = 0
  have h': (∑ i≤p. monom (f (monom (1::'a) i)) i) =
          (∑ x∈{monom (1::'a) x |x. x ≤ p}. smult (f x) x)
  proof (rule sum.reindex-cong)
    show inj-on degree {monom (1::'a) x |x. x ≤ p}
      by (smt (verit) degree-monom-eq inj-on-def mem-Collect-eq zero-neq-one)
    show {..p} = degree ` {monom (1::'a) x |x. x ≤ p}
    proof
      show {..p} ⊆ degree ` {monom (1::'a) x |x. x ≤ p}
      proof
        fix x assume x ∈ {..p}
        then have x = degree (monom (1::'a) x) ∧ x ≤ p
          by (auto simp: degree-monom-eq)
        thus x ∈ degree ` {monom (1::'a) x |x. x ≤ p}
          by blast
      qed
      show degree ` {monom (1::'a) x |x. x ≤ p} ⊆ {..p}
        by (force simp: degree-monom-eq)
    qed
  qed (auto simp: degree-monom-eq smult-monom)

  fix x::'a poly
  assume x ∈ {monom 1 x |x. x ≤ p}
  then obtain y where y ≤ p and x = monom 1 y by blast
  hence f x = coeff (∑ x∈{monom 1 x |x. x ≤ p}. smult (f x) x) y
    by (auto simp: coeff-sum h'[symmetric])
  thus f x = 0
    using h by auto
qed force

lemma dim-degree: poly-vs.dim {x. degree x ≤ n} = n + 1
  using poly-vs.dim-eq-card-independent[OF monom-independent]
  by (auto simp: monom-span[symmetric] card-image image-Collect[symmetric]
      inj-on-def monom-eq-iff')

lemma degree-div:
  fixes p q::('a::idom-divide) poly
  assumes q dvd p
  shows degree (p div q) = degree p − degree q using assms
  by (metis (no-types, lifting) add-diff-cancel-left' degree-0 degree-mult-eq
      diff-add-zero diff-zero div-by-0 dvd-div-eq-0-iff dvd-mult-div-cancel)
```

lemma *lead-coeff-div*:
fixes *p q*::('*a*::{*idom-divide*, *inverse*}) *poly*
assumes *q dvd p*
shows *lead-coeff* (*p div q*) = *lead-coeff p* / *lead-coeff q* **using** *assms*
by (*smt* (*z3*) *div-by-0 dvd-div-mult-self lead-coeff-mult leading-coeff-0-iff nonzero-mult-div-cancel-right*)

lemma *complex-poly-eq*:
r = *map-poly of-real* (*map-poly Re r*) + *smult* i (*map-poly of-real* (*map-poly Im r*))
by (*auto simp*: *poly-eq-iff coeff-map-poly complex-eq*)

lemma *complex-poly-cong*:
(*map-poly Re p* = *map-poly Re q* ∧ *map-poly Im p* = *map-poly Im q*) = (*p* = *q*)
by (*metis complex-poly-eq*)

lemma *map-poly-Im-of-real*: *map-poly Im* (*map-poly of-real p*) = *0*
by (*auto simp*: *poly-eq-iff coeff-map-poly*)

lemma *mult-map-poly-imp-map-poly*:
assumes *map-poly complex-of-real q* = *r* ∗ *map-poly complex-of-real p*
p ≠ *0*
shows *r* = *map-poly complex-of-real* (*map-poly Re r*)
proof −
have *h*: *Im* ∘ (∗) i ∘ *complex-of-real* = *id* **by** *fastforce*
have *map-poly complex-of-real q* = *r* ∗ *map-poly complex-of-real p*
using *assms* **by** *blast*
also have ... = (*map-poly of-real* (*map-poly Re r*) +
smult i (*map-poly of-real* (*map-poly Im r*))) ∗
map-poly complex-of-real p
using *complex-poly-eq* **by** *fastforce*
also have ... = *map-poly of-real* (*map-poly Re r* ∗ *p*) +
smult i (*map-poly of-real* (*map-poly Im r* ∗ *p*))
by (*simp add*: *mult-poly-add-left*)
finally have *map-poly complex-of-real q* =
map-poly of-real (*map-poly Re r* ∗ *p*) +
smult i (*map-poly of-real* (*map-poly Im r* ∗ *p*)) **.**
hence *0* = *map-poly Im* (*map-poly of-real* (*map-poly Re r* ∗ *p*) +
smult i (*map-poly of-real* (*map-poly Im r* ∗ *p*)))
by (*auto simp*: *complex-poly-cong*[*symmetric*] *map-poly-Im-of-real*)
also have ... = *map-poly of-real* (*map-poly Im r* ∗ *p*)
apply (*rule poly-eqI*)
by (*auto simp*: *coeff-map-poly coeff-mult*)
finally have *0* = *map-poly complex-of-real* (*map-poly Im r*) ∗
map-poly complex-of-real p
by *auto*
hence *map-poly complex-of-real* (*map-poly Im r*) = *0* **using** *assms* **by** *fastforce*
thus *?thesis* **apply** (*subst complex-poly-eq*) **by** *auto*

qed

lemma *map-poly-dvd*:
fixes *p q*::*real poly*
assumes *hdvd*: *map-poly complex-of-real p dvd*
map-poly complex-of-real q q ≠ *0*
shows *p dvd q*
proof −
from *hdvd* **obtain** *r*
where *h*:*map-poly complex-of-real q* = *r* ∗ *map-poly complex-of-real p*
by *fastforce*
hence *r* = *map-poly complex-of-real* (*map-poly Re r*)
using *mult-map-poly-imp-map-poly assms* **by** *force*
hence *map-poly complex-of-real q* = *map-poly complex-of-real* (*p* ∗ *map-poly Re r*)
using *h* **by** *auto*
hence *q* = *p* ∗ *map-poly Re r* **using** *of-real-poly-eq-iff* **by** *blast*
thus *p dvd q* **by** *force*
qed

lemma *div-poly-eq-0*:
fixes *p q*::(*'a*::*idom-divide*) *poly*
assumes *q dvd p poly* (*p div q*) *x* = *0 q* ≠ *0*
shows *poly p x* = *0*
using *assms* **by** *fastforce*

lemma *poly-map-poly-of-real-cnj*:
poly (*map-poly of-real p*) (*cnj z*) = *cnj* (*poly* (*map-poly of-real p*) *z*)
apply (*induction p*)
by *auto*

An induction rule on real polynomials, if $P \neq 0$ then either $(X - x)|P$ or $(X - z)(X - cnjz)|P$, we induct by dividing by these polynomials.

lemma *real-poly-roots-induct*:
fixes *P*::*real poly* ⇒ *bool* **and** *p*::*real poly*
assumes *IH-real*: ⋀*p x*. *P p* ⟹ *P* (*p* ∗ [:−*x*, *1*:])
and *IH-complex*: ⋀*p a b*. *b* ≠ *0* ⟹ *P p*
⟹ *P* (*p* ∗ [: *a*∗*a* + *b*∗*b*, −*2*∗*a*, *1* :])
and *H0*: ⋀*a*. *P* [:*a*:]
defines *d* ≡ *degree p*
shows *P p*
using *d-def*
proof (*induction d arbitrary*: *p rule*: *less-induct*)
fix *p*::*real poly*
assume *IH*: (⋀*q*. *degree q* < *degree p* ⟹ *P q*)
show *P p*
proof (*cases* *0* = *degree p*)
fix *p*::*real poly* **assume** *0* = *degree p*
hence ∃ *a*. *p* = [:*a*:]

```
    by (simp add: degree-0-iff)
   thus P p using H0 by blast
  next
   assume hdeg: 0 ≠ degree p
   hence ¬ constant (poly (map-poly of-real p))
    by (metis (no-types, opaque-lifting) constant-def constant-degree of-real-eq-iff
of-real-poly-map-poly)
   then obtain z::complex where h: poly (map-poly of-real p) z = 0
    using fundamental-theorem-of-algebra by blast
   show P p
   proof cases
    assume Im z = 0
    hence z = Re z by (simp add: complex-is-Real-iff)
    moreover have [:−z, 1:] dvd map-poly of-real p
     using h poly-eq-0-iff-dvd by blast
    ultimately have [:−(Re z), 1:] dvd p
     by (smt (z3) dvd-iff-poly-eq-0 h of-real-0 of-real-eq-iff of-real-poly-map-poly)
    hence 2:P (p div [:−Re z, 1:])
     apply (subst IH)
     using hdeg by (auto simp: degree-div)
    moreover have 1:p = (p div [:− Re z, 1:]) ∗ [:−Re z, 1:]
     by (metis ‹[:− Re z, 1:] dvd p› dvd-div-mult-self)
    ultimately show P p
     apply (subst 1)
     by (rule IH-real[OF 2])
   next
    assume Im z ≠ 0
    hence hcnj: cnj z ≠ z by (metis cnj.simps(2) neg-equal-zero)
    have h2: poly (map-poly of-real p) (cnj z) = 0
     using h poly-map-poly-of-real-cnj by force
    have [:−z, 1:] ∗ [:−cnj z, 1:] dvd map-poly of-real p
    proof (rule divides-mult)
     have ⋀c. c dvd [:−z, 1:] ⟹ c dvd [:− cnj z, 1:] ⟹ is-unit c
     proof −
      fix c
       assume h:c dvd [:−z, 1:] hence degree c ≤ 1 using divides-degree by
fastforce
      hence degree c = 0 ∨ degree c = 1 by linarith
      thus c dvd [:− cnj z, 1:] ⟹ is-unit c
      proof
       assume degree c = 0
       moreover have c ≠ 0 using h by fastforce
       ultimately show is-unit c by (simp add: is-unit-iff-degree)
      next
       assume hdeg: degree c = 1
       then obtain x where 1:[:−z, 1:] = x∗c using h by fastforce
       hence degree [:−z, 1:] = degree x + degree c
        by (metis add.inverse-neutral degree-mult-eq mult-cancel-right
            mult-poly-0-left pCons-eq-0-iff zero-neq-neg-one)
```

```
        hence degree x = 0 using hdeg by auto
        then obtain x' where 2: x = [:x':] using degree-0-iff by auto
        assume c dvd [:−cnj z, 1:]
        then obtain y where 3: [:−cnj z, 1:] = y∗c by fastforce
        hence degree [:−cnj z, 1:] = degree y + degree c
          by (metis add.inverse-neutral degree-mult-eq mult-cancel-right
              mult-poly-0-left pCons-eq-0-iff zero-neq-neg-one)
        hence degree y = 0 using hdeg by auto
        then obtain y' where 4: y = [:y':] using degree-0-iff by auto
        moreover from hdeg obtain a b where 5:c = [:a, b:] and 6: b ≠ 0
          by (meson degree-eq-oneE)
        from 1 2 5 6 have x' = inverse b by (auto simp: field-simps)
       moreover from 3 4 5 6 have y' = inverse b by (auto simp: field-simps)
        ultimately have x = y using 2 4 by presburger
      then have z = cnj z using 1 3 by (metis neg-equal-iff-equal pCons-eq-iff)
        thus is-unit c using hcnj by argo
      qed
    qed
    thus coprime [:− z, 1:] [:− cnj z, 1:] by (meson not-coprimeE)
    show [:− z, 1:] dvd map-poly complex-of-real p
      using h poly-eq-0-iff-dvd by auto
    show [:− cnj z, 1:] dvd map-poly complex-of-real p
      using h2 poly-eq-0-iff-dvd by blast
  qed
  moreover have [:− z, 1:] ∗ [:− cnj z, 1:] =
                map-poly of-real [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:]
    by (auto simp: complex-eqI)
  ultimately have hdvd:
    map-poly complex-of-real [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:] dvd
     map-poly complex-of-real p
    by force
  hence [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:] dvd p using map-poly-dvd
    by blast
  hence 2:P (p div [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:])
    apply (subst IH)
    using hdeg by (auto simp: degree-div)
  moreover have 1:
    p = (p div [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:]) ∗
              [:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:]
    apply (subst dvd-div-mult-self)
    using ‹[:Re z∗Re z + Im z∗Im z, −2∗Re z, 1:] dvd p› by auto
  ultimately show P p
    apply (subst 1)
    apply (rule IH-complex[of  Im z - Re z])
    apply (meson ‹Im z ≠ 0›)
    by blast
 qed
qed
qed
```

1.3 The reciprocal polynomial

definition *reciprocal-poly* :: *nat* ⇒ *'a::zero poly* ⇒ *'a poly*
where *reciprocal-poly p P* =
Poly (*rev* ((*coeffs P*) @ (*replicate* (*p* − *degree P*) *0*)))

lemma *reciprocal-0*: *reciprocal-poly p 0* = *0* **by** (*simp add*: *reciprocal-poly-def*)

lemma *reciprocal-1*: *reciprocal-poly p 1* = *monom 1 p*
by (*simp add*: *reciprocal-poly-def monom-altdef Poly-append*)

lemma *coeff-reciprocal*:
assumes *hi*: *i* ≤ *p* **and** *hP*: *degree P* ≤ *p*
shows *coeff* (*reciprocal-poly p P*) *i* = *coeff P* (*p* − *i*)
proof *cases*
assume *i* < *p* − *degree P*
hence *degree P* < *p* − *i* **using** *hP* **by** *linarith*
thus *coeff* (*reciprocal-poly p P*) *i* = *coeff P* (*p* − *i*)
by (*auto simp*: *reciprocal-poly-def nth-default-append coeff-eq-0*)
next
assume *h*: ¬*i* < *p* − *degree P*
show *coeff* (*reciprocal-poly p P*) *i* = *coeff P* (*p* − *i*)
proof *cases*
assume *P* = *0*
thus *coeff* (*reciprocal-poly p P*) *i* = *coeff P* (*p* − *i*)
by (*simp add*: *reciprocal-0*)
next
assume *hP'*: *P* ≠ *0*
have *degree P* ≥ *p* − *i* **using** *h hP* **by** *linarith*
moreover hence (*i* − (*p* − *degree P*)) < *length* (*rev* (*coeffs P*))
using *hP' hP hi* **by** (*auto simp*: *length-coeffs*)
thus *coeff* (*reciprocal-poly p P*) *i* = *coeff P* (*p* − *i*)
by (*auto simp*: *reciprocal-poly-def nth-default-append coeff-eq-0 hP hP'*
nth-default-nth rev-nth calculation coeffs-nth length-coeffs-degree)
qed
qed

lemma *coeff-reciprocal-less*:
assumes *hn*: *p* < *i* **and** *hP*: *degree P* ≤ *p*
shows *coeff* (*reciprocal-poly p P*) *i* = *0*
proof *cases*
assume *P* = *0*
thus *?thesis* **by** (*auto simp*: *reciprocal-0*)
next
assume *P* ≠ *0*
thus *?thesis*
using *hn*
by (*auto simp*: *reciprocal-poly-def nth-default-append*
nth-default-eq-dflt-iff hP length-coeffs)
qed

lemma *reciprocal-monom*:
assumes $n \leq p$
shows *reciprocal-poly p* (*monom a n*) = *monom a* (*p*−*n*)
proof (*cases a=0*)
case *True*
then show *?thesis* **by** (*simp add*: *reciprocal-0*)
next
case *False*
with ‹*n*≤*p*› **show** *?thesis*
apply (*rule-tac poly-eqI*)
by (*metis coeff-monom coeff-reciprocal coeff-reciprocal-less*
diff-diff-cancel diff-le-self lead-coeff-monom not-le-imp-less)
qed

lemma *reciprocal-degree*: *reciprocal-poly* (*degree P*) *P* = *reflect-poly P*
by (*auto simp add*: *reciprocal-poly-def reflect-poly-def*)

lemma *degree-reciprocal*:
fixes *P* :: (*'a*::*zero*) *poly*
assumes *hP*: *degree P* ≤ *p*
shows *degree* (*reciprocal-poly p P*) ≤ *p*
proof (*auto simp add*: *reciprocal-poly-def*)
have *degree* (*reciprocal-poly p P*) ≤
length (*replicate* (*p* − *degree P*) (*0*::*'a*) @ *rev* (*coeffs P*))
by (*metis degree-Poly reciprocal-poly-def rev-append rev-replicate*)
thus *degree* (*Poly* (*replicate* (*p* − *degree P*) *0* @ *rev* (*coeffs P*))) ≤ *p*
by (*smt Suc-le-mono add-Suc-right coeffs-Poly degree-0 hP le-SucE le-SucI*
le-add-diff-inverse2 le-zero-eq length-append length-coeffs-degree
length-replicate length-rev length-strip-while-le reciprocal-0
reciprocal-poly-def rev-append rev-replicate)
qed

lemma *reciprocal-0-iff*:
assumes *hP*: *degree P* ≤ *p*
shows (*reciprocal-poly p P* = *0*) = (*P* = *0*)
proof
assume *h*: *reciprocal-poly p P* = *0*
show *P* = *0*
proof (*rule poly-eqI*)
fix *n*
show *coeff P n* = *coeff 0 n*
proof *cases*
assume *hn*: *n* ≤ *p*
hence *p* − *n* ≤ *p* **by** *auto*
hence *coeff* (*reciprocal-poly p P*) (*p* − *n*) = *coeff P n*
using *hP hn* **by** (*auto simp*: *coeff-reciprocal*)
thus *?thesis* **using** *h* **by** *auto*
next

```
    assume hn: ¬ n ≤ p
    thus ?thesis using hP by (metis coeff-0 dual-order.trans le-degree)
   qed
  qed
next
  assume P = 0
  thus reciprocal-poly p P = 0 using reciprocal-0 by fast
qed

lemma poly-reciprocal:
  fixes P::'a::field poly
  assumes hp: degree P ≤ p and hx: x ≠ 0
  shows poly (reciprocal-poly p P) x = x^p * (poly P (inverse x))
proof −
  have poly (reciprocal-poly p P) x
     = poly ((Poly ((replicate (p − degree P) 0) @ rev (coeffs P)))) x
    by (auto simp add: hx reflect-poly-def reciprocal-poly-def)
  also have ... = poly ((monom 1 (p − degree P)) * (reflect-poly P)) x
    by (auto simp add: reflect-poly-def Poly-append)
  also have ... = x^(p − degree P) *  x ^ degree P * poly P (inverse x)
    by (auto simp add: poly-reflect-poly-nz poly-monom hx)
  also have ... = x^p * poly P (inverse x)
    by (auto simp add: hp power-add[symmetric])
  finally show ?thesis .
qed

lemma reciprocal-fcompose:
  fixes P::('a::{ring-char-0,field}) poly
  assumes hP: degree P ≤ p
  shows reciprocal-poly p P = monom 1 (p − degree P) * fcompose P 1 [:0, 1:]
proof (rule poly-eq-by-eval, cases)
  fix x::'a
  assume hx: x = 0
  hence poly (reciprocal-poly p P) x = coeff P p
    using hP by (auto simp: poly-0-coeff-0 coeff-reciprocal)
  moreover have poly (monom 1 (p − degree P)
    * fcompose P 1 [:0, 1:]) x = coeff P p
  proof cases
    assume degree P = p
    thus ?thesis
      apply (induction P arbitrary: p)
      using hx by (auto simp: poly-monom degree-0-iff fcompose-pCons)
  next
    assume degree P ≠ p
    hence degree P < p using hP by auto
    thus ?thesis
      using hx by (auto simp: poly-monom coeff-eq-0)
  qed
  ultimately show poly (reciprocal-poly p P) x = poly (monom 1 (p − degree P)
```

```
* fcompose P 1 [:0, 1:]) x
    by presburger
next
  fix x::'a assume x ≠ 0
  thus poly (reciprocal-poly p P) x =
       poly (monom 1 (p − degree P) * fcompose P 1 [:0, 1:]) x
    using hP
    by (auto simp: poly-reciprocal poly-fcompose inverse-eq-divide
       poly-monom power-diff)
qed

lemma reciprocal-reciprocal:
  fixes P :: 'a::{field,ring-char-0} poly
  assumes hP: degree P ≤ p
  shows reciprocal-poly p (reciprocal-poly p P) = P
proof (rule poly-eq-by-eval)
  fix x
  show poly (reciprocal-poly p (reciprocal-poly p P)) x = poly P x
  proof cases
    assume x = 0
    thus poly (reciprocal-poly p (reciprocal-poly p P)) x = poly P x
      using hP
      by (auto simp: poly-0-coeff-0 coeff-reciprocal degree-reciprocal)
  next
    assume hx: x ≠ 0
    hence poly (reciprocal-poly p (reciprocal-poly p P)) x
        = x ^ p * (inverse x ^ p * poly P x) using hP
      by (auto simp: poly-reciprocal degree-reciprocal)
    thus poly (reciprocal-poly p (reciprocal-poly p P)) x = poly P x
      using hP hx left-right-inverse-power right-inverse by auto
  qed
qed

lemma reciprocal-smult:
  fixes P :: 'a::idom poly
  assumes h: degree P ≤ p
  shows reciprocal-poly p (smult n P) = smult n (reciprocal-poly p P)
proof cases
  assume n = 0
  thus ?thesis by (auto simp add: reciprocal-poly-def)
next
  assume n ≠ 0
  thus ?thesis
    by (auto simp add: reciprocal-poly-def smult-Poly coeffs-smult
       rev-map[symmetric])
qed

lemma reciprocal-add:
  fixes P Q :: 'a::comm-semiring-0 poly
```

assumes *degree* $P \leq p$ **and** *degree* $Q \leq p$
shows *reciprocal-poly p* $(P + Q)$ = *reciprocal-poly p P* + *reciprocal-poly p Q*
(**is** *?L* = *?R*)
proof (*rule poly-eqI*, *cases*)
fix *n*
assume $n \leq p$
then show *coeff ?L n* = *coeff ?R n*
using *assms* **by** (*auto simp*: *degree-add-le coeff-reciprocal*)
next
fix *n* **assume** $\neg n \leq p$
then show *coeff ?L n* = *coeff ?R n*
using *assms* **by** (*auto simp*: *degree-add-le coeff-reciprocal-less*)
qed

lemma *reciprocal-diff*:
fixes *P Q* :: *'a*::*comm-ring poly*
assumes *degree P* $\leq p$ **and** *degree Q* $\leq p$
shows *reciprocal-poly p* $(P - Q)$ = *reciprocal-poly p P* − *reciprocal-poly p Q*
by (*metis* (*no-types*, *lifting*) *ab-group-add-class.ab-diff-conv-add-uminus assms add-diff-cancel degree-add-le degree-minus diff-add-cancel reciprocal-add*)

lemma *reciprocal-sum*:
fixes *P* :: *'a* ⇒ *'b*::*comm-semiring-0 poly*
assumes *hP*: $\bigwedge k$. *degree* $(P\ k) \leq p$
shows *reciprocal-poly p* $(\sum k{\in}A.\ P\ k)$ = $(\sum k{\in}A.$ *reciprocal-poly p* $(P\ k))$
proof (*induct A rule*: *infinite-finite-induct*)
case (*infinite A*)
then show *?case* **by** (*simp add*: *reciprocal-0*)
next
case *empty*
then show *?case* **by** (*simp add*: *reciprocal-0*)
next
case (*insert x F*)
assume $x \notin F$
and *reciprocal-poly p* (*sum P F*) = $(\sum k{\in}F.$ *reciprocal-poly p* $(P\ k))$
and *finite F*
moreover hence *reciprocal-poly p* (*sum P* (*insert x F*))
= *reciprocal-poly p* (*P x*) + *reciprocal-poly p* (*sum P F*)
by (*auto simp add*: *reciprocal-add hP degree-sum-le*)
ultimately show *reciprocal-poly p* (*sum P* (*insert x F*))
= $(\sum k{\in}$*insert x F*$.$ *reciprocal-poly p* $(P\ k))$
by (*auto simp*: *Groups-Big.comm-monoid-add-class.sum.insert-if*)
qed

lemma *reciprocal-mult*:
fixes *P Q*::*'a*::{*ring-char-0*,*field*} *poly*
assumes *degree* $(P * Q) \leq p$
and *degree P* $\leq p$ **and** *degree Q* $\leq p$
shows *monom 1 p* ∗ *reciprocal-poly p* $(P * Q)$ =

