

Count the Number of Complex Roots

Wenda Li

February 6, 2026

Abstract

Based on evaluating Cauchy indices through remainder sequences [1] [2, Chapter 11], this entry provides an effective procedure to count the number of complex roots (with multiplicity) of a polynomial within a rectangle box or a half-plane. Potential applications of this entry include certified complex root isolation (of a polynomial) and testing the Routh-Hurwitz stability criterion (i.e., to check whether all the roots of some characteristic polynomial have negative real parts).

1 Extra lemmas related to polynomials

```
theory CC-Polynomials-Extra imports  
  Winding-Number-Eval.Missing-Algebraic  
  Winding-Number-Eval.Missing-Transcendental  
  Sturm-Tarski.PolyMisc  
  Budan-Fourier.BF-Misc  
  Polynomial-Interpolation.Ring-Hom-Poly  
begin
```

1.1 Misc

```
lemma poly-linepath-comp':  
  fixes a::'a::{real-normed-vector,comm-semiring-0,real-algebra-1}  
  shows poly p (linepath a b t) = poly (p ∘p [:a, b-a:]) (of-real t)  
  by (auto simp add:poly-pcompose linepath-def scaleR-conv-of-real algebra-simps)
```

```
lemma path-poly-comp[intro]:  
  fixes p::'a::real-normed-field poly  
  shows path g ⇒ path (poly p o g)  
  apply (elim path-continuous-image)  
  by (auto intro:continuous-intros)
```

```
lemma cindex-poly-noroot:  
  assumes  $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$   
  shows cindex-poly a b q p = 0  
  unfolding cindex-poly-def  
  by (metis (lifting) assms mem-Collect-eq sum.neutral)
```

1.2 More polynomial homomorphism interpretations

interpretation *of-real-poly-hom:map-poly-inj-idom-hom of-real ..*

interpretation *Re-poly-hom:map-poly-comm-monoid-add-hom Re*
by *unfold-locales simp-all*

interpretation *Im-poly-hom:map-poly-comm-monoid-add-hom Im*
by *unfold-locales simp-all*

1.3 More about order

lemma *order-normalize[simp]:order x (normalize p) = order x p*
by *(metis dvd-normalize-iff normalize-eq-0-iff order-1 order-2 order-unique-lemma)*

lemma *order-gcd:*

assumes $p \neq 0$ $q \neq 0$

shows $\text{order } x (\text{gcd } p \ q) = \min (\text{order } x \ p) (\text{order } x \ q)$

proof –

define $xx \ op \ oq$ **where** $xx = [- \ x, \ 1:]$ **and** $op = \text{order } x \ p$ **and** $oq = \text{order } x \ q$

obtain pp **where** $pp : p = xx \ ^{op} * pp \ \neg \ xx \ \text{dvd} \ pp$

using *order-decomp[OF <p≠0>,of x,folded xx-def op-def]* **by** *auto*

obtain qq **where** $qq : q = xx \ ^{oq} * qq \ \neg \ xx \ \text{dvd} \ qq$

using *order-decomp[OF <q≠0>,of x,folded xx-def oq-def]* **by** *auto*

define pq **where** $pq = \text{gcd } pp \ qq$

have $p\text{-unfold} : p = (pq * xx \ ^{(\min \ op \ oq)}) * ((pp \ \text{div} \ pq) * xx \ ^{(op - \min \ op \ oq)})$

and $[simp] : \text{coprime } xx \ (pp \ \text{div} \ pq)$ **and** $pp \neq 0$

proof –

have $xx \ ^{op} = xx \ ^{(\min \ op \ oq)} * xx \ ^{(op - \min \ op \ oq)}$

by *(simp flip:power-add)*

moreover have $pp = pq * (pp \ \text{div} \ pq)$

unfolding $pq\text{-def}$ **by** *simp*

ultimately show $p = (pq * xx \ ^{(\min \ op \ oq)}) * ((pp \ \text{div} \ pq) * xx \ ^{(op - \min \ op \ oq)})$

unfolding $pq\text{-def } pp$ **by** *(auto simp:algebra-simps)*

show $\text{coprime } xx \ (pp \ \text{div} \ pq)$

apply *(rule prime-elem-imp-coprime[OF prime-elem-linear-poly[of 1 -x,simplified],folded xx-def])*

using $\langle pp = pq * (pp \ \text{div} \ pq) \rangle pp(2)$ **by** *auto*

qed *(use pp <p≠0> in auto)*

have $q\text{-unfold} : q = (pq * xx \ ^{(\min \ op \ oq)}) * ((qq \ \text{div} \ pq) * xx \ ^{(oq - \min \ op \ oq)})$

and $[simp] : \text{coprime } xx \ (qq \ \text{div} \ pq)$

proof –

have $xx \ ^{oq} = xx \ ^{(\min \ op \ oq)} * xx \ ^{(oq - \min \ op \ oq)}$

by *(simp flip:power-add)*

moreover have $qq = pq * (qq \ \text{div} \ pq)$

unfolding $pq\text{-def}$ **by** *simp*

```

ultimately show  $q = (pq * xx \wedge (\min \text{ op } oq)) * ((qq \text{ div } pq) * xx \wedge (oq - \min \text{ op } oq))$ 
  unfolding pq-def qq by (auto simp: algebra-simps)
show coprime  $xx$  (qq div pq)
  apply (rule prime-elem-imp-coprime[OF
    prime-elem-linear-poly[of 1 - x, simplified], folded xx-def])
  using <math>qq = pq * (qq \text{ div } pq)> qq(2) by auto
qed

have gcd p q = normalize (pq * xx \wedge (\min \text{ op } oq))
proof -
  have coprime (pp div pq * xx \wedge (op - \min \text{ op } oq)) (qq div pq * xx \wedge (oq - \min \text{ op } oq))
  proof (cases op > oq)
    case True
    then have  $oq - \min \text{ op } oq = 0$  by auto
    moreover have coprime (xx \wedge (op - \min \text{ op } oq)) (qq div pq) by auto
    moreover have coprime (pp div pq) (qq div pq)
      using <math>pp \neq 0> div-gcd-coprime pq-def by blast
    ultimately show ?thesis by auto
  next
  case False
  then have  $op - \min \text{ op } oq = 0$  by auto
  moreover have coprime (pp div pq) (xx \wedge (oq - \min \text{ op } oq))
    by (auto simp: coprime-commute)
  moreover have coprime (pp div pq) (qq div pq)
    using <math>pp \neq 0> div-gcd-coprime pq-def by blast
  ultimately show ?thesis by auto
  qed
  then show ?thesis unfolding p-unfold q-unfold
    by (simp add: gcd-mult-left)
  qed
then have order x (gcd p q) = order x pq + order x (xx \wedge (\min \text{ op } oq))
  by (metis assms(1) mult-zero-left order-mult order-normalize p-unfold)
also have ... = order x (xx \wedge (\min \text{ op } oq))
  using pp(2) qq(2) unfolding pq-def xx-def
  by (auto simp add: order-0I poly-eq-0-iff-dvd)
also have ... = min op oq
  unfolding xx-def by (rule order-power-n-n)
also have ... = min (order x p) (order x q) unfolding op-def oq-def by simp
finally show ?thesis .
qed

lemma pderiv-power: pderiv (p \wedge n) = smult (of-nat n) (p \wedge (n-1)) * pderiv p
  apply (cases n)
  using pderiv-power-Suc by auto

```

thm order-pderiv

```

lemma order-pderiv:
  fixes p::'a::{idom,semiring-char-0} poly
  assumes p≠0 poly p x=0
  shows order x p = Suc (order x (pderiv p)) using assms
proof -
  define xx op where xx=[:- x, 1:] and op = order x p
  have op ≠ 0 unfolding op-def using assms order-root by blast
  obtain pp where pp:p = xx ^ op * pp ↯ xx dvd pp
    using order-decomp[OF ‹p≠0›,of x,folded xx-def op-def] by auto
  have p-der:pderiv p = smult (of-nat op) (xx^(op-1)) * pp + xx^op*pderiv
pp
    unfolding pp(1) by (auto simp:pderiv-mult pderiv-power xx-def alge-
bra-simps pderiv-pCons)
  have xx^(op-1) dvd (pderiv p)
    unfolding p-der
  by (metis ‹op ≠ 0› dvd-add-left-iff dvd-mult2 dvd-refl dvd-smult dvd-triv-right
power-eq-iff)
  moreover have ↯ xx^op dvd (pderiv p)
proof
  assume xx ^ op dvd pderiv p
  then have xx ^ op dvd smult (of-nat op) (xx^(op-1) * pp)
    unfolding p-der by (simp add: dvd-add-left-iff)
  then have xx ^ op dvd (xx^(op-1)) * pp
    apply (elim dvd-monic[rotated])
    using ‹op≠0› by (auto simp:lead-coeff-power xx-def)
  then have xx ^ (op-1) * xx dvd (xx^(op-1))
    using ‹↯ xx dvd pp› by (simp add: ‹op ≠ 0› mult.commute power-eq-iff)
  then have xx dvd 1
    using assms(1) pp(1) by auto
  then show False unfolding xx-def by (meson assms(1) dvd-trans one-dvd
order-decomp)
qed
  ultimately have op - 1 = order x (pderiv p)
    using order-unique-lemma[of x op-1 pderiv p,folded xx-def] ‹op≠0›
  by auto
  then show ?thesis using ‹op≠0› unfolding op-def by auto
qed

```

1.4 More about *rsquarefree*

```

lemma rsquarefree-0[simp]: ↯ rsquarefree 0
  unfolding rsquarefree-def by simp

```

```

lemma rsquarefree-times:
  assumes rsquarefree (p*q)
  shows rsquarefree q using assms
proof (induct p rule:poly-root-induct-alt)
  case 0
  then show ?case by simp

```

```

next
  case (no-roots p)
  then have [simp]: p ≠ 0 q ≠ 0 ∧ a. order a p = 0
    using order-0I by auto
  obtain order a (p * q) = 0 ↔ order a q = 0
    order a (p * q) = 1 ↔ order a q = 1 for a
    by (simp add: order-mult)
  then show ?case using ‹rsquarefree (p * q)›
    unfolding rsquarefree-def by simp
next
  case (root a p)
  define pq aa where pq = p * q and aa = [:- a, 1:]
  obtain [simp]: pq ≠ 0 aa ≠ 0 order a aa = 1
    using aa-def pq-def root.premis by force
  have rsquarefree (aa * pq)
    unfolding aa-def pq-def using root(2) by (simp add: algebra-simps)
  then have rsquarefree pq
    unfolding rsquarefree-def by (auto simp add: order-mult)
  from root(1)[OF this[unfolding pq-def]] show ?case .
qed

lemma rsquarefree-smult-iff:
  assumes s ≠ 0
  shows rsquarefree (smult s p) ↔ rsquarefree p
  unfolding rsquarefree-def using assms by (auto simp add: order-smult)

lemma card-roots-within-rsquarefree:
  assumes rsquarefree p
  shows proots-count p s = card (proots-within p s) using assms
proof (induct rule: poly-root-induct[of - λx. x ∈ s])
  case 0
  then have False by simp
  then show ?case by simp
next
  case (no-roots p)
  then show ?case
  by (metis all-not-in-conv card.empty proots-count-def proots-within-iff sum.empty)
next
  case (root a p)
  have proots-count ([:a, - 1:] * p) s = 1 + proots-count p s
  proof (subst proots-count-times)
    show [:a, - 1:] * p ≠ 0
      using root.premis rsquarefree-def by blast
    show proots-count [:a, - 1:] s + proots-count p s = 1 + proots-count p s
      by (metis (no-types, opaque-lifting) add.inverse-inverse add.inverse-neutral
        minus-pCons
        proots-count-pCons-1-iff proots-count-uminus root.hyps(1))
  qed
  also have ... = 1 + card (proots-within p s)

```

```

    by (metis root.hyps(2) root.premis rsquarefree-times)
  also have ... = card (roots-within ([:a, - 1:] * p) s) unfolding roots-within-times

proof (subst card-Un-disjoint)
  have [simp]:p≠0 using root.premis by auto
  show finite (roots-within [:a, - 1:] s) finite (roots-within p s)
    by auto
  show 1 + card (roots-within p s) = card (roots-within [:a, - 1:] s)
    + card (roots-within p s)
    using ⟨a ∈ s⟩
    by (simp add: roots-within-pCons-1-iff(2))
  have poly p a≠0
proof (rule ccontr)
  assume ¬ poly p a ≠ 0
  then have order a p > 0 by (simp add: order-root)
  moreover have order a [:a, - 1:] = 1
    by (metis add.inverse-inverse add.inverse-neutral minus-pCons order-linear
      order-uminus)
  ultimately have order a ([:a, - 1:] * p) > 1
    by (metis root.premis order-mult add-cancel-right-right linorder-neqE-nat
      not-add-less1 rsquarefree-0)
  then show False using ⟨rsquarefree ([:a, - 1:] * p)⟩
    unfolding rsquarefree-def using gr-implies-not0 less-not-refl2 by blast
  qed
  then show roots-within [:a, - 1:] s ∩ roots-within p s = {}
    using roots-within-pCons-1-iff(2) by auto
  qed
  finally show ?case .
qed

lemma rsquarefree-gcd-pderiv:
  fixes p::'a::{factorial-ring-gcd,semiring-gcd-mult-normalize,semiring-char-0} poly
  assumes p≠0
  shows rsquarefree (p div (gcd p (pderiv p)))
proof (cases pderiv p = 0)
  case True
  have poly (unit-factor p) x ≠ 0 for x
    using unit-factor-is-unit[OF ⟨p≠0⟩]
    by (meson assms dvd-trans order-decomp poly-eq-0-iff-dvd unit-factor-dvd)
  then have order x (unit-factor p) = 0 for x
    using order-0I by blast
  then show ?thesis using True ⟨p≠0⟩ unfolding rsquarefree-def by simp
  next
  case False
  define q where q = p div (gcd p (pderiv p))
  have q≠0 unfolding q-def by (simp add: assms dvd-div-eq-0-iff)

  have order-pq:order x p = order x q + min (order x p) (order x (pderiv p))
    for x

```

```

proof –
  have  $p = q * \text{gcd } p \text{ (pderiv } p)$ 
    unfolding  $q\text{-def}$  by  $\text{simp}$ 
  then show  $?thesis$ 
    by ( $\text{metis } CC\text{-Polynomials-Extra.order-gcd } False \text{ assms } \text{order-mult}$ )
qed
have  $\text{order } x \ q = 0 \vee \text{order } x \ q = 1$  for  $x$ 
proof ( $\text{cases } \text{poly } p \ x = 0$ )
  case  $True$ 
    from  $\text{order-pderiv}[OF \ \langle p \neq 0 \rangle \ \text{this}]$ 
    have  $\text{order } x \ p = \text{order } x \ (\text{pderiv } p) + 1$  by  $\text{simp}$ 
    then show  $?thesis$  using  $\text{order-pq}[of \ x]$  by  $\text{auto}$ 
  next
    case  $False$ 
    then have  $\text{order } x \ p = 0$  by ( $\text{simp add: } \text{order-0I}$ )
    then have  $\text{order } x \ q = 0$  using  $\text{order-pq}[of \ x]$  by  $\text{simp}$ 
    then show  $?thesis$  by  $\text{simp}$ 
  qed
then show  $?thesis$  using  $\langle q \neq 0 \rangle$  unfolding  $\text{rsquarefree-def } q\text{-def}$ 
  by  $\text{auto}$ 
qed

lemma  $\text{poly-gcd-pderiv-iff}$ :
  fixes  $p::'a::\{\text{semiring-char-0}, \text{factorial-ring-gcd}, \text{semiring-gcd-mult-normalize}\}$   $\text{poly}$ 
  shows  $\text{poly } (p \ \text{div } (\text{gcd } p \ (\text{pderiv } p))) \ x = 0 \iff \text{poly } p \ x = 0$ 
proof ( $\text{cases } \text{pderiv } p = 0$ )
  case  $True$ 
    then obtain  $a$  where  $p = [a]$  using  $\text{pderiv-iszero}$  by  $\text{auto}$ 
    then show  $?thesis$  by ( $\text{auto simp add: } \text{unit-factor-poly-def}$ )
  next
    case  $False$ 
    then have  $p \neq 0$  using  $\text{pderiv-0}$  by  $\text{blast}$ 
    define  $q$  where  $q = p \ \text{div } (\text{gcd } p \ (\text{pderiv } p))$ 
    have  $q \neq 0$  unfolding  $q\text{-def}$  by ( $\text{simp add: } \langle p \neq 0 \rangle \ \text{dvd-div-eq-0-iff}$ )

have  $\text{order-pq:order } x \ p = \text{order } x \ q + \min (\text{order } x \ p) (\text{order } x \ (\text{pderiv } p))$  for  $x$ 
proof –
  have  $*:p = q * \text{gcd } p \ (\text{pderiv } p)$ 
    unfolding  $q\text{-def}$  by  $\text{simp}$ 
  show  $?thesis$ 
    by ( $\text{metis } * \ CC\text{-Polynomials-Extra.order-gcd } False \ \langle p \neq 0 \rangle \ \text{order-mult}$ )
qed
have  $\text{order } x \ q = 0 \iff \text{order } x \ p = 0$ 
  by ( $\text{metis } \text{order-pderiv } \langle p \neq 0 \rangle \ \text{add.commute add-0 lessI min-0L}$ 
     $\text{min-less-iff-disj } \text{order-0I } \text{order-pq less-irrefl}$ )
then show  $?thesis$ 
  using  $\langle p \neq 0 \rangle \ \langle q \neq 0 \rangle$   $\text{order-root } q\text{-def}$  by  $\text{blast}$ 
qed

```

1.5 Composition of a polynomial and a circular path

lemma *poly-circlepath-tan-eq*:

fixes $z0::\text{complex}$ and $r::\text{real}$ and $p::\text{complex poly}$

defines $q1 \equiv fcompose\ p\ [:(z0+r)*i, z0-r:] [i, 1:]$ and $q2 \equiv [i, 1:] \wedge degree\ p$

assumes $0 \leq t \leq 1$ $t \neq 1/2$

shows $poly\ p\ (circlepath\ z0\ r\ t) = poly\ q1\ (\tan\ (pi*t)) / poly\ q2\ (\tan\ (pi*t))$
(is $?L = ?R$)

proof –

have $?L = poly\ p\ (z0 + r * exp\ (2 * of-real\ pi * i * t))$

unfolding *circlepath* by *simp*

also have ... = $?R$

proof –

define f where $f = (poly\ p \circ (\lambda x::real. z0 + r * exp\ (i * x)))$

have $f\text{-eq}: f\ t = ((\lambda x::real. poly\ q1\ x / poly\ q2\ x) \circ (\lambda x. \tan\ (x/2)))\ t$

when $\cos\ (t / 2) \neq 0$ for t

proof –

have $f\ t = poly\ p\ (z0 + r * (\cos\ t + i * \sin\ t))$

unfolding $f\text{-def}$ *exp-Euler* by *(auto simp add:cos-of-real sin-of-real)*

also have ... = $poly\ p\ ((\lambda x. ((z0-r)*x+(z0+r)*i) / (i+x))\ (\tan\ (t/2)))$

proof –

define tt where $tt = complex-of-real\ (\tan\ (t / 2))$

define rr where $rr = complex-of-real\ r$

have $\S: \cos\ t = (1-tt*tt) / (1 + tt * tt)$

$\sin\ t = 2*tt / (1 + tt * tt)$

unfolding *sin-tan-half*[of $t/2$, *simplified*] *cos-tan-half*[of $t/2$, *OF that, simplified*] *tt-def*

by *(auto simp add:power2-eq-square)*

then have $1 + tt * tt \neq 0$

by *(metis cmod-unit-one division-ring-divide-zero mult-2 mult-zero-right norm-eq-zero*

zero-neq-one)

with \S have $z0 + r * ((\cos\ t) + i * (\sin\ t))$

$= (z0*(1+tt*tt)+rr*(1-tt*tt)+i*rr*2*tt) / (1 + tt * tt)$

apply *(simp flip: rr-def add: add-divide-distrib)*

by *(simp add:algebra-simps)*

also have ... = $((z0-rr)*tt+z0*i+rr*i) / (tt + i)$

proof –

have $tt + i \neq 0$

using $\langle 1 + tt * tt \neq 0 \rangle$

by *(metis i-squared neg-eq-iff-add-eq-0 square-eq-iff)*

then show *?thesis*

using $\langle 1 + tt * tt \neq 0 \rangle$ by *(auto simp add:divide-simps algebra-simps)*

qed

finally have $z0 + r * ((\cos\ t) + i * (\sin\ t)) = ((z0-rr)*tt+z0*i+rr*i) / (tt + i)$.

then show *?thesis* unfolding *tt-def* *rr-def*

by *(auto simp add:algebra-simps power2-eq-square)*

qed

also have ... = $(poly\ p \circ ((\lambda x. ((z0-r)*x+(z0+r)*i) / (i+x))) \circ (\lambda x. \tan$

```

(x/2)) ) t
  unfolding comp-def by (auto simp:tan-of-real)
  also have ... = ((λx::real. poly q1 x / poly q2 x) o (λx. tan (x/2))) t
  unfolding q2-def q1-def
  apply (subst fcompose-poly[symmetric])
  using Re-poly-hom.base.hom-add-eq-zero apply fastforce
  by (auto simp:tan-of-real algebra-simps)
  finally show ?thesis .
qed

have cos (pi * t) ≠ 0 unfolding cos-zero-iff-int2
proof
  assume ∃ i. pi * t = real-of-int i * pi + pi / 2
  then obtain i where pi * t = real-of-int i * pi + pi / 2 by auto
  then have pi * t = pi * (real-of-int i + 1 / 2) by (simp add:algebra-simps)
  then have t = real-of-int i + 1 / 2 by auto
  then show False using ‹0 ≤ t› ‹t ≤ 1› ‹t ≠ 1/2› by auto
qed
from f-eq[of 2*pi*t,simplified,OF this]
show ?thesis
  unfolding f-def comp-def by (auto simp add:algebra-simps)
qed
finally show ?thesis .
qed

```

1.6 Combining two real polynomials into a complex one

definition *cpoly-of*:: real poly \Rightarrow real poly \Rightarrow complex poly **where**
cpoly-of pR pI = map-poly of-real pR + smult i (map-poly of-real pI)

lemma *cpoly-of-eq-0-iff*[iff]:

$$\text{cpoly-of } pR \text{ } pI = 0 \iff pR = 0 \wedge pI = 0$$

proof –

have $pR = 0 \wedge pI = 0$ **when** *cpoly-of* pR pI = 0

proof –

have *complex-of-real* (coeff pR n) + i * *complex-of-real* (coeff pI n) = 0 **for** n

using that **unfolding** *poly-eq-iff* *cpoly-of-def* **by** (auto simp:coeff-map-poly)

then have $\text{coeff } pR \text{ } n = 0 \wedge \text{coeff } pI \text{ } n = 0$ **for** n

using *Complex-eq* *Complex-eq-0* **by** force

then show ?thesis **unfolding** *poly-eq-iff* **by** auto

qed

then show ?thesis **by** (auto simp:cpoly-of-def)

qed

lemma *cpoly-of-decompose*:

$$p = \text{cpoly-of } (\text{map-poly } \text{Re } p) (\text{map-poly } \text{Im } p)$$

unfolding *cpoly-of-def*

by (induct p) (auto simp add: map-poly-map-poly complex-eq)

lemma *cpoly-of-dist-right*:

cpoly-of (*pR***g*) (*pI***g*) = *cpoly-of* *pR* *pI* * (*map-poly of-real* *g*)

unfolding *cpoly-of-def* **by** (*simp add: distrib-right*)

lemma *poly-cpoly-of-real*:

poly (*cpoly-of* *pR* *pI*) (*of-real* *x*) = *Complex* (*poly* *pR* *x*) (*poly* *pI* *x*)

unfolding *cpoly-of-def* **by** (*simp add: Complex-eq*)

lemma *poly-cpoly-of-real-iff*:

shows *poly* (*cpoly-of* *pR* *pI*) (*of-real* *t*) = 0 \longleftrightarrow *poly* *pR* *t* = 0 \wedge *poly* *pI* *t* = 0

unfolding *poly-cpoly-of-real* **using** *Complex-eq-0* **by** *blast*

lemma *order-cpoly-gcd-eq*:

assumes *pR* \neq 0 \vee *pI* \neq 0

shows *order* *t* (*cpoly-of* *pR* *pI*) = *order* *t* (*gcd* *pR* *pI*)

proof –

define *g* **where** *g* = *gcd* *pR* *pI*

have [*simp*]: *g* \neq 0 **unfolding** *g-def* **using** *assms* **by** *auto*

obtain *pr* *pi* **where** *pri*: *pR* = *pr* * *g* *pI* = *pi* * *g* *coprime* *pr* *pi*

unfolding *g-def* **using** *assms*(1) *gcd-coprime-exists* \langle *g* \neq 0 \rangle *g-def* **by** *blast*

then have *pr* \neq 0 \vee *pi* \neq 0 **using** *assms* *mult-zero-left* **by** *blast*

have *order* *t* (*cpoly-of* *pR* *pI*) = *order* *t* (*cpoly-of* *pr* *pi* * (*map-poly of-real* *g*))

unfolding *pri* *cpoly-of-dist-right* **by** *simp*

also have ... = *order* *t* (*cpoly-of* *pr* *pi*) + *order* *t* *g*

using \langle *pr* \neq 0 \vee *pi* \neq 0 \rangle \langle *g* \neq 0 \rangle

by (*simp add: map-poly-order-of-real order-mult*)

also have ... = *order* *t* *g*

proof –

have *poly* (*cpoly-of* *pr* *pi*) *t* \neq 0 **unfolding** *poly-cpoly-of-real-iff*

using \langle *coprime* *pr* *pi* \rangle *coprime-poly-0* **by** *blast*

then have *order* *t* (*cpoly-of* *pr* *pi*) = 0 **by** (*simp add: order-0I*)

then show *?thesis* **by** *auto*

qed

finally show *?thesis* **unfolding** *g-def* .

qed

lemma *cpoly-of-times*:

shows *cpoly-of* *pR* *pI* * *cpoly-of* *qR* *qI* = *cpoly-of* (*pR* * *qR* – *pI* * *qI*) (*pI***qR*+*pR***qI*)

proof –

define *PR* *PI* **where** *PR* = *map-poly complex-of-real* *pR*

and *PI* = *map-poly complex-of-real* *pI*

define *QR* *QI* **where** *QR* = *map-poly complex-of-real* *qR*

and *QI* = *map-poly complex-of-real* *qI*

show *?thesis*

unfolding *cpoly-of-def*

by (*simp add: algebra-simps of-real-poly-hom.hom-minus smult-add-right*

flip: PR-def PI-def QR-def QI-def)

qed

lemma *map-poly-Re-cpoly*[simp]:
map-poly Re (cpoly-of pR pI) = pR
unfolding *cpoly-of-def smult-map-poly*
by (*simp add:map-poly-map-poly Re-poly-hom.hom-add comp-def coeff-map-poly poly-eq-iff*)

lemma *map-poly-Im-cpoly*[simp]:
map-poly Im (cpoly-of pR pI) = pI
unfolding *cpoly-of-def smult-map-poly*
by (*simp add:map-poly-map-poly Im-poly-hom.hom-add comp-def coeff-map-poly poly-eq-iff*)

end

2 An alternative Sturm sequences

theory *Extended-Sturm imports*
Sturm-Tarski.Sturm-Tarski
Winding-Number-Eval.Cauchy-Index-Theorem
CC-Polynomials-Extra

begin

The main purpose of this theory is to provide an effective way to compute *cindexE a b f* when *f* is a rational function. The idea is similar to and based on the evaluation of *cindex-poly* through $\llbracket ?a < ?b; \text{poly } ?p \ ?a \neq 0; \text{poly } ?p \ ?b \neq 0 \rrbracket \implies \text{cindex-poly } ?a \ ?b \ ?q \ ?p = \text{changes-itv-smods } ?a \ ?b \ ?p \ ?q$.

This alternative version of remainder sequences is inspired by the paper "The Fundamental Theorem of Algebra made effective: an elementary real-algebraic proof via Sturm chains" by Michael Eisermann.

hide-const *Permutations.sign*

2.1 Misc

lemma *path-of-real*[simp]:*path (of-real :: real \Rightarrow 'a::real-normed-algebra-1)*
unfolding *path-def* **by** (*rule continuous-on-of-real-id*)

lemma *pathfinish-of-real*[simp]:*pathfinish of-real = 1*
unfolding *pathfinish-def* **by** *simp*

lemma *pathstart-of-real*[simp]:*pathstart of-real = 0*
unfolding *pathstart-def* **by** *simp*

lemma *is-unit-pCons-ex-iff*:
fixes *p::'a::field poly*
shows *is-unit p \longleftrightarrow ($\exists a. a \neq 0 \wedge p = [a]$)*
using *is-unit-poly-iff is-unit-triv*
by (*metis is-unit-pCons-iff*)

lemma *eventually-poly-nz-at-within*:
fixes $x :: 'a::\{idom, euclidean-space\}$
assumes $p \neq 0$
shows *eventually* $(\lambda x. poly\ p\ x \neq 0)$ (at x within S)
proof –
have *eventually* $(\lambda x. poly\ p\ x \neq 0)$ (at x within S)
 $= (\forall_F x\ in\ (at\ x\ within\ S). \forall y \in roots\ p. x \neq y)$
apply (rule *eventually-subst*, rule *eventuallyI*)
by (auto simp add: *roots-def*)
also have ... $= (\forall y \in roots\ p. \forall_F x\ in\ (at\ x\ within\ S). x \neq y)$
apply (*subst eventually-ball-finite-distrib*)
using $\langle p \neq 0 \rangle$ **by** auto
also have ...
unfolding *eventually-at*
by (*metis gt-ex not-less-iff-gr-or-eq zero-less-dist-iff*)
finally show ?thesis .
qed

lemma *sgn-power*:
fixes $x :: 'a::linordered-idom$
shows $sgn\ (x^{\wedge}n) = (if\ n=0\ then\ 1\ else\ if\ even\ n\ then\ |sgn\ x|\ else\ sgn\ x)$
apply (*induct n*)
by (auto simp add: *sgn-mult*)

lemma *poly-divide-filterlim-at-top*:
fixes $p\ q::real\ poly$
defines $ll \equiv (if\ degree\ q < degree\ p\ then$
 $at\ 0$
 $else\ if\ degree\ q = degree\ p\ then$
 $nhds\ (lead-coeff\ q / lead-coeff\ p)$
 $else\ if\ sgn-pos-inf\ q * sgn-pos-inf\ p > 0\ then$
 $at-top$
 $else$
 $at-bot)$
assumes $p \neq 0\ q \neq 0$
shows *filterlim* $(\lambda x. poly\ q\ x / poly\ p\ x)$ $ll\ at-top$
proof –
define pp **where** $pp = (\lambda x. poly\ p\ x / x^{\wedge}(degree\ p))$
define qq **where** $qq = (\lambda x. poly\ q\ x / x^{\wedge}(degree\ q))$
define dd **where** $dd = (\lambda x::real. if\ degree\ p > degree\ q\ then\ 1/x^{\wedge}(degree\ p - degree\ q)\ else$
 $x^{\wedge}(degree\ q - degree\ p))$
have *divide-cong*: $\forall_F x\ in\ at-top. poly\ q\ x / poly\ p\ x = qq\ x / pp\ x * dd\ x$
proof (rule *eventually-at-top-linorderI*[of 1])
fix x **assume** $(x::real) \geq 1$
then have $x \neq 0$ **by** auto
then show $poly\ q\ x / poly\ p\ x = qq\ x / pp\ x * dd\ x$
unfolding *qq-def pp-def dd-def* **using** *assms*
by (auto simp add: *field-simps power-diff*)

qed
have $qppp\text{-tendsto} : ((\lambda x. qq\ x / pp\ x) \longrightarrow \text{lead-coeff}\ q / \text{lead-coeff}\ p)$ *at-top*
proof –
have $(qq \longrightarrow \text{lead-coeff}\ q)$ *at-top*
unfolding $qq\text{-def}$ **using** $\text{poly-divide-tendsto-aux}$ [*of q*]
by $(\text{auto elim!} : \text{filterlim-mono simp:at-top-le-at-infinity})$
moreover have $(pp \longrightarrow \text{lead-coeff}\ p)$ *at-top*
unfolding $pp\text{-def}$ **using** $\text{poly-divide-tendsto-aux}$ [*of p*]
by $(\text{auto elim!} : \text{filterlim-mono simp:at-top-le-at-infinity})$
ultimately show $?thesis$ **using** $\langle p \neq 0 \rangle$ **by** $(\text{auto intro!} : \text{tendsto-eq-intros})$
qed

have $?thesis$ **when** $\text{degree}\ q < \text{degree}\ p$
proof –
have $\text{filterlim}\ (\lambda x. \text{poly}\ q\ x / \text{poly}\ p\ x)$ $(\text{at}\ 0)$ *at-top*
proof $(\text{rule}\ \text{filterlim-atI})$
show $((\lambda x. \text{poly}\ q\ x / \text{poly}\ p\ x) \longrightarrow 0)$ *at-top*
using $\text{poly-divide-tendsto-0-at-infinity}$ [*OF that*]
by $(\text{auto elim} : \text{filterlim-mono simp:at-top-le-at-infinity})$
have $\forall_F x$ *in at-top*. $\text{poly}\ q\ x \neq 0 \ \forall_F x$ *in at-top*. $\text{poly}\ p\ x \neq 0$
using $\text{poly-eventually-not-zero}$ [*OF* $\langle q \neq 0 \rangle$] $\text{poly-eventually-not-zero}$ [*OF* $\langle p \neq 0 \rangle$]
 filter-leD [*OF at-top-le-at-infinity*]
by auto
then show $\forall_F x$ *in at-top*. $\text{poly}\ q\ x / \text{poly}\ p\ x \neq 0$
apply eventually-elim
by auto
qed
then show $?thesis$ **unfolding** ll-def **using** that **by** auto
qed
moreover have $?thesis$ **when** $\text{degree}\ q = \text{degree}\ p$
proof –
have $((\lambda x. \text{poly}\ q\ x / \text{poly}\ p\ x) \longrightarrow \text{lead-coeff}\ q / \text{lead-coeff}\ p)$ *at-top*
using $\text{divide-cong}\ qppp\text{-tendsto}\ \text{that}$ **unfolding** dd-def
by $(\text{auto dest} : \text{tendsto-cong})$
then show $?thesis$ **unfolding** ll-def **using** that **by** auto
qed
moreover have $?thesis$ **when** $\text{degree}\ q > \text{degree}\ p$ $\text{sgn-pos-inf}\ q * \text{sgn-pos-inf}\ p > 0$
proof –
have $\text{filterlim}\ (\lambda x. (qq\ x / pp\ x) * dd\ x)$ *at-top at-top*
proof $(\text{subst}\ \text{filterlim-tendsto-pos-mult-at-top-iff}$ [*OF qppp-tendsto*])
show $0 < \text{lead-coeff}\ q / \text{lead-coeff}\ p$ **using** $\text{that}(2)$ **unfolding** sgn-pos-inf-def
by $(\text{simp add} : \text{zero-less-divide-iff zero-less-mult-iff})$
show $\text{filterlim}\ dd$ *at-top at-top*
unfolding dd-def **using** $\text{that}(1)$
by $(\text{auto intro!} : \text{filterlim-pow-at-top simp:filterlim-ident})$
qed
then have $\text{LIM}\ x$ *at-top*. $\text{poly}\ q\ x / \text{poly}\ p\ x :>$ *at-top*
using filterlim-cong [*OF - - divide-cong*] **by** blast

```

    then show ?thesis unfolding ll-def using that by auto
  qed
  moreover have ?thesis when degree q > degree p  $\neg$  sgn-pos-inf q * sgn-pos-inf
  p > 0
  proof -
    have filterlim ( $\lambda x. (qq\ x / pp\ x) * dd\ x$ ) at-bot at-top
  proof (subst filterlim-tendsto-neg-mult-at-bot-iff[OF qqpp-tendsto])
    show lead-coeff q / lead-coeff p < 0
      using that(2)  $\langle p \neq 0 \rangle$   $\langle q \neq 0 \rangle$  unfolding sgn-pos-inf-def
      by (metis divide-eq-0-iff divide-sgn leading-coeff-0-iff
        linorder-neqE-linordered-idom sgn-divide sgn-greater)
    show filterlim dd at-top at-top
      unfolding dd-def using that(1)
      by (auto intro!:filterlim-pow-at-top simp:filterlim-ident)
  qed
  then have LIM x at-top. poly q x / poly p x  $:\>$  at-bot
    using filterlim-cong[OF - - divide-cong] by blast
  then show ?thesis unfolding ll-def using that by auto
  qed
  ultimately show ?thesis by linarith
  qed

```

lemma poly-divide-filterlim-at-bot:

```

  fixes p q :: real poly
  defines ll  $\equiv$  ( if degree q < degree p then
    at 0
  else if degree q = degree p then
    nhds (lead-coeff q / lead-coeff p)
  else if sgn-neg-inf q * sgn-neg-inf p > 0 then
    at-top
  else
    at-bot)
  assumes p  $\neq$  0 q  $\neq$  0
  shows filterlim ( $\lambda x. poly\ q\ x / poly\ p\ x$ ) ll at-bot
  proof -
    define pp where pp = ( $\lambda x. poly\ p\ x / x^{\wedge}(degree\ p)$ )
    define qq where qq = ( $\lambda x. poly\ q\ x / x^{\wedge}(degree\ q)$ )
    define dd where dd = ( $\lambda x :: real. if\ degree\ p > degree\ q\ then\ 1/x^{\wedge}(degree\ p - degree\ q)\ else$ 
       $x^{\wedge}(degree\ q - degree\ p)$ )
    have divide-cong:  $\forall\ F\ x\ in\ at-bot. poly\ q\ x / poly\ p\ x = qq\ x / pp\ x * dd\ x$ 
  proof (rule eventually-at-bot-linorderI[of - 1])
    fix x assume (x :: real)  $\leq$  -1
    then have x  $\neq$  0 by auto
    then show poly q x / poly p x = qq x / pp x * dd x
      unfolding qq-def pp-def dd-def using assms
      by (auto simp add:field-simps power-diff)
  qed
  have qqpp-tendsto: (( $\lambda x. qq\ x / pp\ x$ )  $\longrightarrow$  lead-coeff q / lead-coeff p) at-bot

```

```

proof –
  have (qq  $\longrightarrow$  lead-coeff q) at-bot
    unfolding qq-def using poly-divide-tendsto-aux[of q]
    by (auto elim!:filterlim-mono simp:at-bot-le-at-infinity)
  moreover have (pp  $\longrightarrow$  lead-coeff p) at-bot
    unfolding pp-def using poly-divide-tendsto-aux[of p]
    by (auto elim!:filterlim-mono simp:at-bot-le-at-infinity)
  ultimately show ?thesis using ⟨p≠0⟩ by (auto intro!:tendsto-eq-intros)
qed

have ?thesis when degree q < degree p
proof –
  have filterlim (λx. poly q x / poly p x) (at 0) at-bot
  proof (rule filterlim-atI)
    show ((λx. poly q x / poly p x)  $\longrightarrow$  0) at-bot
      using poly-divide-tendsto-0-at-infinity[OF that]
      by (auto elim!:filterlim-mono simp:at-bot-le-at-infinity)
    have  $\forall_F x$  in at-bot. poly q x  $\neq 0$   $\forall_F x$  in at-bot. poly p x  $\neq 0$ 
    using poly-eventually-not-zero[OF ⟨q≠0⟩] poly-eventually-not-zero[OF ⟨p≠0⟩]
      filter-leD[OF at-bot-le-at-infinity]
    by auto
    then show  $\forall_F x$  in at-bot. poly q x / poly p x  $\neq 0$ 
    by eventually-elim auto
  qed
  then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree q = degree p
proof –
  have ((λx. poly q x / poly p x)  $\longrightarrow$  lead-coeff q / lead-coeff p) at-bot
    using divide-cong qqpp-tendsto that unfolding dd-def
    by (auto dest:tendsto-cong)
  then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree q > degree p sgn-neg-inf q * sgn-neg-inf p >
0
proof –
  define cc where cc = lead-coeff q / lead-coeff p
  have (cc > 0  $\wedge$  even (degree q – degree p))  $\vee$  (cc < 0  $\wedge$  odd (degree q – degree
p))
proof –
  have even (degree q – degree p)  $\iff$ 
    (even (degree q)  $\wedge$  even (degree p))  $\vee$  (odd (degree q)  $\wedge$  odd (degree p))
    using ⟨degree q > degree p⟩ by auto
  then show ?thesis
    using that ⟨p≠0⟩ ⟨q≠0⟩ unfolding sgn-neg-inf-def cc-def zero-less-mult-iff
      divide-less-0-iff zero-less-divide-iff
    apply (simp add:if-split[of (<) 0] if-split[of (>) 0])
    by argo
qed