```
        reciprocal-poly p P * reciprocal-poly p Q
proof (cases P=0 ∨ Q=0)
  case True
  then show ?thesis using assms(1)
    by (auto simp: reciprocal-fcompose fcompose-mult)
next
  case False
  then show ?thesis
    using assms
    by (auto simp: degree-mult-eq mult-monom reciprocal-fcompose fcompose-mult)
qed

lemma reciprocal-reflect-poly:
  fixes P::'a::{ring-char-0,field} poly
  assumes hP: degree P ≤ p
  shows reciprocal-poly p P = monom 1 (p − degree P) * reflect-poly P
proof (rule poly-eqI)
  fix n
  show coeff (reciprocal-poly p P) n =
    coeff (monom 1 (p − degree P) * reflect-poly P) n
  proof cases
    assume n ≤ p
    thus ?thesis using hP
      by (auto simp: coeff-reciprocal coeff-monom-mult coeff-reflect-poly coeff-eq-0)
  next
    assume ¬ n ≤ p
    thus ?thesis using hP
      by (auto simp: coeff-reciprocal-less coeff-monom-mult coeff-reflect-poly)
  qed
qed

lemma map-poly-reciprocal:
  assumes degree P ≤ p and f 0 = 0
  shows map-poly f (reciprocal-poly p P)  = reciprocal-poly p (map-poly f P)
proof (rule poly-eqI)
  fix n
  show coeff (map-poly f (reciprocal-poly p P)) n =
        coeff (reciprocal-poly p (map-poly f P)) n
  proof (cases n≤p)
    case True
    then show ?thesis
      apply (subst coeff-reciprocal[OF True])
      subgoal by (meson assms(1) assms(2) degree-map-poly-le le-trans)
      by (simp add: assms(1) assms(2) coeff-map-poly coeff-reciprocal)
  next
    case False
    then show ?thesis
      by (metis assms(1) assms(2) coeff-map-poly coeff-reciprocal-less
          degree-map-poly-le dual-order.trans leI)
```

qed
qed

1.4 More about *proots-count*

lemma *proots-count-monom*:
assumes $0 \notin A$
shows *proots-count* (*monom 1 d*) $A = 0$
using *assms* **by** (*auto simp*: *proots-count-def poly-monom*)

lemma *proots-count-reciprocal*:
fixes P::'a::{*ring-char-0*,*field*} *poly*
assumes *hP*: *degree* $P \leq p$ **and** *h0*: $P \neq 0$ **and** *h0'*: $0 \notin A$
shows *proots-count* (*reciprocal-poly p P*) A = *proots-count* P $\{x.\ inverse\ x \in A\}$
proof −
have *proots-count* (*reciprocal-poly p P*) A =
proots-count (*fcompose P 1* [:*0*, *1*:]) A
apply (*subst reciprocal-fcompose*[*OF hP*], *subst proots-count-times*)
subgoal using *h0* **by** (*metis hP reciprocal-0-iff reciprocal-fcompose*)
subgoal using *h0'* **by** (*auto simp*: *proots-count-monom*)
done

also have ... = *proots-count* P $\{x.\ inverse\ x \in A\}$
proof (*rule proots-fcompose-bij-eq*[*symmetric*])
show *bij-betw* ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) A $\{x.\ inverse\ x \in A\}$
proof (*rule bij-betw-imageI*)
show *inj-on* ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) A
by (*simp add*: *inj-on-def*)

show ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) ‘ A = $\{x.\ inverse\ x \in A\}$
proof
show ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) ‘ $A \subseteq \{x.\ inverse\ x \in A\}$
by *force*
show $\{x.\ inverse\ x \in A\} \subseteq$ ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) ‘ A
proof
fix x **assume** $x \in \{x$::'$a.\ inverse\ x \in A\}$
hence x = *poly 1* (*inverse x*) */ poly* [:*0*, *1*:] (*inverse x*) $\wedge$ *inverse* $x \in A$
by (*auto simp*: *inverse-eq-divide*)
thus $x \in$ ($\lambda x.$ *poly 1 x / poly* [:*0*, *1*:] *x*) ‘ A **by** *blast*
qed
qed
qed
next
show $\forall c.\ 1 \neq$ *smult c* [:*0*, *1*:]
by (*metis coeff-pCons-0 degree-1 lead-coeff-1 pCons-0-0 pcompose-0'*
pcompose-smult smult-0-right zero-neq-one)
qed (*auto simp*: *assms infinite-UNIV-char-0*)
finally show *?thesis* **by** *linarith*
qed

lemma *proots-count-reciprocal'*:
fixes *P*::*real poly*
assumes *hP*: *degree P* ≤ *p* **and** *h0*: *P* ≠ *0*
shows *proots-count P* {*x*. *0* < *x* ∧ *x* < *1*} =
proots-count (*reciprocal-poly p P*) {*x*. *1* < *x*}
proof (*subst proots-count-reciprocal*)
show *proots-count P* {*x*. *0* < *x* ∧ *x* < *1*} =
proots-count P {*x*. *inverse x* ∈ *Collect* ((<) *1*)}
apply (*rule arg-cong*[*of* - - *proots-count P*])
using *one-less-inverse-iff* **by** *fastforce*
qed (*use assms* **in** *auto*)

lemma *proots-count-pos*:
assumes *proots-count P S* > *0*
shows ∃ *x* ∈ *S*. *poly P x* = *0*
proof (*rule ccontr*)
assume ¬ (∃ *x*∈*S*. *poly P x* = *0*)
hence ⋀*x*. *x* ∈ *S* ⟹ *poly P x* ≠ *0* **by** *blast*
hence ⋀*x*. *x* ∈ *S* ⟹ *order x P* = *0* **using** *order-0I* **by** *blast*
hence *proots-count P S* = *0* **by** (*force simp*: *proots-count-def*)
thus *False* **using** *assms* **by** *presburger*
qed

lemma *proots-count-of-root-set*:
assumes *P* ≠ *0 R* ⊆ *S* **and** ⋀*x*. *x*∈*R* ⟹ *poly P x* = *0*
shows *proots-count P S* ≥ *card R*
proof −
have *card R* ≤ *card* (*proots-within P S*)
apply (*rule card-mono*)
subgoal using *assms* **by** *auto*
subgoal using *assms*(*2*) *assms*(*3*) **by** (*auto simp*: *proots-within-def*)
done
also have ... ≤ *proots-count P S*
by (*rule card-proots-within-leq*[*OF assms*(*1*)])
finally show *?thesis* **.**
qed

lemma *proots-count-of-root*: **assumes** *P* ≠ *0 x*∈*S poly P x* = *0*
shows *proots-count P S* > *0*
using *proots-count-of-root-set*[*of P* {*x*} *S*] *assms* **by** *force*

1.5 More about *changes*

lemma *changes-nonneg*: *0* ≤ *changes xs*
apply (*induction xs rule*: *changes.induct*)
by *simp-all*

lemma *changes-replicate-0*: **shows** *changes* (*replicate n 0*) = *0*

```
  apply (induction n)
  by auto

lemma changes-append-replicate-0: changes (xs @ replicate n 0) = changes xs
proof (induction xs rule: changes.induct)
  case (2 uu)
  then show ?case
    apply (induction n)
    by auto
qed (auto simp: changes-replicate-0)

lemma changes-scale-Cons:
  fixes xs:: real list assumes hs: s > 0
  shows changes (s * x # xs) = changes (x # xs)
  apply (induction xs rule: changes.induct)
  using assms by (auto simp: mult-less-0-iff zero-less-mult-iff)

lemma changes-scale:
  fixes xs::('a::linordered-idom) list
  assumes hs: ⋀i. i < n ⟹ s i > 0 and hn: length xs ≤ n
  shows changes [s i * (nth-default 0 xs i). i ← [0..<n]] = changes xs
using assms
proof (induction xs arbitrary: s n rule: changes.induct)
  case 1
  show ?case by (auto simp: map-replicate-const changes-replicate-0)
next
  case (2 uu)
  show ?case
  proof (cases n)
    case 0
    then show ?thesis by force
  next
    case (Suc m)
    hence map (λi. s i * nth-default 0 [uu] i) [0..<n] = [s 0 * uu] @ replicate m 0
    proof (induction m arbitrary: n)
      case (Suc m n)
      from Suc have map (λi. s i * nth-default 0 [uu] i) [0..<Suc m] =
                    [s 0 * uu] @ replicate m 0
        by meson
      hence map (λi. s i * nth-default 0 [uu] i) [0..<n] =
            [s 0 * uu] @ replicate m 0 @ [0]
        using Suc by auto
      also have ... = [s 0 * uu] @ replicate (Suc m) 0
        by (simp add: replicate-append-same)
      finally show ?case .
    qed fastforce
    then show ?thesis
      by (metis changes.simps(2) changes-append-replicate-0)
  qed
```

```
next
  case (3 a b xs s n)
  obtain m where hn: n = m + 2
    using 3(5)
    by (metis add-2-eq-Suc' diff-diff-cancel diff-le-self length-Suc-conv
        nat-arith.suc1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
  hence h:
    map (λi. s i * nth-default 0 (a # b # xs) i) [0..<n] =
      s 0 * a # s 1 * b # map (λi.
      (λ i. s (i+2)) i * nth-default 0 (xs) i) [0..<m]
    apply (induction m arbitrary: n)
    by auto
  consider (neg)a*b<0 | (nil)b=0 | (pos)¬a*b<0 ∧ ¬b=0 by linarith
  then show ?case
  proof (cases)
    case neg
    hence
      changes (map (λi. s i * nth-default 0 (a # b # xs) i) [0..<n]) =
        1 + changes (s 1 * b # map (λi. (λ i. s (i+2)) i
                                    * nth-default 0 (xs) i) [0..<m])
      apply (subst h)
      using 3(4)[of 0] 3(4)[of 1] hn
      by (metis (no-types, lifting) changes.simps(3) mult-less-0-iff pos2
          mult-pos-pos one-less-numeral-iff semiring-norm(76) trans-less-add2)
    also have
     ... = 1 + changes (map (λi. s (Suc i) * nth-default 0 (b # xs) i) [0..<Suc m])
      apply (rule arg-cong[of - - λ x. 1 + changes x])
      apply (induction m)
      by auto
    also have ... = changes (a # b # xs)
      apply (subst 3(1)[OF neg])
      using 3 neg hn by auto
    finally show ?thesis .
  next
    case nil
    hence changes (map (λi. s i * nth-default 0 (a # b # xs) i) [0..<n]) =
            changes (s 0 * a # map (λi. (λ i. s (i+2)) i * nth-default 0 (xs) i)
[0..<m])
      apply (subst h)
      using 3(4)[of 0] 3(4)[of 1] hn
      by auto
    also have
      ... = changes (map
              (λi. s (if i = 0 then 0 else Suc i) * nth-default 0 (a # xs) i)
             [0..<Suc m])
      apply (rule arg-cong[of - - λ x. changes x])
      apply (induction m)
      by auto
    also have ... = changes (a # b # xs)
```

```
    apply (subst 3(2))
    using 3 nil hn by auto
  finally show ?thesis .
 next
  case pos
  hence changes (map (λi. s i * nth-default 0 (a # b # xs) i) [0..<n]) =
      changes (s 1 * b # map (λi. (λ i. s (i+2)) i * nth-default 0 (xs) i) [0..<m])
    apply (subst h)
    using 3(4)[of 0] 3(4)[of 1] hn
    by (metis (no-types, lifting) changes.simps(3) divisors-zero
      mult-less-0-iff nat-1-add-1 not-square-less-zero one-less-numeral-iff
      semiring-norm(76) trans-less-add2 zero-less-mult-pos zero-less-two)
  also have
    ... = changes (map (λi. s (Suc i) * nth-default 0 (b # xs) i) [0..<Suc m])
    apply (rule arg-cong[of - - λ x. changes x])
    apply (induction m)
    by auto
  also have ... = changes (a # b # xs)
    apply (subst 3(3))
    using 3 pos hn by auto
  finally show ?thesis .
 qed
qed

lemma changes-scale-const: fixes xs::'a::linordered-idom list
 assumes hs: s ≠ 0
 shows changes (map ((*) s) xs) = changes xs
 apply (induction xs rule: changes.induct)
  apply (simp, force)
 using hs by (auto simp: mult-less-0-iff zero-less-mult-iff)

lemma changes-snoc: fixes xs::'a::linordered-idom list
 shows changes (xs @ [b, a]) = (if a * b < 0 then 1 + changes (xs @ [b])
      else if b = 0 then changes (xs @ [a]) else changes (xs @ [b]))
 apply (induction xs rule: changes.induct)
 subgoal by (force simp: mult-less-0-iff)
 subgoal by (force simp: mult-less-0-iff)
 subgoal by force
 done

lemma changes-rev: fixes xs:: 'a::linordered-idom list
 shows changes (rev xs) = changes xs
 apply (induction xs rule: changes.induct)
 by (auto simp: changes-snoc)

lemma changes-rev-about: fixes xs:: 'a::linordered-idom list
 shows changes (replicate (p − length xs) 0 @ rev xs) = changes xs
proof (induction p)
 case (Suc p)
```

```
  then show ?case
  proof cases
    assume ¬Suc p ≤ length xs
    hence Suc p − length xs = Suc (p − length xs) by linarith
    thus ?case using Suc.IH changes-rev by auto
  qed (auto simp: changes-rev)
qed (auto simp: changes-rev)

lemma changes-add-between:
  assumes a ≤ x and x ≤ b
  shows changes (as @ [a, b] @ bs) = changes (as @ [a, x, b] @ bs)
proof (induction as rule: changes.induct)
  case 1
  then show ?case using assms
    apply (induction bs)
    by (auto simp: mult-less-0-iff)
next
  case (2 c)
  then show ?case
    apply (induction bs)
    using assms by (auto simp: mult-less-0-iff)
next
  case (3 y z as)
  then show ?case
    using assms by (auto simp: mult-less-0-iff)
qed

lemma changes-all-nonneg: assumes ⋀i. nth-default 0 xs i ≥ 0 shows changes
xs = 0
  using assms
proof (induction xs rule: changes.induct)
  case (3 x1 x2 xs)
  moreover assume (⋀i. 0 ≤ nth-default 0 (x1 # x2 # xs) i)
  moreover hence (⋀i. 0 ≤ nth-default 0 (x1 # xs) i)
    and (⋀i. 0 ≤ nth-default 0 (x2 # xs) i)
    and x1 * x2 ≥ 0
  proof −
    fix i
    assume h:(⋀i. 0 ≤ nth-default 0 (x1 # x2 # xs) i)
    show 0 ≤ nth-default 0 (x1 # xs) i
    proof (cases i)
      case 0
      then show ?thesis using h[of 0] by force
    next
      case (Suc nat)
      then show ?thesis using h[of Suc (Suc nat)] by force
    qed
    show 0 ≤ nth-default 0 (x2 # xs) i using h[of Suc i] by simp
    show x1*x2 ≥ 0 using h[of 0] h[of 1] by simp
```

```
  qed
  ultimately show ?case by auto
qed auto

lemma changes-pCons: changes (coeffs (pCons 0 f)) = changes (coeffs f)
  by (auto simp: cCons-def)

lemma changes-increasing:
  assumes ⋀i. i < length xs − 1 ⟹ xs ! (i + 1) ≥ xs ! i
    and length xs > 1
    and hd xs < 0
    and last xs > 0
  shows changes xs = 1
  using assms
proof (induction xs rule:changes.induct)
  case (3 x y xs)
  consider (neg)x∗y<0 | (nil)y=0 | (pos)¬x∗y<0 ∧ ¬y=0 by linarith
  then show ?case
  proof cases
    case neg
    have changes (y # xs) = 0
    proof (rule changes-all-nonneg)
      fix i
      show 0 ≤ nth-default 0 (y # xs) i
      proof (cases i < length (y # xs))
        case True
        then show ?thesis using 3(4)[of i]
          apply (induction i)
          subgoal using 3(6) neg by (fastforce simp: mult-less-0-iff)
          subgoal using 3(4) by (auto simp: nth-default-def)
          done
      next
        case False
        then show ?thesis by (simp add: nth-default-def)
      qed
    qed
    thus changes (x # y # xs) = 1
      using neg by force
  next
    case nil
    hence xs ≠ [] using 3(7) by force
    have h: ⋀i. i < length (x # xs) − 1 ⟹ (x # xs) ! i ≤ (x # xs) ! (i + 1)
    proof −
      fix i assume i < length (x # xs) − 1
      thus (x # xs) ! i ≤ (x # xs) ! (i + 1)
        apply (cases i = 0)
        subgoal using 3(4)[of 0] 3(4)[of 1] ‹xs ≠ []› by force
        using 3(4)[of i+1] by simp
    qed
```

```
    have changes (x # xs) = 1
      apply (rule 3(2))
      using nil h ‹xs ≠ []› 3(6) 3(7) by auto
    thus ?thesis
      using nil by force
  next
    case pos
    hence xs ≠ [] using 3(6) 3(7) by (fastforce simp: mult-less-0-iff)
    have changes (y # xs) = 1
    proof (rule 3(3))
      show ¬ x * y < 0 y ≠ 0
        using pos by auto
      show ⋀i. i < length (y # xs) − 1
        ⟹ (y # xs) ! i ≤ (y # xs) ! (i + 1)
        using 3(4) by force
      show 1 < length (y # xs)
        using ‹xs ≠ []› by force
      show hd (y # xs) < 0
        using 3(6) pos by (force simp: mult-less-0-iff)
      show 0 < last (y # xs)
        using 3(7) by force
    qed
    thus ?thesis using pos by auto
  qed
qed auto

end
```

2 Bernstein Polynomials over the interval [0, 1]

```
theory Bernstein-01
  imports HOL−Computational-Algebra.Computational-Algebra
    Budan-Fourier.Budan-Fourier
    RRI-Misc
begin
```

The theorem of three circles is a statement about the Bernstein coefficients of a polynomial, the coefficients when a polynomial is expressed as a sum of Bernstein polynomials. These coefficients behave nicely under translations and rescaling and are the coefficients of a particular polynomial in the [0, 1] case. We shall define the [0, 1] case now and consider the general case later, deriving all the results by rescaling.

2.1 Definition and basic results

```
definition Bernstein-Poly-01 :: nat ⇒ nat ⇒ real poly where
  Bernstein-Poly-01 j p = (monom (p choose j) j)
                          * (monom 1 (p−j) ∘p [:1, −1:])
```

lemma *degree-Bernstein*:
assumes *hb*: $j \leq p$
shows *degree* (*Bernstein-Poly-01 j p*) = *p*
proof −
have *ha*: *monom* (*p choose j*) *j* ≠ (*0*::*real poly*) **using** *hb* **by** *force*
have *hb*: *monom 1* (*p*−*j*) ∘$_p$ [:*1*, −*1*:] ≠ (*0*::*real poly*)
proof
assume *monom 1* (*p*−*j*) ∘$_p$ [:*1*, −*1*:] = (*0*::*real poly*)
hence *lead-coeff* (*monom 1* (*p* − *j*) ∘$_p$ [:*1*, −*1*:]) = (*0*::*real*)
apply (*subst leading-coeff-0-iff*)
by *simp*
moreover have *lead-coeff* (*monom* (*1*::*real*) (*p* − *j*)
∘$_p$ [:*1*, −*1*:]) = (((− *1*) ^ (*p* − *j*))::*real*)
by (*subst lead-coeff-comp*, *auto simp*: *degree-monom-eq*)
ultimately show *False* **by** *auto*
qed
from *ha hb* **show** *?thesis*
by (*auto simp add*: *Bernstein-Poly-01-def degree-mult-eq*
degree-monom-eq degree-pcompose)
qed

lemma *coeff-gt*:
assumes *hb*: *j* > *p*
shows *Bernstein-Poly-01 j p* = *0*
by (*simp add*: *hb Bernstein-Poly-01-def*)

lemma *degree-Bernstein-le*: *degree* (*Bernstein-Poly-01 j p*) ≤ *p*
apply (*cases j* ≤ *p*)
by (*simp-all add*: *degree-Bernstein coeff-gt*)

lemma *poly-Bernstein-nonneg*:
assumes *x* ≥ *0* **and** *1* ≥ *x*
shows *poly* (*Bernstein-Poly-01 j p*) *x* ≥ *0*
using *assms* **by** (*simp add*: *poly-monom poly-pcompose Bernstein-Poly-01-def*)

lemma *Bernstein-symmetry*:
assumes *j* ≤ *p*
shows (*Bernstein-Poly-01 j p*) ∘$_p$ [:*1*, −*1*:] = *Bernstein-Poly-01* (*p*−*j*) *p*
proof −
have (*Bernstein-Poly-01 j p*) ∘$_p$ [:*1*, −*1*:]
= ((*monom* (*p choose j*) *j*) ∗ (*monom 1* (*p*−*j*) ∘$_p$ [:*1*, −*1*:])) ∘$_p$ [:*1*, −*1*:]
by (*simp add*: *Bernstein-Poly-01-def*)
also have ... = (*monom* (*p choose* (*p*−*j*)) *j* ∗
(*monom 1* (*p*−*j*) ∘$_p$ [:*1*, −*1*:])) ∘$_p$ [:*1*, −*1*:]
by (*fastforce simp*: *binomial-symmetric*[*OF assms*])
also have ... = *monom* (*p choose* (*p*−*j*)) *j* ∘$_p$ [:*1*, −*1*:] ∗
(*monom 1* (*p*−*j*)) ∘$_p$ ([:*1*, −*1*:] ∘$_p$ [:*1*, −*1*:])
by (*force simp*: *pcompose-mult pcompose-assoc*)

also have ... = (*monom* (*p choose* (*p*−*j*)) *j* ∘$_p$ [:*1*, −*1*:]) ∗ *monom 1* (*p*−*j*)
by (*force simp*: *pcompose-pCons*)
also have ... = *smult* (*p choose* (*p*−*j*)) (*monom 1 j* ∘$_p$ [:*1*, −*1*:])
∗ *monom 1* (*p*−*j*)
by (*simp add*: *assms smult-monom pcompose-smult*[*symmetric*])
also have ... = (*monom 1 j* ∘$_p$ [:*1*, −*1*:]) ∗ *monom* (*p choose* (*p*−*j*)) (*p*−*j*)
apply (*subst mult-smult-left*)
apply (*subst mult-smult-right*[*symmetric*])
apply (*subst smult-monom*)
by *force*
also have ... = *Bernstein-Poly-01* (*p*−*j*) *p* **using** *assms*
by (*auto simp*: *Bernstein-Poly-01-def*)
finally show *?thesis* **.**
qed

2.2 *Bernstein-Poly-01* and *reciprocal-poly*

lemma *Bernstein-reciprocal*:
reciprocal-poly p (*Bernstein-Poly-01 i p*)
= *smult* (*p choose i*) ([:−*1*, *1*:]^(*p*−*i*))
proof *cases*
assume *i* ≤ *p*
hence *reciprocal-poly p* (*Bernstein-Poly-01 i p*) =
reciprocal-poly (*degree* (*Bernstein-Poly-01 i p*)) (*Bernstein-Poly-01 i p*)
by (*auto simp*: *degree-Bernstein*)
also have ... = *reflect-poly* (*Bernstein-Poly-01 i p*)
by (*rule reciprocal-degree*)
also have ... = *smult* (*p choose i*) ([:−*1*, *1*:]^(*p*−*i*))
by (*auto simp*: *Bernstein-Poly-01-def reflect-poly-simps monom-altdef*
pcompose-pCons reflect-poly-pCons′ hom-distribs)
finally show *?thesis* **.**
next
assume *h*:¬ *i* ≤ *p*
hence *reciprocal-poly p* (*Bernstein-Poly-01 i p*) = (*0*::*real poly*)
by (*auto simp*: *coeff-gt reciprocal-poly-def*)
also have ... = *smult* (*p choose i*) ([:−*1*, *1*:]^(*p* − *i*)) **using** *h*
by *fastforce*
finally show *?thesis* **.**
qed

lemma *Bernstein-reciprocal-translate*:
reciprocal-poly p (*Bernstein-Poly-01 i p*) ∘$_p$ [:*1*, *1*:] =
monom (*p choose i*) (*p* − *i*)
by (*auto simp*: *Bernstein-reciprocal pcompose-smult pcompose-pCons monom-altdef hom-distribs*)

lemma *coeff-Bernstein-sum-01*: **fixes** *b*::*nat* ⇒ *real* **assumes** *hi*: *p* ≥ *i*
shows
coeff (*reciprocal-poly p*

```
        (∑ x = 0..p. smult (b x) (Bernstein-Poly-01 x p)) ∘_p [:1, 1:])
    (p − i) = (p choose i) ∗ (b i) (is ?L = ?R)
proof −
  define P where P ≡ (∑ x = 0..p. (smult (b x) (Bernstein-Poly-01 x p)))

  have ⋀x. degree (smult (b x) (Bernstein-Poly-01 x p)) ≤ p
  proof −
    fix x
    show degree (smult (b x) (Bernstein-Poly-01 x p)) ≤ p
      apply (cases x ≤ p)
      by (auto simp: degree-Bernstein coeff-gt)
  qed
  hence reciprocal-poly p P =
        (∑ x = 0..p. reciprocal-poly p (smult (b x) (Bernstein-Poly-01 x p)))
    apply (subst P-def)
    apply (rule reciprocal-sum)
    by presburger
  also have
    ... = (∑ x = 0..p. (smult (b x ∗ (p choose x)) ([:−1, 1:]^(p−x))))
  proof (rule sum.cong)
    fix x assume x ∈ {0..p}
    hence x ≤ p by simp
    thus reciprocal-poly p (smult (b x) (Bernstein-Poly-01 x p)) =
         smult ((b x) ∗ (p choose x)) ([:−1, 1:]^(p−x))
      by (auto simp add: reciprocal-smult degree-Bernstein Bernstein-reciprocal)
  qed (simp)
  finally have
    reciprocal-poly p P =
    (∑ x = 0..p. (smult ((b x) ∗ (p choose x)) ([:−1, 1:]^(p−x)))) .
  hence
    (reciprocal-poly p P) ∘_p [:1, 1:] =
    (∑ x = 0..p. (smult ((b x) ∗ (p choose x)) ([:−1, 1:]^(p−x))) ∘_p [:1, 1:])
    by (simp add: pcompose-sum pcompose-add)
  also have ... = (∑ x = 0..p. (monom ((b x) ∗ (p choose x)) (p − x)))
  proof (rule sum.cong)
    fix x assume x ∈ {0..p}
    hence x ≤ p by simp
    thus smult (b x ∗ (p choose x)) ([:− 1, 1:] ^ (p − x)) ∘_p [:1, 1:] =
         monom (b x ∗ (p choose x)) (p − x)
      by (simp add: hom-distribs pcompose-smult pcompose-pCons monom-altdef)
  qed (simp)
  finally have (reciprocal-poly p P) ∘_p [:1, 1:] =
              (∑ x = 0..p. (monom ((b x) ∗ (p choose x)) (p − x))) .
  hence ?L = (∑ x = 0..p. if p − x = p − i then b x ∗ real (p choose x) else 0)
    by (auto simp add: P-def coeff-sum)
  also have ... = (∑ x = 0..p. if x = i then b x ∗ real (p choose x) else 0)
  proof (rule sum.cong)
    fix x assume x ∈ {0..p}
    hence x ≤ p by simp
```

```
    thus (if p − x = p − i then b x ∗ real (p choose x) else 0) =
        (if x = i then b x ∗ real (p choose x) else 0) using hi
      by (auto simp add: leI)
  qed (simp)
  also have ... = ?R by simp
  finally show ?thesis .
qed

lemma Bernstein-sum-01: assumes hP: degree P ≤ p
  shows
  P = (∑ j = 0..p. smult
     (inverse (real (p choose j)) ∗
      coeff (reciprocal-poly p P ∘_p [:1, 1:]) (p−j))
   (Bernstein-Poly-01 j p))
proof −
  define Q where Q ≡ reciprocal-poly p P ∘_p [:1, 1:]
  from hP Q-def have hQ: degree Q ≤ p
    by (auto simp: degree-reciprocal degree-pcompose)
  have reciprocal-poly p (∑ j = 0..p.
       smult (inverse (real (p choose j)) ∗ coeff Q (p−j))
       (Bernstein-Poly-01 j p)) ∘_p [:1, 1:] = Q
  proof (rule poly-eqI)
    fix n
    show coeff (reciprocal-poly p (∑ j = 0..p.
         smult (inverse (real (p choose j)) ∗ coeff Q (p−j))
         (Bernstein-Poly-01 j p)) ∘_p [:1, 1:]) n = coeff Q n
      (is ?L = ?R)
    proof cases
      assume hn: n ≤ p
      hence ?L = coeff (reciprocal-poly p (∑ j = 0..p.
           smult (inverse (real (p choose j)) ∗ coeff Q (p−j))
           (Bernstein-Poly-01 j p)) ∘_p [:1, 1:]) (p − (p − n))
        by force
      also have ... = (p choose (p−n)) ∗
                      (inverse (real (p choose (p−n))) ∗
                       coeff Q (p−(p−n)))
        apply (subst coeff-Bernstein-sum-01)
        by auto
      also have ... = ?R using hn
        by fastforce
      finally show ?L = ?R .
    next
      assume hn: ¬ n ≤ p
      have degree (∑ j = 0..p.
           smult (inverse (real (p choose j)) ∗ coeff Q (p − j))
           (Bernstein-Poly-01 j p)) ≤ p
      proof (rule degree-sum-le)
        fix q assume q ∈ {0..p}
        hence q ≤ p by fastforce
```