```

moreover have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x) \text{ at-top at-bot}$
when $cc > 0 \text{ even } (degree \ q - degree \ p)$
proof ($\text{subst filterlim-tendsto-pos-mult-at-top-iff}[OF \ qppp-tendsto]$)
show $0 < \text{lead-coeff } q / \text{lead-coeff } p$ **using** $\langle cc > 0 \rangle$ **unfolding** $cc\text{-def}$ **by** auto
show $\text{filterlim } dd \text{ at-top at-bot}$
unfolding $dd\text{-def}$ **using** $\langle degree \ q > degree \ p \rangle$ $\text{that}(2)$
by ($\text{auto intro!}:\text{filterlim-pow-at-bot-even simp:filterlim-ident}$)
qed
moreover have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x) \text{ at-top at-bot}$
when $cc < 0 \text{ odd } (degree \ q - degree \ p)$
proof ($\text{subst filterlim-tendsto-neg-mult-at-top-iff}[OF \ qppp-tendsto]$)
show $0 > \text{lead-coeff } q / \text{lead-coeff } p$ **using** $\langle cc < 0 \rangle$ **unfolding** $cc\text{-def}$ **by** auto
show $\text{filterlim } dd \text{ at-bot at-bot}$
unfolding $dd\text{-def}$ **using** $\langle degree \ q > degree \ p \rangle$ $\text{that}(2)$
by ($\text{auto intro!}:\text{filterlim-pow-at-bot-odd simp:filterlim-ident}$)
qed
ultimately have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x) \text{ at-top at-bot}$
by blast
then have $LIM \ x \text{ at-bot. poly } q \ x / \text{poly } p \ x \text{ :> at-top}$
using $\text{filterlim-cong}[OF \ - \ - \ \text{divide-cong}]$ **by** blast
then show $?thesis$ **unfolding** $ll\text{-def}$ **using** that **by** auto
qed
moreover have $?thesis$ **when** $degree \ q > degree \ p \wedge \text{sgn-neg-inf } q * \text{sgn-neg-inf } p > 0$
proof –
define cc **where** $cc = \text{lead-coeff } q / \text{lead-coeff } p$
have $(cc < 0 \wedge \text{even } (degree \ q - degree \ p)) \vee (cc > 0 \wedge \text{odd } (degree \ q - degree \ p))$
proof –
have $\text{even } (degree \ q - degree \ p) \iff$
 $(\text{even } (degree \ q) \wedge \text{even } (degree \ p)) \vee (\text{odd } (degree \ q) \wedge \text{odd } (degree \ p))$
using $\langle degree \ q > degree \ p \rangle$ **by** auto
then show $?thesis$
using $\text{that } \langle p \neq 0 \rangle \langle q \neq 0 \rangle$ **unfolding** $\text{sgn-neg-inf-def } cc\text{-def } \text{zero-less-mult-iff}$
 $\text{divide-less-0-iff } \text{zero-less-divide-iff}$
apply ($\text{simp add:if-split}[of \ (<) \ 0] \ \text{if-split}[of \ (>) \ 0]$)
by ($\text{metis leading-coeff-0-iff linorder-neqE-linordered-idom}$)
qed
moreover have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x) \text{ at-bot at-bot}$
when $cc < 0 \text{ even } (degree \ q - degree \ p)$
proof ($\text{subst filterlim-tendsto-neg-mult-at-bot-iff}[OF \ qppp-tendsto]$)
show $0 > \text{lead-coeff } q / \text{lead-coeff } p$ **using** $\langle cc < 0 \rangle$ **unfolding** $cc\text{-def}$ **by** auto
show $\text{filterlim } dd \text{ at-top at-bot}$
unfolding $dd\text{-def}$ **using** $\langle degree \ q > degree \ p \rangle$ $\text{that}(2)$
by ($\text{auto intro!}:\text{filterlim-pow-at-bot-even simp:filterlim-ident}$)
qed
moreover have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x) \text{ at-bot at-bot}$
when $cc > 0 \text{ odd } (degree \ q - degree \ p)$
proof ($\text{subst filterlim-tendsto-pos-mult-at-bot-iff}[OF \ qppp-tendsto]$)

show $0 < \text{lead-coeff } q / \text{lead-coeff } p$ **using** $\langle cc > 0 \rangle$ **unfolding** $cc\text{-def}$ **by** $auto$
show $\text{filterlim } dd \text{ at-bot at-bot}$
unfolding $dd\text{-def}$ **using** $\langle \text{degree } q > \text{degree } p \rangle$ **that**(2)
by $(auto \text{ intro!}:\text{filterlim-pow-at-bot-odd simp}:\text{filterlim-ident})$
qed
ultimately have $\text{filterlim } (\lambda x. (qq \ x / pp \ x) * dd \ x)$ at-bot at-bot
by $blast$
then have $LIM \ x \ \text{at-bot. } poly \ q \ x / poly \ p \ x \ :> \ \text{at-bot}$
using $\text{filterlim-cong}[OF \ - \ - \ \text{divide-cong}]$ **by** $blast$
then show $?thesis$ **unfolding** $ll\text{-def}$ **using** $that$ **by** $auto$
qed
ultimately show $?thesis$ **by** linarith
qed

lemma $sgnx\text{-poly-times}$:
assumes $F = \text{at-bot} \vee F = \text{at-top} \vee F = \text{at-right } x \vee F = \text{at-left } x$
shows $sgnx \ (poly \ (p * q)) \ F = sgnx \ (poly \ p) \ F * sgnx \ (poly \ q) \ F$
(is $?PQ = ?P * ?Q$ **)**
proof $-$
have $(poly \ p \ \text{has-}sgnx \ ?P) \ F$
 $(poly \ q \ \text{has-}sgnx \ ?Q) \ F$
by $(rule \ sgnx\text{-able-}sgnx; use \ \text{assms } sgnx\text{-able-poly in } blast) +$
from $\text{has-}sgnx\text{-times}[OF \ \text{this}]$
have $(poly \ (p * q) \ \text{has-}sgnx \ ?P * ?Q) \ F$
by $(simp \ \text{flip}:\text{poly-mult})$
moreover have $(poly \ (p * q) \ \text{has-}sgnx \ ?PQ) \ F$
by $(rule \ sgnx\text{-able-}sgnx; use \ \text{assms } sgnx\text{-able-poly in } blast) +$
ultimately show $?thesis$
using $\text{has-}sgnx\text{-unique } \text{assms}$ **by** $auto$
qed

lemma $sgnx\text{-poly-plus}$:
assumes $poly \ p \ x = 0 \ \text{poly } q \ x \neq 0$ **and** $F: F = \text{at-right } x \vee F = \text{at-left } x$
shows $sgnx \ (poly \ (p + q)) \ F = sgnx \ (poly \ q) \ F$ **(is** $?L = ?R$ **)**
proof $-$
have $((poly \ (p + q)) \ \text{has-}sgnx \ ?R) \ F$
proof $-$
have $sgnx \ (poly \ q) \ F = sgn \ (poly \ q \ x)$
using $F \ \text{assms}(2) \ sgnx\text{-poly-nz}(1) \ sgnx\text{-poly-nz}(2)$ **by** presburger
moreover have $((\lambda x. poly \ (p + q) \ x) \ \text{has-}sgnx \ sgn \ (poly \ q \ x)) \ F$
proof $(rule \ \text{tendsto-nonzero-has-}sgnx)$
have $((poly \ p) \ \longrightarrow 0) \ F$
by $(metis \ F \ \text{assms}(1) \ \text{poly-tendsto}(2) \ \text{poly-tendsto}(3))$
then have $((\lambda x. poly \ p \ x + poly \ q \ x) \ \longrightarrow poly \ q \ x) \ F$
apply $(elim \ \text{tendsto-add}[\text{where } a = 0, \ \text{simplified}])$
using $F \ \text{poly-tendsto}(2) \ \text{poly-tendsto}(3)$ **by** $blast$
then show $((\lambda x. poly \ (p + q) \ x) \ \longrightarrow poly \ q \ x) \ F$

```

    by auto
  qed fact
  ultimately show ?thesis by metis
qed
from has-sgnx-imp-sgnx[OF this] F
show ?thesis by auto
qed

```

```

lemma sign-r-pos-plus-imp:
  assumes sign-r-pos p x sign-r-pos q x
  shows sign-r-pos (p+q) x
  using assms unfolding sign-r-pos-def
  by eventually-elim auto

```

```

lemma cindex-poly-combine:
  assumes a<b b<c
  shows cindex-poly a b q p + jump-poly q p b + cindex-poly b c q p = cindex-poly
  a c q p
proof (cases p≠0)
  case True
  define A B C D where A = {x. poly p x = 0 ∧ a < x ∧ x < c}
    and B = {x. poly p x = 0 ∧ a < x ∧ x < b}
    and C = (if poly p b = 0 then {b} else {})
    and D = {x. poly p x = 0 ∧ b < x ∧ x < c}
  let ?sum=sum (λx. jump-poly q p x)

  have cindex-poly a c q p = ?sum A
    unfolding cindex-poly-def A-def by simp
  also have ... = ?sum (B ∪ C ∪ D)
    apply (rule arg-cong2[where f=?sum])
    unfolding A-def B-def C-def D-def using less-linear assms by auto
  also have ... = ?sum B + ?sum C + ?sum D
proof -
  have finite B finite C finite D
    unfolding B-def C-def D-def using True
    by (auto simp add: poly-roots-finite)
  moreover have B ∩ C = {} C ∩ D = {} B ∩ D = {}
    unfolding B-def C-def D-def using assms by auto
  ultimately show ?thesis
    by (subst sum.union-disjoint; auto)+
qed
also have ... = cindex-poly a b q p + jump-poly q p b + cindex-poly b c q p
proof -
  have ?sum C = jump-poly q p b
    unfolding C-def using jump-poly-not-root by auto
  then show ?thesis unfolding cindex-poly-def B-def D-def
    by auto

```

qed
 finally show ?thesis by simp
 qed auto

lemma coprime-linear-comp: — TODO: need to be generalised

fixes b c::real
 defines r0 ≡ [:b,c:]
 assumes coprime p q c≠0
 shows coprime (p ∘_p r0) (q ∘_p r0)
 proof –
 define g where g = gcd (p ∘_p r0) (q ∘_p r0)
 define p' where p' = (p ∘_p r0) div g
 define q' where q' = (q ∘_p r0) div g
 define r1 where r1 = [-b/c,1/c:]

have r-id:
 r0 ∘_p r1 = [:0,1:]
 r1 ∘_p r0 = [:0,1:]
 unfolding r0-def r1-def using ⟨c≠0⟩
 by (simp add: pcompose-pCons)+

have p = (g ∘_p r1) * (p' ∘_p r1)

proof –
 from r-id have p = p ∘_p (r0 ∘_p r1)
 by (metis pcompose-idR)
 also have ... = (g * p') ∘_p r1
 unfolding g-def p'-def by (auto simp:pcompose-assoc)
 also have ... = (g ∘_p r1) * (p' ∘_p r1)
 unfolding pcompose-mult by simp
 finally show ?thesis .

qed

moreover have q = (g ∘_p r1) * (q' ∘_p r1)

proof –
 from r-id have q = q ∘_p (r0 ∘_p r1)
 by (metis pcompose-idR)
 also have ... = (g * q') ∘_p r1
 unfolding g-def q'-def by (auto simp:pcompose-assoc)
 also have ... = (g ∘_p r1) * (q' ∘_p r1)
 unfolding pcompose-mult by simp
 finally show ?thesis .

qed

ultimately have (g ∘_p r1) dvd gcd p q by simp

then have g ∘_p r1 dvd 1

using ⟨coprime p q⟩ by auto

from pcompose-hom.hom-dvd-1[OF this]

have is-unit (g ∘_p (r1 ∘_p r0))

by (auto simp:pcompose-assoc)

then have is-unit g

using r-id pcompose-idR by auto

then show $\text{coprime } (p \circ_p r0) (q \circ_p r0)$ **unfolding** $g\text{-def}$
using is-unit-gcd **by** blast
qed

lemma $\text{finite-ReZ-segments-poly-rectpath}$:
 $\text{finite-ReZ-segments } (\text{poly } p \circ \text{rectpath } a \ b) \ z$
unfolding rectpath-def Let-def path-compose-join
by $((\text{subst } \text{finite-ReZ-segments-joinpaths}$
 $\quad | \text{intro } \text{path-poly-comp } \text{conjI});$
 $(\text{simp add:poly-linepath-comp } \text{finite-ReZ-segments-poly-of-real } \text{path-compose-join}$
 $\quad \text{pathfinish-compose } \text{pathstart-compose } \text{poly-pcompose})?)+$

lemma $\text{valid-path-poly-linepath}$:
fixes $a \ b :: 'a :: \text{real-normed-field}$
shows $\text{valid-path } (\text{poly } p \ o \ \text{linepath } a \ b)$
proof $(\text{rule } \text{valid-path-compose})$
show $\text{valid-path } (\text{linepath } a \ b)$ **by** simp
show $\bigwedge x. x \in \text{path-image } (\text{linepath } a \ b) \implies \text{poly } p \ \text{field-differentiable at } x$
by simp
show $\text{continuous-on } (\text{path-image } (\text{linepath } a \ b)) (\text{deriv } (\text{poly } p))$
unfolding deriv-pderiv **by** $(\text{auto intro:continuous-intros})$
qed

lemma $\text{valid-path-poly-rectpath}$: $\text{valid-path } (\text{poly } p \ o \ \text{rectpath } a \ b)$
unfolding rectpath-def Let-def path-compose-join
by $(\text{simp add: } \text{pathfinish-compose } \text{pathstart-compose } \text{valid-path-poly-linepath})$

2.2 Sign difference

definition $\text{psign-diff} :: \text{real poly} \implies \text{real poly} \implies \text{real} \implies \text{int}$ **where**
 $\text{psign-diff } p \ q \ x = (\text{if } \text{poly } p \ x = 0 \wedge \text{poly } q \ x = 0 \text{ then}$
 $\quad 1 \ \text{else } |\text{sign } (\text{poly } p \ x) - \text{sign } (\text{poly } q \ x)|)$

lemma psign-diff-alt :
assumes $\text{coprime } p \ q$
shows $\text{psign-diff } p \ q \ x = |\text{sign } (\text{poly } p \ x) - \text{sign } (\text{poly } q \ x)|$
unfolding psign-diff-def **by** $(\text{meson } \text{assms } \text{coprime-poly-0})$

lemma $\text{psign-diff-0[simp]}$:
 $\text{psign-diff } 0 \ q \ x = 1$
 $\text{psign-diff } p \ 0 \ x = 1$
unfolding psign-diff-def **by** $(\text{auto } \text{simp add:sign-def})$

lemma $\text{psign-diff-poly-commute}$:
 $\text{psign-diff } p \ q \ x = \text{psign-diff } q \ p \ x$
unfolding psign-diff-def
by $(\text{metis } \text{abs-minus-commute } \text{gcd.commute})$

lemma *normalize-real-poly*:
normalize p = smult (1/lead-coeff p) (p::real poly)
unfolding *normalize-poly-def*
by (*metis Missing-Polynomial.unit-factor-field inverse-eq-divide normalize-poly-def normalize-poly-old-def*)

lemma *psign-diff-cancel*:
assumes *poly r x ≠ 0*
shows *psign-diff (r*p) (r*q) x = psign-diff p q x*
proof –
have *poly (r * p) x = 0 ⟷ poly p x = 0*
by (*simp add: assms*)
moreover have *poly (r * q) x = 0 ⟷ poly q x = 0* **by** (*simp add: assms*)
moreover have $|sign (poly (r * p) x) - sign (poly (r * q) x)|$
 $= |sign (poly p x) - sign (poly q x)|$
proof –
have $|sign (poly (r * p) x) - sign (poly (r * q) x)|$
 $= |sign (poly r x) * (sign (poly p x) - sign (poly q x))|$
by (*simp add: algebra-simps sign-times*)
also have ... $= |sign (poly r x)|$
 $* |sign (poly p x) - sign (poly q x)|$
unfolding *abs-mult* **by** *simp*
also have ... $= |sign (poly p x) - sign (poly q x)|$
by (*simp add: Sturm-Tarski.sign-def assms*)
finally show *?thesis* .
qed
ultimately show *?thesis*
unfolding *psign-diff-def* **by** *auto*
qed

lemma *psign-diff-clear*: *psign-diff p q x = psign-diff 1 (p * q) x*
unfolding *psign-diff-def*
apply (*simp add: sign-times*)
by (*simp add: sign-def*)

lemma *psign-diff-linear-comp*:
fixes *b c::real*
defines *h ≡ (λp. pcompose p [:b,c:])*
shows *psign-diff (h p) (h q) x = psign-diff p q (c * x + b)*
unfolding *psign-diff-def h-def poly-pcompose*
by (*smt (verit, del-Insts) mult.commute mult-eq-0-iff poly-0 poly-pCons*)

2.3 Alternative definition of cross

definition *cross-alt* :: *real poly ⇒ real poly ⇒ real ⇒ real ⇒ int* **where**
cross-alt p q a b = psign-diff p q a - psign-diff p q b

lemma *cross-alt-0* [*simp*]:
cross-alt 0 q a b = 0

$cross-alt\ p\ 0\ a\ b = 0$
unfolding $cross-alt-def$ **by** $simp-all$

lemma $cross-alt-poly-commute$:
 $cross-alt\ p\ q\ a\ b = cross-alt\ q\ p\ a\ b$
unfolding $cross-alt-def$ **using** $psign-diff-poly-commute$ **by** $auto$

lemma $cross-alt-clear$:
 $cross-alt\ p\ q\ a\ b = cross-alt\ 1\ (p*q)\ a\ b$
unfolding $cross-alt-def$ **using** $psign-diff-clear$ **by** $metis$

lemma $cross-alt-alt$:
 $cross-alt\ p\ q\ a\ b = sign\ (poly\ (p*q)\ b) - sign\ (poly\ (p*q)\ a)$
apply $(subst\ cross-alt-clear)$
unfolding $cross-alt-def\ psign-diff-def$ **by** $(auto\ simp\ add:sign-def)$

lemma $cross-alt-coprime-0$:
assumes $coprime\ p\ q\ p=0 \vee q=0$
shows $cross-alt\ p\ q\ a\ b=0$
proof –
have $?thesis$ **when** $p=0$
proof –
have $is-unit\ q$ **using** $that\ \langle coprime\ p\ q \rangle$
by $simp$
then obtain a **where** $a \neq 0\ q = [:a:]$ **using** $is-unit-pCons-ex-iff$ **by** $blast$
then show $?thesis$ **using** $that\ unfolding\ cross-alt-def$ **by** $auto$
qed
moreover have $?thesis$ **when** $q=0$
proof –
have $is-unit\ p$ **using** $that\ \langle coprime\ p\ q \rangle$
by $simp$
then obtain a **where** $a \neq 0\ p = [:a:]$ **using** $is-unit-pCons-ex-iff$ **by** $blast$
then show $?thesis$ **using** $that\ unfolding\ cross-alt-def$ **by** $auto$
qed
ultimately show $?thesis$ **using** $\langle p=0 \vee q=0 \rangle$ **by** $auto$
qed

lemma $cross-alt-cancel$:
assumes $poly\ q\ a \neq 0\ poly\ q\ b \neq 0$
shows $cross-alt\ (q * r)\ (q * s)\ a\ b = cross-alt\ r\ s\ a\ b$
unfolding $cross-alt-def$ **using** $psign-diff-cancel\ assms$ **by** $auto$

lemma $cross-alt-noroot$:
assumes $a < b$ **and** $\forall x. a \leq x \wedge x \leq b \longrightarrow poly\ (p*q)\ x \neq 0$
shows $cross-alt\ p\ q\ a\ b = 0$
proof –
define pq **where** $pq = p*q$
have $cross-alt\ p\ q\ a\ b = psign-diff\ 1\ pq\ a - psign-diff\ 1\ pq\ b$
apply $(subst\ cross-alt-clear)$

```

  unfolding cross-alt-def pq-def by simp
  also have ... = |1 - sign (poly pq a)| - |1 - sign (poly pq b)|
  unfolding psign-diff-def by simp
  also have ... = sign (poly pq b) - sign (poly pq a)
  unfolding sign-def by auto
  also have ... = 0
  proof (rule ccontr)
    assume sign (poly pq b) - sign (poly pq a) ≠ 0
    then have poly pq a * poly pq b < 0
      by (smt (verit, best) Sturm-Tarski.sign-def assms(1) assms(2)
          divisors-zero eq-iff-diff-eq-0 pq-def zero-less-mult-pos zero-less-mult-pos2)
    from poly-IVT[OF ‹a < b› this]
    have ∃ x > a. x < b ∧ poly pq x = 0 .
    then show False using ‹∀ x. a ≤ x ∧ x ≤ b ⟶ poly (p*q) x ≠ 0› ‹a < b›
      apply (fold pq-def)
      by auto
  qed
  finally show ?thesis .
  qed