```
      thus degree (smult (inverse (real (p choose q)) *
           coeff Q (p − q)) (Bernstein-Poly-01 q p)) ≤ p
        by (auto simp add: degree-Bernstein degree-smult-le)
    qed simp
    hence degree (reciprocal-poly p (∑ j = 0..p.
        smult (inverse (real (p choose j)) * coeff Q (p − j))
        (Bernstein-Poly-01 j p)) ∘p [:1, 1:]) ≤ p
      by (auto simp add: degree-pcompose degree-reciprocal)
    hence ?L = 0 using hn by (auto simp add: coeff-eq-0)
    thus ?L = ?R using hQ hn by (simp add: coeff-eq-0)
  qed
 qed
 hence reciprocal-poly p P ∘p [:1, 1:] =
      reciprocal-poly p (∑ j = 0..p.
      smult (inverse (real (p choose j)) *
      coeff (reciprocal-poly p P ∘p [:1, 1:]) (p−j))
      (Bernstein-Poly-01 j p)) ∘p [:1, 1:]
   by (auto simp: degree-reciprocal degree-pcompose Q-def)
 hence reciprocal-poly p P ∘p ([:1, 1:] ∘p [:−1, 1:]) =
      reciprocal-poly p (∑ j = 0..p. smult (inverse (real (p choose j)) *
      coeff (reciprocal-poly p P ∘p [:1, 1:]) (p−j))
      (Bernstein-Poly-01 j p)) ∘p ([:1, 1:] ∘p [:−1, 1:])
   by (auto simp: pcompose-assoc)
 hence reciprocal-poly p P = reciprocal-poly p (∑ j = 0..p.
      smult (inverse (real (p choose j)) *
      coeff (reciprocal-poly p P ∘p [:1, 1:]) (p−j)) (Bernstein-Poly-01 j p))
   by (auto simp: pcompose-pCons)
 hence reciprocal-poly p (reciprocal-poly p P) =
      reciprocal-poly p (reciprocal-poly p (∑ j = 0..p.
      smult (inverse (real (p choose j)) *
      coeff (reciprocal-poly p P ∘p [:1, 1:]) (p−j)) (Bernstein-Poly-01 j p)))
   by argo
 thus P = (∑ j = 0..p. smult (inverse (real (p choose j)) *
     coeff (reciprocal-poly p P ∘p [:1, 1:]) (p−j)) (Bernstein-Poly-01 j p))
   using hP by (auto simp: reciprocal-reciprocal degree-sum-le degree-smult-le
          degree-Bernstein degree-add-le)
qed

lemma Bernstein-Poly-01-span1:
 assumes hP: degree P ≤ p
 shows P ∈ poly-vs.span {Bernstein-Poly-01 x p | x. x ≤ p}
proof −
 have Bernstein-Poly-01 x p
      ∈ poly-vs.span {Bernstein-Poly-01 x p |x. x ≤ p}
   if x ∈ {0..p} for x
 proof −
   have ∃ n. Bernstein-Poly-01 x p = Bernstein-Poly-01 n p ∧ n ≤ p
     using that by force
   then show
```

Bernstein-Poly-01 x p ∈ poly-vs.span {Bernstein-Poly-01 n p |n. n ≤ p}
by (*simp add*: *poly-vs.span-base*)
qed
thus *?thesis*
apply (*subst Bernstein-sum-01*[*OF hP*])
apply (*rule poly-vs.span-sum*)
apply (*rule poly-vs.span-scale*)
by *blast*
qed

lemma *Bernstein-Poly-01-span*:
poly-vs.span {Bernstein-Poly-01 x p | x. x ≤ p}
= {x. degree x ≤ p}
apply (*subst monom-span*[*symmetric*])
apply (*subst poly-vs.span-eq*)
by (*auto simp*: *monom-span degree-Bernstein-le*
Bernstein-Poly-01-span1 degree-monom-eq)

2.3 Bernstein coefficients and changes

definition *Bernstein-coeffs-01* :: *nat* ⇒ *real poly* ⇒ *real list* **where**
Bernstein-coeffs-01 p P =
[(*inverse* (*real* (*p choose j*)) ∗
coeff (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]) (*p*−*j*)). *j* ← [*0*..<(*p*+*1*)]]

lemma *length-Bernstein-coeffs-01*: *length* (*Bernstein-coeffs-01 p P*) = *p* + *1*
by (*auto simp*: *Bernstein-coeffs-01-def*)

lemma *nth-default-Bernstein-coeffs-01*: **assumes** *degree P* ≤ *p*
shows *nth-default 0* (*Bernstein-coeffs-01 p P*) *i* =
inverse (*p choose i*) ∗ *coeff* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]) (*p*−*i*)
apply (*cases p* = *i*)
using *assms* **by** (*auto simp*: *Bernstein-coeffs-01-def nth-default-append*
nth-default-Cons Nitpick.case-nat-unfold binomial-eq-0)

lemma *Bernstein-coeffs-01-sum*: **assumes** *degree P* ≤ *p*
shows *P* = (∑ *j* = *0..p*. *smult* (*nth-default 0* (*Bernstein-coeffs-01 p P*) *j*)
(*Bernstein-Poly-01 j p*))
apply (*subst nth-default-Bernstein-coeffs-01*[*OF assms*])
apply (*subst Bernstein-sum-01*[*OF assms*])
by *argo*

definition *Bernstein-changes-01* :: *nat* ⇒ *real poly* ⇒ *int* **where**
Bernstein-changes-01 p P = *nat* (*changes* (*Bernstein-coeffs-01 p P*))

lemma *Bernstein-changes-01-def′*:
Bernstein-changes-01 p P = *nat* (*changes* [(*inverse* (*real* (*p choose j*)) ∗
coeff (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]) (*p*−*j*)). *j* ← [*0*..<*p* + *1*]])
by (*simp add*: *Bernstein-changes-01-def Bernstein-coeffs-01-def*)

```
lemma Bernstein-changes-01-eq-changes:
  assumes hP: degree P ≤ p
  shows Bernstein-changes-01 p P =
        changes (coeffs ((reciprocal-poly p P) ∘p [:1, 1:]))
proof (subst Bernstein-changes-01-def')
  have h:
    map (λj. inverse (real (p choose j)) *
     coeff (reciprocal-poly p P ∘p [:1, 1:]) (p − j)) [0..<p + 1] =
     map (λj. inverse (real (p choose j)) *
     nth-default 0 [nth-default 0 (coeffs (reciprocal-poly p P ∘p [:1, 1:]))
                   (p − j). j ← [0..<p + 1]] j) [0..<p + 1]
  proof (rule map-cong)
    fix x
    assume x ∈ set [0..<p+1]
    hence hx: x ≤ p by fastforce
    moreover have 1:
      length (map (λj. nth-default 0
       (coeffs (reciprocal-poly p P ∘p [:1, 1:])) (p − j)) [0..<p + 1]) ≤ Suc p
      by force
    moreover have length (coeffs (reciprocal-poly p P ∘p [:1, 1:])) ≤ Suc p
    proof (cases P=0)
      case False
      then have reciprocal-poly p P ∘p [:1, 1:] ≠ 0
        using hP by (simp add: Missing-Polynomial.pcompose-eq-0 reciprocal-0-iff)
      moreover have Suc (degree (reciprocal-poly p P ∘p [:1, 1:])) ≤ Suc p
        using hP by (auto simp: degree-pcompose degree-reciprocal)
      ultimately show ?thesis
        using length-coeffs-degree by force
    qed (auto simp: reciprocal-0)
    ultimately have h:
      nth-default 0 (map (λj. nth-default 0 (coeffs
       (reciprocal-poly p P ∘p [:1, 1:])) (p − j)) [0..<p + 1]) x =
       nth-default 0 (coeffs (reciprocal-poly p P ∘p [:1, 1:])) (p − x)
      (is ?L = ?R)
    proof −
      have ?L = (map (λj. nth-default 0 (coeffs
            (reciprocal-poly p P ∘p [:1, 1:])) (p − j)) [0..<p + 1]) ! x
        using hx by (auto simp: nth-default-nth)
      also have ... =  nth-default 0
          (coeffs (reciprocal-poly p P ∘p [:1, 1:])) (p − [0..<p + 1] ! x)
        apply (subst nth-map)
        using hx by auto
      also have ... = ?R
        apply (subst nth-upt)
        using hx by auto
      finally show ?thesis .
    qed
    show inverse (real (p choose x)) *
```

```
      coeff (reciprocal-poly p P ∘ₚ [:1, 1:]) (p − x) =
      inverse (real (p choose x)) ∗
      nth-default 0 (map (λj. nth-default 0
      (coeffs (reciprocal-poly p P ∘ₚ [:1, 1:])) (p − j)) [0..<p + 1]) x
    apply (subst h)
    apply (subst nth-default-coeffs-eq)
    by blast
qed auto

have 1:
  rev (map (λj. nth-default 0 (coeffs (reciprocal-poly p P ∘ₚ [:1, 1:])))
   (p − j)) [0..<p + 1]) = map (λj. nth-default 0 (coeffs
   (reciprocal-poly p P ∘ₚ [:1, 1:])) j) [0..<p + 1]
proof (subst rev-map, rule map-cong′)
  have ⋀q. (q ≥ p ⟶ rev [q−p..<q+1] = map ((−) q) [0..<p+1])
  proof (induction p)
    case 0
    then show ?case by simp
  next
    case (Suc p)
    have IH: ⋀q. (q ≥ p ⟶ rev [q−p..<q+1] = map ((−) q) [0..<p+1])
      using Suc.IH by blast
    show ?case
    proof
      assume hq: Suc p ≤ q
      then have h: rev [q − p..<q + 1] = map ((−) (q)) [0..<p + 1]
        apply (subst IH)
        using hq by auto
      have [q − Suc p..<q + 1] = (q − Suc p) # [q − p..<q + 1]
        by (simp add: Suc-diff-Suc Suc-le-lessD hq upt-conv-Cons)
      hence rev [q − Suc p..<q + 1] = rev [q − p..<q + 1] @ [q − Suc p]
        by force
      also have ... = map ((−) (q)) [0..<p + 1] @ [q − Suc p]
        using h by blast
      also have ... = map ((−) q) [0..<Suc p + 1]
        by force
      finally show rev [q − Suc p..<q + 1] = map ((−) q) [0..<Suc p + 1] .
    qed
  qed
  thus rev [0..<p + 1] = map ((−) p) [0..<p + 1]
    by force
next
  fix y
  assume y ∈ set [0..<p + 1]
  hence y ≤ p by fastforce
  thus nth-default 0 (coeffs (reciprocal-poly p P ∘ₚ [:1, 1:])) (p − (p − y)) =
       nth-default 0 (coeffs (reciprocal-poly p P ∘ₚ [:1, 1:])) y
    by fastforce
qed
```

```
have 2: ⋀ f. f ≠ 0 ⟶ degree f ≤ p ⟶
        map (nth-default 0 (coeffs f)) [0..<p + 1] =
        coeffs f @ replicate (p − degree f) 0
proof (induction p)
  case 0
  then show ?case by (auto simp: degree-0-iff)
next
  fix f
  case (Suc p)
  hence IH: (f ≠ 0 ⟶
             degree f ≤ p ⟶
             map (nth-default 0 (coeffs f)) [0..<p + 1] =
             coeffs f @ replicate (p − degree f) 0) by blast
  then show ?case
  proof (cases)
    assume h': Suc p = degree f
    hence h: [0..<Suc p + 1] = [0..<length (coeffs f)]
      by (metis add-is-0 degree-0 length-coeffs plus-1-eq-Suc zero-neq-one)
    thus ?thesis
      apply (subst h)
      apply (subst map-nth-default)
      using h' by fastforce
  next
    assume h': Suc p ≠ degree f
    show ?thesis
    proof
      assume hf: f ≠ 0
      show degree f ≤ Suc p ⟶
          map (nth-default 0 (coeffs f)) [0..<Suc p + 1] =
          coeffs f @ replicate (Suc p − degree f) 0
      proof
        assume degree f ≤ Suc p
        hence 1: degree f ≤ p using h' by fastforce
        hence 2: map (nth-default 0 (coeffs f)) [0..<p + 1] =
              coeffs f @ replicate (p − degree f) 0 using IH hf by blast
        have map (nth-default 0 (coeffs f)) [0..<Suc p + 1] =
             map (nth-default 0 (coeffs f)) [0..<p + 1] @
                [nth-default 0 (coeffs f) (Suc p)]
          by fastforce
        also have
          ... = coeffs f @ replicate (p − degree f) 0 @ [coeff f (Suc p)]
          using 2
          by (auto simp: nth-default-coeffs-eq)
        also have ... = coeffs f @ replicate (p − degree f) 0 @ [0]
          using ‹degree f ≤ Suc p› h' le-antisym le-degree by blast
        also have ... = coeffs f @ replicate (Suc p − degree f) 0 using 1
          by (simp add: Suc-diff-le replicate-app-Cons-same)
        finally show map (nth-default 0 (coeffs f)) [0..<Suc p + 1] =
```

```
            coeffs f @ replicate (Suc p − degree f) 0 .
      qed
    qed
  qed
qed

thus int (nat (changes (map (λj. inverse (real (p choose j)) *
      coeff (reciprocal-poly p P ∘p [:1, 1:]) (p − j)) [0..<p + 1]))) =
      changes (coeffs (reciprocal-poly p P ∘p [:1, 1:]))
proof cases
  assume hP: P = 0
  show int (nat (changes (map (λj. inverse (real (p choose j)) *
        coeff (reciprocal-poly p P ∘p [:1, 1:]) (p − j)) [0..<p + 1]))) =
        changes (coeffs (reciprocal-poly p P ∘p [:1, 1:])) (is ?L = ?R)
  proof −
    have ?L = int (nat (changes (map (λj. 0::real) [0..<p+1])))
      using hP by (auto simp: reciprocal-0 changes-nonneg)
    also have ... = 0
      apply (induction p)
      by (auto simp: map-replicate-trivial changes-nonneg
          replicate-app-Cons-same)
    also have 0 = changes ([]::real list) by simp
    also have ... = ?R using hP by (auto simp: reciprocal-0)
    finally show ?thesis .
  qed
next
  assume hP': P ≠ 0
  thus ?thesis
    apply (subst h)
    apply (subst changes-scale)
      apply auto[2]
    apply (subst changes-rev[symmetric])
    apply (subst 1)
    apply (subst 2)
    apply (simp add: pcompose-eq-0 hP reciprocal-0-iff)
    using assms apply (auto simp: degree-reciprocal)[1]
    by (auto simp: changes-append-replicate-0 changes-nonneg)
  qed
qed

lemma Bernstein-changes-01-test: fixes P::real poly
  assumes hP: degree P ≤ p and h0: P ≠ 0
  shows proots-count P {x. 0 < x ∧ x < 1} ≤ Bernstein-changes-01 p P ∧
      even (Bernstein-changes-01 p P − proots-count P {x. 0 < x ∧ x < 1})
proof −
  let ?Q = (reciprocal-poly p P) ∘p [:1, 1:]

  have 1: changes (coeffs ?Q) ≥ proots-count ?Q {x. 0 < x} ∧
      even (changes (coeffs ?Q) − proots-count ?Q {x. 0 < x})
```

```
    apply (rule descartes-sign)
    by (simp add: Missing-Polynomial.pcompose-eq-0 h0 hP reciprocal-0-iff)

  have ((+) (1::real) ‘ Collect ((<) (0::real))) = {x. (1::real)<x}
  proof
    show {x::real. 1 < x} ⊆ (+) 1 ‘ Collect ((<) 0)
    proof
      fix x::real assume x ∈ {x. 1 < x}
      hence 1 < x by simp
      hence −1 + x ∈ Collect ((<) 0) by auto
      hence 1 + (−1 + x) ∈ (+) 1 ‘ Collect ((<) 0) by blast
      thus x ∈ (+) 1 ‘ Collect ((<) 0) by argo
    qed
  qed auto
  hence 2: proots-count P {x. 0 < x ∧ x < 1} = proots-count ?Q {x. 0 < x}
    using assms
    by (auto simp: proots-pcompose reciprocal-0-iff proots-count-reciprocal')

  show ?thesis
    apply (subst Bernstein-changes-01-eq-changes[OF hP])
    apply (subst Bernstein-changes-01-eq-changes[OF hP])
    apply (subst 2)
    apply (subst 2)
    by (rule 1)
qed
```

2.4 Expression as a Bernstein sum

```
lemma Bernstein-coeffs-01-0: Bernstein-coeffs-01 p 0 = replicate (p+1) 0
  by (auto simp: Bernstein-coeffs-01-def reciprocal-0 map-replicate-trivial
      replicate-append-same)

lemma Bernstein-coeffs-01-1: Bernstein-coeffs-01 p 1 = replicate (p+1) 1
proof −
  have Bernstein-coeffs-01 p 1 =
     map (λj. inverse (real (p choose j)) ∗
     coeff (∑k≤p. smult (real (p choose k)) ([:0, 1:] ^ k)) (p − j)) [0..<(p+1)]
    by (auto simp: Bernstein-coeffs-01-def reciprocal-1 monom-altdef
        hom-distribs pcompose-pCons poly-0-coeff-0[symmetric] poly-binomial)
  also have ... = map (λj. inverse (real (p choose j)) ∗
           real (p choose (p − j))) [0..<(p+1)]
    by (auto simp: monom-altdef[symmetric] coeff-sum binomial)
  also have ... = map (λj. 1) [0..<(p+1)]
    apply (rule map-cong)
    subgoal by argo
    subgoal apply (subst binomial-symmetric)
      by auto
    done
  also have ... = replicate (p+1) 1
```

```
    by (auto simp: map-replicate-trivial replicate-append-same)
  finally show ?thesis .
qed

lemma Bernstein-coeffs-01-x: assumes p ≠ 0
  shows Bernstein-coeffs-01 p (monom 1 1) = [i/p. i ← [0..<(p+1)]]
proof −
  have
    Bernstein-coeffs-01 p (monom 1 1) = map (λj. inverse (real (p choose j)) *
     coeff (monom 1 (p − Suc 0) ∘p [:1, 1:]) (p − j)) [0..<(p+1)]
    using assms by (auto simp: Bernstein-coeffs-01-def reciprocal-monom)
  also have
    ... = map (λj. inverse (real (p choose j)) *
    (∑ k≤p − Suc 0. coeff (monom (real (p − 1 choose k)) k) (p − j))) [0..<(p+1)]
     by (auto simp: monom-altdef hom-distribs pcompose-pCons poly-binomial co-
eff-sum)
  also have... = map (λj. inverse (real (p choose j)) *
           real (p − 1 choose (p − j))) [0..<(p+1)]
    by auto
  also have ... = map (λj. j/p) [0..<(p+1)]
  proof (rule map-cong)
    fix x assume x ∈ set [0..<(p+1)]
    hence x ≤ p by force
    thus inverse (real (p choose x)) * real (p − 1 choose (p − x)) =
         real x / real p
    proof (cases x = 0)
      show x = 0 ⟹ ?thesis
        using assms by fastforce
      assume 1: x ≤ p and 2: x ≠ 0
      hence p − x ≤ p − 1 by force
      hence (p − 1 choose (p − x)) = (p − 1 choose (x − 1))
        apply (subst binomial-symmetric)
        using 1 2 by auto
      hence x * (p choose x) = p * (p − 1 choose (p − x))
        using 2 times-binomial-minus1-eq by simp
       hence real x * real (p choose x) = real p * real (p − 1 choose (p − x))
        by (metis of-nat-mult)
       thus ?thesis using 1 2
        by (auto simp: divide-simps)
    qed
  qed blast
  finally show ?thesis .
qed

lemma Bernstein-coeffs-01-add:
  assumes degree P ≤ p and degree Q ≤ p
  shows nth-default 0 (Bernstein-coeffs-01 p (P + Q)) i =
    nth-default 0 (Bernstein-coeffs-01 p P) i +
    nth-default 0 (Bernstein-coeffs-01 p Q) i
```

using *assms* **by** (*auto simp*: *nth-default-Bernstein-coeffs-01 degree-add-le reciprocal-add pcompose-add algebra-simps*)

lemma *Bernstein-coeffs-01-smult*:
assumes *degree* $P \leq p$
shows *nth-default* 0 (*Bernstein-coeffs-01* p (*smult* a P)) i = a $*$ *nth-default* 0 (*Bernstein-coeffs-01* p P) i
using *assms*
by (*auto simp*: *nth-default-Bernstein-coeffs-01 reciprocal-smult pcompose-smult*)

end

3 Bernstein Polynomials over any finite interval

theory *Bernstein*
imports *Bernstein-01*
begin

3.1 Definition and relation to Bernstein Polynomials over [0, 1]

definition *Bernstein-Poly* :: *nat* ⇒ *nat* ⇒ *real* ⇒ *real* ⇒ *real poly* **where**
Bernstein-Poly j p c d = *smult* ((*p choose j*)/(d − c)^p) (((*monom 1 j*) ∘$_p$ [:−c, 1:]) $*$ (*monom 1* (p−j) ∘$_p$ [:d, −1:]))

lemma *Bernstein-Poly-altdef*:
assumes $c \neq d$ **and** $j \leq p$
shows *Bernstein-Poly* j p c d = *smult* (*p choose j*) ([:−c/(d−c), 1/(d−c):]^j $*$ [:d/(d−c), −1/(d−c):]^(p−j))
(**is** *?L* = *?R*)
proof −
have *?L* = *smult* (*p choose j*) (*smult* ((1/(d − c))^j) (*smult* ((1/(d − c))^(p−j)) ([:−c, 1:]^j $*$ [:d, −1:]^(p−j))))
using *assms* **by** (*auto simp*: *Bernstein-Poly-def monom-altdef hom-distribs pcompose-pCons smult-eq-iff field-simps power-add*[*symmetric*])
also have ... = *?R*
apply (*subst mult-smult-right*[*symmetric*])
apply (*subst mult-smult-left*[*symmetric*])
apply (*subst smult-power*)
apply (*subst smult-power*)
by *auto*
finally show *?thesis* **.**
qed

lemma *Bernstein-Poly-nonneg*:
assumes $c \leq x$ **and** $x \leq d$
shows *poly* (*Bernstein-Poly* j p c d) $x \geq 0$
using *assms* **by** (*auto simp*: *Bernstein-Poly-def poly-pcompose poly-monom*)

lemma *Bernstein-Poly-01*: *Bernstein-Poly j p 0 1 = Bernstein-Poly-01 j p*
by (*auto simp*: *Bernstein-Poly-def Bernstein-Poly-01-def monom-altdef*)

lemma *Bernstein-Poly-rescale*:
assumes $a \neq b$
shows *Bernstein-Poly j p c d* $\circ_p$ [:*a*, *1*:] $\circ_p$ [:*0*, *b*−*a*:]
= *Bernstein-Poly j p* ((*c*−*a*)/(*b*−*a*)) ((*d*−*a*)/(*b*−*a*))
(**is** *?L* = *?R*)
proof −
have *?R* = *smult* (*real* (*p choose j*)
/ ((*d* − *a*) / (*b* − *a*) − (*c* − *a*) / (*b* − *a*)) ^ *p*)
([:− ((*c* − *a*) / (*b* − *a*)), *1*:] ^ *j*
∗ [:(*d* − *a*) / (*b* − *a*), − *1*:] ^ (*p* − *j*))
by (*auto simp*: *Bernstein-Poly-def monom-altdef hom-distribs pcompose-pCons*)
also have ... = *smult* (*real* (*p choose j*) / ((*d* − *c*) / (*b* − *a*)) ^ *p*)
([:− ((*c* − *a*) / (*b* − *a*)), *1*:] ^ *j* ∗ [:(*d* − *a*) / (*b* − *a*), − *1*:]
^ (*p* − *j*))
by *argo*
also have ... = *smult* (*real* (*p choose j*) / (*d* − *c*) ^ *p*)
(*smult* ((*b* − *a*) ^ (*p* − *j*)) (*smult* ((*b* − *a*) ^ *j*)
([:− ((*c* − *a*) / (*b* − *a*)), *1*:] ^ *j* ∗ [:(*d* − *a*) / (*b* − *a*), − *1*:]
^ (*p* − *j*))))
by (*auto simp*: *power-add*[*symmetric*] *power-divide*)
also have ... = *smult* (*real* (*p choose j*) / (*d* − *c*) ^ *p*)
([:− (*c* − *a*), *b* − *a*:] ^ *j* ∗ [:*d* − *a*, −(*b* − *a*):] ^ (*p* − *j*))
apply (*subst mult-smult-left*[*symmetric*])
apply (*subst mult-smult-right*[*symmetric*])
using *assms* **by** (*auto simp*: *smult-power*)
also have ... = *?L*
using *assms*
by (*auto simp*: *Bernstein-Poly-def monom-altdef pcompose-mult pcompose-smult hom-distribs pcompose-pCons*)
finally show *?thesis* **by** *presburger*
qed

lemma *Bernstein-Poly-rescale-01*:
assumes $c \neq d$
shows *Bernstein-Poly j p c d* $\circ_p$ [:*c*, *1*:] $\circ_p$ [:*0*, *d*−*c*:]
= *Bernstein-Poly-01 j p*
apply (*subst Bernstein-Poly-rescale*)
using *assms* **by** (*auto simp*: *Bernstein-Poly-01*)

lemma *Bernstein-Poly-eq-rescale-01*:
assumes $c \neq d$
shows *Bernstein-Poly j p c d* = *Bernstein-Poly-01 j p*
$\circ_p$ [:*0*, *1*/(*d*−*c*):] $\circ_p$ [:−*c*, *1*:]
apply (*subst Bernstein-Poly-rescale-01*[*symmetric*])

using *assms* **by** (*auto simp*: *pcompose-pCons pcompose-assoc*[*symmetric*])

lemma *coeff-Bernstein-sum*:
fixes *b*::*nat* ⇒ *real* **and** *p*::*nat* **and** *c d*::*real*
defines $P \equiv (\sum j = 0..p.\ (smult\ (b\ j)\ (Bernstein\text{-}Poly\ j\ p\ c\ d)))$
assumes $i \leq p$ **and** $c \neq d$
shows $coeff\ ((reciprocal\text{-}poly\ p\ (P \circ_p [:c,\ 1:]$
$\circ_p [:0,\ d-c:])) \circ_p [:1,\ 1:])\ (p - i) = (p\ choose\ i) * (b\ i)$
proof −
have *h*: $P \circ_p [:c,\ 1:] \circ_p [:0,\ d-c:]$
$= (\sum j = 0..p.\ (smult\ (b\ j)\ (Bernstein\text{-}Poly\text{-}01\ j\ p)))$
using *assms*
by (*auto simp*: *P-def pcompose-sum pcompose-smult*
pcompose-add Bernstein-Poly-rescale-01)
then show *?thesis*
using *coeff-Bernstein-sum-01 assms* **by** *simp*
qed

lemma *Bernstein-sum*:
assumes $c \neq d$ **and** $degree\ P \leq p$
shows $P = (\sum j = 0..p.\ smult\ (inverse\ (real\ (p\ choose\ j))$
$*\ coeff\ (reciprocal\text{-}poly\ p\ (P \circ_p [:c,\ 1:] \circ_p [:0,\ d-c:])$
$\circ_p [:1,\ 1:])\ (p-j))\ (Bernstein\text{-}Poly\ j\ p\ c\ d))$
apply (*subst Bernstein-Poly-eq-rescale-01*)
subgoal using *assms* **by** *blast*
subgoal
apply (*subst pcompose-smult*[*symmetric*])
apply (*subst pcompose-sum*[*symmetric*])
apply (*subst pcompose-smult*[*symmetric*])
apply (*subst pcompose-sum*[*symmetric*])
apply (*subst Bernstein-sum-01*[*symmetric*])
using *assms* **by** (*auto simp*: *degree-pcompose pcompose-assoc*[*symmetric*]
pcompose-pCons)
done

lemma *Bernstein-Poly-span1*:
assumes $c \neq d$ **and** $degree\ P \leq p$
shows $P \in poly\text{-}vs.span\ \{Bernstein\text{-}Poly\ x\ p\ c\ d \mid x.\ x \leq p\}$
proof (*subst Bernstein-sum*[*OF assms*], *rule poly-vs.span-sum*)
fix *x* :: *nat*
assume $x \in \{0..p\}$
then have $\exists n.\ Bernstein\text{-}Poly\ x\ p\ c\ d = Bernstein\text{-}Poly\ n\ p\ c\ d \wedge n \leq p$
by *auto*
then have
$Bernstein\text{-}Poly\ x\ p\ c\ d \in poly\text{-}vs.span\ \{Bernstein\text{-}Poly\ n\ p\ c\ d \mid n.\ n \leq p\}$
by (*simp add*: *poly-vs.span-base*)
thus $smult\ (inverse\ (real\ (p\ choose\ x)) *$
$coeff\ (reciprocal\text{-}poly\ p\ (P \circ_p [:c,\ 1:] \circ_p [:0,\ d - c:]) \circ_p [:1,\ 1:])$
$(p - x))\ (Bernstein\text{-}Poly\ x\ p\ c\ d)$

```
      ∈ poly-vs.span {Bernstein-Poly x p c d |x. x ≤ p}
   by (rule poly-vs.span-scale)
qed

lemma Bernstein-Poly-span:
  assumes c ≠ d
  shows poly-vs.span {Bernstein-Poly x p c d | x. x ≤ p} = {x. degree x ≤ p}
proof (subst Bernstein-Poly-01-span[symmetric], subst poly-vs.span-eq, rule conjI)
  show {Bernstein-Poly x p c d |x. x ≤ p}
     ⊆ poly-vs.span {Bernstein-Poly-01 x p |x. x ≤ p}
    apply (subst Setcompr-subset)
    apply (rule allI, rule impI)
    apply (rule Bernstein-Poly-01-span1)
    using assms by (auto simp: degree-Bernstein-le Bernstein-Poly-eq-rescale-01
                  degree-pcompose)

  show {Bernstein-Poly-01 x p |x. x ≤ p}
     ⊆ poly-vs.span {Bernstein-Poly x p c d |x. x ≤ p}
    apply (subst Setcompr-subset)
    apply (rule allI, rule impI)
    apply (rule Bernstein-Poly-span1)
    using assms by (auto simp: degree-Bernstein-le)
qed

lemma Bernstein-Poly-independent: assumes c ≠ d
  shows poly-vs.independent {Bernstein-Poly x p c d | x. x ∈ {..p}}
proof (rule poly-vs.card-le-dim-spanning)
  show {Bernstein-Poly x p c d |x. x ∈ {.. p}} ⊆ {x. degree x ≤ p}
    using assms
    by (auto simp: degree-Bernstein Bernstein-Poly-eq-rescale-01 degree-pcompose)
  show {x. degree x ≤ p} ⊆ poly-vs.span {Bernstein-Poly x p c d |x. x ∈ {..p}}
    using assms by (auto simp: Bernstein-Poly-span1)
  show finite {Bernstein-Poly x p c d |x. x ∈ {..p}} by fastforce
  show card {Bernstein-Poly x p c d |x. x ∈ {..p}} ≤ poly-vs.dim {x. degree x ≤
p}
    apply (rule le-trans)
     apply (subst image-Collect[symmetric], rule card-image-le, force)
    by (force simp: dim-degree)
qed
```

3.2 Bernstein coefficients and changes over any interval

```
definition Bernstein-coeffs ::
  nat ⇒ real ⇒ real ⇒ real poly ⇒ real list where
  Bernstein-coeffs p c d P =
    [(inverse (real (p choose j)) *
      coeff (reciprocal-poly p (P ∘p [:c, 1:] ∘p [:0, d−c:]) ∘p [:1, 1:]) (p−j)).
     j ← [0..<(p+1)]]
```

lemma *Bernstein-coeffs-eq-rescale*: **assumes** $c \neq d$
shows *Bernstein-coeffs p c d P* = *Bernstein-coeffs-01 p* (*P* $\circ_p$ [:*c*, *1*:] $\circ_p$ [:*0*, *d*−*c*:])
using *assms* **by** (*auto simp*: *pcompose-pCons pcompose-assoc*[*symmetric*]
Bernstein-coeffs-def Bernstein-coeffs-01-def)

lemma *nth-default-Bernstein-coeffs*: **assumes** *degree P* ≤ *p*
shows *nth-default 0* (*Bernstein-coeffs p c d P*) *i* =
inverse (*p choose i*) ∗ *coeff*
(*reciprocal-poly p* (*P* $\circ_p$ [:*c*, *1*:] $\circ_p$ [:*0*, *d*−*c*:]) $\circ_p$ [:*1*, *1*:]) (*p*−*i*)
apply (*cases p* = *i*)
using *assms* **by** (*auto simp*: *Bernstein-coeffs-def nth-default-append*
nth-default-Cons Nitpick.case-nat-unfold binomial-eq-0)

lemma *Bernstein-coeffs-sum*: **assumes** $c \neq d$ **and** *hP*: *degree P* ≤ *p*
shows *P* = (∑ *j* = *0*..*p*. *smult* (*nth-default 0* (*Bernstein-coeffs p c d P*) *j*)
(*Bernstein-Poly j p c d*))
apply (*subst nth-default-Bernstein-coeffs*[*OF hP*])
apply (*subst Bernstein-sum*[*OF assms*])
by *argo*

definition *Bernstein-changes* :: *nat* ⇒ *real* ⇒ *real* ⇒ *real poly* ⇒ *int* **where**
Bernstein-changes p c d P = *nat* (*changes* (*Bernstein-coeffs p c d P*))

lemma *Bernstein-changes-eq-rescale*: **assumes** $c \neq d$ **and** *degree P* ≤ *p*
shows *Bernstein-changes p c d P* =
Bernstein-changes-01 p (*P* $\circ_p$ [:*c*, *1*:] $\circ_p$ [:*0*, *d*−*c*:])
using *assms* **by** (*auto simp*: *Bernstein-coeffs-eq-rescale Bernstein-changes-def*
Bernstein-changes-01-def)

This is related and mostly equivalent to previous Descartes test [3]

lemma *Bernstein-changes-test*:
fixes *P*::*real poly*
assumes *degree P* ≤ *p* **and** *P* ≠ *0* **and** *c* < *d*
shows *proots-count P* {*x*. *c* < *x* ∧ *x* < *d*} ≤ *Bernstein-changes p c d P* ∧
even (*Bernstein-changes p c d P* − *proots-count P* {*x*. *c* < *x* ∧ *x* < *d*})
proof −
define *Q* **where** *Q*=*P* $\circ_p$ [:*c*, *1*:] $\circ_p$ [:*0*, *d* − *c*:]

have *int* (*proots-count Q* {*x*. *0* < *x* ∧ *x* < *1*})
≤ *Bernstein-changes-01 p Q* ∧
even (*Bernstein-changes-01 p Q* −
int (*proots-count Q* {*x*. *0* < *x* ∧ *x* < *1*}))
unfolding *Q-def*
apply (*rule Bernstein-changes-01-test*)
subgoal using *assms* **by** *fastforce*
subgoal using *assms* **by** (*auto simp*: *pcompose-eq-0*)
done
moreover have *proots-count P* {*x*. *c* < *x* ∧ *x* < *d*} =

```
        proots-count Q {x. 0 < x ∧ x < 1}
    unfolding Q-def
  proof (subst proots-pcompose)
    have poly [:c, 1:] ` poly [:0, d − c:] ` {x. 0 < x ∧ x < 1} =
       {x. c < x ∧ x < d} (is ?L = ?R)
    proof
      have c + x * (d − c) < d if x < 1 for x
      proof −
        have x * (d − c) < 1 * (d − c)
          using ‹c < d› that by force
        then show ?thesis by fastforce
      qed
      then show ?L ⊆ ?R
        using assms by auto
    next
      show ?R ⊆ ?L
      proof
        fix x::real assume x ∈ ?R
        hence c < x and x < d by auto
        thus x ∈ ?L
        proof (subst image-eqI)
          show x = poly [:c, 1:] (x − c) by force
          assume c < x and x < d
          thus x − c ∈ poly [:0, d − c:] ` {x. 0 < x ∧ x < 1}
          proof (subst image-eqI)
            show x − c = poly [:0, d − c:] ((x − c)/(d − c))
              using assms by fastforce
            assume c < x and x < d
            thus (x − c) / (d − c) ∈ {x. 0 < x ∧ x < 1}
              by auto
          qed fast
        qed fast
      qed
    qed
    then show proots-count P {x. c < x ∧ x < d} =
       proots-count (P ∘p [:c, 1:])
       (poly [:0, d − c:] ` {x. 0 < x ∧ x < 1})
      using assms by (auto simp:proots-pcompose)
    show P ∘p [:c, 1:] ≠ 0
      by (simp add: pcompose-eq-0 assms(2))
    show degree [:0, d − c:] = 1
      using assms by auto
  qed
  moreover have Bernstein-changes p c d P = Bernstein-changes-01 p Q
    unfolding Q-def
    apply (rule Bernstein-changes-eq-rescale)
    using assms by auto
  ultimately show ?thesis by auto
qed
```

3.3 The control polygon of a polynomial

definition *control-points* ::
  nat ⇒ *real* ⇒ *real* ⇒ *real poly* ⇒ (*real* × *real*) *list*
where
  control-points p c d P =
   [(((*real i*)∗*d* + (*real* (*p* − *i*))∗*c*)/*p*,
     nth-default 0 (*Bernstein-coeffs p c d P*) *i*).
     i ← [*0*..<(*p*+*1*)]]

lemma *line-above*:
  fixes *a b c d* :: *real* **and** *p* :: *nat* **and** *P* :: *real poly*
  assumes *hline*: ⋀*i*. *i* ≤ *p* ⟹ *a* ∗ (((*real i*)∗*d* + (*real* (*p* − *i*))∗*c*)/*p*) + *b* ≥
              nth-default 0 (*Bernstein-coeffs p c d P*) *i*
  and *hp*: *p* ≠ *0* **and** *hcd*: *c* ≠ *d* **and** *hP*: *degree P* ≤ *p*
  shows ⋀*x*. *c* ≤ *x* ⟹ *x* ≤ *d* ⟹ *a*∗*x* + *b* ≥ *poly P x*
proof −
  fix *x*
  assume *hc*: *c* ≤ *x* **and** *hd*: *x* ≤ *d*

  have *bern-eq*:*Bernstein-coeffs p c d* [:*b*, *a*:] =
        [*a*∗(*real i* ∗ *d* + *real* (*p* − *i*) ∗ *c*)/*p* + *b*. *i* ← [*0*..<(*p*+*1*)]]
  proof −
    have *Bernstein-coeffs p c d* [:*b*, *a*:] = *map* (*nth-default 0*
        (*Bernstein-coeffs-01 p* ([:*b*, *a*:] ∘*p* [:*c*, *1*:] ∘*p* [:*0*, *d* − *c*:])))
       [*0*..<*p*+*1*]
      apply (*subst Bernstein-coeffs-eq-rescale*[*OF hcd*])
      apply (*subst map-nth-default*[*symmetric*])
      apply (*subst length-Bernstein-coeffs-01*)
      by *blast*
    also have
      ... = *map* (λ*i*. *a* ∗ (*real i* ∗ *d* + *real* (*p* − *i*) ∗ *c*) / *real p* + *b*) [*0*..<*p* + *1*]
    proof (*rule map-cong*)
      fix *x* **assume** *hx*: *x* ∈ *set* [*0*..<*p* + *1*]
      have *nth-default 0* (*Bernstein-coeffs-01 p*
          ([:*b*, *a*:] ∘*p* [:*c*, *1*:] ∘*p* [:*0*, *d* − *c*:])) *x* =
          nth-default 0 (*Bernstein-coeffs-01 p*
          (*smult* (*b* + *a*∗*c*) *1* + *smult* (*a*∗(*d* − *c*)) (*monom 1 1*))) *x*
      proof−
        have [:*b*, *a*:] ∘*p* [:*c*, *1*:] ∘*p* [:*0*, *d* − *c*:] =
              smult (*b* + *a*∗*c*) *1* + *smult* (*a*∗(*d* − *c*)) (*monom 1 1*)
          by (*simp add*: *monom-altdef pcompose-pCons*)
        then show *?thesis* **by** *auto*
      qed
      also have ... =
         nth-default 0 (*Bernstein-coeffs-01 p* (*smult* (*b* + *a* ∗ *c*) *1*)) *x* +
         nth-default 0 (*Bernstein-coeffs-01 p* (*smult* (*a* ∗ (*d* − *c*)) (*monom 1 1*))) *x*
        apply (*subst Bernstein-coeffs-01-add*)
        using *hp* **by** (*auto simp*: *degree-monom-eq*)
      also have ... =

```
        (b + a*c) * nth-default 0 (Bernstein-coeffs-01 p 1) x +
        (a*(d − c)) * nth-default 0 (Bernstein-coeffs-01 p (monom 1 1)) x
    apply (subst Bernstein-coeffs-01-smult)
    using hp by (auto simp: Bernstein-coeffs-01-smult degree-monom-eq)
  also have ... =
      (b + a * c) * (if x < p + 1 then 1 else 0) +
       a * (d − c) * (real (nth-default 0 [0..<p + 1] x) / real p)
    apply (subst Bernstein-coeffs-01-1, subst Bernstein-coeffs-01-x[OF hp])
    apply (subst nth-default-replicate-eq, subst nth-default-map-eq[of - 0])
    by auto
  also have ... =
        (b + a * c) * (if x < p + 1 then 1 else 0) +
        a * (d − c) * (real ([0..<p + 1] ! x) / real p)
    apply (subst nth-default-nth)
    using hx by auto
  also have ... = (b + a * c) * (if x < p + 1 then 1 else 0) +
        a * (d − c) * (real (0 + x) / real p)
    apply (subst nth-upt)
    using hx by auto
  also have ... = a * (real x * d + real (p − x) * c) / real p + b
    apply (subst of-nat-diff)
    using hx hp by (auto simp: field-simps)
  finally show nth-default 0 (Bernstein-coeffs-01 p
                ([:b, a:] ∘p [:c, 1:] ∘p [:0, d − c:])) x =
                a * (real x * d + real (p − x) * c) / real p + b .
 qed ?thesis
 finally show ?thesis .
qed

have nth-default-geq:nth-default 0 (Bernstein-coeffs p c d [:b, a:]) i ≥
       nth-default 0 (Bernstein-coeffs p c d P) i for i
proof −
  show nth-default 0 (Bernstein-coeffs p c d [:b, a:]) i ≥
       nth-default 0 (Bernstein-coeffs p c d P) i
  proof cases
    define p1 where p1 ≡ p+1
    assume h: i ≤ p
    hence nth-default 0 (Bernstein-coeffs p c d P) i ≤
          a * (((real i)*d + (real (p − i))*c)/p) + b
      by (rule hline)
    also have ... = nth-default 0 (map (λi. a * (real i * d
      + real (p − i) * c) / real p + b) [0..<p + 1]) i
      apply (subst p1-def[symmetric])
      using h apply (auto simp: nth-default-def)
      by (auto simp: p1-def)
    also have ... = nth-default 0 (Bernstein-coeffs p c d [:b, a:]) i
      using bern-eq by simp
    finally show ?thesis .
  next
```

```
    assume h: ¬i ≤ p
    thus ?thesis
      using assms
      by (auto simp: nth-default-def Bernstein-coeffs-eq-rescale
                     length-Bernstein-coeffs-01)
  qed
qed

have poly P x = (∑ k = 0..p.
      poly (smult (nth-default 0 (Bernstein-coeffs p c d P) k)
      (Bernstein-Poly k p c d)) x)
  apply (subst Bernstein-coeffs-sum[OF hcd hP])
  by (rule poly-sum)
also have ... ≤ (∑ k = 0..p.
    poly (smult (nth-default 0 (Bernstein-coeffs p c d [:b, a:]) k)
      (Bernstein-Poly k p c d)) x)
  apply (rule sum-mono)
  using mult-right-mono[OF nth-default-geq] Bernstein-Poly-nonneg[OF hc hd]
  by auto
also have ... = poly [:b, a:] x
  apply (subst(2) Bernstein-coeffs-sum[of c d [:b, a:] p])
  using assms apply auto[2]
  by (rule poly-sum[symmetric])
also have ... = a*x + b by force
finally show poly P x ≤ a*x + b .
qed

end
```

4 Normal Polynomials

```
theory Normal-Poly
  imports RRI-Misc
begin
```

Here we define normal polynomials as defined in Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer Berlin Heidelberg, Berlin, Heidelberg (2016).

```
definition normal-poly :: ('a::{comm-ring-1,ord}) poly ⇒ bool where
normal-poly p ≡
  (p ≠ 0) ∧
  (∀ i. 0 ≤ coeff p i) ∧
  (∀ i. coeff p i * coeff p (i+2) ≤ (coeff p (i+1))^2) ∧
  (∀ i j k. i ≤ j ⟶ j ≤ k ⟶ 0 < coeff p i
      ⟶ 0 < coeff p k ⟶ 0 < coeff p j)

lemma normal-non-zero: normal-poly p ⟹ p ≠ 0
  using normal-poly-def by blast
```

lemma *normal-coeff-nonneg*: *normal-poly p* $\Longrightarrow$ *0* $\leq$ *coeff p i*
using *normal-poly-def* **by** *metis*

lemma *normal-poly-coeff-mult*:
normal-poly p $\Longrightarrow$ *coeff p i* $*$ *coeff p (i+2)* $\leq$ (*coeff p (i+1)*)^2
using *normal-poly-def* **by** *blast*

lemma *normal-poly-pos-interval*:
normal-poly p $\Longrightarrow$ *i* $\leq$ *j* $\Longrightarrow$ *j* $\leq$ *k* $\Longrightarrow$ *0* $<$ *coeff p i* $\Longrightarrow$ *0* $<$ *coeff p k*
$\Longrightarrow$ *0* $<$ *coeff p j*
using *normal-poly-def* **by** *blast*

lemma *normal-polyI*:
assumes (*p* $\neq$ *0*)
and ($\bigwedge$ *i*. *0* $\leq$ *coeff p i*)
and ($\bigwedge$ *i*. *coeff p i* $*$ *coeff p (i+2)* $\leq$ (*coeff p (i+1)*)^2)
and ($\bigwedge$ *i j k*. *i* $\leq$ *j* $\Longrightarrow$ *j* $\leq$ *k* $\Longrightarrow$ *0* $<$ *coeff p i* $\Longrightarrow$ *0* $<$ *coeff p k* $\Longrightarrow$ *0* $<$ *coeff p j*)
shows *normal-poly p*
using *assms* **by** (*force simp*: *normal-poly-def*)

lemma *linear-normal-iff*:
fixes *x*::*real*
shows *normal-poly* [:$-x$, *1*:] $\longleftrightarrow$ *x* $\leq$ *0*
proof
assume *normal-poly* [:$-x$, *1*:]
thus *x* $\leq$ *0* **using** *normal-coeff-nonneg*[*of* [:$-x$, *1*:] *0*] **by** *auto*
next
assume *x* $\leq$ *0*
then have *0* $\leq$ *coeff* [:$-$ *x*, *1*:] *i* **for** *i*
by (*cases i*) (*simp-all add*: *pCons-one*)
moreover have *0* $<$ *coeff* [:$-$ *x*, *1*:] *j*
if *i* $\leq$ *j j* $\leq$ *k 0* $<$ *coeff* [:$-$ *x*, *1*:] *i*
0 $<$ *coeff* [:$-$ *x*, *1*:] *k* **for** *i j k*
apply (*cases k=0* $\vee$ *i=0*)
subgoal using *that*
by (*smt* (*z3*) *bot-nat-0.extremum-uniqueI degree-pCons-eq-if*
le-antisym le-degree not-less-eq-eq)
subgoal using *that*
by (*smt* (*z3*) *One-nat-def degree-pCons-eq-if le-degree less-one*
not-le one-neq-zero pCons-one verit-la-disequality)
done
ultimately show *normal-poly* [:$-x$, *1*:]
unfolding *normal-poly-def* **by** *auto*
qed

lemma *quadratic-normal-iff*:
fixes *z*::*complex*

```
  shows normal-poly [:(cmod z)^2, -2*Re z, 1:]
        ⟷ Re z ≤ 0 ∧ 4*(Re z)^2 ≥ (cmod z)^2
proof
  assume normal-poly [:(cmod z)^2, - 2 * Re z, 1:]
  hence -2*Re z ≥ 0 ∧ (cmod z)^2 ≥ 0 ∧ (-2*Re z)^2 ≥ (cmod z)^2
    using normal-coeff-nonneg[of - 1] normal-poly-coeff-mult[of - 0]
    by fastforce
  thus Re z ≤ 0 ∧ 4*(Re z)^2 ≥ (cmod z)^2
    by auto
next
  assume asm:Re z ≤ 0 ∧ 4*(Re z)^2 ≥ (cmod z)^2
  define P where P=[:(cmod z)^2, - 2 * Re z, 1:]

  have 0 ≤ coeff P i for i
    unfolding P-def using asm
    apply (cases i=0∨i=1∨i=2)
    by (auto simp:numeral-2-eq-2 coeff-eq-0)
  moreover have coeff P i * coeff P (i + 2) ≤ (coeff P (i + 1))^2 for i
    apply (cases i=0∨i=1∨i=2)
    using asm
    unfolding P-def by (auto simp:coeff-eq-0)
  moreover have 0 < coeff P j
    if 0 < coeff P k 0 < coeff P i j ≤ k i ≤ j
    for i j k
    using that unfolding P-def
    apply (cases k=0 ∨ k=1 ∨ k=2)
    subgoal using asm
      by (smt (z3) One-nat-def Suc-1 bot-nat-0.extremum-uniqueI
          coeff-pCons-0 coeff-pCons-Suc le-Suc-eq
          zero-less-power2)
    subgoal by (auto simp:coeff-eq-0)
    done
  moreover have P≠0 unfolding P-def by auto
  ultimately show normal-poly P
    unfolding normal-poly-def by blast
qed

lemma normal-of-no-zero-root:
  fixes f::real poly
  assumes hzero: poly f 0 ≠ 0 and hdeg: i ≤ degree f
    and hnorm: normal-poly f
  shows 0 < coeff f i
proof −
  have coeff f 0 > 0 using hzero normal-coeff-nonneg[OF hnorm]
    by (metis eq-iff not-le-imp-less poly-0-coeff-0)
  moreover have coeff f (degree f) > 0 using normal-coeff-nonneg[OF hnorm]
normal-non-zero[OF hnorm]
    by (meson dual-order.irrefl eq-iff eq-zero-or-degree-less not-le-imp-less)
  moreover have 0 ≤ i by simp
```

ultimately show *0 < coeff f i* **using** *hdeg normal-poly-pos-interval*[*OF hnorm*]
by *blast*
qed

lemma *normal-divide-x*:
fixes *f*::*real poly*
assumes *hnorm*: *normal-poly* (*f*∗[:*0*,*1*:])
shows *normal-poly f*
proof (*rule normal-polyI*)
show *f* ≠ *0*
using *normal-non-zero*[*OF hnorm*] **by** *auto*
next
fix *i*
show *0* ≤ *coeff f i*
using *normal-coeff-nonneg*[*OF hnorm*, *of Suc i*] **by** (*simp add*: *coeff-pCons*)
next
fix *i*
show *coeff f i* ∗ *coeff f* (*i* + *2*) ≤ (*coeff f* (*i* + *1*))^2
using *normal-poly-coeff-mult*[*OF hnorm*, *of Suc i*] **by** (*simp add*: *coeff-pCons*)
next
fix *i j k*
show *i* ≤ *j* ⟹ *j* ≤ *k* ⟹ *0* < *coeff f i* ⟹ *0* < *coeff f k* ⟹ *0* < *coeff f j*
using *normal-poly-pos-interval*[*of - Suc i Suc j Suc k*, *OF hnorm*]
by (*simp add*: *coeff-pCons*)
qed

lemma *normal-mult-x*:
fixes *f*::*real poly*
assumes *hnorm*: *normal-poly f*
shows *normal-poly* (*f* ∗ [:*0*, *1*:])
proof (*rule normal-polyI*)
show *f* ∗ [:*0*, *1*:] ≠ *0*
using *normal-non-zero*[*OF hnorm*] **by** *auto*
next
fix *i*
show *0* ≤ *coeff* (*f* ∗ [:*0*, *1*:]) *i*
using *normal-coeff-nonneg*[*OF hnorm*, *of i−1*] **by** (*cases i*, *auto simp*: *coeff-pCons*)
next
fix *i*
show *coeff* (*f* ∗ [:*0*, *1*:]) *i* ∗ *coeff* (*f* ∗ [:*0*, *1*:]) (*i* + *2*) ≤ (*coeff* (*f* ∗ [:*0*, *1*:]) (*i* + *1*))^2
using *normal-poly-coeff-mult*[*OF hnorm*, *of i−1*] **by** (*cases i*, *auto simp*: *coeff-pCons*)
next
fix *i j k*
show *i* ≤ *j* ⟹ *j* ≤ *k* ⟹ *0* < *coeff* (*f* ∗ [:*0*, *1*:]) *i* ⟹ *0* < *coeff* (*f* ∗ [:*0*, *1*:]) *k* ⟹ *0* < *coeff* (*f* ∗ [:*0*, *1*:]) *j*
using *normal-poly-pos-interval*[*of - i−1 j−1 k−1*, *OF hnorm*]