```

lemma *cross-alt-linear-comp*:

```

  fixes b c :: real
  defines h ≡ (λp. pcompose p [:b,c:])
  shows cross-alt (h p) (h q) lb ub = cross-alt p q (c * lb + b) (c * ub + b)
  unfolding cross-alt-def h-def
  by (subst (1 2) psign-diff-linear-comp; simp)

```

2.4 Alternative sign variation sequence

fun *changes-alt*:: ('a :: linordered-idom) list ⇒ int **where**

```

  changes-alt [] = 0 |
  changes-alt [-] = 0 |
  changes-alt (x1 # x2 # xs) = abs(sign x1 - sign x2) + changes-alt (x2 # xs)

```

definition *changes-alt-poly-at*:: ('a :: linordered-idom) poly list ⇒ 'a ⇒ int **where**
changes-alt-poly-at ps a = *changes-alt (map (λp. poly p a) ps)*

definition *changes-alt-itv-smods*:: real ⇒ real ⇒ real poly ⇒ real poly ⇒ int
where

```

  changes-alt-itv-smods a b p q = (let ps = smods p q
    in changes-alt-poly-at ps a - changes-alt-poly-at ps b)

```

lemma *changes-alt-itv-smods-rec*:

```

  assumes a < b coprime p q
  shows changes-alt-itv-smods a b p q = cross-alt p q a b + changes-alt-itv-smods
    a b q (-(p mod q))
  proof (cases p = 0 ∨ q = 0 ∨ q dvd p)

```

case *True*
moreover have $p=0 \vee q=0 \implies ?thesis$
using *cross-alt-coprime-0*
unfolding *changes-alt-itv-smods-def changes-alt-poly-at-def* **by** *fastforce*
moreover have $\llbracket p \neq 0; q \neq 0; p \bmod q = 0 \rrbracket \implies ?thesis$
unfolding *changes-alt-itv-smods-def changes-alt-poly-at-def cross-alt-def*
psign-diff-alt[OF ‹coprime p q›]
by (*simp add:sign-times*)
ultimately show *?thesis*
by *auto (auto elim: dvdE)*
next
case *False*
hence $p \neq 0 \ q \neq 0 \ p \bmod q \neq 0$ **by** *auto*
then obtain *ps* **where** $ps:smods \ p \ q = p \# q \# -(p \bmod q) \# ps \ smods \ q \ (- (p \bmod q)) = q \# -(p \bmod q) \# ps$
by *auto*
define *changes-diff* **where** $changes-diff \equiv \lambda x. changes-alt-poly-at \ (p \# q \# -(p \bmod q) \# ps) \ x$
– changes-alt-poly-at (q#-(p mod q)#ps) x
have $changes-diff \ a \ - \ changes-diff \ b = cross-alt \ p \ q \ a \ b$
unfolding *changes-diff-def changes-alt-poly-at-def cross-alt-def*
psign-diff-alt[OF ‹coprime p q›]
by *simp*
thus *?thesis* **unfolding** *changes-alt-itv-smods-def changes-diff-def changes-alt-poly-at-def ps*
by *force*
qed

2.5 jumpF on polynomials

definition *jumpF-polyR*:: $real \ poly \Rightarrow real \ poly \Rightarrow real \Rightarrow real$ **where**
jumpF-polyR q p a = jumpF (\lambda x. poly q x / poly p x) (at-right a)

definition *jumpF-polyL*:: $real \ poly \Rightarrow real \ poly \Rightarrow real \Rightarrow real$ **where**
jumpF-polyL q p a = jumpF (\lambda x. poly q x / poly p x) (at-left a)

definition *jumpF-poly-top*:: $real \ poly \Rightarrow real \ poly \Rightarrow real$ **where**
jumpF-poly-top q p = jumpF (\lambda x. poly q x / poly p x) at-top

definition *jumpF-poly-bot*:: $real \ poly \Rightarrow real \ poly \Rightarrow real$ **where**
jumpF-poly-bot q p = jumpF (\lambda x. poly q x / poly p x) at-bot

lemma *jumpF-polyR-0[simp]*: $jumpF-polyR \ 0 \ p \ a = 0 \ jumpF-polyR \ q \ 0 \ a = 0$
unfolding *jumpF-polyR-def* **by** *auto*

lemma *jumpF-polyL-0[simp]*: $jumpF-polyL \ 0 \ p \ a = 0 \ jumpF-polyL \ q \ 0 \ a = 0$
unfolding *jumpF-polyL-def* **by** *auto*

lemma *jumpF-polyR-mult-cancel*:
assumes $p' \neq 0$
shows $\text{jumpF-polyR } (p' * q) (p' * p) a = \text{jumpF-polyR } q p a$
unfolding *jumpF-polyR-def*
proof (*rule jumpF-cong*)
obtain ub **where** $a < ub \ \forall z. a < z \wedge z \leq ub \longrightarrow \text{poly } p' z \neq 0$
using *next-non-root-interval[OF ‹p'≠0›,of a]* **by** *auto*
then show $\forall_F x \text{ in at-right } a. \text{poly } (p' * q) x / \text{poly } (p' * p) x = \text{poly } q x / \text{poly } p x$
apply (*unfold eventually-at-right*)
apply (*intro exI[where x=ub]*)
by *auto*
qed *simp*

lemma *jumpF-polyL-mult-cancel*:
assumes $p' \neq 0$
shows $\text{jumpF-polyL } (p' * q) (p' * p) a = \text{jumpF-polyL } q p a$
unfolding *jumpF-polyL-def*
proof (*rule jumpF-cong*)
obtain lb **where** $lb < a \ \forall z. lb \leq z \wedge z < a \longrightarrow \text{poly } p' z \neq 0$
using *last-non-root-interval[OF ‹p'≠0›,of a]* **by** *auto*
then show $\forall_F x \text{ in at-left } a. \text{poly } (p' * q) x / \text{poly } (p' * p) x = \text{poly } q x / \text{poly } p x$
apply (*unfold eventually-at-left*)
apply (*intro exI[where x=lb]*)
by *auto*
qed *simp*

lemma *jumpF-poly-noroot*:
assumes $\text{poly } p a \neq 0$
shows $\text{jumpF-polyL } q p a = 0 \ \text{jumpF-polyR } q p a = 0$
subgoal unfolding *jumpF-polyL-def* **using** *assms*
apply (*intro jumpF-not-infinity*)
by (*auto intro!:continuous-intros*)
subgoal unfolding *jumpF-polyR-def* **using** *assms*
apply (*intro jumpF-not-infinity*)
by (*auto intro!:continuous-intros*)
done

lemma *jumpF-polyR-coprime'*:
assumes $\text{poly } p x \neq 0 \vee \text{poly } q x \neq 0$
shows $\text{jumpF-polyR } q p x = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{poly } p x = 0 \text{ then } \text{if } \text{sign-r-pos } p x \longleftrightarrow \text{poly } q x > 0 \text{ then } 1/2 \text{ else } -1/2 \text{ else } 0)$
proof (*cases p=0 ∨ q=0 ∨ poly p x≠0*)
case *True*
then show *?thesis* **using** *jumpF-poly-noroot* **by** *fastforce*
next
case *False*

then have *asm:p≠0 q≠0 poly p x=0* **by** *auto*
then have *poly q x≠0* **using** *assms* **using** *coprime-poly-0* **by** *blast*
have *?thesis* **when** *sign-r-pos p x* \longleftrightarrow *poly q x>0*
proof –
have (*poly p has-sgnx sgn (poly q x)*) (*at-right x*)
by (*metis False* \langle *poly q x ≠ 0* \rangle *has-sgnx-imp-sgnx lt-ex order-less-not-sym*
poly-has-sgnx-values(2) *sgn-greater*
sgn-real-def sign-r-pos-sgnx-iff that trivial-limit-at-right-real zero-less-one)
then have *LIM x at-right x. poly q x / poly p x := at-top*
apply (*subst filterlim-divide-at-bot-at-top-iff[of - poly q x]*)
apply (*auto simp add:* \langle *poly q x≠0* \rangle)
by (*metis asm(3) poly-tendsto(3)*)
then have *jumpF-polyR q p x = 1/2*
unfolding *jumpF-polyR-def jumpF-def* **by** *auto*
then show *?thesis* **using** *that False* **by** *auto*
qed
moreover have *?thesis* **when** \neg (*sign-r-pos p x* \longleftrightarrow *poly q x>0*)
proof –
have (*poly p has-sgnx - sgn (poly q x)*) (*at-right x*)
proof –
have (*0::real*) $< 1 \vee \neg$ (*1::real*) $< 0 \wedge$ *sign-r-pos p x*
 \vee (*poly p has-sgnx - sgn (poly q x)*) (*at-right x*)
by *simp*
then show *?thesis*
by (*metis (no-types) False* \langle *poly q x ≠ 0* \rangle *add.inverse-inverse has-sgnx-imp-sgnx*
neg-less-0-iff-less poly-has-sgnx-values(2) sgn-if sgn-less sign-r-pos-sgnx-iff
that trivial-limit-at-right-real)
qed
then have *LIM x at-right x. poly q x / poly p x := at-bot*
apply (*subst filterlim-divide-at-bot-at-top-iff[of - poly q x]*)
apply (*auto simp add:* \langle *poly q x≠0* \rangle)
by (*metis asm(3) poly-tendsto(3)*)
then have *jumpF-polyR q p x = - 1/2*
unfolding *jumpF-polyR-def jumpF-def* **by** *auto*
then show *?thesis* **using** *that False* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *jumpF-polyR-coprime:*
assumes *coprime p q*
shows *jumpF-polyR q p x = (if p ≠ 0 \wedge q ≠ 0 \wedge poly p x=0 then*
if sign-r-pos p x \longleftrightarrow poly q x>0 then 1/2 else - 1/2
else 0)
apply (*rule jumpF-polyR-coprime'*)
using *assms coprime-poly-0* **by** *blast*

```

lemma jumpF-polyL-coprime':
  assumes poly p x≠0 ∨ poly q x≠0
  shows jumpF-polyL q p x = (if p ≠ 0 ∧ q ≠ 0 ∧ poly p x=0 then
    if even (order x p) ⟷ sign-r-pos p x ⟷ poly q x>0 then 1/2 else
  - 1/2 else 0)
proof (cases p=0 ∨ q=0 ∨ poly p x≠0)
  case True
  then show ?thesis using jumpF-poly-noroot by fastforce
next
  case False
  then have asm:p≠0 q≠0 poly p x=0 by auto
  then have poly q x≠0 using assms using coprime-poly-0 by blast
  have ?thesis when even (order x p) ⟷ sign-r-pos p x ⟷ poly q x>0
  proof -
    consider (lt) poly q x>0 | (gt) poly q x<0 using ⟨poly q x≠0⟩ by linarith
    then have sgnx (poly p) (at-left x) = sgn (poly q x)
      apply cases
      subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF ⟨p≠0⟩,of x]
        apply (subst poly-sgnx-left-right[OF ⟨p≠0⟩])
        by auto
      subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF ⟨p≠0⟩,of x]
        apply (subst poly-sgnx-left-right[OF ⟨p≠0⟩])
        by auto
      done
    then have (poly p has-sgnx sgn (poly q x)) (at-left x)
      by (metis sgnx-able-poly(2) sgnx-able-sgnx)
    then have LIM x at-left x. poly q x / poly p x :=> at-top
      apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
      apply (auto simp add:⟨poly q x≠0⟩)
      by (metis asm(3) poly-tendsto(2))
    then have jumpF-polyL q p x = 1/2
      unfolding jumpF-polyL-def jumpF-def by auto
    then show ?thesis using that False by auto
  qed
  moreover have ?thesis when ¬ (even (order x p) ⟷ sign-r-pos p x ⟷ poly
  q x>0)
  proof -
    consider (lt) poly q x>0 | (gt) poly q x<0 using ⟨poly q x≠0⟩ by linarith
    then have sgnx (poly p) (at-left x) = - sgn (poly q x)
      apply cases
      subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF ⟨p≠0⟩,of x]
        apply (subst poly-sgnx-left-right[OF ⟨p≠0⟩])
        by auto
      subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF ⟨p≠0⟩,of x]
        apply (subst poly-sgnx-left-right[OF ⟨p≠0⟩])
        by auto
      done
    then have (poly p has-sgnx - sgn (poly q x)) (at-left x)
      by (metis sgnx-able-poly(2) sgnx-able-sgnx)
  
```

```

then have LIM x at-left x. poly q x / poly p x :> at-bot
  apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
  apply (auto simp add:⟨poly q x≠0⟩)
  by (metis asm(3) poly-tendsto(2))
then have jumpF-polyL q p x = - 1/2
  unfolding jumpF-polyL-def jumpF-def by auto
then show ?thesis using that False by auto
qed
ultimately show ?thesis by auto
qed

lemma jumpF-polyL-coprime:
  assumes coprime p q
  shows jumpF-polyL q p x = (if p ≠ 0 ∧ q ≠ 0 ∧ poly p x=0 then
    if even (order x p) ↔ sign-r-pos p x ↔ poly q x>0 then 1/2 else
    - 1/2 else 0)
  apply (rule jumpF-polyL-coprime')
  using assms coprime-poly-0 by blast

lemma jumpF-times:
  assumes tendsto:(f ⟶ c) F and c≠0 F≠bot
  shows jumpF (λx. f x * g x) F = sgn c * jumpF g F
proof -
  have c>0 ∨ c<0 using ⟨c≠0⟩ by auto
  moreover have ?thesis when c>0
  proof -
    note filterlim-tendsto-pos-mult-at-top-iff[OF tendsto ⟨c>0⟩,of g]
    moreover note filterlim-tendsto-pos-mult-at-bot-iff[OF tendsto ⟨c>0⟩,of g]
    moreover have sgn c = 1 using ⟨c>0⟩ by auto
    ultimately show ?thesis unfolding jumpF-def by auto
  qed
  moreover have ?thesis when c<0
  proof -
    define atbot where atbot = filterlim g at-bot F
    define attop where attop = filterlim g at-top F
    have jumpF (λx. f x * g x) F = (if atbot then 1 / 2 else if attop then - 1 / 2
    else 0)
    proof -
      note filterlim-tendsto-neg-mult-at-top-iff[OF tendsto ⟨c<0⟩,of g]
      moreover note filterlim-tendsto-neg-mult-at-bot-iff[OF tendsto ⟨c<0⟩,of g]
      ultimately show ?thesis unfolding jumpF-def atbot-def attop-def by auto
    qed
    also have ... = - (if attop then 1 / 2 else if atbot then - 1 / 2 else 0)
    proof -
      have False when atbot attop
        using filterlim-at-top-at-bot[OF - - ⟨F≠bot⟩] that unfolding atbot-def
        attop-def by auto
      then show ?thesis by fastforce
    qed
  qed

```

```

    also have ... = sgn c * jumpF g F
      using ⟨c<0⟩ unfolding jumpF-def attop-def atbot-def by auto
    finally show ?thesis .
  qed
  ultimately show ?thesis by auto
qed

lemma jumpF-polyR-inverse-add:
  assumes coprime p q
  shows jumpF-polyR q p x + jumpF-polyR p q x = jumpF-polyR 1 (q*p) x
  proof (cases p=0 ∨ q=0)
    case True
      then show ?thesis by auto
    next
      case False
        have jumpF-add:
          jumpF-polyR q p x = jumpF-polyR 1 (q*p) x when poly p x=0 coprime p q for
          p q
        proof (cases p=0)
          case True
            then show ?thesis by auto
          next
            case False
              have poly q x≠0 using that coprime-poly-0 by blast
              then have q≠0 by auto
              moreover have sign-r-pos p x = (0 < poly q x) ↔ sign-r-pos (q * p) x
                using sign-r-pos-mult[OF ⟨q≠0⟩ ⟨p≠0⟩] sign-r-pos-rec[OF ⟨q≠0⟩] ⟨poly q
                x≠0⟩
                by auto
              ultimately show ?thesis using ⟨poly p x=0⟩
                unfolding jumpF-polyR-coprime[OF ⟨coprime p q⟩,of x] jumpF-polyR-coprime[of
                q*p 1 x,simplified]
                by auto
            qed
          have False when poly p x=0 poly q x=0
            using ⟨coprime p q⟩ that coprime-poly-0 by blast
          moreover have ?thesis when poly p x=0 poly q x≠0
            proof –
              have jumpF-polyR p q x = 0 using jumpF-poly-noroot[OF ⟨poly q x≠0⟩] by
              auto
              then show ?thesis using jumpF-add[OF ⟨poly p x=0⟩ ⟨coprime p q⟩] by auto
            qed
          moreover have ?thesis when poly p x≠0 poly q x=0
            proof –
              have jumpF-polyR q p x = 0 using jumpF-poly-noroot[OF ⟨poly p x≠0⟩] by
              auto
              then show ?thesis using jumpF-add[OF ⟨poly q x=0⟩,of p] ⟨coprime p q⟩
                by (simp add: ac-simps)
            qed
          qed
  qed