```
    apply (cases i, force)
    apply (cases j, force)
    apply (cases k, force)
    by (auto simp: coeff-pCons)
qed

lemma normal-poly-general-coeff-mult:
  fixes f::real poly
  assumes normal-poly f and h ≤ j
  shows coeff f (h+1) * coeff f (j+1) ≥ coeff f h * coeff f (j+2)
using assms proof (induction j)
  case 0
  then show ?case
    using normal-poly-coeff-mult by (auto simp: power2-eq-square)[1]
next
  case (Suc j)
  then show ?case
  proof (cases h = Suc j)
    assume h = Suc j normal-poly f
    thus ?thesis
      using normal-poly-coeff-mult by (auto simp: power2-eq-square)
  next
    assume (normal-poly f ⟹
      h ≤ j ⟹ coeff f h * coeff f (j + 2) ≤ coeff f (h + 1) * coeff f (j + 1))
      normal-poly f and h: h ≤ Suc j h ≠ Suc j
    hence IH: coeff f h * coeff f (j + 2) ≤ coeff f (h + 1) * coeff f (j + 1)
      by linarith
    show ?thesis
    proof (cases coeff f (Suc j + 1) = 0, cases coeff f (Suc j + 2) = 0)
      show coeff f (Suc j + 1) = 0 ⟹ coeff f (Suc j + 2) = 0 ⟹
        coeff f h * coeff f (Suc j + 2) ≤ coeff f (h + 1) * coeff f (Suc j + 1)
        by (metis assms(1) mult-zero-right normal-coeff-nonneg)
    next
      assume 1: coeff f (Suc j + 1) = 0 coeff f (Suc j + 2) ≠ 0
      hence coeff f (Suc j + 2) > 0 ¬coeff f (Suc j + 1) > 0
        using normal-coeff-nonneg[of f Suc j + 2] assms(1) by auto
      hence coeff f h > 0 ⟹ False
        using normal-poly-pos-interval[of f h Suc j + 1 Suc j + 2] assms(1) h by
force
      hence coeff f h = 0
        using normal-coeff-nonneg[OF assms(1)] less-eq-real-def by auto
      thus coeff f h * coeff f (Suc j + 2) ≤ coeff f (h + 1) * coeff f (Suc j + 1)
        using 1 by fastforce
    next
      assume 1: coeff f (Suc j + 1) ≠ 0
      show coeff f h * coeff f (Suc j + 2) ≤ coeff f (h + 1) * coeff f (Suc j + 1)
      proof (cases coeff f (Suc j) = 0)
        assume 2: coeff f (Suc j) = 0
        hence coeff f (Suc j + 1) > 0 ¬coeff f (Suc j) > 0
```

```
        using normal-coeff-nonneg[of f Suc j + 1] assms(1) 1 by auto
      hence coeff f h > 0 ⟹ False
       using normal-poly-pos-interval[of f h Suc j Suc j + 1] assms(1) h by force
      hence coeff f h = 0
        using normal-coeff-nonneg[OF assms(1)] less-eq-real-def by auto
      thus coeff f h * coeff f (Suc j + 2) ≤ coeff f (h + 1) * coeff f (Suc j + 1)
        by (simp add: assms(1) normal-coeff-nonneg)
    next
      assume 2: coeff f (Suc j) ≠ 0
     from normal-poly-coeff-mult[OF assms(1), of Suc j] normal-coeff-nonneg[OF
assms(1), of Suc j]
        normal-coeff-nonneg[OF assms(1), of Suc (Suc j)] 1 2
      have 3: coeff f (Suc j + 1) / coeff f (Suc j) ≥ coeff f (Suc j + 2) / coeff f
(Suc j + 1)
        by (auto simp: power2-eq-square divide-simps algebra-simps)
      have (coeff f h * coeff f (j + 2)) * (coeff f (Suc j + 2) / coeff f (Suc j +
1)) ≤ (coeff f (h + 1) * coeff f (j + 1)) * (coeff f (Suc j + 1) / coeff f (Suc j))
        apply (rule mult-mono[OF IH])
        using 3 by (simp-all add: assms(1) normal-coeff-nonneg)
      thus coeff f h * coeff f (Suc j + 2) ≤ coeff f (h + 1) * coeff f (Suc j + 1)
        using 1 2 by fastforce
    qed
  qed
 qed
qed

lemma normal-mult:
 fixes f g::real poly
 assumes hf: normal-poly f and hg: normal-poly g
 defines df ≡ degree f and dg ≡ degree g
 shows normal-poly (f*g)
using df-def hf proof (induction df arbitrary: f)
```

We shall first show that without loss of generality we may assume *poly f 0* $\neq$ *0*, this is done by induction on the degree, if 0 is a root then we derive the result from *f*/[:*0*,*1*:].

```
 fix f::real poly fix i::nat
 assume 0 = degree f and hf: normal-poly f
 then obtain a where f = [:a:] using degree-0-iff by auto
 then show normal-poly (f*g)
  apply (subst normal-polyI)
  subgoal using normal-non-zero[OF hf] normal-non-zero[OF hg] by auto
  subgoal
   using normal-coeff-nonneg[of - 0, OF hf] normal-coeff-nonneg[OF hg]
   by simp
  subgoal
   using normal-coeff-nonneg[of - 0, OF hf] normal-poly-coeff-mult[OF hg]
   by (auto simp: algebra-simps power2-eq-square mult-left-mono)[1]
  subgoal
```

```
      using normal-non-zero[OF hf] normal-coeff-nonneg[of - 0, OF hf] nor-
mal-poly-pos-interval[OF hg]
    by (simp add: zero-less-mult-iff)
  subgoal by simp
  done
next
 case (Suc df)
 then show ?case
 proof (cases poly f 0 = 0)
  assume poly f 0 = 0 and hf:normal-poly f
  moreover then obtain f' where hdiv: f = f'*[:0,1:]
   by (smt (verit) dvdE mult.commute poly-eq-0-iff-dvd)
  ultimately have hf': normal-poly f' using normal-divide-x by blast
  assume Suc df = degree f
   hence degree f' = df using hdiv normal-non-zero[OF hf'] by (auto simp:
degree-mult-eq)
  moreover assume ⋀f. df = degree f ⟹ normal-poly f ⟹ normal-poly (f *
g)
  ultimately have normal-poly (f'*g) using hf' by blast
  thus normal-poly (f*g) using hdiv normal-mult-x by fastforce
 next
  assume hf: normal-poly f and hf0: poly f 0 ≠ 0
  define dg where dg ≡ degree g
  show normal-poly (f * g)
  using dg-def hg proof (induction dg arbitrary: g)
```

Similarly we may assume *poly g 0 ≠ 0*.

```
    fix g::real poly fix i::nat
    assume 0 = degree g and hg: normal-poly g
    then obtain a where g = [:a:] using degree-0-iff by auto
    then show normal-poly (f*g)
     apply (subst normal-polyI)
     subgoal
       using normal-non-zero[OF hg] normal-non-zero[OF hf] by auto
     subgoal
       using normal-coeff-nonneg[of - 0, OF hg] normal-coeff-nonneg[OF hf]
       by simp
     subgoal
       using normal-coeff-nonneg[of - 0, OF hg] normal-poly-coeff-mult[OF hf]
       by (auto simp: algebra-simps power2-eq-square mult-left-mono)
     subgoal
       using normal-non-zero[OF hf] normal-coeff-nonneg[of - 0, OF hg]
         normal-poly-pos-interval[OF hf]
       by (simp add: zero-less-mult-iff)
     by simp
  next
    case (Suc dg)
    then show ?case
    proof (cases poly g 0 = 0)
```

assume *poly g 0 = 0* **and** *hg*:*normal-poly g*
moreover then obtain g' **where** *hdiv*: $g = g' * [:0,1:]$
by (*smt* (*verit*) *dvdE mult.commute poly-eq-0-iff-dvd*)
ultimately have *hg'*: *normal-poly g'* **using** *normal-divide-x* **by** *blast*
assume *Suc dg = degree g*
hence *degree g' = dg* **using** *hdiv normal-non-zero*[*OF hg'*] **by** (*auto simp*: *degree-mult-eq*)
moreover assume $\bigwedge g.$ *dg = degree g* $\Longrightarrow$ *normal-poly g* $\Longrightarrow$ *normal-poly* $(f * g)$
ultimately have *normal-poly* $(f*g')$ **using** *hg'* **by** *blast*
thus *normal-poly* $(f*g)$ **using** *hdiv normal-mult-x* **by** *fastforce*
next

It now remains to show that $(fg)_i \geq 0$. This follows by decomposing $\{(h,j) \in \mathbf{Z}^2 | h > j\} = \{(h,j) \in \mathbf{Z}^2 | h \leq j\} \cup \{(h, h-1) \in \mathbf{Z}^2 | h \in \mathbf{Z}\}$. Note in order to avoid working with infinite sums over integers all these sets are bounded, which adds some complexity compared to the proof of lemma 2.55 in Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer Berlin Heidelberg, Berlin, Heidelberg (2016).

assume *hg0*: *poly g 0* $\neq$ *0* **and** *hg*: *normal-poly g*
have $f * g \neq 0$ **using** *hf hg* **by** (*simp add*: *normal-non-zero Suc.prems*)
moreover have $\bigwedge i.$ *coeff* $(f*g)$ $i \geq 0$
apply (*subst coeff-mult*, *rule sum-nonneg*, *rule mult-nonneg-nonneg*)
using *normal-coeff-nonneg*[*OF hf*] *normal-coeff-nonneg*[*OF hg*] **by** *auto*
moreover have
coeff $(f*g)$ i $*$ *coeff* $(f*g)$ $(i+2)$ $\leq$ (*coeff* $(f*g)$ $(i+1)$)^2 **for** i
proof −

$(fg)_{i+1}^2 - (fg)_i (fg)_{i+2} = \left(\sum_x f_x g_{i+1-x}\right)^2 - \left(\sum_x f_x g_{i+2-x}\right)\left(\sum_x f_x g_{i-x}\right)$

have (*coeff* $(f*g)$ $(i+1)$)^2 − *coeff* $(f*g)$ i $*$ *coeff* $(f*g)$ $(i+2)$ =
$(\sum x \leq i+1.$ *coeff f x* $*$ *coeff g* $(i + 1 - x)) *$
$(\sum x \leq i+1.$ *coeff f x* $*$ *coeff g* $(i + 1 - x)) -$
$(\sum x \leq i+2.$ *coeff f x* $*$ *coeff g* $(i + 2 - x)) *$
$(\sum x \leq i.$ *coeff f x* $*$ *coeff g* $(i - x))$
by (*auto simp*: *coeff-mult power2-eq-square algebra-simps*)

$\ldots = \sum_{x,y} f_x g_{i+1-x} f_y g_{i+1-y} - \sum_{x,y} f_x g_{i+2-x} f_y g_{i-y}$

also have ... =
$(\sum x \leq i+1.$ $\sum y \leq i+1.$ *coeff f x* $*$ *coeff g* $(i + 1 - x) *$
coeff f y $*$ *coeff g* $(i + 1 - y)) -$
$(\sum x \leq i+2.$ $\sum y \leq i.$ *coeff f x* $*$ *coeff g* $(i + 2 - x) *$
coeff f y $*$ *coeff g* $(i - y))$
by (*subst sum-product*, *subst sum-product*, *auto simp*: *algebra-simps*)

$\ldots = \sum_{h \leq j} f_h g_{i+1-h} f_j g_{i+1-j} + \sum_{h > j} f_h g_{i+1-h} f_j g_{i+1-j} - \sum_{h \leq j} f_h g_{i+2-h} f_j g_{i-j} - \sum_{h > j} f_h g_{i+2-h} f_j g_{i-j}$

also have ... =
$(\sum (h,\, j) \in \{(h,\, j).\ i+1 \geq j \wedge j \geq h\}.$

```
        coeff f h * coeff g (i + 1 - h) * coeff f j * coeff g (i + 1 - j)) +
      (∑(h, j)∈{(h, j). i+1 ≥ h ∧ h > j}.
        coeff f h * coeff g (i + 1 - h) * coeff f j * coeff g (i + 1 - j)) -
      ((∑(h, j)∈{(h, j). i ≥ j ∧ j ≥ h}.
        coeff f h * coeff g (i + 2 - h) * coeff f j * coeff g (i - j)) +
      (∑(h, j)∈{(h, j). i + 2 ≥ h ∧ h > j ∧ i ≥ j}.
        coeff f h * coeff g (i + 2 - h) * coeff f j * coeff g (i - j)))
    proof -
      have (∑x≤i + 1. ∑y≤i + 1. coeff f x * coeff g (i + 1 - x) * coeff f y
* coeff g (i + 1 - y)) =
        (∑(h, j)∈{(h, j). j ≤ i + 1 ∧ h ≤ j}.
          coeff f h * coeff g (i + 1 - h) * coeff f j * coeff g (i + 1 - j)) +
        (∑(h, j)∈{(h, j). h ≤ i + 1 ∧ j < h}.
          coeff f h * coeff g (i + 1 - h) * coeff f j * coeff g (i + 1 - j))
        proof (subst sum.union-disjoint[symmetric])
          have H:{(h, j). j ≤ i + 1 ∧ h ≤ j} ⊆ {..i+1} × {..i+1}
              {(h, j). h ≤ i + 1 ∧ j < h} ⊆ {..i+1} × {..i+1}
              finite ({..i+1} × {..i+1})
            by (fastforce, fastforce, fastforce)
          show finite {(h, j). j ≤ i + 1 ∧ h ≤ j}
            apply (rule finite-subset) using H by (blast, blast)
          show finite {(h, j). h ≤ i + 1 ∧ j < h}
            apply (rule finite-subset) using H by (blast, blast)
          show {(h, j). j ≤ i + 1 ∧ h ≤ j} ∩ {(h, j). h ≤ i + 1 ∧ j < h} = {}
            by fastforce
          show (∑x≤i + 1. ∑y≤i + 1. coeff f x * coeff g (i + 1 - x) * coeff f
y * coeff g (i + 1 - y)) =
            (∑(h, j)∈{(h, j). j ≤ i + 1 ∧ h ≤ j} ∪ {(h, j). h ≤ i + 1 ∧ j < h}.
              coeff f h * coeff g (i + 1 - h) * coeff f j * coeff g (i + 1 - j))
            apply (subst sum.cartesian-product, rule sum.cong)
            apply force by blast
        qed
      moreover have (∑x≤i + 2. ∑y≤i. coeff f x * coeff g (i + 2 - x) *
coeff f y * coeff g (i - y)) =
        (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
          coeff f h * coeff g (i + 2 - h) * coeff f j * coeff g (i - j)) +
        (∑(h, j)∈{(h, j). i + 2 ≥ h ∧ h > j ∧ i ≥ j}.
          coeff f h * coeff g (i + 2 - h) * coeff f j * coeff g (i - j))
        proof (subst sum.union-disjoint[symmetric])
          have H:{(h, j). j ≤ i ∧ h ≤ j} ⊆ {..i+2} × {..i}
              {(h, j). i + 2 ≥ h ∧ h > j ∧ i ≥ j} ⊆ {..i+2} × {..i}
              finite ({..i+2} × {..i})
            by (fastforce, fastforce, fastforce)
          show finite {(h, j). j ≤ i ∧ h ≤ j}
            apply (rule finite-subset) using H by (blast, blast)
          show finite {(h, j). i + 2 ≥ h ∧ h > j ∧ i ≥ j}
            apply (rule finite-subset) using H by (blast, blast)
        show {(h, j). j ≤ i ∧ h ≤ j} ∩ {(h, j). i + 2 ≥ h ∧ h > j ∧ i ≥ j} = {}
          by fastforce
```

show ($\sum x{\le}i + 2.\ \sum y{\le}i.$ *coeff f x* $*$ *coeff g* $(i + 2 - x) *$ *coeff f y* $*$ *coeff g* $(i - y)) =$
$(\sum (h, j){\in}\{(h, j).\ j \le i \wedge h \le j\} \cup \{(h, j).\ i + 2 \ge h \wedge h > j \wedge i \ge j\}.$
coeff f h $*$ *coeff g* $(i + 2 - h) *$ *coeff f j* $*$ *coeff g* $(i - j))$
apply (*subst sum.cartesian-product*, *rule sum.cong*)
apply *force* **by** *blast*
qed
ultimately show *?thesis* **by** *presburger*
qed

$\ldots = \sum_{h\le j} f_h g_{i+1-h} f_j g_{i+1-j} + \sum_{h\le j} f_{j+1} g_{i-j} f_{h-2} g_{i+2-h} + \sum_h f_h g_{i+1-h} f_{h-1} g_{i+2-h} - \sum_{h\le j} f_h g_{i+2-h} f_j g_{i-j} - \sum_{h\le j} f_{j+1} g_{i+1-j} f_{h-2} g_{i+1-h} - \sum_h f_h g_{i+2-h} f_{h-1} g_{i+1-h}$

also have ... =
$(\sum (h, j){\in}\{(h, j).\ j \le i + 1 \wedge h \le j\}.$
coeff f h $*$ *coeff g* $(i + 1 - h) *$ *coeff f j* $*$ *coeff g* $(i + 1 - j)) +$
$(\sum (h, j){\in}\{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\}.$
coeff f $(j{+}1) *$ *coeff g* $(i - j) *$ *coeff f* $(h{-}1) *$ *coeff g* $(i + 2 - h)) +$
$(\sum h{\in}\{1..i{+}1\}.$
coeff f h $*$ *coeff g* $(i + 1 - h) *$ *coeff f* $(h{-}1) *$ *coeff g* $(i + 2 - h)) -$
$((\sum (h, j){\in}\{(h, j).\ j \le i \wedge h \le j\}.$
coeff f h $*$ *coeff g* $(i + 2 - h) *$ *coeff f j* $*$ *coeff g* $(i - j)) +$
$(\sum (h, j){\in}\{(h, j).\ j \le i + 1 \wedge h \le j \wedge 0 < h\}.$
coeff f $(j{+}1) *$ *coeff g* $(i + 1 - j) *$ *coeff f* $(h{-}1) *$ *coeff g* $(i + 1 - h)) +$
$(\sum h{\in}\{1..i{+}1\}.$
coeff f h $*$ *coeff g* $(i + 2 - h) *$ *coeff f* $(h{-}1) *$ *coeff g* $(i + 1 - h)))$
proof −
have $(\sum (h, j){\in}\{(h, j).\ h \le i + 1 \wedge j < h\}.$
coeff f h $*$ *coeff g* $(i + 1 - h) *$ *coeff f j* $*$ *coeff g* $(i + 1 - j)) =$
$(\sum (h, j){\in}\{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\}.$
coeff f $(j + 1) *$ *coeff g* $(i - j) *$ *coeff f* $(h - 1) *$ *coeff g* $(i + 2 - h)) +$
$(\sum h = 1..i + 1.$ *coeff f h* $*$ *coeff g* $(i + 1 - h) *$ *coeff f* $(h - 1) *$ *coeff g* $(i + 2 - h))$
proof −
have *1*: $(\sum (h, j){\in}\{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\}.$
coeff f $(j + 1) *$ *coeff g* $(i - j) *$ *coeff f* $(h - 1) *$ *coeff g* $(i + 2 - h)) =$
$(\sum (h, j){\in}\{(h, j).\ h \le i + 1 \wedge j < h \wedge h \ne j + 1\}.$
coeff f h $*$ *coeff g* $(i + 1 - h) *$ *coeff f j* $*$ *coeff g* $(i + 1 - j))$
proof (*rule sum.reindex-cong*)
show $\{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\} = (\lambda(h, j).\ (j{+}1, h{-}1))\ `\ \{(h, j).\ h \le i + 1 \wedge j < h \wedge h \ne j + 1\}$
proof
show $(\lambda(h, j).\ (j + 1, h - 1))\ `\ \{(h, j).\ h \le i + 1 \wedge j < h \wedge h \ne j + 1\} \subseteq \{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\}$
by *fastforce*
show $\{(h, j).\ j \le i \wedge h \le j \wedge 0 < h\} \subseteq (\lambda(h, j).\ (j + 1, h - 1))\ `$

$\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}$
proof
fix x
assume $x \in \{(h, j).\ j \leq i \wedge h \leq j \wedge 0 < h\}$
then obtain $h\ j$ **where** $x = (h, j)\ j \leq i\ h \leq j\ 0 < h$ **by** *blast*
hence $j + 1 \leq i + 1 \wedge h - 1 < j + 1 \wedge j + 1 \neq h - 1 + 1 \wedge x = ((h - 1) + 1, (j + 1) - 1)$
by *auto*
thus $x \in (\lambda(h, j).\ (j + 1, h - 1))\ `\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}$
by (*auto simp*: *image-iff*)
qed
qed
show *inj-on* $(\lambda(h, j).\ (j + 1, h - 1))\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}$
proof
fix $x\ y$::$nat \times nat$
assume $x \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}\ y \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}$
thus $(case\ x\ of\ (h, j) \Rightarrow (j + 1, h - 1)) = (case\ y\ of\ (h, j) \Rightarrow (j + 1, h - 1)) \Longrightarrow x = y$
by *auto*
qed
show $\bigwedge x.\ x \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\} \Longrightarrow$
$(case\ case\ x\ of\ (h, j) \Rightarrow (j + 1, h - 1)\ of$
$(h, j) \Rightarrow coeff\ f\ (j + 1) * coeff\ g\ (i - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 2 - h)) =$
$(case\ x\ of\ (h, j) \Rightarrow coeff\ f\ h * coeff\ g\ (i + 1 - h) * coeff\ f\ j * coeff\ g\ (i + 1 - j))$
by *fastforce*
qed
have *2*: $(\sum h = 1..i + 1.\ coeff\ f\ h * coeff\ g\ (i + 1 - h) * coeff\ f\ (h - 1) * coeff\ g\ (i + 2 - h)) =$
$(\sum (h, j) \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}.$
$coeff\ f\ h * coeff\ g\ (i + 1 - h) * coeff\ f\ j * coeff\ g\ (i + 1 - j))$
proof (*rule sum.reindex-cong*)
show $\{1..i + 1\} = fst\ `\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
proof
show $\{1..i + 1\} \subseteq fst\ `\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
proof
fix x
assume $x \in \{1..i + 1\}$
hence $x \leq i + 1 \wedge x - 1 < x \wedge x = x - 1 + 1 \wedge x = fst\ (x, x-1)$
by *auto*
thus $x \in fst\ `\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
by *blast*
qed
show $fst\ `\ \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\} \subseteq \{1..i + 1\}$
by *force*

qed
show *inj-on fst* $\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
proof
fix *x y*
assume $x \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
$y \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
hence $x = (fst\ x, fst\ x - 1)$ $y = (fst\ y, fst\ y - 1)$ *fst x > 0 fst y > 0*
by *auto*
thus $fst\ x = fst\ y \Longrightarrow x = y$ **by** *presburger*
qed
show $\bigwedge x.\ x \in \{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\} \Longrightarrow$
*coeff f (fst x) * coeff g (i + 1 − fst x) * coeff f (fst x − 1) * coeff g (i + 2 − fst x) =*
*(case x of (h, j) ⇒ coeff f h * coeff g (i + 1 − h) * coeff f j * coeff g (i + 1 − j))*
by *fastforce*
qed
have *H*: $\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\} \subseteq \{0..i+1\}\times\{0..i+1\}$
$\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\} \subseteq \{0..i+1\}\times\{0..i+1\}$
finite $(\{0..i+1\}\times\{0..i+1\})$
by (*fastforce*, *fastforce*, *fastforce*)
have *finite* $\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h \neq j + 1\}$
finite $\{(h, j).\ h \leq i + 1 \wedge j < h \wedge h = j + 1\}$
apply (*rule finite-subset*) **using** *H* **apply** (*simp*, *simp*)
apply (*rule finite-subset*) **using** *H* **apply** (*simp*, *simp*)
done
thus *?thesis*
apply (*subst 1*, *subst 2*, *subst sum.union-disjoint*[*symmetric*])
apply *auto*[*3*]
apply (*rule sum.cong*)
by *auto*
qed
moreover have $(\sum (h, j) \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i\}.$
*coeff f h * coeff g (i + 2 − h) * coeff f j * coeff g (i − j)) =*
$(\sum (h, j) \in \{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\}.$
*coeff f (j + 1) * coeff g (i + 1 − j) * coeff f (h − 1) * coeff g (i + 1 − h)) +*
$(\sum h = 1..i + 1.$ *coeff f h * coeff g (i + 2 − h) * coeff f (h − 1) * coeff g (i + 1 − h))*
proof −
have *1*: $(\sum (h, j) \in \{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\}.$
*coeff f (j + 1) * coeff g (i + 1 − j) * coeff f (h − 1) * coeff g (i + 1 − h)) =*
$(\sum (h, j) \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}.$
*coeff f h * coeff g (i + 2 − h) * coeff f j * coeff g (i − j))*
proof (*rule sum.reindex-cong*)
show $\{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\} = (\lambda(h, j).\ (j+1, h-1))$ ‘
$\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}$
proof

show $(\lambda(h, j).\ (j + 1, h - 1))$ ‘ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\} \subseteq \{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\}$
by *fastforce*
show $\{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\} \subseteq (\lambda(h, j).\ (j + 1, h - 1))$ ‘ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}$
proof
fix x
assume $x \in \{(h, j).\ j \leq i + 1 \wedge h \leq j \wedge 0 < h\}$
then obtain $h\ j$ **where** $x = (h, j)\ j \leq i + 1\ h \leq j\ 0 < h$ **by** *blast*
hence $j + 1 \leq i + 2 \wedge h - 1 < j + 1 \wedge h - 1 \leq i \wedge j + 1 \neq h - 1 + 1 \wedge x = ((h - 1) + 1, (j + 1) - 1)$
by *auto*
thus $x \in (\lambda(h, j).\ (j + 1, h - 1))$ ‘ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}$
by (*auto simp*: *image-iff*)
qed
qed
show *inj-on* $(\lambda(h, j).\ (j + 1, h - 1))$ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}$
proof
fix $x\ y$::$nat \times nat$
assume $x \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}\ y \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\}$
thus $(case\ x\ of\ (h, j) \Rightarrow (j + 1, h - 1)) = (case\ y\ of\ (h, j) \Rightarrow (j + 1, h - 1)) \Longrightarrow x = y$
by *auto*
qed
show $\bigwedge x.\ x \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h \neq j + 1\} \Longrightarrow$
$(case\ case\ x\ of\ (h, j) \Rightarrow (j + 1, h - 1)\ of$
$(h, j) \Rightarrow coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h)) =$
$(case\ x\ of\ (h, j) \Rightarrow coeff\ f\ h * coeff\ g\ (i + 2 - h) * coeff\ f\ j * coeff\ g\ (i - j))$
by *fastforce*
qed
have *2*: $(\sum h = 1..i + 1.\ coeff\ f\ h * coeff\ g\ (i + 2 - h) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h)) =$
$(\sum (h, j) \in \{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h = j + 1\}.$
$coeff\ f\ h * coeff\ g\ (i + 2 - h) * coeff\ f\ j * coeff\ g\ (i - j))$
proof (*rule sum.reindex-cong*)
show $\{1..i + 1\} = fst$ ‘ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h = j + 1\}$
proof
show $\{1..i + 1\} \subseteq fst$ ‘ $\{(h, j).\ h \leq i + 2 \wedge j < h \wedge j \leq i \wedge h = j + 1\}$
proof
fix x
assume $x \in \{1..i + 1\}$
hence $x \leq i + 2 \wedge x - 1 < x \wedge x - 1 \leq i \wedge x = x - 1 + 1 \wedge x$

```
= fst (x, x−1)
                    by auto
                  thus x ∈ fst ‘ {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1}
                    by blast
                qed
                show fst ‘ {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1} ⊆ {1..i
+ 1}
                  by force
              qed
              show inj-on fst {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1}
              proof
                fix x y
                assume x ∈ {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1}
                       y ∈ {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1}
               hence x = (fst x, fst x − 1) y = (fst y, fst y − 1) fst x > 0 fst y > 0
                  by auto
                thus fst x = fst y ⟹ x = y by presburger
              qed
              show ⋀x. x ∈ {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1} ⟹
                coeff f (fst x) ∗ coeff g (i + 2 − fst x) ∗ coeff f (fst x − 1) ∗ coeff g
(i + 1 − fst x) =
                (case x of (h, j) ⇒ coeff f h ∗ coeff g (i + 2 − h) ∗ coeff f j ∗ coeff
g (i − j))
                  by fastforce
            qed
              have H: {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h ≠ j + 1} ⊆
{0..i+2}×{0..i}
                {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1} ⊆ {0..i+2}×{0..i}
                  finite ({0..i+2}×{0..i})
              by (fastforce, fastforce, fastforce)
            have finite {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h ≠ j + 1}
                finite {(h, j). h ≤ i + 2 ∧ j < h ∧ j ≤ i ∧ h = j + 1}
              apply (rule finite-subset) using H apply (simp, simp)
              apply (rule finite-subset) using H apply (simp, simp)
              done
            thus ?thesis
              apply (subst 1, subst 2, subst sum.union-disjoint[symmetric])
                apply auto[3]
              apply (rule sum.cong)
              by auto
          qed
          ultimately show ?thesis
            by algebra
        qed
```

$\ldots = \sum_{h\leq j} f_h f_j \left(g_{i+1-h}g_{i+1-j} - g_{i+2-h}g_{i-j}\right) + \sum_{h\leq j} f_{j+1}f_{h-1}\left(g_{i-j}g_{i+2-h} - g_{i+1-j}f_j g_{i+1-h}\right)$

Note we have to also consider the edge cases caused by making these sums finite.