```

moreover have *?thesis* **when** $\text{poly } p \ x \neq 0 \ \text{poly } q \ x \neq 0$
by (*simp add: jumpF-poly-noroot(2) that(1) that(2)*)
ultimately show *?thesis* **by** *auto*
qed

lemma *jumpF-polyL-inverse-add:*
assumes *coprime p q*
shows $\text{jumpF-polyL } q \ p \ x + \text{jumpF-polyL } p \ q \ x = \text{jumpF-polyL } 1 \ (q * p) \ x$
proof (*cases p=0 \vee q=0*)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
have *jumpF-add:*
 $\text{jumpF-polyL } q \ p \ x = \text{jumpF-polyL } 1 \ (q * p) \ x$ **when** $\text{poly } p \ x = 0$ *coprime p q* **for**
 $p \ q$
proof (*cases p=0*)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
have $\text{poly } q \ x \neq 0$ **using** *that coprime-poly-0* **by** *blast*
then have $q \neq 0$ **by** *auto*
moreover have $\text{sign-r-pos } p \ x = (0 < \text{poly } q \ x) \longleftrightarrow \text{sign-r-pos } (q * p) \ x$
using *sign-r-pos-mult[OF $\langle q \neq 0 \rangle \langle p \neq 0 \rangle$] sign-r-pos-rec[OF $\langle q \neq 0 \rangle$] $\langle \text{poly } q \ x \neq 0 \rangle$*
by *auto*
moreover have $\text{order } x \ p = \text{order } x \ (q * p)$
by (*metis $\langle \text{poly } q \ x \neq 0 \rangle$ add-cancel-right-left divisors-zero order-mult order-root*)
ultimately show *?thesis* **using** $\langle \text{poly } p \ x = 0 \rangle$
unfolding *jumpF-polyL-coprime[OF $\langle \text{coprime } p \ q \rangle$, of x] jumpF-polyL-coprime[of q * p 1 x, simplified]*
by *auto*
qed
have *False* **when** $\text{poly } p \ x = 0 \ \text{poly } q \ x = 0$
using $\langle \text{coprime } p \ q \rangle$ *that coprime-poly-0* **by** *blast*
moreover have *?thesis* **when** $\text{poly } p \ x = 0 \ \text{poly } q \ x \neq 0$
proof –
have $\text{jumpF-polyL } p \ q \ x = 0$ **using** *jumpF-poly-noroot[OF $\langle \text{poly } q \ x \neq 0 \rangle$] by auto*
then show *?thesis* **using** *jumpF-add[OF $\langle \text{poly } p \ x = 0 \rangle \langle \text{coprime } p \ q \rangle$] by auto*
qed
moreover have *?thesis* **when** $\text{poly } p \ x \neq 0 \ \text{poly } q \ x = 0$
proof –
have $\text{jumpF-polyL } q \ p \ x = 0$ **using** *jumpF-poly-noroot[OF $\langle \text{poly } p \ x \neq 0 \rangle$] by auto*
then show *?thesis* **using** *jumpF-add[OF $\langle \text{poly } q \ x = 0 \rangle$, of p] $\langle \text{coprime } p \ q \rangle$*
by (*simp add: ac-simps*)

qed
moreover have *?thesis* **when** $\text{poly } p \ x \neq 0 \ \text{poly } q \ x \neq 0$
by (*simp add: jumpF-poly-noroot that(1) that(2)*)
ultimately show *?thesis* **by** *auto*
qed

lemma *jumpF-polyL-smult-1*:
 $\text{jumpF-polyL } (\text{smult } c \ q) \ p \ x = \text{sgn } c * \text{jumpF-polyL } q \ p \ x$
proof (*cases c=0*)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then show *?thesis*
unfolding *jumpF-polyL-def*
apply (*subst jumpF-times[of λ-. c,symmetric]*)
by *auto*
qed

lemma *jumpF-polyR-smult-1*:
 $\text{jumpF-polyR } (\text{smult } c \ q) \ p \ x = \text{sgn } c * \text{jumpF-polyR } q \ p \ x$
proof (*cases c=0*)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then show *?thesis*
unfolding *jumpF-polyR-def* **using** *False*
apply (*subst jumpF-times[of λ-. c,symmetric]*)
by *auto*
qed

lemma
shows *jumpF-polyR-mod:jumpF-polyR* $q \ p \ x = \text{jumpF-polyR } (q \ \text{mod } p) \ p \ x$ **and**
 $\text{jumpF-polyL-mod:jumpF-polyL } q \ p \ x = \text{jumpF-polyL } (q \ \text{mod } p) \ p \ x$
proof –
define *f* **where** $f = (\lambda x. \text{poly } (q \ \text{div } p) \ x)$
define *g* **where** $g = (\lambda x. \text{poly } (q \ \text{mod } p) \ x / \text{poly } p \ x)$
have *jumpF-eq:jumpF* $(\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ (\text{at } y \ \text{within } S) = \text{jumpF } g \ (\text{at } y \ \text{within } S)$
when $p \neq 0$ **for** $y \ S$
proof –
let $?F = \text{at } y \ \text{within } S$
have $\forall_F \ x \ \text{in } \text{at } y \ \text{within } S. \ \text{poly } p \ x \neq 0$
using *eventually-poly-nz-at-within*[*OF* $\langle p \neq 0 \rangle$, *of* $y \ S$] .
then have *eventually* $(\lambda x. (\text{poly } q \ x / \text{poly } p \ x) = (f \ x + g \ x)) \ ?F$
proof (*rule eventually-mono*)

```

fix x
assume P: poly p x ≠ 0
have poly q x = poly (q div p * p + q mod p) x
  by simp
also have ... = f x * poly p x + poly (q mod p) x
  by (simp only: poly-add poly-mult f-def g-def)
moreover have poly (q mod p) x = g x * poly p x
  using P by (simp add: g-def)
ultimately show poly q x / poly p x = f x + g x
  using P by simp
qed
then have jumpF (λx. poly q x / poly p x) ?F = jumpF (λx. f x + g x) ?F
  by (intro jumpF-cong, auto)
also have ... = jumpF g ?F
proof -
  have (f → f y) (at y within S)
    unfolding f-def by (intro tendsto-intros)
  from filterlim-tendsto-add-at-bot-iff[OF this, of g] filterlim-tendsto-add-at-top-iff[OF
this, of g]
    show ?thesis unfolding jumpF-def by auto
  qed
finally show ?thesis .
qed
show jumpF-polyR q p x = jumpF-polyR (q mod p) p x
  apply (cases p=0)
  subgoal by auto
  subgoal using jumpF-eq unfolding g-def jumpF-polyR-def by auto
  done
show jumpF-polyL q p x = jumpF-polyL (q mod p) p x
  apply (cases p=0)
  subgoal by auto
  subgoal using jumpF-eq unfolding g-def jumpF-polyL-def by auto
  done
qed

lemma
assumes order x p ≤ order x r
shows jumpF-polyR-order-leq: jumpF-polyR (r+q) p x = jumpF-polyR q p x
  and jumpF-polyL-order-leq: jumpF-polyL (r+q) p x = jumpF-polyL q p x
proof -
define f g h where f=(λx. poly (q + r) x / poly p x)
  and g=(λx. poly q x / poly p x)
  and h=(λx. poly r x / poly p x)

have ∃ c. h -x → c if p≠0 r≠0
proof -
define xo where xo=[:- x, 1:] ^ order x p
obtain p' where p = xo * p' ∩ [:- x, 1:] dvd p'
  using order-decomp[OF ⟨p≠0⟩, of x] unfolding xo-def by auto

```

```

define r' where r' = r div xo
define h' where h' = (λx. poly r' x / poly p' x)

have ∀F x in at x. h x = h' x
proof -
  obtain S where open S x ∈ S by blast
  moreover have h w = h' w if w ∈ S w ≠ x for w
  proof -
    have r = xo * r'
    proof -
      have xo dvd r
      unfolding xo-def using ⟨r ≠ 0⟩ assms
      by (subst order-divides) simp
      then show ?thesis unfolding r'-def by simp
    qed
  moreover have poly xo w ≠ 0
  unfolding xo-def using ⟨w ≠ x⟩ by simp
  moreover note ⟨p = xo * p'⟩
  ultimately show ?thesis
  unfolding h-def h'-def by auto
  qed
  ultimately show ?thesis
  unfolding eventually-at-topological by auto
  qed
  moreover have h' - x → h' x
  proof -
    have poly p' x ≠ 0
    using ⟨¬ [- x, 1:] dvd p'⟩ poly-eq-0-iff-dvd by blast
    then show ?thesis
    unfolding h'-def
    by (auto intro!: tendsto-eq-intros)
  qed
  ultimately have h - x → h' x
  using tendsto-cong by auto
  then show ?thesis by auto
  qed
then obtain c where left:(h → c) (at-left x)
  and right:(h → c) (at-right x)
  if p ≠ 0 r ≠ 0
  unfolding filterlim-at-split by auto

show jumpF-polyR (r+q) p x = jumpF-polyR q p x
proof (cases p=0 ∨ r=0)
  case False
  have jumpF-polyR (r+q) p x =
    (if filterlim (λx. h x + g x) at-top (at-right x)
     then 1 / 2
     else if filterlim (λx. h x + g x) at-bot (at-right x)
     then - 1 / 2 else 0)

```

unfolding *jumpF-polyR-def jumpF-def g-def h-def*
by (*simp add:poly-add add-divide-distrib*)
also have ... =
 (*if filterlim g at-top (at-right x) then 1 / 2*
 else if filterlim g at-bot (at-right x) then - 1 / 2 else 0)
using *filterlim-tendsto-add-at-top-iff[OF right]*
 filterlim-tendsto-add-at-bot-iff[OF right] False
by *simp*
also have ... = *jumpF-polyR q p x*
unfolding *jumpF-polyR-def jumpF-def g-def* **by** *simp*
finally show *jumpF-polyR (r + q) p x = jumpF-polyR q p x* .
qed *auto*

show *jumpF-polyL (r+q) p x = jumpF-polyL q p x*
proof (*cases p=0 ∨ r=0*)
 case *False*
 have *jumpF-polyL (r+q) p x =*
 (*if filterlim (λx. h x + g x) at-top (at-left x)*
 then 1 / 2
 else if filterlim (λx. h x + g x) at-bot (at-left x)
 then - 1 / 2 else 0)
 unfolding *jumpF-polyL-def jumpF-def g-def h-def*
 by (*simp add:poly-add add-divide-distrib*)
 also have ... =
 (*if filterlim g at-top (at-left x) then 1 / 2*
 else if filterlim g at-bot (at-left x) then - 1 / 2 else 0)
 using *filterlim-tendsto-add-at-top-iff[OF left]*
 filterlim-tendsto-add-at-bot-iff[OF left] False
 by *simp*
 also have ... = *jumpF-polyL q p x*
 unfolding *jumpF-polyL-def jumpF-def g-def* **by** *simp*
 finally show *jumpF-polyL (r + q) p x = jumpF-polyL q p x* .
qed *auto*
qed

lemma

assumes *order x q < order x r q ≠ 0*
shows *jumpF-polyR-order-le:jumpF-polyR (r+q) p x = jumpF-polyR q p x*
 and *jumpF-polyL-order-le:jumpF-polyL (r+q) p x = jumpF-polyL q p x*
proof –
 have *jumpF-polyR (r+q) p x = jumpF-polyR q p x*
 jumpF-polyL (r+q) p x = jumpF-polyL q p x
 if *p=0 ∨ r=0 ∨ order x p ≤ order x r*
 using *jumpF-polyR-order-leq jumpF-polyL-order-leq* **that** **by** *auto*
moreover have
 jumpF-polyR (r+q) p x = jumpF-polyR q p x
 jumpF-polyL (r+q) p x = jumpF-polyL q p x
 if *p ≠ 0 r ≠ 0 order x p > order x r*
proof –

```

define xo where  $xo = [- x, 1] \wedge \text{order } x \ q$ 
have [simp]:  $xo \neq 0$  unfolding xo-def by simp
have  $xo \cdot q : \text{order } x \ xo = \text{order } x \ q$ 
  unfolding xo-def by (meson order-power-n-n)
obtain q' where  $q : q = xo * q'$  and  $\neg [- x, 1] \ \text{dvd } q'$ 
  using order-decomp[OF  $\langle q \neq 0 \rangle$ , of x] unfolding xo-def by auto
from this(2)
have poly  $q' \ x \neq 0$  using poly-eq-0-iff-dvd by blast
define p' r' where  $p' = p \ \text{div } xo$  and  $r' = r \ \text{div } xo$ 
have  $p : p = xo * p'$ 
proof -
  have  $\text{order } x \ q < \text{order } x \ p$ 
    using assms(1) less-trans that(3) by blast
  then have  $xo \ \text{dvd } p$ 
    unfolding xo-def by (metis less-or-eq-imp-le order-divides)
  then show ?thesis by (simp add: p'-def)
qed
have  $r : r = xo * r'$ 
proof -
  have  $xo \ \text{dvd } r$ 
    unfolding xo-def by (meson assms(1) less-or-eq-imp-le order-divides)
  then show ?thesis by (simp add: r'-def)
qed
have poly  $r' \ x = 0$ 
proof -
  have  $\text{order } x \ r = \text{order } x \ xo + \text{order } x \ r'$ 
    unfolding r using  $\langle r \neq 0 \rangle$  r order-mult by blast
  with  $xo \cdot q$  have  $\text{order } x \ r' = \text{order } x \ r - \text{order } x \ q$ 
    by auto
  then have  $\text{order } x \ r' > 0$ 
    using  $\langle \text{order } x \ r < \text{order } x \ p \rangle$  assms(1) by linarith
  then show poly  $r' \ x = 0$  using order-root by blast
qed
have poly  $p' \ x = 0$ 
proof -
  have  $\text{order } x \ p = \text{order } x \ xo + \text{order } x \ p'$ 
    unfolding p using  $\langle p \neq 0 \rangle$  p order-mult by blast
  with  $xo \cdot q$  have  $\text{order } x \ p' = \text{order } x \ p - \text{order } x \ q$ 
    by auto
  then have  $\text{order } x \ p' > 0$ 
    using  $\langle \text{order } x \ r < \text{order } x \ p \rangle$  assms(1) by linarith
  then show poly  $p' \ x = 0$  using order-root by blast
qed

```

have *jumpF-polyL* $(r+q) \ p \ x = \text{jumpF-polyL } (xo * (r'+q')) \ (xo * p') \ x$
 unfolding *p q r* by (*simp add: algebra-simps*)
also have ... = *jumpF-polyL* $(r'+q') \ p' \ x$
 by (*rule jumpF-polyL-mult-cancel*) *simp*
also have ... = (*if even* $(\text{order } x \ p')$) = (*sign-r-pos* $p' \ x$)

$= (0 < \text{poly } (r' + q') x) \text{ then } 1 / 2 \text{ else } - 1 / 2)$
proof –
have $\text{poly } (r' + q') x \neq 0$
using $\langle \text{poly } q' x \neq 0 \rangle \langle \text{poly } r' x = 0 \rangle$ **by** *auto*
then show *?thesis*
apply (*subst jumpF-polyL-coprime'*)
subgoal by *simp*
subgoal by (*smt (verit) $\langle p \neq 0 \rangle \langle \text{poly } p' x = 0 \rangle \text{mult.commute}$*
mult-zero-left p poly-0)
done
qed
also have ... = (*if even (order x p') = (sign-r-pos p' x*
 $= (0 < \text{poly } q' x) \text{ then } 1 / 2 \text{ else } - 1 / 2)$
using $\langle \text{poly } r' x = 0 \rangle$ **by** *auto*
also have ... = *jumpF-polyL q' p' x*
apply (*subst jumpF-polyL-coprime'*)
subgoal using $\langle \text{poly } q' x \neq 0 \rangle$ **by** *blast*
subgoal using $\langle p \neq 0 \rangle \langle \text{poly } p' x = 0 \rangle$ *assms(2) p q* **by** *simp*
done
also have ... = *jumpF-polyL q p x*
unfolding $p q$ **by** (*subst jumpF-polyL-mult-cancel simp-all*)
finally show *jumpF-polyL (r+q) p x = jumpF-polyL q p x* .

have *jumpF-polyR (r+q) p x = jumpF-polyR (xo * (r'+q')) (xo*p') x*
unfolding $p q r$ **by** (*simp add:algebra-simps*)
also have ... = *jumpF-polyR (r'+q') p' x*
by (*rule jumpF-polyR-mult-cancel simp*)
also have ... = (*if sign-r-pos p' x = (0 < poly (r' + q') x*
 $\text{then } 1 / 2 \text{ else } - 1 / 2)$
proof –
have $\text{poly } (r' + q') x \neq 0$
using $\langle \text{poly } q' x \neq 0 \rangle \langle \text{poly } r' x = 0 \rangle$ **by** *auto*
then show *?thesis*
apply (*subst jumpF-polyR-coprime'*)
subgoal by *simp*
subgoal
by (*smt (verit) $\langle p \neq 0 \rangle \langle \text{poly } p' x = 0 \rangle \text{mult.commute}$*
mult-zero-left p poly-0)
done
qed
also have ... = (*if sign-r-pos p' x = (0 < poly q' x*
 $\text{then } 1 / 2 \text{ else } - 1 / 2)$
using $\langle \text{poly } r' x = 0 \rangle$ **by** *auto*
also have ... = *jumpF-polyR q' p' x*
apply (*subst jumpF-polyR-coprime'*)
subgoal using $\langle \text{poly } q' x \neq 0 \rangle$ **by** *blast*
subgoal using $\langle p \neq 0 \rangle \langle \text{poly } p' x = 0 \rangle$ *assms(2) p q* **by** *force*
done
also have ... = *jumpF-polyR q p x*

unfolding $p\ q$ **by** (*subst jumpF-polyR-mult-cancel*) *simp-all*
finally show $\text{jumpF-polyR } (r+q)\ p\ x = \text{jumpF-polyR } q\ p\ x$.
qed
ultimately show
 $\text{jumpF-polyR } (r+q)\ p\ x = \text{jumpF-polyR } q\ p\ x$
 $\text{jumpF-polyL } (r+q)\ p\ x = \text{jumpF-polyL } q\ p\ x$
by force +
qed

lemma *jumpF-poly-top-0[simp]*: $\text{jumpF-poly-top } 0\ p = 0\ \text{jumpF-poly-top } q\ 0 = 0$
unfolding *jumpF-poly-top-def* **by** *auto*

lemma *jumpF-poly-bot-0[simp]*: $\text{jumpF-poly-bot } 0\ p = 0\ \text{jumpF-poly-bot } q\ 0 = 0$
unfolding *jumpF-poly-bot-def* **by** *auto*

lemma *jumpF-poly-top-code*:
 $\text{jumpF-poly-top } q\ p = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{degree } q > \text{degree } p \text{ then}$
 $\quad \text{if } \text{sgn-pos-inf } q * \text{sgn-pos-inf } p > 0 \text{ then } 1/2 \text{ else } -1/2 \text{ else } 0)$
proof (*cases* $p \neq 0 \wedge q \neq 0 \wedge \text{degree } q > \text{degree } p$)
case *True*
have *?thesis when* $\text{sgn-pos-inf } q * \text{sgn-pos-inf } p > 0$
proof –
have $\text{LIM } x \text{ at-top. } \text{poly } q\ x / \text{poly } p\ x := \text{at-top}$
using *poly-divide-filterlim-at-top[of p q]* *True* **that** **by** *auto*
then have $\text{jumpF } (\lambda x. \text{poly } q\ x / \text{poly } p\ x) \text{ at-top} = 1/2$
unfolding *jumpF-def* **by** *auto*
then show *?thesis* **unfolding** *jumpF-poly-top-def* **using** *that True* **by** *auto*
qed
moreover have *?thesis when* $\neg \text{sgn-pos-inf } q * \text{sgn-pos-inf } p > 0$
proof –
have $\text{LIM } x \text{ at-top. } \text{poly } q\ x / \text{poly } p\ x := \text{at-bot}$
using *poly-divide-filterlim-at-top[of p q]* *True* **that** **by** *auto*
then have $\text{jumpF } (\lambda x. \text{poly } q\ x / \text{poly } p\ x) \text{ at-top} = -1/2$
unfolding *jumpF-def* **by** *auto*
then show *?thesis* **unfolding** *jumpF-poly-top-def* **using** *that True* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*

next
case *False*
define P **where** $P = (\neg (\text{LIM } x \text{ at-top. } \text{poly } q\ x / \text{poly } p\ x := \text{at-bot})$
 $\quad \wedge \neg (\text{LIM } x \text{ at-top. } \text{poly } q\ x / \text{poly } p\ x := \text{at-top}))$
have P **when** $p=0 \vee q=0$
unfolding P -*def* **using** *that*
by (*auto elim!*:*filterlim-at-bot-nhds filterlim-at-top-nhds*)
moreover have P **when** $p \neq 0\ q \neq 0\ \text{degree } p > \text{degree } q$
proof –
have $\text{LIM } x \text{ at-top. } \text{poly } q\ x / \text{poly } p\ x := \text{at } 0$
using *poly-divide-filterlim-at-top[OF that(1,2)] that(3)* **by** *auto*
then show *?thesis* **unfolding** P -*def*

by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds simp:filterlim-at)
 qed
 moreover have P when $p \neq 0 \ q \neq 0 \ \text{degree } p = \text{degree } q$
 proof -
 have $((\lambda x. \text{poly } q \ x / \text{poly } p \ x) \longrightarrow \text{lead-coeff } q / \text{lead-coeff } p) \text{ at-top}$
 using poly-divide-filterlim-at-top[OF that(1,2)] using that by auto
 then show ?thesis unfolding P-def
 by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds)
 qed
 ultimately have P using False by fastforce
 then have $\text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \text{ at-top} = 0$
 unfolding jumpF-def P-def by auto
 then show ?thesis unfolding jumpF-poly-top-def using False by presburger
 qed

lemma jumpF-poly-bot-code:

$\text{jumpF-poly-bot } q \ p = (\text{if } p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{degree } q > \text{degree } p \ \text{then}$
 $\quad \text{if } \text{sgn-neg-inf } q * \text{sgn-neg-inf } p > 0 \ \text{then } 1/2 \ \text{else } -1/2 \ \text{else } 0)$
 proof (cases $p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{degree } q > \text{degree } p$)
 case True
 have ?thesis when $\text{sgn-neg-inf } q * \text{sgn-neg-inf } p > 0$
 proof -
 have $LIM \ x \ \text{at-bot. } \text{poly } q \ x / \text{poly } p \ x \ :> \ \text{at-top}$
 using poly-divide-filterlim-at-bot[of p q] True that by auto
 then have $\text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ \text{at-bot} = 1/2$
 unfolding jumpF-def by auto
 then show ?thesis unfolding jumpF-poly-bot-def using that True by auto
 qed
 moreover have ?thesis when $\neg \text{sgn-neg-inf } q * \text{sgn-neg-inf } p > 0$
 proof -
 have $LIM \ x \ \text{at-bot. } \text{poly } q \ x / \text{poly } p \ x \ :> \ \text{at-bot}$
 using poly-divide-filterlim-at-bot[of p q] True that by auto
 then have $\text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ \text{at-bot} = -1/2$
 unfolding jumpF-def by auto
 then show ?thesis unfolding jumpF-poly-bot-def using that True by auto
 qed
 ultimately show ?thesis by auto

next

case False

define P where $P = (\neg (LIM \ x \ \text{at-bot. } \text{poly } q \ x / \text{poly } p \ x \ :> \ \text{at-bot})$
 $\quad \wedge \neg (LIM \ x \ \text{at-bot. } \text{poly } q \ x / \text{poly } p \ x \ :> \ \text{at-top}))$

have P when $p=0 \ \vee \ q=0$

unfolding P-def using that

by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds)

moreover have P when $p \neq 0 \ q \neq 0 \ \text{degree } p > \text{degree } q$

proof -

have $LIM \ x \ \text{at-bot. } \text{poly } q \ x / \text{poly } p \ x \ :> \ \text{at } 0$

using poly-divide-filterlim-at-bot[OF that(1,2)] that(3) by auto

then show ?thesis unfolding P-def

by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds simp:filterlim-at)

qed

moreover have P when $p \neq 0$ $q \neq 0$ degree $p =$ degree q

proof –

 have $((\lambda x. \text{poly } q \ x / \text{poly } p \ x) \longrightarrow \text{lead-coeff } q / \text{lead-coeff } p)$ at-bot

 using poly-divide-filterlim-at-bot[OF that(1,2)] using that by auto

 then show ?thesis unfolding P-def

 by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds)

qed

ultimately have P using False by fastforce

then have $\text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x)$ at-bot = 0

 unfolding jumpF-def P-def by auto

then show ?thesis unfolding jumpF-poly-bot-def using False by presburger

qed

lemma jump-poly-jumpF-poly:

shows $\text{jump-poly } q \ p \ x = \text{jumpF-polyR } q \ p \ x - \text{jumpF-polyL } q \ p \ x$

proof (cases $p=0 \vee q=0$)

 case True

 then show ?thesis by auto

next

 case False

 have $*\text{jump-poly } q \ p \ x = \text{jumpF-polyR } q \ p \ x - \text{jumpF-polyL } q \ p \ x$

 if coprime $q \ p$ for $q \ p$

 proof (cases $p=0 \vee q=0 \vee \text{poly } p \ x \neq 0$)

 case True

 moreover have ?thesis if $p=0 \vee q=0$ using that by auto

 moreover have ?thesis if $\text{poly } p \ x \neq 0$

 by (simp add: jumpF-poly-noroot(1) jumpF-poly-noroot(2) jump-poly-not-root that)

 ultimately show ?thesis by blast

 next

 case False

 then have $p \neq 0$ $q \neq 0$ $\text{poly } p \ x = 0$ by auto

 have $\text{jump-poly } q \ p \ x = \text{jump } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ x$

 using jump-jump-poly by simp

 also have $\text{real-of-int } \dots = \text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) (\text{at-right } x) -$

$\text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) (\text{at-left } x)$

 proof (rule jump-jumpF)

 have $\text{poly } q \ x \neq 0$ by (meson False coprime-poly-0 that)

 then show isCont (inverse $\circ (\lambda x. \text{poly } q \ x / \text{poly } p \ x)$) x

 unfolding comp-def by simp

 define l where $l = \text{sgnx } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) (\text{at-left } x)$

 define r where $r = \text{sgnx } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) (\text{at-right } x)$

 show $(\lambda x. \text{poly } q \ x / \text{poly } p \ x) \text{ has-sgnx } l$ (at-left x)

 unfolding l-def by (auto intro!:sgnx-intros sgnx-able-sgnx)

 show $(\lambda x. \text{poly } q \ x / \text{poly } p \ x) \text{ has-sgnx } r$ (at-right x)

```

    unfolding r-def by (auto intro!:sgnx-intros sgnx-able-sgnx)
  show l≠0 unfolding l-def
    apply (subst sgnx-divide)
    using poly-sgnx-values[OF ⟨p≠0⟩, of x] poly-sgnx-values[OF ⟨q≠0⟩, of x]
    by auto
  show r≠0 unfolding r-def
    apply (subst sgnx-divide)
    using poly-sgnx-values[OF ⟨p≠0⟩, of x] poly-sgnx-values[OF ⟨q≠0⟩, of x]
    by auto
qed
also have ... = jumpF-polyR q p x - jumpF-polyL q p x
  unfolding jumpF-polyR-def jumpF-polyL-def by simp
finally show ?thesis .
qed

```

```

obtain p' q' g where pq:p=g*p' q=g*q' and coprime q' p' g=gcd p q
  using gcd-coprime-exists[of p q]
  by (metis False coprime-commute gcd-coprime-exists gcd-eq-0-iff mult.commute)
then have g≠0 using False mult-zero-left by blast
then have jump-poly q p x = jump-poly q' p' x
  unfolding pq using jump-poly-mult by auto
also have ... = jumpF-polyR q' p' x - jumpF-polyL q' p' x
  using *[OF ⟨coprime q' p'⟩] .
also have ... = jumpF-polyR q p x - jumpF-polyL q p x
  unfolding pq using ⟨g≠0⟩ jumpF-polyL-mult-cancel jumpF-polyR-mult-cancel
by auto
finally show ?thesis .
qed

```

2.6 The extended Cauchy index on polynomials

definition *cindex-polyE*:: *real* \Rightarrow *real* \Rightarrow *real poly* \Rightarrow *real poly* \Rightarrow *real* **where**
cindex-polyE a b q p = *jumpF-polyR* q p a + *cindex-poly* a b q p - *jumpF-polyL* q p b

definition *cindex-poly-ubd*::*real poly* \Rightarrow *real poly* \Rightarrow *int* **where**
cindex-poly-ubd q p = (THE l. (\forall_F r in at-top. *cindexE* (-r) r (λx . *poly* q x / *poly* p x) = of-int l))

lemma *cindex-polyE-0*[*simp*]: *cindex-polyE* a b 0 p = 0 *cindex-polyE* a b q 0 = 0
unfolding *cindex-polyE-def* **by** *auto*

lemma *cindex-polyE-mult-cancel*:
fixes p q p'::*real poly*
assumes p'≠0
shows *cindex-polyE* a b (p' * q) (p' * p) = *cindex-polyE* a b q p
unfolding *cindex-polyE-def*
using *cindex-poly-mult*[OF ⟨p'≠0⟩] *jumpF-polyL-mult-cancel*[OF ⟨p'≠0⟩]
jumpF-polyR-mult-cancel[OF ⟨p'≠0⟩]

by *simp*

lemma *cindexE-eq-cindex-polyE*:

assumes $a < b$

shows $cindexE\ a\ b\ (\lambda x. poly\ q\ x / poly\ p\ x) = cindex-polyE\ a\ b\ q\ p$

proof (*cases* $p=0 \vee q=0$)

case *True*

then show *?thesis* by (*auto simp add: cindexE-constI*)

next

case *False*

then have $p \neq 0\ q \neq 0$ by *auto*

define g where $g = gcd\ p\ q$

define $p'\ q'$ where $p' = p\ div\ g$ and $q' = q\ div\ g$

define f' where $f' = (\lambda x. poly\ q'\ x / poly\ p'\ x)$

have $g \neq 0$ using *False g-def* by *auto*

have $pq-f: p = g * p'\ q = g * q'$ and *coprime* $p'\ q'$

unfolding *g-def p'-def q'-def*

apply *simp-all*

using *False div-gcd-coprime* by *blast*

have $cindexE\ a\ b\ (\lambda x. poly\ q\ x / poly\ p\ x) = cindexE\ a\ b\ (\lambda x. poly\ q'\ x / poly\ p'\ x)$

proof –

define f where $f = (\lambda x. poly\ q\ x / poly\ p\ x)$

define f' where $f' = (\lambda x. poly\ q'\ x / poly\ p'\ x)$

have *jumpF* $f\ (at-right\ x) = jumpF\ f'\ (at-right\ x)$ for x

proof (*rule jumpF-cong*)

obtain ub where $x < ub\ \forall z. x < z \wedge z \leq ub \longrightarrow poly\ g\ z \neq 0$

using *next-non-root-interval[OF ‹g≠0›, of x]* by *auto*

then show $\forall_F\ x\ in\ at-right\ x. f\ x = f'\ x$

unfolding *eventually-at-right f-def f'-def pq-f*

apply (*intro exI[where x=ub]*)

by *auto*

qed *simp*

moreover have *jumpF* $f\ (at-left\ x) = jumpF\ f'\ (at-left\ x)$ for x

proof (*rule jumpF-cong*)

obtain lb where $lb < x\ \forall z. lb \leq z \wedge z < x \longrightarrow poly\ g\ z \neq 0$

using *last-non-root-interval[OF ‹g≠0›, of x]* by *auto*

then show $\forall_F\ x\ in\ at-left\ x. f\ x = f'\ x$

unfolding *eventually-at-left f-def f'-def pq-f*

apply (*intro exI[where x=lb]*)

by *auto*

qed *simp*

ultimately show *?thesis* unfolding *cindexE-def*

apply (*fold f-def f'-def*)

by *auto*

qed

also have ... = *jumpF* $f'\ (at-right\ a) + real-of-int\ (cindex\ a\ b\ f') - jumpF\ f'$
(*at-left\ b*)

unfolding *f'-def*

apply (*rule cindex-eq-cindexE-divide*)

```

subgoal using ⟨a<b⟩ .
subgoal
proof –
  have finite (proots (q*p'))
    using False poly-roots-finite pq-f(1) pq-f(2) by auto
  then show finite {x. (poly q' x = 0 ∨ poly p' x = 0) ∧ a ≤ x ∧ x ≤ b}
    by (elim rev-finite-subset) auto
qed
subgoal using ⟨coprime p' q'⟩ poly-gcd-0-iff by force
subgoal by (auto intro:continuous-intros)
subgoal by (auto intro:continuous-intros)
done
also have ... = cindex-polyE a b q' p'
  using cindex-eq-cindex-poly unfolding cindex-polyE-def jumpF-polyR-def jumpF-polyL-def
f'-def
  by auto
also have ... = cindex-polyE a b q p
  using cindex-polyE-mult-cancel[OF ⟨g≠0⟩] unfolding pq-f by auto
finally show ?thesis .
qed

lemma cindex-polyE-cross:
  fixes p::real poly and a b::real
  assumes a<b
  shows cindex-polyE a b 1 p = cross-alt 1 p a b / 2
proof (induct degree p arbitrary:p rule:nat-less-induct)
  case induct:1
  have ?case when p=0
    using that unfolding cross-alt-def by auto
  moreover have ?case when p≠0 and noroot:{x. a < x ∧ x < b ∧ poly p x = 0}
  = {}
proof –
  have cindex-polyE a b 1 p = jumpF-polyR 1 p a - jumpF-polyL 1 p b
proof –
  have cindex-poly a b 1 p = 0 unfolding cindex-poly-def
    apply (rule sum.neutral)
    using that by auto
  then show ?thesis unfolding cindex-polyE-def by auto
qed
also have ... = cross-alt 1 p a b / 2
proof –
  define f where f = (λx. 1 / poly p x)
  define ja where ja = jumpF f (at-right a)
  define jb where jb = jumpF f (at-left b)
  define right where right = (λR. R ja (0::real) ∨ (continuous (at-right a) f
  ∧ R (poly p a) 0))
  define left where left = (λR. R jb (0::real) ∨ (continuous (at-left b) f
  ∧ R (poly p b) 0))

```

note $ja\text{-alt}=\text{jumpF-polyR-coprime}[of\ p\ 1\ a,\text{unfolded}\ \text{jumpF-polyR-def},\text{simplified},\text{folded}\ f\text{-def}\ ja\text{-def}]$

note $jb\text{-alt}=\text{jumpF-polyL-coprime}[of\ p\ 1\ b,\text{unfolded}\ \text{jumpF-polyL-def},\text{simplified},\text{folded}\ f\text{-def}\ jb\text{-def}]$

have $[simp]: 0 < ja \iff \text{jumpF-polyR}\ 1\ p\ a = 1/2\ 0 > ja \iff \text{jumpF-polyR}\ 1\ p\ a = -1/2$
 $0 < jb \iff \text{jumpF-polyL}\ 1\ p\ b = 1/2\ 0 > jb \iff \text{jumpF-polyL}\ 1\ p\ b = -1/2$

unfolding $ja\text{-def}\ jb\text{-def}\ \text{jumpF-polyR-def}\ \text{jumpF-polyL-def}\ f\text{-def}\ \text{jumpF-def}$
by *auto*

have $[simp]:$

$\text{poly}\ p\ a \neq 0 \implies \text{continuous}\ (\text{at-right}\ a)\ f$

$\text{poly}\ p\ b \neq 0 \implies \text{continuous}\ (\text{at-left}\ b)\ f$

unfolding $f\text{-def}$ **by** $(\text{auto}\ \text{intro!}:\ \text{continuous-intros})$

have $\text{not-right-left}:\ \text{False}$ **when** $(\text{right}\ \text{greater} \wedge \text{left}\ \text{less} \vee \text{right}\ \text{less} \wedge \text{left}\ \text{greater})$

proof –

have $[simp]: f\ a > 0 \iff \text{poly}\ p\ a > 0\ f\ a < 0 \iff \text{poly}\ p\ a < 0$

$f\ b > 0 \iff \text{poly}\ p\ b > 0\ f\ b < 0 \iff \text{poly}\ p\ b < 0$

unfolding $f\text{-def}$ **by** *auto*

have $\text{continuous-on}\ \{a < .. < b\}\ f$

unfolding $f\text{-def}$ **using** noroot **by** $(\text{auto}\ \text{intro!}:\ \text{continuous-intros})$

then **have** $\exists x > a.\ x < b \wedge f\ x = 0$

apply $(\text{elim}\ \text{jumpF-IVT}[OF\ \langle a < b \rangle,\ of\ f])$

using $\text{that}\ \text{unfolding}\ \text{right-def}\ \text{left-def}$ **by** $(\text{fold}\ ja\text{-def}\ jb\text{-def},\ \text{auto})$

then **show** False **using** noroot **using** $f\text{-def}$ **by** *auto*

qed

have $?thesis$ **when** $\text{poly}\ p\ a > 0 \wedge \text{poly}\ p\ b > 0 \vee \text{poly}\ p\ a < 0 \wedge \text{poly}\ p\ b < 0$

using $\text{that}\ \text{jumpF-poly-noroot}$

unfolding $\text{cross-alt-def}\ \text{psign-diff-def}$ **by** *auto*

moreover **have** False **when** $\text{poly}\ p\ a > 0 \wedge \text{poly}\ p\ b < 0 \vee \text{poly}\ p\ a < 0 \wedge \text{poly}\ p\ b > 0$

apply $(\text{rule}\ \text{not-right-left})$

unfolding $\text{right-def}\ \text{left-def}$ **using** that **by** *auto*

moreover **have** $?thesis$ **when** $\text{poly}\ p\ a = 0\ \text{poly}\ p\ b > 0 \vee \text{poly}\ p\ b < 0$

proof –

have $ja > 0 \vee ja < 0$ **using** $ja\text{-alt}\ \langle p \neq 0 \rangle\ \langle \text{poly}\ p\ a = 0 \rangle$ **by** *argo*

moreover **have** False **when** $ja > 0 \wedge \text{poly}\ p\ b < 0 \vee ja < 0 \wedge \text{poly}\ p\ b > 0$

apply $(\text{rule}\ \text{not-right-left})$

unfolding $\text{right-def}\ \text{left-def}$ **using** that **by** *fastforce*

moreover **have** $?thesis$ **when** $ja > 0 \wedge \text{poly}\ p\ b > 0 \vee ja < 0 \wedge \text{poly}\ p\ b < 0$

using $\text{that}\ \text{jumpF-poly-noroot}\ \langle \text{poly}\ p\ a = 0 \rangle$

unfolding $\text{cross-alt-def}\ \text{psign-diff-def}$ **by** *auto*

ultimately **show** $?thesis$ **using** $\text{that}\ \text{jumpF-poly-noroot}$ **unfolding** cross-alt-def

by *auto*

qed

moreover **have** $?thesis$ **when** $\text{poly}\ p\ b = 0\ \text{poly}\ p\ a > 0 \vee \text{poly}\ p\ a < 0$

proof –

```

    have  $jb > 0 \vee jb < 0$  using  $jb\text{-alt } \langle p \neq 0 \rangle \langle \text{poly } p \text{ } b = 0 \rangle$  by argo
    moreover have  $\text{False}$  when  $jb > 0 \wedge \text{poly } p \text{ } a < 0 \vee jb < 0 \wedge \text{poly } p \text{ } a > 0$ 
      apply (rule not-right-left)
      unfolding right-def left-def using that by fastforce
    moreover have ?thesis when  $jb > 0 \wedge \text{poly } p \text{ } a > 0 \vee jb < 0 \wedge \text{poly } p \text{ } a < 0$ 
      using that jumpF-poly-noroot  $\langle \text{poly } p \text{ } b = 0 \rangle$ 
      unfolding cross-alt-def psign-diff-def by auto
    ultimately show ?thesis using that jumpF-poly-noroot unfolding cross-alt-def
  by auto
  qed
  moreover have ?thesis when  $\text{poly } p \text{ } a = 0 \text{ } \text{poly } p \text{ } b = 0$ 
  proof -
    have  $jb > 0 \vee jb < 0$  using  $jb\text{-alt } \langle p \neq 0 \rangle \langle \text{poly } p \text{ } b = 0 \rangle$  by argo
    moreover have  $ja > 0 \vee ja < 0$  using  $ja\text{-alt } \langle p \neq 0 \rangle \langle \text{poly } p \text{ } a = 0 \rangle$  by argo
    moreover have  $\text{False}$  when  $ja > 0 \wedge jb < 0 \vee ja < 0 \wedge jb > 0$ 
      apply (rule not-right-left)
      unfolding right-def left-def using that by fastforce
    moreover have ?thesis when  $ja > 0 \wedge jb > 0 \vee ja < 0 \wedge jb < 0$ 
      using that jumpF-poly-noroot  $\langle \text{poly } p \text{ } b = 0 \rangle \langle \text{poly } p \text{ } a = 0 \rangle$ 
      unfolding cross-alt-def psign-diff-def by auto
    ultimately show ?thesis by blast
  qed
  ultimately show ?thesis by argo
  qed
  finally show ?thesis .
  qed
  moreover have ?case when  $p \neq 0$  and  $\text{no-empty}:\{x. a < x \wedge x < b \wedge \text{poly } p \text{ } x = 0\}$ 
} ≠ {}
  proof -
    define roots where  $\text{roots} \equiv \{x. a < x \wedge x < b \wedge \text{poly } p \text{ } x = 0\}$ 
    have finite roots unfolding roots-def using poly-roots-finite[OF  $\langle p \neq 0 \rangle$ ] by
  auto
    define max-r where  $\text{max-r} \equiv \text{Max roots}$ 
    hence  $\text{poly } p \text{ } \text{max-r} = 0$  and  $a < \text{max-r}$  and  $\text{max-r} < b$ 
      using Max-in[OF  $\langle \text{finite roots} \rangle$ ] no-empty unfolding roots-def by auto
    define max-rp where  $\text{max-rp} \equiv [-\text{max-r}, 1:]^{\wedge} \text{order } \text{max-r } p$ 
    then obtain p' where p'-def:  $p = p' * \text{max-rp}$  and  $\neg [-\text{max-r}, 1:] \text{ dvd } p'$ 
      by (metis  $\langle p \neq 0 \rangle$  mult.commute order-decomp)
    hence  $p' \neq 0$  and  $\text{max-rp} \neq 0$  and  $\text{max-r-nz}:\text{poly } p' \text{ } \text{max-r} \neq 0$ 

    using  $\langle p \neq 0 \rangle$  by (auto simp add: dvd-iff-poly-eq-0)
    define max-r-sign where  $\text{max-r-sign} \equiv \text{if odd } (\text{order } \text{max-r } p) \text{ then } -1 \text{ else } 1::\text{int}$ 
    define roots' where  $\text{roots}' \equiv \{x. a < x \wedge x < b \wedge \text{poly } p' \text{ } x = 0\}$ 

    have  $\text{cindex-polyE } a \ b \ 1 \ p = \text{jumpF-polyR } 1 \ p \ a + (\sum x \in \text{roots}. \text{jump-poly } 1 \ p \ x) - \text{jumpF-polyL } 1 \ p \ b$ 
      unfolding cindex-polyE-def cindex-poly-def roots-def by (simp, meson)
    also have ... =  $\text{max-r-sign} * \text{cindex-poly } a \ b \ 1 \ p' + \text{jump-poly } 1 \ p \ \text{max-r}$ 
      +  $\text{max-r-sign} * \text{jumpF-polyR } 1 \ p' \ a - \text{jumpF-polyL } 1 \ p' \ b$ 