```
      also have ... =
```

```
(∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j}.
   coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j))) +
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
  coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff
g (i + 2 − h) * coeff g (i − j))) +
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
   coeff f (j+1) * coeff f (h−1) * (coeff g (i − j) * coeff g (i + 2 − h)
− coeff g (i + 1 − j) * coeff g (i + 1 − h))) −
((∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j ∧ 0 < h}.
   coeff f (j+1) * coeff g (i + 1 − j) * coeff f (h−1) * coeff g (i + 1 −
h))) (is ?L = ?R)
  proof −
   have ?R =
   (∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j}.
      coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j))) +
   ((∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
      coeff f h * coeff f j * coeff g (i + 1 − h) * coeff g (i + 1 − j)) −
   (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
      coeff f h * coeff f j * coeff g (i + 2 − h) * coeff g (i − j))) +
   ((∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
    coeff f (j+1) * coeff f (h−1) * coeff g (i − j) * coeff g (i + 2 − h)) −
   (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
     coeff f (j+1) * coeff f (h−1) * coeff g (i + 1 − j) * coeff g (i + 1 −
h))) −
   ((∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j ∧ 0 < h}.
     coeff f (j+1) * coeff g (i + 1 − j) * coeff f (h−1) * coeff g (i + 1 −
h)))
    apply (subst sum-subtractf[symmetric], subst sum-subtractf[symmetric])
     by (auto simp: algebra-simps split-beta)
    also have ... =
      ((∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j}.
         coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j))) +
      (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
        coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j)))) −
       (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.
          coeff f h * coeff f j * coeff g (i + 2 − h) * coeff g (i − j)) +
      (∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
          coeff f (j + 1) * coeff f (h − 1) * coeff g (i − j) * coeff g (i + 2
− h)) −
      ((∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
        coeff f (j + 1) * coeff g (i + 1 − j) * coeff f (h − 1) * coeff g (i +
1 − h)) +
      (∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j ∧ 0 < h}.
        coeff f (j + 1) * coeff g (i + 1 − j) * coeff f (h − 1) * coeff g (i +
1 − h)))
     by (auto simp: algebra-simps)
    also have ... = ?L
    proof −
     have (∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j}.
```

coeff f h ∗ *coeff f j* ∗ (*coeff g* (*i* + *1* − *h*) ∗ *coeff g* (*i* + *1* − *j*))) +
(∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j*}.
coeff f h ∗ *coeff f j* ∗ (*coeff g* (*i* + *1* − *h*) ∗ *coeff g* (*i* + *1* − *j*))) =
(∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* + *1* ∧ *h* ≤ *j*}.
coeff f h ∗ *coeff f j* ∗ (*coeff g* (*i* + *1* − *h*) ∗ *coeff g* (*i* + *1* − *j*)))
proof (*subst sum.union-disjoint*[*symmetric*])
have {(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j*} ⊆ {..*i* + *1*} × {..*i* + *1*}
{(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j*} ⊆ {..*i* + *1*} × {..*i* + *1*}
by (*fastforce*, *fastforce*)
thus *finite* {(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j*} *finite* {(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j*}
by (*auto simp*: *finite-subset*)
show {(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j*} ∩ {(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j*} = {}
by *fastforce*
qed (*rule sum.cong*, *auto*)
moreover have (∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *0* < *h*}.
coeff f (*j* + *1*) ∗ *coeff g* (*i* + *1* − *j*) ∗ *coeff f* (*h* − *1*) ∗ *coeff g* (*i* + *1* − *h*)) +
(∑ (*h*, *j*)∈{(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j* ∧ *0* < *h*}.
coeff f (*j* + *1*) ∗ *coeff g* (*i* + *1* − *j*) ∗ *coeff f* (*h* − *1*) ∗ *coeff g* (*i* + *1* − *h*)) =
(∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* + *1* ∧ *h* ≤ *j* ∧ *0* < *h*}.
coeff f (*j* + *1*) ∗ *coeff g* (*i* + *1* − *j*) ∗ *coeff f* (*h* − *1*) ∗ *coeff g* (*i* + *1* − *h*))
proof (*subst sum.union-disjoint*[*symmetric*])
have {(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *0* < *h*} ⊆ {..*i* + *1*} × {..*i* + *1*}
{(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j* ∧ *0* < *h*} ⊆ {..*i* + *1*} × {..*i* + *1*}
by (*fastforce*, *fastforce*)
thus *finite* {(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *0* < *h*} *finite* {(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j* ∧ *0* < *h*}
by (*auto simp*: *finite-subset*)
show {(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *0* < *h*} ∩ {(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j* ∧ *0* < *h*} = {}
by *fastforce*
qed (*rule sum.cong*, *auto*)
ultimately show *?thesis*
by (*auto simp*: *algebra-simps*)
qed
finally show *?thesis* **by** *presburger*
qed

$\ldots = \sum_{h \le j} (f_h f_j - f_{j+1} f_{h-1}) (g_{i+1-h} g_{i+1-j} - g_{i+2-h} g_{i-j})$

also have ... =
(∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *0* < *h*}.
−(*coeff f h* ∗ *coeff f j* − *coeff f* (*j*+*1*) ∗ *coeff f* (*h*−*1*)) ∗ (*coeff g* (*i* − *j*) ∗ *coeff g* (*i* + *2* − *h*) − *coeff g* (*i* + *1* − *j*) ∗ *coeff g* (*i* + *1* − *h*))) +
(∑ (*h*, *j*)∈{(*h*, *j*). *j* ≤ *i* ∧ *h* ≤ *j* ∧ *h* = *0*}.
coeff f h ∗ *coeff f j* ∗ (*coeff g* (*i* + *1* − *h*) ∗ *coeff g* (*i* + *1* − *j*) − *coeff g* (*i* + *2* − *h*) ∗ *coeff g* (*i* − *j*))) +
(∑ (*h*, *j*)∈{(*h*, *j*). *j* = *i* + *1* ∧ *h* ≤ *j*}.

*coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j))) −*
((∑(h, j)∈{(h, j). j = i + 1 ∧ h ≤ j ∧ 0 < h}.
*coeff f (j+1) * coeff g (i + 1 − j) * coeff f (h−1) * coeff g (i + 1 − h))) (***is** *?L = ?R*)

proof −

have *(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j}.*
*coeff f h * coeff f j **
*(coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff g (i + 2 − h) * coeff g (i − j))) =*
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
*coeff f h * coeff f j **
*(coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff g (i + 2 − h) * coeff g (i − j))) +*
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 = h}.
*coeff f h * coeff f j **
*(coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff g (i + 2 − h) * coeff g (i − j)))*

proof (*subst sum.union-disjoint*[*symmetric*])

have *{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h} ⊆ {..i}×{..i} {(h, j). j ≤ i ∧ h ≤ j ∧ 0 = h} ⊆ {..i}×{..i}*

by (*force*, *force*)

thus *finite {(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h} finite {(h, j). j ≤ i ∧ h ≤ j ∧ 0 = h}*

by (*auto simp*: *finite-subset*)

show *{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h} ∩ {(h, j). j ≤ i ∧ h ≤ j ∧ 0 = h} = {}*

by *fast*

qed (*rule sum.cong*, *auto*)

moreover have *(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.*
*(−coeff f h * coeff f j + coeff f (j + 1) * coeff f (h − 1)) **
*(coeff g (i − j) * coeff g (i + 2 − h) − coeff g (i + 1 − j) * coeff g (i + 1 − h))) =*
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
*coeff f h * coeff f j * (coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff g (i + 2 − h) * coeff g (i − j))) +*
(∑(h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
*coeff f (j + 1) * coeff f (h − 1) **
*(coeff g (i − j) * coeff g (i + 2 − h) − coeff g (i + 1 − j) * coeff g (i + 1 − h)))*

by (*subst sum.distrib*[*symmetric*], *rule sum.cong*, *fast*, *auto simp*: *algebra-simps*)

ultimately show *?thesis*

by (*auto simp*: *algebra-simps*)

qed

. . . ≥ 0 by *normal-poly-general-coeff-mult*

also have *... ≥ 0*

```
proof −
  have 0 ≤ (∑ (h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}.
              − (coeff f h * coeff f j − coeff f (j + 1) * coeff f (h − 1)) *
               (coeff g (i − j) * coeff g (i + 2 − h) − coeff g (i + 1 − j) *
coeff g (i + 1 − h)))
  proof (rule sum-nonneg)
    fix x assume x ∈ {(h, j). j ≤ i ∧ h ≤ j ∧ 0 < h}
    then obtain h j where H: x = (h, j) j ≤ i h ≤ j 0 < h by fast
    hence h − 1 ≤ j − 1 by force
    hence 1: coeff f h * coeff f j − coeff f (j + 1) * coeff f (h − 1) ≥ 0
      using normal-poly-general-coeff-mult[OF hf, of h−1 j−1] H
      by (auto simp: algebra-simps)
    from H have i − j ≤ i − h by force
    hence 2: coeff g (i − j) * coeff g (i + 2 − h) − coeff g (i + 1 − j) *
coeff g (i + 1 − h) ≤ 0
      using normal-poly-general-coeff-mult[OF hg, of i − j i − h] H
      by (smt (verit, del-insts) Nat.add-diff-assoc2 le-trans)
    show 0 ≤ (case x of
     (h, j) ⇒
       − (coeff f h * coeff f j − coeff f (j + 1) * coeff f (h − 1)) *
       (coeff g (i − j) * coeff g (i + 2 − h) −
        coeff g (i + 1 − j) * coeff g (i + 1 − h)))
          apply (subst H(1), subst split, rule mult-nonpos-nonpos, subst
neg-le-0-iff-le)
      subgoal using 1 by blast
      subgoal using 2 by blast
      done
  qed
  moreover have 0 ≤ (∑ (h, j)∈{(h, j). j ≤ i ∧ h ≤ j ∧ h = 0}.
                coeff f h * coeff f j *
               (coeff g (i + 1 − h) * coeff g (i + 1 − j) − coeff g (i + 2 − h)
* coeff g (i − j)))
  proof (rule sum-nonneg)
    fix x assume x ∈ {(h, j). j ≤ i ∧ h ≤ j ∧ h = 0}
    then obtain h j where H: x = (h, j) j ≤ i h ≤ j h = 0 by fast
    have 1: coeff f h * coeff f j ≥ 0
      by (simp add: hf normal-coeff-nonneg)
    from H have i − j ≤ i − h by force
    hence 2: coeff g (i − j) * coeff g (i + 2 − h) − coeff g (i + 1 − j) *
coeff g (i + 1 − h) ≤ 0
      using normal-poly-general-coeff-mult[OF hg, of i − j i − h] H
      by (smt (verit, del-insts) Nat.add-diff-assoc2 le-trans)
    show 0 ≤ (case x of
     (h, j) ⇒
       coeff f h * coeff f j *
       (coeff g (i + 1 − h) * coeff g (i + 1 − j) −
        coeff g (i + 2 − h) * coeff g (i − j)))
      apply (subst H(1), subst split, rule mult-nonneg-nonneg)
      subgoal using 1 by blast
```

subgoal using *2* **by** *argo*
done
qed
moreover have $0 \leq (\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j\}.\ coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) -$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h))$
proof −
have $(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j\}.\ coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) -$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h)) =$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\}.\ coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) +$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.\ coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) -$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h))$
proof (*subst sum.union-disjoint*[*symmetric*])
have $\{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\} = \{(0, i + 1)\}$
$\{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\} = \{1..i+1\} \times \{i + 1\}$
by (*fastforce*, *force*)
thus *finite* $\{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\}$
finite $\{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}$
by *auto*
show $\{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\} \cap \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\} = \{\}$
by *fastforce*
have $\{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\} \cup \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\} = \{(h, j).\ j = i + 1 \wedge h \leq j\}$
by *fastforce*
thus $(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j\}.\ coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) -$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h)) =$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\} \cup \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ h * coeff\ f\ j * (coeff\ g\ (i + 1 - h) * coeff\ g\ (i + 1 - j))) -$
$(\sum (h, j) \in \{(h, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$coeff\ f\ (j + 1) * coeff\ g\ (i + 1 - j) * coeff\ f\ (h - 1) * coeff\ g\ (i + 1 - h))$
by *presburger*
qed
also have ... =

$(\sum (h,\, j) \in \{(h,\, j).\ j = i + 1 \wedge h \leq j \wedge h = 0\}.\ \mathit{coeff}\ f\ h * \mathit{coeff}\ f\ j * (\mathit{coeff}\ g\ (i + 1 - h) * \mathit{coeff}\ g\ (i + 1 - j))) +$
$(\sum (h,\, j) \in \{(h,\, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$(\mathit{coeff}\ f\ h * \mathit{coeff}\ f\ j - \mathit{coeff}\ f\ (j + 1) * \mathit{coeff}\ f\ (h - 1)) * (\mathit{coeff}\ g\ (i + 1 - h) * \mathit{coeff}\ g\ (i + 1 - j)))$
by (*subst add-diff-eq*[*symmetric*], *subst sum-subtractf*[*symmetric*], *subst add-left-cancel*, *rule sum.cong*, *auto simp*: *algebra-simps*)
also have $\ldots \geq 0$
proof (*rule add-nonneg-nonneg*)
show $0 \leq (\sum (h,\, j) \in \{(h,\, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}.$
$(\mathit{coeff}\ f\ h * \mathit{coeff}\ f\ j - \mathit{coeff}\ f\ (j + 1) * \mathit{coeff}\ f\ (h - 1)) *$
$(\mathit{coeff}\ g\ (i + 1 - h) * \mathit{coeff}\ g\ (i + 1 - j)))$
proof (*rule sum-nonneg*)
fix x **assume** $x \in \{(h,\, j).\ j = i + 1 \wedge h \leq j \wedge 0 < h\}$
then obtain $h\ j$ **where** H: $x = (h,\, j)\ j = i + 1\ h \leq j\ 0 < h$ **by** *fast*
hence $h - 1 \leq j - 1$ **by** *force*
hence *1*: $\mathit{coeff}\ f\ h * \mathit{coeff}\ f\ j - \mathit{coeff}\ f\ (j + 1) * \mathit{coeff}\ f\ (h - 1) \geq 0$
using *normal-poly-general-coeff-mult*[*OF hf*, *of h−1 j−1*] *H*
by (*auto simp*: *algebra-simps*)
hence *2*: $0 \leq \mathit{coeff}\ g\ (i + 1 - h) * \mathit{coeff}\ g\ (i + 1 - j)$
by (*meson hg mult-nonneg-nonneg normal-coeff-nonneg*)
show $0 \leq$ (*case x of*
$(h,\, j) \Rightarrow$
$(\mathit{coeff}\ f\ h * \mathit{coeff}\ f\ j - \mathit{coeff}\ f\ (j + 1) * \mathit{coeff}\ f\ (h - 1)) *$
$(\mathit{coeff}\ g\ (i + 1 - h) * \mathit{coeff}\ g\ (i + 1 - j)))$
apply (*subst H*(*1*), *subst split*, *rule mult-nonneg-nonneg*)
subgoal using *1* **by** *blast*
subgoal using *2* **by** *blast*
done
qed
qed (*rule sum-nonneg*, *auto simp*: *hf hg normal-coeff-nonneg*)[*1*]
finally show *?thesis* **.**
qed
ultimately show *?thesis* **by** *auto*
qed
finally show $\mathit{coeff}\ (f * g)\ i * \mathit{coeff}\ (f * g)\ (i + 2) \leq (\mathit{coeff}\ (f * g)\ (i + 1))^2$ **by** (*auto simp*: *power2-eq-square*)
qed
moreover have $\bigwedge i\ j\ k.\ i \leq j \Longrightarrow j \leq k \Longrightarrow 0 < \mathit{coeff}\ (f{*}g)\ i \Longrightarrow 0 < \mathit{coeff}\ (f{*}g)\ k \Longrightarrow 0 < \mathit{coeff}\ (f{*}g)\ j$
proof −
fix $j\ k$
assume $0 < \mathit{coeff}\ (f * g)\ k$
hence $k \leq \mathit{degree}\ (f * g)$ **using** *le-degree* **by** *force*
moreover assume $j \leq k$
ultimately have $j \leq \mathit{degree}\ (f * g)$ **by** *auto*
hence *1*: $j \leq \mathit{degree}\ f + \mathit{degree}\ g$
by (*simp add*: *degree-mult-eq hf hg normal-non-zero*)
show $0 < \mathit{coeff}\ (f * g)\ j$

```
        apply (subst coeff-mult, rule sum-pos2[of - min j (degree f)], simp, simp)
          apply (rule mult-pos-pos, rule normal-of-no-zero-root, simp add: hf0,
simp)
         using hf apply auto[1]
          apply (rule normal-of-no-zero-root)
           apply (simp add: hg0)
         using 1 apply force
         using hg apply auto[1]
         by (simp add: hf hg normal-coeff-nonneg)
       qed
       ultimately show normal-poly (f*g)
        by (rule normal-polyI)
     qed
   qed
 qed
qed

lemma normal-poly-of-roots:
 fixes p::real poly
 assumes ⋀z. poly (map-poly complex-of-real p) z = 0
      ⟹ Re z ≤ 0 ∧ 4*(Re z)^2 ≥ (cmod z)^2
    and lead-coeff p = 1
 shows normal-poly p
 using assms
proof (induction p rule: real-poly-roots-induct)
 fix p::real poly and x::real
 assume lead-coeff (p * [:- x, 1:]) = 1
 hence 1: lead-coeff p = 1
    by (metis coeff-degree-mult lead-coeff-pCons(1) mult-cancel-left1 pCons-one
zero-neq-one)
 assume h: (⋀z. poly (map-poly complex-of-real (p * [:- x, 1:])) z = 0 ⟹
              Re z ≤ 0 ∧ (cmod z)^2 ≤ 4 * (Re z)^2)
 hence 2: (⋀z. poly (map-poly complex-of-real p) z = 0 ⟹
              Re z ≤ 0 ∧ (cmod z)^2 ≤ 4 * (Re z)^2)
   by (metis four-x-squared mult-zero-left of-real-poly-map-mult poly-mult)
 have 3: normal-poly [:-x, 1:]
   apply (subst linear-normal-iff,
      subst Re-complex-of-real[symmetric], rule conjunct1)
   by (rule h[of x], subst of-real-poly-map-poly[symmetric], force)
 assume (⋀z. poly (map-poly complex-of-real p) z = 0
         ⟹ Re z ≤ 0 ∧ (cmod z)^2 ≤ 4 * (Re z)^2) ⟹
        lead-coeff p = 1 ⟹ normal-poly p
 hence normal-poly p using 1 2 by fast
 then show normal-poly (p * [:-x, 1:])
   using 3 by (rule normal-mult)
next
 fix p::real poly and a b::real
 assume lead-coeff (p * [:a * a + b * b, - 2 * a, 1:]) = 1
 hence 1: lead-coeff p = 1
```

by (*smt* (*verit*) *coeff-degree-mult lead-coeff-pCons*(*1*) *mult-cancel-left1 pCons-eq-0-iff pCons-one*)
assume *h*: (⋀*z*. *poly* (*map-poly complex-of-real* (*p* ∗ [:*a* ∗ *a* + *b* ∗ *b*, − *2* ∗ *a*, *1*:])) *z* = *0* ⟹
Re z ≤ *0* ∧ (*cmod z*)^2 ≤ *4* ∗ (*Re z*)^2)
hence *2*: (⋀*z*. *poly* (*map-poly complex-of-real p*) *z* = *0* ⟹
Re z ≤ *0* ∧ (*cmod z*)^2 ≤ *4* ∗ (*Re z*)^2)
proof −
fix *z* :: *complex*
assume *poly* (*map-poly complex-of-real p*) *z* = *0*
then have ∀ *q*. *0* = *poly* (*map-poly complex-of-real* (*p* ∗ *q*)) *z*
by *simp*
then show *Re z* ≤ *0* ∧ (*cmod z*)^2 ≤ *4* ∗ (*Re z*)^2
using *h* **by** *presburger*
qed
have *3*: [:*a* ∗ *a* + *b* ∗ *b*, − *2* ∗ *a*, *1*:] = [:*cmod* (*a* + i∗*b*) ^ *2*, −*2* ∗ *Re* (*a* + i∗*b*), *1*:]
by (*force simp*: *cmod-def power2-eq-square*)
interpret *map-poly-idom-hom complex-of-real* **..**
have *4*: *normal-poly* [:*a* ∗ *a* + *b* ∗ *b*, − *2* ∗ *a*, *1*:]
apply (*subst 3*, *subst quadratic-normal-iff*)
apply (*rule h*, *unfold hom-mult poly-mult*)
by (*auto simp*: *algebra-simps*)
assume (⋀*z*. *poly* (*map-poly complex-of-real p*) *z* = *0* ⟹ *Re z* ≤ *0* ∧ (*cmod z*)^2 ≤ *4* ∗ (*Re z*)^2) ⟹
lead-coeff p = *1* ⟹ *normal-poly p*
hence *normal-poly p* **using** *1 2* **by** *fast*
then show *normal-poly* (*p* ∗ [:*a* ∗ *a* + *b* ∗ *b*, − *2* ∗ *a*, *1*:])
using *4* **by** (*rule normal-mult*)
next
fix *a*::*real*
assume *lead-coeff* [:*a*:] = *1*
moreover have ⋀*i j k*.
lead-coeff [:*a*:] = *1* ⟹
i ≤ *j* ⟹
j ≤ *k* ⟹ *0* < *coeff* [:*a*:] *i* ⟹ *0* < *coeff* [:*a*:] *k* ⟹ *0* < *coeff* [:*a*:] *j*
by (*metis bot-nat-0.extremum-uniqueI coeff-eq-0 degree-pCons-0 leI*
less-numeral-extra(*3*))
ultimately show *normal-poly* [:*a*:]
apply (*subst normal-polyI*)
by (*auto simp*:*pCons-one*)
qed

lemma *normal-changes*:
fixes *f*::*real poly*
assumes *hf*: *normal-poly f* **and** *hx*: *x* > *0*
defines *df* ≡ *degree f*
shows *changes* (*coeffs* (*f*∗[:−*x*,*1*:])) = *1*
using *df-def hf*

proof (*induction df arbitrary*: *f*)
case *0*
then obtain *a* **where** *f* = [:*a*:] **using** *degree-0-iff* **by** *auto*
thus *changes* (*coeffs* (*f*∗[:−*x*, *1*:])) = *1*
using *normal-non-zero*[*OF* ‹*normal-poly f*›] *hx*
by (*auto simp*: *algebra-simps zero-less-mult-iff mult-less-0-iff*)
next
case (*Suc df*)
then show *?case*
proof (*cases poly f 0 = 0*)
assume *poly f 0 = 0* **and** *hf*:*normal-poly f*
moreover then obtain *f′* **where** *hdiv*: *f* = *f′*∗[:*0*, *1*:]
by (*smt* (*verit*) *dvdE mult.commute poly-eq-0-iff-dvd*)
ultimately have *hf′*: *normal-poly f′* **using** *normal-divide-x* **by** *blast*
assume *Suc df* = *degree f*
hence *degree f′* = *df* **using** *hdiv normal-non-zero*[*OF hf′*] **by** (*auto simp*: *degree-mult-eq*)
moreover assume ⋀*f*::*real poly*. *df* = *degree f* ⟹ *normal-poly f* ⟹ *changes* (*coeffs* (*f* ∗ [:− *x*, *1*:])) = *1*
ultimately have *changes* (*coeffs* (*f′* ∗ [:− *x*, *1*:])) = *1* **using** *hf′* **by** *fast*
thus *changes* (*coeffs* (*f* ∗ [:− *x*, *1*:])) = *1*
apply (*subst hdiv*, *subst mult-pCons-right*, *subst smult-0-left*, *subst add-0*)
apply (*subst mult-pCons-left*, *subst smult-0-left*, *subst add-0*)
by (*subst changes-pCons*, *auto*)
next
assume *hf*:*normal-poly f* **and** *poly f 0* ≠ *0*
hence *h′*: ⋀*i*. *i* ≤ *degree f* ⟹ *coeff f i* > *0*
by (*auto simp*: *normal-of-no-zero-root*)
hence ⋀*i*. *i* < *degree f* − *1* ⟹ (*coeff f i*)/(*coeff f* (*i*+*1*)) ≤ (*coeff f* (*i*+*1*))/(*coeff f* (*i*+*2*))
using *normal-poly-coeff-mult*[*OF hf*] *normal-coeff-nonneg*[*OF hf*]
by (*auto simp*: *divide-simps power2-eq-square*)
hence *h″*: ⋀*i*. *i* < *degree f* − *1* ⟹ (*coeff f i*)/(*coeff f* (*i*+*1*)) − *x* ≤ (*coeff f* (*i*+*1*))/(*coeff f* (*i*+*2*)) − *x*
by *fastforce*
have *hdeg*: *degree* (*pCons 0 f* − *smult x f*) = *degree f* + *1*
apply (*subst diff-conv-add-uminus*)
apply (*subst degree-add-eq-left*)
by (*auto simp*: *hf normal-non-zero*)

let *?f* = λ *z w*. λ*i*. *if i*=*0 then z*/(*x* ∗ *coeff f 0*) *else* (*if i* = *degree* (*pCons 0 f* − *smult x f*) *then w*/(*lead-coeff f*) *else inverse* (*coeff f i*))

have *1*: ⋀*z w*. *0* < *z* ⟹ *0* < *w* ⟹ *changes* (*coeffs* (*f* ∗ [:−*x*, *1*:])) =
changes (−*z* # *map* (λ*i*. (*coeff f* (*i*−*1*))/(*coeff f i*) − *x*) [*1*..<*degree* (*pCons 0 f* − *smult x f*)] @ [*w*])
proof −
fix *z w* :: *real*
assume *hz*: *0* < *z* **and** *hw*: *0* < *w*

have $-z \# map\ (\lambda i.\ (coeff\ f\ (i-1))/(coeff\ f\ i) - x)\ [1..<degree\ (pCons\ 0\ f - smult\ x\ f)] \ @\ [w] =$
$map\ (\lambda i.\ if\ i = 0\ then\ -z\ else\ if\ i = degree\ (pCons\ 0\ f - smult\ x\ f)\ then\ w\ else$
$(coeff\ f\ (i-1))/(coeff\ f\ i) - x)\ [0..<degree\ (pCons\ 0\ f - smult\ x\ f) + 1]$
proof (*rule nth-equalityI*)
fix i **assume** $i < length\ (- z \# map\ (\lambda i.\ coeff\ f\ (i - 1)\ /\ coeff\ f\ i - x)\ [1..<degree\ (pCons\ 0\ f - smult\ x\ f)] \ @\ [w])$
hence $i \leq degree\ (pCons\ 0\ f - smult\ x\ f)$
using *hdeg Suc.hyps*(*2*) **by** *auto*
then consider $(a) i = 0 \mid (b)(0 < i \wedge i < degree\ (pCons\ 0\ f - smult\ x\ f)) \mid (c) i = degree\ (pCons\ 0\ f - smult\ x\ f)$
by *fastforce*
then show $(- z \#$
$map\ (\lambda i.\ coeff\ f\ (i - 1)\ /\ coeff\ f\ i - x)$
$[1..<degree\ (pCons\ 0\ f - smult\ x\ f)] \ @$
$[w])\ !\ i =$
$map\ (\lambda i.\ if\ i = 0\ then - z$
$else\ if\ i = degree\ (pCons\ 0\ f - smult\ x\ f)\ then\ w$
$else\ coeff\ f\ (i - 1)\ /\ coeff\ f\ i - x)$
$[0..<degree\ (pCons\ 0\ f - smult\ x\ f) + 1]\ !\ i$
apply (*cases*)
by (*auto simp*: *nth-append*)
qed (*force simp*: *hdeg*)

also have $... = [?f\ z\ w\ i * (nth\text{-}default\ 0\ (coeffs\ (f * [:-x,\ 1:]))\ i).$
$i \leftarrow [0..<Suc\ (degree\ (pCons\ 0\ f - smult\ x\ f))]]$
proof (*rule map-cong*)
fix i **assume** $i \in set\ [0..<Suc\ (degree\ (pCons\ 0\ f - smult\ x\ f))]$
then consider $(a) i = 0 \mid (b)(0 \neq i \wedge i < degree\ (pCons\ 0\ f - smult\ x\ f)) \mid (c) i = degree\ (pCons\ 0\ f - smult\ x\ f)$
by *fastforce*
then show $(if\ i = 0\ then - z$
$else\ if\ i = degree\ (pCons\ 0\ f - smult\ x\ f)\ then\ w$
$else\ coeff\ f\ (i - 1)\ /\ coeff\ f\ i - x) =$
$(if\ i = 0\ then\ z\ /\ (x * coeff\ f\ 0)$
$else\ if\ i = degree\ (pCons\ 0\ f - smult\ x\ f)\ then\ w\ /\ lead\text{-}coeff\ f$
$else\ inverse\ (coeff\ f\ i)) *$
$nth\text{-}default\ 0\ (coeffs\ (f * [:-x,\ 1:]))\ i$
proof (*cases*)
case (*a*)
thus *?thesis* **using** *hx* ‹*poly f 0* $\neq$ *0*› **by** (*auto simp*: *nth-default-coeffs-eq poly-0-coeff-0*)
next
case (*b*)
thus *?thesis* **using** *hx h'*[*of i*] *hdeg*
by (*auto simp*: *field-simps nth-default-coeffs-eq coeff-pCons nat.split poly-0-coeff-0*)

```
      next
        case (c)
        thus ?thesis using hdeg by (auto simp: nth-default-coeffs-eq coeff-eq-0)
      qed
    qed force

    finally have 1: − z #
      map (λi. coeff f (i − 1) / coeff f i − x) [1..<degree (pCons 0 f − smult x
f)] @ [w] =
      map (λi. (if i = 0 then z / (x ∗ coeff f 0)
            else if i = degree (pCons 0 f − smult x f) then w / lead-coeff f
                 else inverse (coeff f i)) ∗
          nth-default 0 (coeffs (f ∗ [:− x, 1:])) i)
       [0..<Suc (degree (pCons 0 f − smult x f))] .

    have f ∗ [:−x, 1:] ≠ 0 using hdeg by force

    show changes (coeffs (f ∗ [:− x, 1:])) =
        changes
         (− z #
          map (λi. coeff f (i − 1) / coeff f i − x)
           [1..<degree (pCons 0 f − smult x f)] @
          [w])
      apply (subst 1)
      apply (rule changes-scale[symmetric])
      subgoal using hz hw hx h′ hdeg by auto
      subgoal using hdeg ‹f ∗ [:−x, 1:] ≠ 0›
        by (auto simp: length-coeffs)
      done
  qed

  hence changes (coeffs (f ∗ [:− x, 1:])) =
      changes
       (− (max 1 (−(coeff f 0 / coeff f 1 − x))) #
        map (λi. coeff f (i − 1) / coeff f i − x)
         [1..<degree (pCons 0 f − smult x f)] @
        [max 1 (coeff f (degree f − 1) / lead-coeff f − x)])
    by force

  also have ... = 1
  proof (rule changes-increasing)
    fix i
    assume i < length
          (− max 1 (− (coeff f 0 / coeff f 1 − x)) #
          map (λi. coeff f (i − 1) / coeff f i − x) [1..<degree (pCons 0 f − smult
x f)] @
           [max 1 (coeff f (degree f − 1) / lead-coeff f − x)]) − 1
    hence i < degree (pCons 0 f − smult x f)
      using hdeg Suc.hyps(2) by fastforce
```

```
    then consider (a)i = 0 | (b)0 ≠ i ∧ i < degree (pCons 0 f − smult x f) −
1 |
      (c)i = degree (pCons 0 f − smult x f) − 1
      by fastforce
    then show (− max 1 (− (coeff f 0 / coeff f 1 − x)) #
       map (λi. coeff f (i − 1) / coeff f i − x)
        [1..<degree (pCons 0 f − smult x f)] @
       [max 1 (coeff f (degree f − 1) / lead-coeff f − x)]) !
      i
      ≤ (− max 1 (− (coeff f 0 / coeff f 1 − x)) #
          map (λi. coeff f (i − 1) / coeff f i − x)
           [1..<degree (pCons 0 f − smult x f)] @
          [max 1 (coeff f (degree f − 1) / lead-coeff f − x)]) !
         (i + 1)
    proof (cases)
      case a
      then show ?thesis by (auto simp: nth-append)
    next
      case b
      have coeff f (i − 1) ∗ coeff f (i − 1 + 2) ≤ (coeff f (i − 1 + 1))^2
        by (rule normal-poly-coeff-mult[OF hf, of i − 1])
      hence coeff f (i − 1) / coeff f i ≤ coeff f i / coeff f (i + 1)
        using h'[of i] h'[of i+1] h'[of i−1] h' b hdeg
        by (auto simp: power2-eq-square divide-simps)
      then show ?thesis
        using b by (auto simp: nth-append)
    next
      case c
      then show ?thesis using hdeg by (auto simp: nth-append not-le)
    qed
  qed auto

  finally show changes (coeffs (f ∗ [:−x, 1:])) = 1 .
 qed
qed

end
```

5 Proof of the theorem of three circles

```
theory Three-Circles
 imports Bernstein Normal-Poly
begin
```

The theorem of three circles is a result in real algebraic geometry about the number of real roots in an interval. It says if the number of roots in certain circles in the complex plane are zero or one then the number of roots in the circles is equal to the sign changes of the Bernstein coefficients on that interval for which the circles intersect the real line. This can then be used

to determine if an interval has a real root in the bisection procedure, which is more efficient than Descartes' rule of signs.

The proof here follows Theorem 10.50 in Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer Berlin Heidelberg, Berlin, Heidelberg (2016).

This theorem has also been fomalised in Coq [4]. The relationship between this theorem and root isolation has been elaborated in Eigenwillig's PhD thesis [2].

5.1 No sign changes case

declare *degree-pcompose*[*simp del*]

corollary *descartes-sign-zero*:
  fixes *p*::*real poly*
  assumes $\bigwedge$*x*::*complex*. *poly* (*map-poly of-real p*) *x* = *0* $\Longrightarrow$ *Re x* $\leq$ *0*
    and *lead-coeff p* = *1*
  shows *coeff p i* $\geq$ *0*
  using *assms*
proof (*induction p arbitrary*: *i rule*: *real-poly-roots-induct*)
  case (*1 p x*)
  interpret *map-poly-idom-hom complex-of-real* **..**
  have *h*: $\bigwedge$ *i*. *0* $\leq$ *coeff p i*
    apply (*rule 1*(*1*))
    using *1*(*2*) **apply** (*metis lambda-zero of-real-poly-map-mult poly-mult*)
    using *1*(*3*) **apply** (*metis lead-coeff-1 lead-coeff-mult lead-coeff-pCons*(*1*)
                mult-cancel-right2 pCons-one zero-neq-one)
    done
  have *x* $\leq$ *0*
    apply (*subst Re-complex-of-real*[*symmetric*])
    apply (*rule 1*(*2*))
    apply (*subst hom-mult*)
    by (*auto*)
  thus *?case*
    apply (*cases i*)
    subgoal using *h*[*of i*] *h*[*of i*−*1*]
      by (*fastforce simp*: *coeff-pCons mult-nonneg-nonpos2*)
    subgoal using *h*[*of i*] *h*[*of i*−*1*] *mult-left-mono-neg*
      by (*fastforce simp*: *coeff-pCons*)
    done
next
  case (*2 p a b*)
  interpret *map-poly-idom-hom complex-of-real* **..**
  have *h*: $\bigwedge$ *i*. *0* $\leq$ *coeff p i*
    apply (*rule 2*(*2*))
    using *2*(*3*) **apply** (*metis lambda-zero of-real-poly-map-mult poly-mult*)
    using *2*(*4*) **apply** (*metis lead-coeff-1 lead-coeff-mult lead-coeff-pCons*(*1*)
                mult-cancel-right2 pCons-one zero-neq-one)

```
    done
  have Re (a + b * i) ≤ 0
    apply (rule 2(3))
    apply (subst hom-mult)
    by (auto simp: algebra-simps)
  hence 1: 0 ≤ - 2 * a by fastforce
  have 2: 0 ≤ a * a + b * b by fastforce
  have ⋀ x. 0 ≤ coeff [:a * a + b * b, - 2 * a, 1:] x
  proof -
    fix x
    show 0 ≤ coeff [:a * a + b * b, - 2 * a, 1:] x
      using 2 apply (cases x = 0, fastforce)
      using 1 apply (cases x = 1, fastforce)
      apply (cases x = 2, fastforce simp: coeff-pCons)
      by (auto simp: coeff-eq-0)
  qed
  thus ?case
    apply (subst mult.commute, subst coeff-mult)
    apply (rule sum-nonneg, rule mult-nonneg-nonneg[OF - h])
    by auto
next
  case (3 a)
  then show ?case
    by (smt (z3) bot-nat-0.extremum-uniqueI degree-1 le-degree
        lead-coeff-pCons(2) pCons-one)
qed

definition circle-01-diam :: complex set where
circle-01-diam =
 {x. cmod (x - (of-nat 1 :: complex)/(of-nat 2)) < (real 1)/(real 2)}

lemma pos-real-map:
  {x::complex. 1 / x ∈ (λx. x + 1) ‘ {x. 0 < Re x}} = circle-01-diam
proof
  show {x. 1 / x ∈ (λx. x + 1) ‘ {x. 0 < Re x}} ⊆ circle-01-diam
  proof
    fix x assume x ∈ {x. 1 / x ∈ (λx. x + 1) ‘ {x. 0 < Re x}}
    then obtain y where h: 1 / x = y + 1 and hy: 0 < Re y by blast
    hence hy': y = 1 / x - 1 by fastforce
    hence hy'': y + 1 ≠ 0 using h hy by fastforce
    hence x = 1 / (y + 1) using h
      by (metis div-by-1 divide-divide-eq-right mult.left-neutral)
    have |Re y - 1| < |Re y + 1| using hy by simp
    hence cmod (y - 1) < cmod (y + 1)
      by (smt (z3) cmod-Re-le-iff minus-complex.simps(1) minus-complex.simps(2)
          one-complex.simps plus-complex.simps(1) plus-complex.simps(2))
    hence cmod ((y - 1)/(y + 1)) < 1
      by (smt (verit) divide-less-eq-1-pos nonzero-norm-divide zero-less-norm-iff)
    thus x ∈ circle-01-diam using hy' hy''
```

```
    by (auto simp: field-simps norm-minus-commute circle-01-diam-def)
  qed
  show circle-01-diam ⊆ {x. 1 / x ∈ (λx. x + 1) ` {x. 0 < Re x}}
  proof
    fix x assume x ∈ circle-01-diam
    hence cmod (x − 1 / 2) ∗ 2 < 1 by (auto simp: circle-01-diam-def)
    hence h: x ≠ 0 and cmod (x − 1 / 2) ∗ cmod 2 < 1 by auto
    hence cmod (2∗x − 1) < 1
      by (smt (verit) dbl-simps(3) dbl-simps(5) div-self times-divide-eq-left
          left-diff-distrib-numeral mult.commute mult-numeral-1
          norm-eq-zero norm-mult norm-numeral norm-one numeral-One)
    hence cmod (((1/x − 1) − 1)/((1/x − 1) + 1)) < 1
      by (auto simp: divide-simps norm-minus-commute)
    hence cmod (((1/x − 1) − 1)/ cmod ((1/x − 1) + 1)) < 1
      by (metis (no-types, lifting) abs-norm-cancel norm-divide norm-of-real)
    hence cmod ((1/x − 1) − 1) < cmod ((1/x − 1) + 1) using h
      by (smt (verit) diff-add-cancel divide-eq-0-iff divide-less-eq-1-pos
          norm-divide norm-of-real zero-less-norm-iff zero-neq-one)
    hence |Re (1/x − 1) − 1| < |Re (1/x − 1) + 1|
      by (smt (z3) cmod-Re-le-iff minus-complex.simps(1) minus-complex.simps(2)
          one-complex.simps plus-complex.simps(1) plus-complex.simps(2))
    hence 0 < Re (1/x − 1) by linarith
    moreover have 1 / x = (1/x − 1) + 1 by simp
    ultimately have 0 < Re (1/x − 1) ∧ 1 / x = (1/x − 1) + 1 by blast
    hence ∃xa. 0 < Re xa ∧ 1 / x = xa + 1 by blast
    thus x ∈ {x. 1 / x ∈ (λx. x + 1) ` {x. 0 < Re x}} by blast
  qed
qed

lemma one-circle-01: fixes P::real poly assumes hP: degree P ≤ p and P ≠ 0
  and proots-count (map-poly of-real P) circle-01-diam = 0
shows Bernstein-changes-01 p P = 0
proof −
  let ?Q = (reciprocal-poly p P) ∘p [:1, 1:]
  have hQ: ?Q ≠ 0
    using assms
    by (simp add: Missing-Polynomial.pcompose-eq-0 reciprocal-0-iff)

  hence 1: changes (coeffs ?Q) ≥ proots-count ?Q {x. 0 < x} ∧
       even (changes (coeffs ?Q) − proots-count ?Q {x. 0 < x})
    by (rule descartes-sign)

  have hdeg: degree (map-poly complex-of-real P) ≤ p
    by (rule le-trans, rule degree-map-poly-le, auto simp: assms)
  have hx: ⋀x. 1 + x = 0 ⟹ 0 < Re x ⟹ False
  proof −
    fix x::complex assume 1 + x = 0
    hence x = −1 by algebra
    thus 0 < Re x ⟹ False by auto
```

qed

have *2*: *proots-count* (*map-poly of-real P*) *circle-01-diam* =
proots-count (*map-poly of-real ?Q*) {*x*. *0* < *Re x*}
apply (*subst pos-real-map*[*symmetric*])
apply (*subst of-real-hom.map-poly-pcompose*)
apply (*subst map-poly-reciprocal*) **using** *assms* **apply** *auto*[*2*]
apply (*subst proots-pcompose*)
using *assms* **apply** (*auto simp*: *reciprocal-0-iff degree-map-poly*)[*2*]
apply (*subst proots-count-reciprocal*)
using *assms* **apply** (*auto simp*: *degree-map-poly inverse-eq-divide*)[*2*]
using *hx* **apply** *fastforce*
by (*auto simp*: *inverse-eq-divide algebra-simps*)

hence *3*:*proots-count* (*map-poly of-real ?Q*) {*x*. *0* < *Re x*} = *0*
using *assms*(*3*) **by** *presburger*

hence ⋀*x*::*complex*.
poly (*map-poly of-real* (*smult* (*inverse* (*lead-coeff ?Q*)) *?Q*)) *x* = *0* ⟹
Re x ≤ *0*
proof *cases*
fix *x*::*complex* **show** *Re x* ≤ *0* ⟹ *Re x* ≤ *0* **by** *fast*
assume ¬*Re x* ≤ *0* **hence** *h*:*0* < *Re x* **by** *simp*
assume *poly* (*map-poly of-real* (*smult* (*inverse* (*lead-coeff ?Q*)) *?Q*)) *x* = *0*
hence *h2*:*poly* (*map-poly of-real ?Q*) *x* = *0* **by** *fastforce*
hence *order x* (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])) > *0*
using *assms* **by** (*fastforce simp*: *order-root pcompose-eq-0 reciprocal-0-iff*)
hence *proots-count* (*map-poly of-real ?Q*) {*x*. *0* < *Re x*} ≠ *0*
proof −
have *h3*: *finite* {*x*. *poly* (*map-poly complex-of-real*
(*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])) *x* = *0*}
apply (*rule poly-roots-finite*)
using *assms* **by** (*fastforce simp*: *order-root pcompose-eq-0 reciprocal-0-iff*)
have *0* < *order x* (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]))
using *h2 assms* **by** (*fastforce simp*: *order-root pcompose-eq-0 reciprocal-0-iff*)
also have ... ≤ (∑ *r*∈{*x*. *0* < *Re x* ∧
poly (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])) *x* =
0}.
order r (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])))
apply (*rule member-le-sum*) **using** *h h2 h3* **by** *auto*
finally have
0 < (∑ *r*∈{*x*. *0* < *Re x* ∧
poly (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])) *x* = *0*}.
order r (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]))) .
thus
0 < *order x* (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:])) ⟹
proots-count (*map-poly complex-of-real* (*reciprocal-poly p P* ∘$_p$ [:*1*, *1*:]))
{*x*. *0* < *Re x*} ≠ *0*
by (*auto simp*: *proots-count-def proots-within-def*)

```
    qed
    thus Re x ≤ 0 using 3 by blast
  qed
  hence ⋀i. coeff (smult (inverse (lead-coeff ?Q)) ?Q) i ≥ 0
    apply (frule descartes-sign-zero)
    using assms by (fastforce simp: pcompose-eq-0 reciprocal-0-iff)
  hence changes (coeffs (smult (inverse (lead-coeff ?Q)) ?Q)) = 0
    by (subst changes-all-nonneg, auto simp: nth-default-coeffs-eq)
  hence changes (coeffs ?Q) = 0
    using hQ by (auto simp: coeffs-smult changes-scale-const)

  thus ?thesis
    apply (subst Bernstein-changes-01-eq-changes[OF hP])
    by blast
qed

definition circle-diam :: real ⇒ real ⇒ complex set where
circle-diam l r = {x. cmod ((x − l) − (r − l)/2) < (r − l)/2}

lemma circle-diam-rescale: assumes l < r
  shows circle-diam l r = (λ x . (x∗(r − l) + l)) ‘ circle-01-diam
proof
  show circle-diam l r ⊆ (λx. x ∗ (complex-of-real r − complex-of-real l) +
      complex-of-real l) ‘ circle-01-diam
  proof
    fix x assume x ∈ circle-diam l r
    hence cmod ((x − l) − (r − l)/2) < (r − l)/2 by (auto simp: circle-diam-def)
    hence cmod ((r − l) ∗ ((x − l)/(r − l) − 1/2)) < (r − l)/2 using assms
      by (subst right-diff-distrib, fastforce)
    hence (r − l) ∗ cmod ((x − l)/(r − l) − 1/2) < (r − l) ∗ 1/2
      apply (subst(2) abs-of-pos[symmetric])
      subgoal using assms by argo
      subgoal
        apply (subst norm-scaleR[symmetric])
        by (simp add: scaleR-conv-of-real)
      done
    hence cmod ((x − l)/(r − l) − 1/2) < 1/2
      apply (subst mult-less-cancel-left-pos[of r−l,symmetric])
      using assms by auto
    hence
      cmod ((x−l)/(r−l) − 1 / 2) ∗ 2 < 1 ∧
      x = (x−l)/(r−l) ∗ (complex-of-real r − complex-of-real l) + complex-of-real l
      by force
    thus x ∈ (λx. x ∗ (complex-of-real r − complex-of-real l) + complex-of-real l) ‘
        circle-01-diam
      by (force simp: circle-01-diam-def)
  qed
  show (λx. x ∗ (complex-of-real r − complex-of-real l) + complex-of-real l) ‘
      circle-01-diam ⊆ circle-diam l r
```

```
  proof
    fix x::complex
    assume
      x ∈ (λx. x * (complex-of-real r − complex-of-real l) + complex-of-real l) ‘
       circle-01-diam
    then obtain y::complex where x = y * (r − l) + l cmod (y − 1/2) < 1/2
      by (fastforce simp: circle-01-diam-def)
    moreover hence y = (x − l) / (r − l) using assms by force
    ultimately have cmod ((x − l) / (r − l) − 1/2) < 1/2 by presburger
    hence (r − l) * (cmod ((x − l) / (r − l) − 1/2)) < (r − l) * (1/2)
      apply (subst mult-less-cancel-left-pos)
      using assms by auto
    hence cmod ((x − l) − (r − l)/2) < (r − l)/2
      apply (subst(asm) (2) abs-of-pos[symmetric])
      using assms apply argo
      apply (subst(asm) norm-scaleR[symmetric])
      by (smt (verit, del-insts)
          ‹x = y * complex-of-real (r − l) + complex-of-real l›
          ‹y = (x − complex-of-real l) / complex-of-real (r − l)›
          add-diff-cancel divide-divide-eq-right divide-numeral-1 mult.commute
          of-real-1 of-real-add of-real-divide one-add-one scaleR-conv-of-real
          scale-right-diff-distrib times-divide-eq-right)
    thus x ∈ circle-diam l r
      by (force simp: circle-diam-def)
  qed
qed

lemma one-circle: fixes P::real poly assumes l < r
  and proots-count (map-poly of-real P) (circle-diam l r) = 0
  and P ≠ 0
  and degree P ≤ p
shows Bernstein-changes p l r P = 0
proof (subst Bernstein-changes-eq-rescale)
  show l ≠ r using assms(1) by force
  show degree P ≤ p using assms(4) by blast
  show Bernstein-changes-01 p (P ∘p [:l, 1:] ∘p [:0, r − l:]) = 0
  proof (rule one-circle-01)
    show degree (P ∘p [:l, 1:] ∘p [:0, r − l:]) ≤ p
      using assms(4) by (force simp: degree-pcompose)
    show P ∘p [:l, 1:] ∘p [:0, r − l:] ≠ 0
      using assms by (smt (z3) degree-0-iff gr-zeroI pCons-eq-0-iff pCons-eq-iff
                    pcompose-eq-0)

    have proots-count (map-poly of-real P) (circle-diam l r) =
         proots-count (map-poly complex-of-real (P ∘p [:l, 1:] ∘p [:0, r − l:]))
         circle-01-diam
      apply (subst of-real-hom.map-poly-pcompose)
      apply (subst proots-pcompose)
       apply (metis assms(3) degree-eq-zeroE of-real-poly-eq-0-iff
```

pCons-eq-iff pCons-one pcompose-eq-0 zero-neq-one)
using *assms*(*1*) **apply** *fastforce*
apply (*subst of-real-hom.map-poly-pcompose*)
apply (*subst proots-pcompose*)
apply (*auto simp*: *assms*(*3*))[*2*]
apply (*subst circle-diam-rescale*[*OF assms*(*1*)])
apply (*rule arg-cong*[*of - - proots-count* (*map-poly complex-of-real P*)])
by *fastforce*

thus *proots-count* (*map-poly complex-of-real* (*P* ∘$_p$ [:*l*, *1*:] ∘$_p$ [:*0*, *r* − *l*:]))
circle-01-diam = *0*
using *assms*(*2*) **by** *presburger*
qed
qed

5.2 One sign change case

definition *upper-circle-01* :: *complex set* **where**
upper-circle-01 = {*x*. *cmod* (*x* − (*1/2* + *sqrt*(*3*)/*6* ∗ i)) < *sqrt 3 / 3*}

lemma *upper-circle-map*:
{*x*::*complex*. *1 / x* ∈ (λ*x*. *x* + *1*) ‘ {*x*. *Im x* < *sqrt 3* ∗ *Re x*}} = *upper-circle-01*
proof
show {*x*::*complex*. *1 / x* ∈ (λ*x*. *x* + *1*) ‘ {*x*. *Im x* < *sqrt 3* ∗ *Re x*}} ⊆ *upper-circle-01*
proof
fix *x*
assume *x* ∈ {*x*. *1 / x* ∈ (λ*x*. *x* + *1*) ‘ {*x*. *Im x* < *sqrt 3* ∗ *Re x*}}
then obtain *y* **where** *1 / x* = *y* + *1* **and** *h*: *Im y* < *sqrt 3* ∗ *Re y* **by** *fastforce*
hence *hy*: *y* = *1/x* − *1* **by** *simp*
hence *hx*: *x* = *1*/(*y*+*1*) **by** *auto*
from *h* **have** *hy1*: *y* ≠ −*1* **by** *fastforce*
hence *hx0*: *x* ≠ *0* **using** *hy* **by** *fastforce*
from *h* **have** *0* < *Re* ((i + *sqrt 3*) ∗ *y*) **by** *fastforce*
hence *cmod* ((i + *sqrt 3*) ∗ *y* − *1*) < *cmod* ((i + *sqrt 3*) ∗ *y* + *1*)
by (*auto simp*: *cmod-def power2-eq-square algebra-simps*)
hence *1*: *cmod* (((i + *sqrt 3*) ∗ *y* − *1*)/((i + *sqrt 3*) ∗ *y* + *1*)) < *1*
by (*auto simp*: *norm-divide divide-simps*)
also have *cmod* (((i + *sqrt 3*) ∗ *y* − *1*)/((i + *sqrt 3*) ∗ *y* + *1*)) =
cmod (((i + *sqrt 3*) ∗ *y* ∗ *x* − *x*)/((i + *sqrt 3*) ∗ *y* ∗ *x* + *x*))
apply (*subst mult-divide-mult-cancel-right*[*symmetric*, *OF hx0*])
apply (*subst ring-distribs*(*2*)[*of - - x*])
apply (*subst left-diff-distrib*[*of - - x*])
by *simp*
also have ... = *cmod*
(((−*1* − *complex-of-real* (*sqrt 3*) − i) ∗ *x* + (*complex-of-real* (*sqrt 3*) + i)) /
((*1* − *complex-of-real* (*sqrt 3*) − i) ∗ *x* + (*complex-of-real* (*sqrt 3*) + i)))
by (*auto simp*: *hy algebra-simps hx0*)