```

proof –
have $(\sum x \in \text{roots}. \text{jump-poly } 1 \ p \ x) = \text{max-r-sign} * \text{cindex-poly } a \ b \ 1 \ p' + \text{jump-poly } 1 \ p \ \text{max-r}$
proof –
have $(\sum x \in \text{roots}. \text{jump-poly } 1 \ p \ x) = (\sum x \in \text{roots}'. \text{jump-poly } 1 \ p \ x) + \text{jump-poly } 1 \ p \ \text{max-r}$
proof –
have $\text{roots} = \text{insert } \text{max-r} \ \text{roots}'$
unfolding $\text{roots-def } \text{roots}'\text{-def } p'\text{-def}$
using $\langle \text{poly } p \ \text{max-r}=0 \rangle \langle a < \text{max-r} \rangle \langle \text{max-r} < b \rangle \langle p \neq 0 \rangle \text{order-root}$
apply $(\text{subst } \text{max-rp-def})$
by auto
moreover **have** $\text{finite } \text{roots}'$
unfolding $\text{roots}'\text{-def}$ **using** $\text{poly-roots-finite}[OF \ \langle p' \neq 0 \rangle]$ **by** auto
moreover **have** $\text{max-r} \notin \text{roots}'$
unfolding $\text{roots}'\text{-def}$ **using** max-r-nz
by auto
ultimately show $?thesis$ **using** $\text{sum.insert}[of \ \text{roots}' \ \text{max-r}]$ **by** auto
qed
moreover **have** $(\sum x \in \text{roots}'. \text{jump-poly } 1 \ p \ x) = \text{max-r-sign} * \text{cindex-poly } a \ b \ 1 \ p'$
proof –
have $(\sum x \in \text{roots}'. \text{jump-poly } 1 \ p \ x) = (\sum x \in \text{roots}'. \text{max-r-sign} * \text{jump-poly } 1 \ p' \ x)$
proof $(\text{rule } \text{sum.cong}, \text{rule } \text{refl})$
fix x **assume** $x \in \text{roots}'$
hence $x \neq \text{max-r}$ **using** max-r-nz **unfolding** $\text{roots}'\text{-def}$
by auto
hence $\text{poly } \text{max-rp} \ x \neq 0$ **using** poly-power-n-eq **unfolding** max-rp-def
by auto
hence $\text{order } x \ \text{max-rp}=0$ **by** $(\text{metis } \text{order-root})$
moreover **have** $\text{jump-poly } 1 \ \text{max-rp} \ x=0$
using $\langle \text{poly } \text{max-rp} \ x \neq 0 \rangle$ **by** $(\text{metis } \text{jump-poly-not-root})$
moreover **have** $x \in \text{roots}$
using $\langle x \in \text{roots}' \rangle$ **unfolding** $\text{roots-def } \text{roots}'\text{-def } p'\text{-def}$ **by** auto
hence $x < \text{max-r}$
using $\text{Max-ge}[OF \ \langle \text{finite } \text{roots} \rangle, of \ x] \ \langle x \neq \text{max-r} \rangle$ **by** $(\text{fold } \text{max-r-def}, \text{auto})$
hence $\text{sign } (\text{poly } \text{max-rp} \ x) = \text{max-r-sign}$
using $\langle \text{poly } \text{max-rp} \ x \neq 0 \rangle$ **unfolding** $\text{max-r-sign-def } \text{max-rp-def } \text{sign-def}$
by $(\text{subst } \text{poly-power}, \text{simp } \text{add:linorder-class.not-less } \text{zero-less-power-eq})$
ultimately show $\text{jump-poly } 1 \ p \ x = \text{max-r-sign} * \text{jump-poly } 1 \ p' \ x$
using $\text{jump-poly-1-mult}[of \ p' \ x \ \text{max-rp}]$ **unfolding** $p'\text{-def}$
by $(\text{simp } \text{add: } \langle \text{poly } \text{max-rp} \ x \neq 0 \rangle)$
qed
also **have** $\dots = \text{max-r-sign} * (\sum x \in \text{roots}'. \text{jump-poly } 1 \ p' \ x)$
by $(\text{simp } \text{add: } \text{sum-distrib-left})$
also **have** $\dots = \text{max-r-sign} * \text{cindex-poly } a \ b \ 1 \ p'$
unfolding $\text{cindex-poly-def } \text{roots}'\text{-def}$ **by** meson
finally show $?thesis$.

```

qed
ultimately show ?thesis by simp
qed
moreover have jumpF-polyR 1 p a = max-r-sign * jumpF-polyR 1 p' a
proof -
  define f where f = (λx. 1 / poly max-rp x)
  define g where g = (λx. 1 / poly p' x)
  have jumpF-polyR 1 p a = jumpF (λx. f x * g x) (at-right a)
    unfolding jumpF-polyR-def f-def g-def p'-def
    by (auto simp add:field-simps)
  also have ... = sgn (f a) * jumpF g (at-right a)
  proof (rule jumpF-times)
    have [simp]: poly max-rp a ≠ 0
      unfolding max-rp-def using ‹max-r>a› by auto
    show (f ⟶ f a) (at-right a) f a ≠ 0
      unfolding f-def by (auto intro:tendsto-intros)
  qed auto
  also have ... = max-r-sign * jumpF-polyR 1 p' a
  proof -
    have sgn (f a) = max-r-sign
      unfolding max-r-sign-def f-def max-rp-def using ‹a<max-r›
      by (auto simp add:sgn-power)
    then show ?thesis unfolding jumpF-polyR-def g-def by auto
  qed
  finally show ?thesis .
qed
moreover have jumpF-polyL 1 p b = jumpF-polyL 1 p' b
proof -
  define f where f = (λx. 1 / poly max-rp x)
  define g where g = (λx. 1 / poly p' x)
  have jumpF-polyL 1 p b = jumpF (λx. f x * g x) (at-left b)
    unfolding jumpF-polyL-def f-def g-def p'-def
    by (auto simp add:field-simps)
  also have ... = sgn (f b) * jumpF g (at-left b)
  proof (rule jumpF-times)
    have [simp]: poly max-rp b ≠ 0
      unfolding max-rp-def using ‹max-r<b› by auto
    show (f ⟶ f b) (at-left b) f b ≠ 0
      unfolding f-def by (auto intro:tendsto-intros)
  qed auto
  also have ... = jumpF-polyL 1 p' b
  proof -
    have sgn (f b) = 1
      unfolding max-r-sign-def f-def max-rp-def using ‹b>max-r›
      by (auto simp add:sgn-power)
    then show ?thesis unfolding jumpF-polyL-def g-def by auto
  qed
  finally show ?thesis .
qed

```

```

ultimately show ?thesis by auto
qed
also have ... = max-r-sign * cindex-polyE a b 1 p' + jump-poly 1 p max-r
  + (max-r-sign - 1) * jumpF-polyL 1 p' b
  unfolding cindex-polyE-def roots'-def by (auto simp add: algebra-simps)
also have ... = max-r-sign * cross-alt 1 p' a b / 2 + jump-poly 1 p max-r
  + (max-r-sign - 1) * jumpF-polyL 1 p' b
proof -
  have degree max-rp > 0 unfolding max-rp-def degree-linear-power
    using ⟨poly p max-r = 0⟩ order-root ⟨p ≠ 0⟩ by blast
  then have degree p' < degree p unfolding p'-def
    using degree-mult-eq[OF ⟨p' ≠ 0⟩ ⟨max-rp ≠ 0⟩] by auto
  from induct[rule-format, OF this]
  have cindex-polyE a b 1 p' = real-of-int (cross-alt 1 p' a b) / 2 by auto
  then show ?thesis by auto
qed
also have ... = real-of-int (cross-alt 1 p a b) / 2
proof -
  have sjump-p: jump-poly 1 p max-r = (if odd (order max-r p) then sign (poly
p' max-r) else 0)
  proof -
    note max-r-nz
    moreover then have poly max-rp max-r = 0
      using ⟨poly p max-r = 0⟩ p'-def by auto
    ultimately have jump-poly 1 p max-r = sign (poly p' max-r) * jump-poly
1 max-rp max-r
      unfolding p'-def using jump-poly-1-mult[of p' max-r max-rp]
      by auto
    also have ... = (if odd (order max-r max-rp) then sign (poly p' max-r) else
0)
  proof -
    have sign-r-pos max-rp max-r
      unfolding max-rp-def using sign-r-pos-power by auto
    then show ?thesis using ⟨max-rp ≠ 0⟩ unfolding jump-poly-def by auto
  qed
  also have ... = (if odd (order max-r p) then sign (poly p' max-r) else 0)
  proof -
    have order max-r p' = 0 by (simp add: ⟨poly p' max-r ≠ 0⟩ order-0I)
    then have order max-r max-rp = order max-r p
      unfolding p'-def using ⟨p' ≠ 0⟩ ⟨max-rp ≠ 0⟩
      apply (subst order-mult)
      by auto
    then show ?thesis by auto
  qed
  finally show ?thesis .
qed
have ?thesis when even (order max-r p)
proof -
  have sign (poly p x) = (sign (poly p' x)::int) when x ≠ max-r for x

```

```

proof –
  have sign (poly max-rp x) = (1::int)
    unfolding max-rp-def using  $\langle \text{even (order max-r p)} \rangle$  that
    apply (simp add:sign-power)
    by (simp add: Sturm-Tarski.sign-def)
  then show ?thesis unfolding p'-def by (simp add:sign-times)
qed
from this[of a] this[of b]  $\langle a < \text{max-r} \rangle \langle \text{max-r} < b \rangle$ 
have cross-alt 1 p' a b = cross-alt 1 p a b
  unfolding cross-alt-def psign-diff-def by auto
then show ?thesis using that unfolding max-r-sign-def sjump-p by auto
qed
moreover have ?thesis when odd (order max-r p)
proof –
let ?thesis2 = sign (poly p' max-r) * 2 – cross-alt 1 p' a b – 4 * jumpF-polyL
1 p' b
  = cross-alt 1 p a b
have ?thesis2 when poly p' b=0
proof –
  have jumpF-polyL 1 p' b = 1/2  $\vee$  jumpF-polyL 1 p' b=-1/2
  using jumpF-polyL-coprime[of p' 1 b,simplified]  $\langle p' \neq 0 \rangle \langle \text{poly } p' b=0 \rangle$  by
auto
  moreover have poly p' max-r > 0  $\vee$  poly p' max-r < 0
  using max-r-nz by auto
moreover have False when poly p' max-r > 0  $\wedge$  jumpF-polyL 1 p' b=-1/2
   $\vee$  poly p' max-r < 0  $\wedge$  jumpF-polyL 1 p' b=1/2
proof –
  define f where f = ( $\lambda x. 1 / \text{poly } p' x$ )
  have noroots:poly p' x  $\neq 0$  when  $x \in \{\text{max-r} < .. < b\}$  for x
  proof (rule ccontr)
    assume  $\neg$  poly p' x  $\neq 0$ 
    then have poly p x = 0 unfolding p'-def by auto
    then have x  $\in$  roots unfolding roots-def using that  $\langle a < \text{max-r} \rangle$  by auto
    then have x  $\leq$  max-r using Max-ge[OF  $\langle \text{finite roots} \rangle$ ] unfolding
max-r-def by auto
    moreover have x > max-r using that by auto
    ultimately show False by auto
  qed
have continuous-on  $\{\text{max-r} < .. < b\}$  f
  unfolding f-def using noroots by (auto intro!:continuous-intros)
moreover have continuous (at-right max-r) f
  unfolding f-def using max-r-nz
  by (auto intro!:continuous-intros)
moreover have f max-r > 0  $\wedge$  jumpF f (at-left b) < 0
   $\vee$  f max-r < 0  $\wedge$  jumpF f (at-left b) > 0
  using that unfolding f-def jumpF-polyL-def by auto
ultimately have  $\exists x > \text{max-r}. x < b \wedge f x = 0$ 
  apply (intro jumpF-IVT[OF  $\langle \text{max-r} < b \rangle$ ])

```

```

    by auto
    then show False using noroots unfolding f-def by auto
  qed
  moreover have ?thesis when poly p' max-r > 0 ∧ jumpF-polyL 1 p' b = 1/2
    ∨ poly p' max-r < 0 ∧ jumpF-polyL 1 p' b = -1/2
  proof -
    have poly max-rp a < 0 poly max-rp b > 0
    unfolding max-rp-def using ⟨odd (order max-r p)⟩ ⟨a < max-r⟩ ⟨max-r < b⟩
    by (simp-all add:zero-less-power-eq)
    then have cross-alt 1 p a b = - cross-alt 1 p' a b
    unfolding cross-alt-def p'-def using ⟨poly p' b = 0⟩
    apply (simp add:sign-times)
    by (auto simp add: Sturm-Tarski.sign-def psign-diff-def zero-less-mult-iff)
    with that show ?thesis by auto
  qed
  ultimately show ?thesis by blast
  qed
  moreover have ?thesis2 when poly p' b ≠ 0
  proof -
    have [simp]: jumpF-polyL 1 p' b = 0
    using jumpF-polyL-coprime[of p' 1 b, simplified] ⟨poly p' b ≠ 0⟩ by auto
    have [simp]: poly max-rp a < 0 poly max-rp b > 0
    unfolding max-rp-def using ⟨odd (order max-r p)⟩ ⟨a < max-r⟩ ⟨max-r < b⟩
    by (simp-all add:zero-less-power-eq)
    have poly p' b > 0 ∨ poly p' b < 0
    using ⟨poly p' b ≠ 0⟩ by auto
    moreover have poly p' max-r > 0 ∨ poly p' max-r < 0
    using max-r-nz by auto
    moreover have ?thesis when poly p' b > 0 ∧ poly p' max-r > 0
    using that unfolding cross-alt-def p'-def psign-diff-def
    apply (simp add:sign-times)
    by (simp add: Sturm-Tarski.sign-def)
    moreover have ?thesis when poly p' b < 0 ∧ poly p' max-r < 0
    using that unfolding cross-alt-def p'-def psign-diff-def
    apply (simp add:sign-times)
    by (simp add: Sturm-Tarski.sign-def)
    moreover have False when poly p' b > 0 ∧ poly p' max-r < 0 ∨ poly p'
    b < 0 ∧ poly p' max-r > 0
    proof -
      have ∃ x > max-r. x < b ∧ poly p' x = 0
      apply (rule poly-IVT[OF ⟨max-r < b⟩, of p'])
      using that mult-less-0-iff by blast
      then obtain x where max-r < x < b poly p x = 0 unfolding p'-def by
    auto
    then have x ∈ roots using ⟨a < max-r⟩ unfolding roots-def by auto
    then have x ≤ max-r unfolding max-r-def using Max-ge[OF ⟨finite
    roots⟩] by auto
    then show False using ⟨max-r < x⟩ by auto
  qed

```

```

      ultimately show ?thesis by blast
    qed
    ultimately have ?thesis2 by auto
    then show ?thesis unfolding max-r-sign-def sjump-p using that by simp
  qed
  ultimately show ?thesis by auto
  qed
  finally show ?thesis .
  qed
  ultimately show ?case by fast
  qed

```

```

lemma cindex-polyE-inverse-add:
  fixes p q::real poly
  assumes cp:coprime p q
  shows cindex-polyE a b q p + cindex-polyE a b p q = cindex-polyE a b 1 (q*p)
  unfolding cindex-polyE-def
  using cindex-poly-inverse-add[OF cp,symmetric] jumpF-polyR-inverse-add[OF
cp,symmetric]
  jumpF-polyL-inverse-add[OF cp,symmetric]
  by auto

```

```

lemma cindex-polyE-inverse-add-cross:
  fixes p q::real poly
  assumes a < b coprime p q
  shows cindex-polyE a b q p + cindex-polyE a b p q = cross-alt p q a b / 2
  apply (subst cindex-polyE-inverse-add[OF ‹coprime p q›])
  apply (subst cindex-polyE-cross[OF ‹a<b›])
  apply (subst mult.commute)
  apply (subst (2) cross-alt-clear)
  by simp

```

```

lemma cindex-polyE-inverse-add-cross':
  fixes p q::real poly
  assumes a < b poly p a≠0 ∨ poly q a≠0 poly p b≠0 ∨ poly q b≠0
  shows cindex-polyE a b q p + cindex-polyE a b p q = cross-alt p q a b / 2
  proof -
    define g1 where g1 = gcd p q
    obtain p' q' where pq:p=g1*p' q=g1*q' and coprime p' q'
      unfolding g1-def
    by (metis assms(2) coprime-commute div-gcd-coprime dvd-mult-div-cancel gcd-dvd1

      gcd-dvd2 order-root)
    have [simp]:g1≠0
      unfolding g1-def using assms(2) by force

    have cindex-polyE a b q' p' + cindex-polyE a b p' q' = (cross-alt p' q' a b) / 2
      using cindex-polyE-inverse-add-cross[OF ‹a<b› ‹coprime p' q'›] .
    moreover have cindex-polyE a b p' q' = cindex-polyE a b p q

```

```

unfolding pq
apply (subst cindex-polyE-mult-cancel)
by simp-all
moreover have cindex-polyE a b q' p' = cindex-polyE a b q p
unfolding pq
apply (subst cindex-polyE-mult-cancel)
by simp-all
moreover have cross-alt p' q' a b = cross-alt p q a b
unfolding pq
apply (subst cross-alt-cancel)
subgoal using assms(2) g1-def poly-gcd-0-iff by blast
subgoal using assms(3) g1-def poly-gcd-0-iff by blast
by simp
ultimately show ?thesis by auto
qed

```

```

lemma cindex-polyE-smult-1:
  fixes p q::real poly and c::real
  shows cindex-polyE a b (smult c q) p = (sgn c) * cindex-polyE a b q p
proof -
  have real-of-int (sign c) = sgn c
  by (simp add: sgn-if)
  then show ?thesis
    unfolding cindex-polyE-def jumpF-polyL-smult-1 jumpF-polyR-smult-1 cindex-poly-smult-1
    by (auto simp add: algebra-simps)
qed

```

```

lemma cindex-polyE-smult-2:
  fixes p q::real poly and c::real
  shows cindex-polyE a b q (smult c p) = (sgn c) * cindex-polyE a b q p
proof (cases c=0)
  case True
  then show ?thesis by simp
next
  case False
  then have cindex-polyE a b q (smult c p)
    = cindex-polyE a b ([:1/c:]*q) ([:1/c:]*(smult c p))
  apply (subst cindex-polyE-mult-cancel)
  by simp-all
  also have ... = cindex-polyE a b (smult (1/c) q) p
  by simp
  also have ... = (sgn (1/c)) * cindex-polyE a b q p
  using cindex-polyE-smult-1 by simp
  also have ... = (sgn c) * cindex-polyE a b q p
  by simp
  finally show ?thesis .
qed

```

```

lemma cindex-polyE-mod:
  fixes  $p q :: \text{real poly}$ 
  shows  $\text{cindex-polyE } a \ b \ q \ p = \text{cindex-polyE } a \ b \ (q \ \text{mod } p) \ p$ 
  unfolding cindex-polyE-def
  apply (subst cindex-poly-mod)
  apply (subst jumpF-polyR-mod)
  apply (subst jumpF-polyL-mod)
  by simp

lemma cindex-polyE-rec:
  fixes  $p q :: \text{real poly}$ 
  assumes  $a < b$  coprime  $p \ q$ 
  shows  $\text{cindex-polyE } a \ b \ q \ p = \text{cross-alt } q \ p \ a \ b / 2 + \text{cindex-polyE } a \ b \ (- (p \ \text{mod } q)) \ q$ 
  proof -
    note cindex-polyE-inverse-add-cross[OF assms]
    moreover have  $\text{cindex-polyE } a \ b \ (- (p \ \text{mod } q)) \ q = - \text{cindex-polyE } a \ b \ p \ q$ 
      using cindex-polyE-mod cindex-polyE-smult-1[of a b -1]
      by auto
    ultimately show ?thesis by (auto simp add:field-simps cross-alt-poly-commute)
  qed

lemma cindex-polyE-changes-alt-itv-mods:
  assumes  $a < b$  coprime  $p \ q$ 
  shows  $\text{cindex-polyE } a \ b \ q \ p = \text{changes-alt-itv-smods } a \ b \ p \ q / 2$  using  $\langle \text{coprime } p \ q \rangle$ 
  proof (induct smods p q arbitrary:p q)
    case Nil
    then have  $p=0$  by (metis smods-nil-eq)
    then show ?case by (simp add:changes-alt-itv-smods-def changes-alt-poly-at-def)
  next
    case (Cons x xs)
    then have  $p \neq 0$  by auto
    have ?case when  $q=0$ 
      using that by (simp add:changes-alt-itv-smods-def changes-alt-poly-at-def)
    moreover have ?case when  $q \neq 0$ 
    proof -
      define  $r$  where  $r \equiv - (p \ \text{mod } q)$ 
      obtain  $ps$  where  $ps:\text{smods } p \ q = p \# q \# ps \ \text{smods } q \ r = q \# ps$  and  $xs = q \# ps$ 
        unfolding r-def using  $\langle q \neq 0 \rangle \langle p \neq 0 \rangle \langle x \# xs = \text{smods } p \ q \rangle$ 
        by (metis list.inject smods.simps)
      from Cons.prems  $\langle q \neq 0 \rangle$  have coprime  $q \ r$ 
        by (simp add: r-def ac-simps)
      then have  $\text{cindex-polyE } a \ b \ r \ q = \text{real-of-int } (\text{changes-alt-itv-smods } a \ b \ q \ r) /$ 
        2
        apply (rule-tac Cons.hyps(1))
        using  $ps \langle xs = q \# ps \rangle$  by simp-all
      moreover have  $\text{changes-alt-itv-smods } a \ b \ p \ q = \text{cross-alt } p \ q \ a \ b + \text{changes-alt-itv-smods}$ 