```
    also have
      ... = cmod ((−1 − complex-of-real (sqrt 3) − i) ∗ x +
                  (complex-of-real (sqrt 3) + i)) /
            cmod (( 1 − complex-of-real (sqrt 3) − i) ∗ x +
                  (complex-of-real (sqrt 3) + i))
      by (auto simp: norm-divide)

    finally have
      cmod ((−1 − complex-of-real (sqrt 3) − i) ∗ x + (complex-of-real (sqrt 3) +
i))
       < cmod ((1 − complex-of-real (sqrt 3) − i) ∗ x + (complex-of-real (sqrt 3)
+ i))
    proof (subst divide-less-eq-1-pos[symmetric], subst zero-less-norm-iff)
      show (1 − complex-of-real (sqrt 3) − i) ∗ x + (complex-of-real (sqrt 3) + i)
≠ 0
      proof
        have −i + 1 ≠ complex-of-real (sqrt 3) by (auto simp: complex-eq-iff)
        moreover assume
          (1 − complex-of-real (sqrt 3) − i) ∗ x + (complex-of-real (sqrt 3) + i) = 0
        ultimately have
          x = (−complex-of-real (sqrt 3) − i)/(1 − complex-of-real (sqrt 3) − i)
          by (auto simp: divide-simps algebra-simps)
        thus False
          using h by (auto simp: hy field-simps Im-divide Re-divide)
      qed
    qed

    hence cmod (x − (1/2 + sqrt(3)/6 ∗ i)) < sqrt 3 / 3
      apply (subst(3) abs-of-pos[symmetric, of 3]) apply auto[1]
      apply (subst real-sqrt-abs2[symmetric], subst real-sqrt-divide[symmetric])
      apply (subst cmod-def, subst real-sqrt-less-iff)
      apply (rule mult-right-less-imp-less[of - sqrt 3 /3])
      by (auto simp: cmod-def power2-eq-square algebra-simps)

    thus x ∈ upper-circle-01
      by (auto simp: upper-circle-01-def)
  qed

  show upper-circle-01 ⊆ {x. 1 / x ∈ (λx. x + 1) ‘ {x. sqrt 3 ∗ Re x > Im x}}
  proof
    fix x assume x ∈ upper-circle-01
    hence cmod (x − (1/2 + sqrt(3)/6 ∗ i)) < sqrt 3 / 3
      by (force simp: upper-circle-01-def)
    hence sqrt ((Re x − 1/2)^2 + (Im x − sqrt(3)/6)^2) < sqrt (1/3)
      by (auto simp: cmod-def sqrt-divide-self-eq real-sqrt-inverse[symmetric])
    hence 1: − Im x ∗ sqrt 3 + (Im x ∗ (Im x ∗ 3) + Re x ∗ (Re x ∗ 3)) < Re x
∗ 3
      by (auto simp: power2-eq-square algebra-simps)
    have 2: − Im x + (Im x ∗ (Im x ∗ sqrt 3) + Re x ∗ (Re x ∗ sqrt 3)) < Re x ∗
```

```
sqrt 3
      apply (rule mult-right-less-imp-less[of - sqrt 3])
       apply (subst mult.assoc[of - sqrt 3], subst real-sqrt-mult-self)
      using 1 by (auto simp: algebra-simps)
    have sqrt 3 + (−Im x) / (Im x * Im x + Re x * Re x) <
         Re x * sqrt 3 / (Im x * Im x + Re x * Re x)
      apply (rule mult-right-less-imp-less[of - (Im x * Im x + Re x * Re x)])
       apply (rule subst, rule arg-cong2[of - - - - (<)])
         prefer 3 apply (rule 2)
        apply (subst distrib-right)
      using 2 apply auto
      by (auto simp: algebra-simps)

    hence 0 < − Im (1/x−1) + sqrt 3 * Re (1/x−1)
      by (auto simp: power2-eq-square algebra-simps Re-divide Im-divide)
    hence sqrt 3 * Re (1/x−1) > Im (1/x−1)
      by argo
    hence (1/x−1) ∈ {x. sqrt 3 * Re x > Im x} by fast
    moreover have 1/x = (1/x−1) + 1 by simp
    ultimately show x ∈ {x. 1 / x ∈ (λx. x + 1) ‘ {x. sqrt 3 * Re x > Im x}}
      by blast
  qed
qed

definition lower-circle-01 :: complex set where
lower-circle-01 = {x. cmod (x − (1/2 − sqrt(3)/6 * i)) < sqrt 3 / 3}

lemma cnj-upper-circle-01: cnj ‘ upper-circle-01 = lower-circle-01
proof
  show cnj ‘ upper-circle-01 ⊆ lower-circle-01
  proof
    fix x assume x ∈ cnj ‘ upper-circle-01
    then obtain y where y ∈ upper-circle-01 and x = cnj y by blast
    thus x ∈ lower-circle-01
      apply (subst lower-circle-01-def, subst complex-mod-cnj[symmetric])
      by (auto simp add: upper-circle-01-def del: complex-mod-cnj)
  qed
  show lower-circle-01 ⊆ cnj ‘ upper-circle-01
  proof
    fix x assume x ∈ lower-circle-01
    hence cnj x ∈ upper-circle-01 and x = cnj (cnj x)
      apply (subst upper-circle-01-def, subst complex-mod-cnj[symmetric])
      by (auto simp add: lower-circle-01-def del: complex-mod-cnj)
    thus x ∈ cnj ‘ upper-circle-01
      by blast
  qed
qed

lemma lower-circle-map:
```

```
  {x::complex. 1 / x ∈ (λx. x + 1) ‘ {x. Im x > − sqrt 3 ∗ Re x}} = lower-circle-01
proof (subst cnj-upper-circle-01[symmetric], subst upper-circle-map[symmetric])
  have {x. 1 / x ∈ (λx. x + 1) ‘ {x. − sqrt 3 ∗ Re x < Im x}} =
        {x. 1 / x ∈ (λx. x + 1) ‘ {x. sqrt 3 ∗ Re (cnj x) > Im (cnj x)}}
    by auto
  also have ... = {x. 1 / x ∈ (λx. x + 1) ‘ cnj ‘ {x. sqrt 3 ∗ Re x > Im x}}
    apply (subst(2) bij-image-Collect-eq)
     apply (metis bijI' complex-cnj-cnj)
    by (auto simp: inj-def inj-imp-inv-eq[of - cnj])
  also have ... = {x. 1 / x ∈ cnj ‘ (λx. x + 1) ‘ {x. sqrt 3 ∗ Re x > Im x}}
    by (auto simp: image-image)
  also have ... = {x. cnj (1 / x) ∈ (λx. x + 1) ‘ {x. sqrt 3 ∗ Re x > Im x}}
    by (metis (no-types, lifting) complex-cnj-cnj image-iff)
  also have ... = cnj ‘ {x. 1 / x ∈ (λx. x + 1) ‘ {x. sqrt 3 ∗ Re x > Im x}}
    apply (subst(2) bij-image-Collect-eq)
     apply (metis bijI' complex-cnj-cnj)
    by (auto simp: inj-def inj-imp-inv-eq[of - cnj])
  finally show {x. 1 / x ∈ (λx. x + 1) ‘ {x. − sqrt 3 ∗ Re x < Im x}} =
          cnj ‘ {x. 1 / x ∈ (λx. x + 1) ‘ {x. Im x < sqrt 3 ∗ Re x}} .
qed

lemma two-circles-01:
  fixes P::real poly
  assumes hP: degree P ≤ p and hP0: P ≠ 0 and hp0: p ≠ 0
  and h: proots-count (map-poly of-real P)
         (upper-circle-01 ∪ lower-circle-01) = 1
shows Bernstein-changes-01 p P = 1
proof (subst Bernstein-changes-01-eq-changes[OF hP])
  let ?Q = reciprocal-poly p P ∘p [:1, 1:]
  have hQ0: ?Q ≠ 0 using hP0
    by (simp add: pcompose-eq-0 hP reciprocal-0-iff)

  from h obtain x' where hroot': poly (map-poly of-real P) x' = 0
    and hx':x' ∈ upper-circle-01 ∪ lower-circle-01
    using proots-count-pos by (metis less-numeral-extra(1))

  obtain x where hxx': x' = complex-of-real x
  proof (cases Im x' = 0)
    assume Im x' = 0 and h: ⋀x. x' = complex-of-real x ⟹ thesis
    then show thesis using h[of Re x'] by (simp add: complex-is-Real-iff)
  next
    assume hx'': Im x' ≠ 0
    have 1: card {x', cnj x'} = 2
    proof (subst card-2-iff)
      from hx'' have x' ≠ cnj x' and {x', cnj x'} = {x', cnj x'}
        by (metis cnj.simps(2) neg-equal-zero, argo)
      thus ∃x y. {x', cnj x'} = {x, y} ∧ x ≠ y by blast
    qed
    moreover have {x', cnj x'} ⊆ upper-circle-01 ∪ lower-circle-01 using hx'
```

```
    apply (rule UnE)
    by (auto simp: cnj-upper-circle-01[symmetric])
  moreover have ⋀x. x ∈ {x', cnj x'} ⟹ poly (map-poly of-real P) x = 0
    using hroot' poly-map-poly-of-real-cnj by auto
  ultimately have
    proots-count (map-poly of-real P) (upper-circle-01 ∪ lower-circle-01) ≥ 2
    apply (subst 1[symmetric])
    apply (rule proots-count-of-root-set)
    using assms(2) of-real-poly-eq-0-iff by (blast, blast, blast)
  thus thesis using assms(4) by linarith
qed
hence hroot: poly P x = 0
  using hroot' by (metis of-real-0 of-real-eq-iff of-real-poly-map-poly)
have that: 3 * sqrt (x * x + 1 / 3 - x) < sqrt 3 using hx'
  apply (rule UnE)
  by (auto simp: cmod-def power2-eq-square algebra-simps upper-circle-01-def
      lower-circle-01-def hxx')
have hx: 0 < x ∧ x < 1
proof -
  have 3 * sqrt (x * x + 1 / 3 - x) = sqrt (9 * (x * x + 1 / 3 - x))
    by (subst real-sqrt-mult, simp)
  hence 9 * (x * x + 1 / 3 - x) < 3 using that real-sqrt-less-iff by metis
  hence x*x - x < 0 by auto
  thus 0 < x ∧ x < 1
    using mult-eq-0-iff mult-less-cancel-right-disj by fastforce
qed

let ?y = 1/x - 1
from hroot hx assms have poly ?Q ?y = 0
  by (auto simp: poly-pcompose poly-reciprocal)
hence [:-?y, 1:] dvd ?Q using poly-eq-0-iff-dvd by blast
then obtain R where hR: ?Q = R * [:-?y, 1:] by auto
hence hR0: R ≠ 0 using hQ0 by force
interpret map-poly-idom-hom complex-of-real ..


have normal-poly (smult (inverse (lead-coeff (reciprocal-poly p P ∘p [:1, 1:]))) R)
proof (rule normal-poly-of-roots)
  show ⋀z. poly (map-poly complex-of-real
      (smult (inverse (lead-coeff (reciprocal-poly p P ∘p [:1, 1:]))) R)) z = 0 ⟹
      Re z ≤ 0 ∧ (cmod z)^2 ≤ 4 * (Re z)^2
  proof -
    fix z
    assume
      poly (map-poly complex-of-real
       (smult (inverse (lead-coeff (reciprocal-poly p P ∘p [:1, 1:]))) R)) z = 0
    hence hroot2: poly (map-poly complex-of-real R) z = 0
      by (auto simp: map-poly-smult hQ0)
    show Re z ≤ 0 ∧ (cmod z)^2 ≤ 4 * (Re z)^2
```

proof (*rule ccontr*)
assume $\neg\ (Re\ z \leq 0 \wedge (cmod\ z)^2 \leq 4 * (Re\ z)^2)$
hence *1*: $0 < Re\ z \vee 4 * (Re\ z)^2 < (cmod\ z)^2$ **by** *linarith*
hence *hz*: $z \neq -1$ **by** *force*
have $Im\ z > -\ sqrt\ 3 * Re\ z \vee sqrt\ 3 * Re\ z > Im\ z$
proof (*cases* $Im\ z \geq sqrt\ 3 * Re\ z$, *cases* $-\ sqrt\ 3 * Re\ z \geq Im\ z$)
assume *2*: $sqrt\ 3 * Re\ z \leq Im\ z\ Im\ z \leq -\ sqrt\ 3 * Re\ z$
hence $sqrt\ 3 * Re\ z \leq sqrt\ 3 * -\ Re\ z$ **by** *force*
hence $Re\ z \leq -\ Re\ z$
apply (*rule mult-left-le-imp-le*)
by *fastforce*
hence $Re\ z \leq 0$ **by** *simp*
moreover have $(Im\ z)\hat{}2 \leq (-sqrt\ 3 * Re\ z)\hat{}2$
apply (*subst power2-eq-square*, *subst power2-eq-square*)
apply (*rule square-bounded-le*)
using *2* **by** *auto*
ultimately have *False* **using** *1*
by (*auto simp*: *power2-eq-square cmod-def algebra-simps*)
thus *?thesis* **by** *fast*
qed *auto*

hence $z \in \{z.\ -\ sqrt\ 3 * Re\ z < Im\ z\} \cup \{z.\ Im\ z < sqrt\ 3 * Re\ z\}$
by *blast*

hence *1*: $inverse\ (1 + z) \in$ *upper-circle-01* $\cup$ *lower-circle-01*
by (*force simp*: *inverse-eq-divide upper-circle-map*[*symmetric*]
lower-circle-map[*symmetric*])

have *hRdeg'*: $degree\ R < p$
apply (*rule less-le-trans*[*of degree R degree ?Q*])
apply (*subst hR*, *subst degree-mult-eq*[*OF hR0*], *fastforce*, *fastforce*)
by (*auto simp*: *degree-pcompose degree-reciprocal hP*)
hence *hRdeg*: $degree\ R \leq p$ **by** *fastforce*
have *2*: $map\text{-}poly\ complex\text{-}of\text{-}real\ (reciprocal\text{-}poly\ p\ (R \circ_p [:-1, 1:])) \neq 0$
apply (*subst of-real-poly-eq-0-iff*, *subst reciprocal-0-iff*)
apply (*force simp*: *hRdeg degree-pcompose*)
using *hR0 pcompose-eq-0*
by (*metis degree-eq-zeroE gr-zeroI pCons-eq-iff pCons-one zero-neq-one*)
have *3*:
$poly\ (map\text{-}poly\ complex\text{-}of\text{-}real\ (reciprocal\text{-}poly\ p\ (R \circ_p [:-1, 1:])))$
$(inverse\ (1 + z)) = 0$
apply (*subst map-poly-reciprocal*)
using *hRdeg* **apply** (*force simp*: *degree-pcompose*)
apply *auto*[*1*]
apply (*subst poly-reciprocal*)
using *hRdeg* **apply** (*force simp*: *degree-map-poly degree-pcompose*)
using *hz* **apply** (*metis inverse-nonzero-iff-nonzero neg-eq-iff-add-eq-0*)
by (*auto simp*: *of-real-hom.map-poly-pcompose poly-pcompose hroot2*)

```
    have proots-count (map-poly of-real (reciprocal-poly p (R ∘p [:−1, 1:])))
          (upper-circle-01 ∪ lower-circle-01) > 0
      by (rule proots-count-of-root[OF 2 1 3])
    moreover have proots-count
                (map-poly complex-of-real
                (reciprocal-poly p ([:1 − 1 / x, 1:] ∘p [:− 1, 1:])))
                (upper-circle-01 ∪ lower-circle-01) > 0
      apply (subst map-poly-reciprocal)
       using hp0 less-eq-Suc-le apply (simp add: degree-pcompose)
      apply simp
       apply (subst proots-count-reciprocal)
       using hp0 less-eq-Suc-le
          apply (simp add: degree-pcompose degree-map-poly)
         apply (auto simp: pcompose-pCons)[1]
        apply (auto simp: cmod-def power2-eq-square real-sqrt-divide
              real-div-sqrt upper-circle-01-def lower-circle-01-def)[1]
       apply (subst of-real-hom.map-poly-pcompose)
       apply (subst proots-pcompose, fastforce, force)
       apply (subst lower-circle-map[symmetric])
       apply (subst upper-circle-map[symmetric])
       apply (rule proots-count-of-root[of - of-real (1/x − 1)])
        apply simp
       apply (auto simp: bij-image-Collect-eq bij-def inj-def image-iff
                   inverse-eq-divide inj-imp-inv-eq[of - λ x. x+1] hx simp del:
surj-plus-right)[1]
       by force

      ultimately have proots-count
           (map-poly complex-of-real (reciprocal-poly p (R ∘p [:− 1, 1:])))
           (upper-circle-01 ∪ lower-circle-01) +
       proots-count
        (map-poly complex-of-real
          (reciprocal-poly p ([:1 − 1 / x, 1:] ∘p [:− 1, 1:])))
        (upper-circle-01 ∪ lower-circle-01) > 1
      by fastforce
    also have ... = proots-count (map-poly complex-of-real
            (monom 1 p ∗ reciprocal-poly p (?Q ∘p [:− 1, 1:])))
     (upper-circle-01 ∪ lower-circle-01)
      apply (subst eq-commute, subst hR, subst pcompose-mult)
      apply (subst reciprocal-mult, subst degree-mult-eq)
      using hR0 apply (fastforce simp: pcompose-eq-0)
        apply (fastforce simp: pcompose-pCons)
      using hRdeg′ apply (simp add: degree-pcompose)
      using hRdeg apply (simp add: degree-pcompose)
      using hp0 apply (auto simp: degree-pcompose)[1]
      apply (subst hom-mult)
      apply (subst proots-count-times)
      using hp0 hRdeg′ hR0
      apply (fastforce simp: reciprocal-0-iff degree-pcompose pcompose-eq-0
```

```
        pcompose-pCons)
    by simp
   also have ... = proots-count
    (map-poly complex-of-real
     (reciprocal-poly p (reciprocal-poly p P ∘p [:1, 1:] ∘p [:− 1, 1:])))
    (upper-circle-01 ∪ lower-circle-01)
    apply (subst hom-mult)
    apply (subst proots-count-times)
    using hp0 hP hP0
    apply (auto simp: map-poly-reciprocal degree-pcompose
        degree-reciprocal of-real-hom.map-poly-pcompose
        reciprocal-0-iff degree-map-poly pcompose-eq-0)[1]
    apply (subst map-poly-monom, fastforce)
    apply (subst of-real-1, subst proots-count-monom)
    apply (auto simp: cmod-def power2-eq-square real-sqrt-divide
        real-div-sqrt upper-circle-01-def lower-circle-01-def)[1]
    by presburger
   also have ... = 1
    by (auto simp: pcompose-assoc[symmetric] pcompose-pCons
      reciprocal-reciprocal hP h)
   finally show False by blast
  qed
 qed
 show lead-coeff
     (smult (inverse (lead-coeff (reciprocal-poly p P ∘p [:1, 1:]))) R) = 1
  by (auto simp: hR degree-add-eq-right hR0 coeff-eq-0)
qed

 hence changes (coeffs (smult (inverse (lead-coeff ?Q)) ?Q)) = 1
  apply (subst hR, subst mult-smult-left[symmetric], rule normal-changes)
  by (auto simp: hx)

 moreover have changes (coeffs (reciprocal-poly p P ∘p [:1, 1:])) =
   changes (coeffs (smult (inverse (lead-coeff (reciprocal-poly p P ∘p [:1, 1:]))))
    (reciprocal-poly p P ∘p [:1, 1:])))
  by (auto simp: pcompose-eq-0 reciprocal-0-iff hP hP0 coeffs-smult
    changes-scale-const[symmetric])

 ultimately show changes (coeffs (reciprocal-poly p P ∘p [:1, 1:])) = 1 by argo
qed

definition upper-circle :: real ⇒ real ⇒ complex set where
upper-circle l r = {x::complex.
 cmod ((x−of-real l)/(of-real (r−l)) − (1/2 + of-real (sqrt(3))/6 * i)) < sqrt 3 /
3}

lemma upper-circle-rescale: assumes l < r
 shows upper-circle l r = (λ x . (x*(r − l)) + l)) ‘ upper-circle-01
proof
```

show *upper-circle l r* ⊆
(λ*x*. *x* ∗ (*of-real r* − *of-real l*) + *of-real l*) ‘ *upper-circle-01*
apply (*rule subsetI*)
apply (*rule image-eqI*[*of* - - (- − *of-real l*)/(*of-real r* − *of-real l*)])
using *assms* **apply** (*auto simp*: *divide-simps*)[*1*]
by (*auto simp*: *upper-circle-01-def upper-circle-def*)
show (λ*x*. *x* ∗ (*of-real r* − *of-real l*) + *of-real l*) ‘ *upper-circle-01* ⊆
upper-circle l r
apply (*rule subsetI*, *subst*(*asm*) *image-iff*)
using *assms* **by** (*auto simp*: *upper-circle-01-def upper-circle-def*)
qed

definition *lower-circle* :: *real* ⇒ *real* ⇒ *complex set* **where**
lower-circle l r = {*x*::*complex*.
cmod ((*x*−*of-real l*)/(*of-real* (*r*−*l*)) − (*1/2* − *of-real* (*sqrt*(*3*))/*6* ∗ i)) < *sqrt 3* / *3*}

lemma *lower-circle-rescale*:
assumes *l* < *r*
shows *lower-circle l r* = (λ *x* . (*x*∗(*r* − *l*) + *l*)) ‘ *lower-circle-01*
proof
show *lower-circle l r* ⊆ (λ*x*. *x* ∗ (*of-real r* − *of-real l*) + *of-real l*) ‘
lower-circle-01
apply (*rule subsetI*)
apply (*rule image-eqI*[*of* - - (- − *of-real l*)/(*of-real r* − *of-real l*)])
using *assms* **apply** (*auto simp*: *divide-simps*)[*1*]
by (*auto simp*: *lower-circle-01-def lower-circle-def*)
show (λ*x*. *x* ∗ (*of-real r* − *of-real l*) + *of-real l*) ‘ *lower-circle-01* ⊆
lower-circle l r
apply (*rule subsetI*, *subst*(*asm*) *image-iff*)
using *assms* **by** (*auto simp*: *lower-circle-01-def lower-circle-def*)
qed

lemma *two-circles*:
fixes *P*::*real poly* **and** *l r*::*real*
assumes *hlr*: *l* < *r*
and *hP*: *degree P* ≤ *p*
and *hP0*: *P* ≠ *0*
and *hp0*: *p* ≠ *0*
and *h*: *proots-count* (*map-poly of-real P*)
(*upper-circle l r* ∪ *lower-circle l r*) = *1*
shows *Bernstein-changes p l r P* = *1*
proof −
have *1*: *Bernstein-changes p l r P* =
Bernstein-changes-01 p (*P* ∘$_p$ [:*l*, *1*:] ∘$_p$ [:*0*, *r* − *l*:])
using *assms* **by** (*force simp*: *Bernstein-changes-eq-rescale*)
have *proots-count* (*map-poly complex-of-real* (*P* ∘$_p$ [:*l*, *1*:] ∘$_p$ [:*0*, *r* − *l*:]))
(*upper-circle-01* ∪ *lower-circle-01*) = *1*
using *assms*

by (*auto simp*: *upper-circle-rescale lower-circle-rescale proots-pcompose image-Un of-real-hom.map-poly-pcompose pcompose-eq-0 image-image algebra-simps*)
thus *?thesis*
apply (*subst 1*)
apply (*rule two-circles-01*)
using *hP* **apply** (*force simp*: *degree-pcompose*)
using *hP0 hlr* **apply** (*fastforce simp*: *pcompose-eq-0*)
using *hp0* **apply** *blast*
by *blast*
qed

5.3 The theorem of three circles

theorem *three-circles*:
fixes *P*::*real poly* **and** *l r*::*real*
assumes $l < r$
and *hP*: *degree* $P \leq p$
and *hP0*: $P \neq 0$
and *hp0*: $p \neq 0$
shows *proots-count* (*map-poly of-real P*) (*circle-diam l r*) $= 0 \Longrightarrow$
Bernstein-changes p l r P $= 0$
and *proots-count* (*map-poly of-real P*)
(*upper-circle l r* $\cup$ *lower-circle l r*) $= 1 \Longrightarrow$
Bernstein-changes p l r P $= 1$
apply (*rule one-circle*)
using *assms* **apply** *auto*[*4*]
apply (*rule two-circles*)
using *assms* **by** *auto*

end

6 Acknowledgements

The work has been jointly supported by the Cambridge Mathematics Placements (CMP) Programme and the ERC Advanced Grant ALEXANDRIA (Project GA 742178).

References

[1] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[2] A. Eigenwillig. Real root isolation for exact and approximate polynomials using descartes' rule of signs. 2008.

[3] W. Li and L. C. Paulson. Counting polynomial roots in isabelle/hol: a formal proof of the budan–fourier theorem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 52–64, 2019.

[4] J. Zsidó. Theorem of three circles in coq. *Journal of automated reasoning*, 53(2):105–127, 2014.