```

```

a b q r
  using changes-alt-itv-smods-rec[OF ‹a<b› ‹coprime p q›,folded r-def] .
  moreover have cindex-polyE a b q p = real-of-int (cross-alt q p a b) / 2 +
cindex-polyE a b r q
  using cindex-polyE-rec[OF ‹a<b› ‹coprime p q›,folded r-def] .
  ultimately show ?case
    by (auto simp add:field-simps cross-alt-poly-commute)
qed
ultimately show ?case by blast
qed

lemma cindex-poly-ubd-eventually:
  shows  $\forall_F r$  in at-top. cindexE (-r) r ( $\lambda x$ . poly q x / poly p x) = of-int (cindex-poly-ubd
q p)
proof -
  define f where f=( $\lambda x$ . poly q x / poly p x)
  obtain R where R-def:  $R > 0$  roots p  $\subseteq \{-R <..<R\}$ 
  if  $p \neq 0$ 
  proof (cases p=0)
    case True
      then show ?thesis using that[of 1] by auto
    next
      case False
        then have finite (roots p) by auto
        from finite-ball-include[OF this,of 0]
        obtain r where  $r > 0$  and r-ball:roots p  $\subseteq$  ball 0 r
          by auto
        have roots p  $\subseteq \{-r <..<r\}$ 
        proof
          fix x assume x  $\in$  roots p
          then have x $\in$ ball 0 r using r-ball by auto
          then have abs x<r using mem-ball-0 by auto
          then show x  $\in \{-r <..<r\}$  using ‹r>0› by auto
        qed
        then show ?thesis using that[of r] False ‹r>0› by auto
      qed
  define l where l=(if p=0 then 0 else cindex-poly (-R) R q p)
  define P where P=( $\lambda l$ . ( $\forall_F r$  in at-top. cindexE (-r) r f = of-int l))
  have P l
  proof (cases p=0 )
    case True
      then show ?thesis
        unfolding P-def f-def l-def using True
        by (auto intro!: eventuallyI cindexE-constI)
    next
      case False
        have P l unfolding P-def
        proof (rule eventually-at-top-linorderI[of R])
          fix r assume  $R \leq r$ 

```

```

then have cindexE (- r) r f = cindex-polyE (-r) r q p
unfolding f-def using R-def[OF ⟨p≠0⟩] by (auto intro: cindexE-eq-cindex-polyE)
also have ... = of-int (cindex-poly (- r) r q p)
proof -
  have jumpF-polyR q p (- r) = 0
  apply (rule jumpF-poly-noroot)
  using ⟨R≤r⟩ R-def[OF ⟨p≠0⟩] by auto
  moreover have jumpF-polyL q p r = 0
  apply (rule jumpF-poly-noroot)
  using ⟨R≤r⟩ R-def[OF ⟨p≠0⟩] by auto
  ultimately show ?thesis unfolding cindex-polyE-def by auto
qed
also have ... = of-int (cindex-poly (- R) R q p)
proof -
  define rs where rs={x. poly p x = 0 ∧ - r < x ∧ x < r}
  define Rs where Rs={x. poly p x = 0 ∧ - R < x ∧ x < R}
  have rs=Rs
  using R-def[OF ⟨p≠0⟩] ⟨R≤r⟩ unfolding rs-def Rs-def by force
  then show ?thesis
  unfolding cindex-poly-def by (fold rs-def Rs-def,auto)
qed
also have ... = of-int l unfolding l-def using False by auto
finally show cindexE (- r) r f = real-of-int l .
qed
then show ?thesis unfolding P-def by auto
qed
moreover have x=l when P x for x
proof -
  have ∀F r in at-top. cindexE (- r) r f = real-of-int x
  ∀F r in at-top. cindexE (- r) r f = real-of-int l
  using ⟨P x⟩ ⟨P l⟩ unfolding P-def by auto
  from eventually-conj[OF this]
  have ∀F r::real in at-top. real-of-int x = real-of-int l
  by (elim eventually-mono,auto)
  then have real-of-int x = real-of-int l by auto
  then show ?thesis by simp
qed
ultimately have P (THE x. P x) using theI[of P l] by blast
then show ?thesis unfolding P-def f-def cindex-poly-ubd-def by auto
qed

lemma cindex-poly-ubd-0:
  assumes p=0 ∨ q=0
  shows cindex-poly-ubd q p = 0
proof -
  have ∀F r in at-top. cindexE (-r) r (λx. poly q x/poly p x) = 0
  apply (rule eventuallyI)
  using assms by (auto intro:cindexE-constI)
  from eventually-conj[OF this cindex-poly-ubd-eventually[of q p]]

```

```

have  $\forall_F r::\text{real in at-top. (cindex-poly-ubd } q \ p) = (0::\text{int})$ 
  apply (elim eventually-mono)
  by auto
then show ?thesis by auto
qed

lemma cindex-poly-ubd-code:
  shows cindex-poly-ubd  $q \ p = \text{changes-R-smods } p \ q$ 
proof (cases  $p=0$ )
  case True
  then show ?thesis using cindex-poly-ubd-0 by auto
next
  case False
  define ps where  $ps \equiv \text{smods } p \ q$ 
  have  $p \in \text{set } ps$  using ps-def  $\langle p \neq 0 \rangle$  by auto
  obtain lb where  $lb: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb$ 
    and lb-sgn:  $\forall x \leq lb. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$ 
    and  $lb < 0$ 
  using root-list-lb[OF no-0-in-smods, of  $p \ q$ , folded ps-def]
  by auto
  obtain ub where  $ub: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x < ub$ 
    and ub-sgn:  $\forall x \geq ub. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-pos-inf } p$ 
    and  $ub > 0$ 
  using root-list-ub[OF no-0-in-smods, of  $p \ q$ , folded ps-def]
  by auto
  define f where  $f = (\lambda t. \text{poly } q \ t / \text{poly } p \ t)$ 
  define P where  $P = (\lambda l. (\forall_F r \text{ in at-top. cindexE } (-r) \ r \ f = \text{of-int } l))$ 
  have P (changes-R-smods  $p \ q$ ) unfolding P-def
  proof (rule eventually-at-top-linorderI[of  $\max |lb| \ |ub| + 1$ ])
    fix r assume  $r\text{-asm}: r \geq \max |lb| \ |ub| + 1$ 
    have cindexE  $(-r) \ r \ f = \text{cindex-polyE } (-r) \ r \ q \ p$ 
      unfolding f-def using r-asm by (auto intro: cindexE-eq-cindex-polyE)
    also have ... = of-int (cindex-poly  $(-r) \ r \ q \ p$ )
    proof -
      have jumpF-polyR  $q \ p \ (-r) = 0$ 
        apply (rule jumpF-poly-noroot)
        using r-asm lb[rule-format, OF  $\langle p \in \text{set } ps \rangle$ , of  $-r$ ] by linarith
      moreover have jumpF-polyL  $q \ p \ r = 0$ 
        apply (rule jumpF-poly-noroot)
        using r-asm ub[rule-format, OF  $\langle p \in \text{set } ps \rangle$ , of  $r$ ] by linarith
      ultimately show ?thesis unfolding cindex-polyE-def by auto
    qed
  also have ... = of-int (changes-itv-smods  $(-r) \ r \ p \ q$ )
    apply (rule cindex-poly-changes-itv-mods[THEN arg-cong])
    using r-asm lb[rule-format, OF  $\langle p \in \text{set } ps \rangle$ , of  $-r$ ] ub[rule-format, OF  $\langle p \in \text{set } ps \rangle$ , of  $r$ ]
    by linarith+
  also have ... = of-int (changes-R-smods  $p \ q$ )
  proof -

```

have $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p (-r))) \text{ ps} = \text{map } \text{sgn-neg-inf } \text{ps}$
and $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p r)) \text{ ps} = \text{map } \text{sgn-pos-inf } \text{ps}$
using $\text{lb-sgn}[\text{THEN } \text{spec, of } -r, \text{simplified}] \text{ ub-sgn}[\text{THEN } \text{spec, of } r, \text{simplified}]$
r-asm
by *auto*
hence $\text{changes-poly-at } \text{ps } (-r) = \text{changes-poly-neg-inf } \text{ps}$
 $\wedge \text{changes-poly-at } \text{ps } r = \text{changes-poly-pos-inf } \text{ps}$
unfolding $\text{changes-poly-neg-inf-def } \text{changes-poly-at-def } \text{changes-poly-pos-inf-def}$
by $(\text{subst } (1 \ 3) \ \text{changes-map-sgn-eq,metis } \text{map-map})$
thus *?thesis* **unfolding** $\text{changes-R-smods-def } \text{changes-itv-smods-def } \text{ps-def}$
by *metis*
qed
finally show $\text{cindexE } (-r) \text{ r f} = \text{of-int } (\text{changes-R-smods } p \ q) .$
qed
moreover have $x = \text{changes-R-smods } p \ q$ **when** $P \ x$ **for** x
proof –
have $\forall_F \ r \text{ in } \text{at-top. } \text{cindexE } (-r) \text{ r f} = \text{real-of-int } (\text{changes-R-smods } p \ q)$
 $\forall_F \ r \text{ in } \text{at-top. } \text{cindexE } (-r) \text{ r f} = \text{real-of-int } x$
using $\langle P \ (\text{changes-R-smods } p \ q) \rangle \langle P \ x \rangle$ **unfolding** $P\text{-def}$ **by** *auto*
from *eventually-conj[OF this]*
have $\forall_F \ (r::\text{real}) \text{ in } \text{at-top. } \text{of-int } x = \text{of-int } (\text{changes-R-smods } p \ q)$
by $(\text{elim } \text{eventually-mono, auto})$
then have $\text{of-int } x = \text{of-int } (\text{changes-R-smods } p \ q)$
using *eventually-const-iff* **by** *auto*
then show *?thesis* **using** *of-int-eq-iff* **by** *blast*
qed
ultimately have $(\text{THE } x. P \ x) = \text{changes-R-smods } p \ q$
using *the-equality*[*of P changes-R-smods p q*] **by** *blast*
then show *?thesis* **unfolding** $\text{cindex-poly-ubd-def } P\text{-def } f\text{-def}$ **by** *auto*
qed

lemma *cindexE-ubd-poly*: $\text{cindexE-ubd } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) = \text{cindex-poly-ubd } q$
 p
proof $(\text{cases } p=0)$
case *True*
then show *?thesis* **using** *cindex-poly-ubd-0* **unfolding** *cindexE-ubd-def*
by *auto*
next
case *False*
define $mx \ mn$ **where** $mx = \text{Max } \{x. \text{poly } p \ x = 0\}$ **and** $mn = \text{Min } \{x. \text{poly } p \ x = 0\}$
define rr **where** $rr = 1 + (\text{max } |mx| \ |mn|)$
have $rr: -rr < x \wedge x < rr$ **when** $\text{poly } p \ x = 0$ **for** x
proof –
have $\text{finite } \{x. \text{poly } p \ x = 0\}$ **using** $\langle p \neq 0 \rangle$ *poly-roots-finite* **by** *blast*
then have $mn \leq x \leq mx$
using *Max-ge Min-le* **that** **unfolding** $mn\text{-def } mx\text{-def}$ **by** *simp-all*
then show *?thesis* **unfolding** $rr\text{-def}$ **by** *auto*

```

qed
define f where f=(λx. poly q x / poly p x)
have ∀F r in at-top. cindexE (- r) r f = cindexE-ubd f
proof (rule eventually-at-top-linorderI[of rr])
  fix r assume r ≥ rr
  define R1 R2 where R1={x. jumpF f (at-right x) ≠ 0 ∧ - r ≤ x ∧ x < r}
    and R2 = {x. jumpF f (at-right x) ≠ 0}
  define L1 L2 where L1={x. jumpF f (at-left x) ≠ 0 ∧ - r < x ∧ x ≤ r}
    and L2={x. jumpF f (at-left x) ≠ 0}
  have R1=R2
  proof -
    have jumpF f (at-right x) = 0 when ¬ (- r ≤ x ∧ x < r) for x
    proof -
      have jumpF f (at-right x) = jumpF-polyR q p x
      unfolding f-def jumpF-polyR-def by simp
      also have ... = 0
      apply (rule jumpF-poly-noroot)
      using that ⟨r ≥ rr⟩ by (auto dest:rr)
      finally show ?thesis .
    qed
    then show ?thesis unfolding R1-def R2-def by blast
  qed
  moreover have L1=L2
  proof -
    have jumpF f (at-left x) = 0 when ¬ (- r < x ∧ x ≤ r) for x
    proof -
      have jumpF f (at-left x) = jumpF-polyL q p x
      unfolding f-def jumpF-polyL-def by simp
      also have ... = 0
      apply (rule jumpF-poly-noroot)
      using that ⟨r ≥ rr⟩ by (auto dest:rr)
      finally show ?thesis .
    qed
    then show ?thesis unfolding L1-def L2-def by blast
  qed
  ultimately show cindexE (- r) r f = cindexE-ubd f
  unfolding cindexE-def cindexE-ubd-def
  apply (fold R1-def R2-def L1-def L2-def)
  by auto
qed
moreover have ∀F r in at-top. cindexE (- r) r f = cindex-poly-ubd q p
  using cindex-poly-ubd-eventually unfolding f-def by auto
ultimately have ∀F r in at-top. cindexE (- r) r f = cindexE-ubd f
  ∧ cindexE (- r) r f = cindex-poly-ubd q p
  using eventually-conj by auto
then have ∀F (r::real) in at-top. cindexE-ubd f = cindex-poly-ubd q p
  by (elim eventually-mono) auto
then show ?thesis unfolding f-def by auto
qed

```

lemma *cindex-polyE-noroot*:
assumes $a < b \ \forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0$
shows $\text{cindex-polyE } a \ b \ q \ p = 0$
proof –
have $\text{jumpF-polyR } q \ p \ a = 0$
apply (*rule jumpF-poly-noroot*)
using *assms* **by** *auto*
moreover **have** $\text{jumpF-polyL } q \ p \ b = 0$
apply (*rule jumpF-poly-noroot*)
using *assms* **by** *auto*
moreover **have** $\text{cindex-poly } a \ b \ q \ p = 0$
apply (*rule cindex-poly-noroot*)
using *assms* **by** *auto*
ultimately show *?thesis* **unfolding** *cindex-polyE-def* **by** *auto*
qed

lemma *cindex-polyE-combine*:
assumes $a < b \ b < c$
shows $\text{cindex-polyE } a \ b \ q \ p + \text{cindex-polyE } b \ c \ q \ p = \text{cindex-polyE } a \ c \ q \ p$
proof –
define $A \ B$ **where** $A = \text{cindex-poly } a \ b \ q \ p - \text{jumpF-polyL } q \ p \ b$
and $B = \text{jumpF-polyR } q \ p \ b + \text{cindex-poly } b \ c \ q \ p$
have $\text{cindex-polyE } a \ b \ q \ p + \text{cindex-polyE } b \ c \ q \ p =$
 $\text{jumpF-polyR } q \ p \ a + (A + B) - \text{jumpF-polyL } q \ p \ c$
unfolding *cindex-polyE-def* *A-def* *B-def* **by** *auto*
also have $\dots = \text{jumpF-polyR } q \ p \ a + \text{cindex-poly } a \ c \ q \ p - \text{jumpF-polyL } q \ p \ c$
proof –
have $A + B = \text{cindex-poly } a \ b \ q \ p + (\text{jumpF-polyR } q \ p \ b - \text{jumpF-polyL } q \ p \ b)$
 $+ \text{cindex-poly } b \ c \ q \ p$
unfolding *A-def* *B-def* **by** *auto*
also have $\dots = \text{cindex-poly } a \ b \ q \ p + \text{real-of-int } (\text{jump-poly } q \ p \ b) + \text{cindex-poly}$
 $b \ c \ q \ p$
using *jump-poly-jumpF-poly* **by** *auto*
also have $\dots = \text{cindex-poly } a \ c \ q \ p$
using *assms*
apply (*subst* (3) *cindex-poly-combine[symmetric, of - b]*)
by *auto*
finally show *?thesis* **by** *auto*
qed
also have $\dots = \text{cindex-polyE } a \ c \ q \ p$
unfolding *cindex-polyE-def* **by** *simp*
finally show *?thesis* .
qed

lemma *cindex-polyE-linear-comp*:
fixes $b \ c :: \text{real}$
defines $h \equiv (\lambda p. \text{pcompose } p \ [: b, c :])$
assumes $lb < ub \ c \neq 0$

shows $cindex\text{-}polyE\ lb\ ub\ (h\ q)\ (h\ p) =$
 $(if\ 0 < c\ then\ cindex\text{-}polyE\ (c * lb + b)\ (c * ub + b)\ q\ p$
 $else - cindex\text{-}polyE\ (c * ub + b)\ (c * lb + b)\ q\ p)$

proof –
have $cindex\text{-}polyE\ lb\ ub\ (h\ q)\ (h\ p) = cindexE\ lb\ ub\ (\lambda x. poly\ (h\ q)\ x / poly\ (h\ p)\ x)$
apply ($subst\ cindexE\text{-}eq\text{-}cindex\text{-}polyE[symmetric, OF\ \langle lb < ub \rangle]$)
by *simp*
also have $... = cindexE\ lb\ ub\ ((\lambda x. poly\ q\ x / poly\ p\ x) \circ (\lambda x. c * x + b))$
unfolding *comp-def h-def poly-pcompose* **by** (*simp add: algebra-simps*)
also have $... = (if\ 0 < c\ then\ cindexE\ (c * lb + b)\ (c * ub + b)\ (\lambda x. poly\ q\ x / poly\ p\ x)$
 $else - cindexE\ (c * ub + b)\ (c * lb + b)\ (\lambda x. poly\ q\ x / poly\ p\ x))$
apply ($subst\ cindexE\text{-}linear\text{-}comp[OF\ \langle c \neq 0 \rangle]$)
by *simp*
also have $... = (if\ 0 < c\ then\ cindex\text{-}polyE\ (c * lb + b)\ (c * ub + b)\ q\ p$
 $else - cindex\text{-}polyE\ (c * ub + b)\ (c * lb + b)\ q\ p)$

proof –
have $cindexE\ (c * lb + b)\ (c * ub + b)\ (\lambda x. poly\ q\ x / poly\ p\ x)$
 $= cindex\text{-}polyE\ (c * lb + b)\ (c * ub + b)\ q\ p\ \mathbf{if}\ c > 0$
apply ($subst\ cindexE\text{-}eq\text{-}cindex\text{-}polyE$)
using *that* $\langle lb < ub \rangle$ **by** *auto*
moreover have $cindexE\ (c * ub + b)\ (c * lb + b)\ (\lambda x. poly\ q\ x / poly\ p\ x)$
 $= cindex\text{-}polyE\ (c * ub + b)\ (c * lb + b)\ q\ p\ \mathbf{if}\ \neg c > 0$
apply ($subst\ cindexE\text{-}eq\text{-}cindex\text{-}polyE$)
using *that* *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed
finally show *?thesis* .
qed

lemma *cindex-polyE-product'*:
fixes $p\ r\ q\ s :: real$ **and** $a\ b :: real$
assumes $a < b$ *coprime* $q\ p$ *coprime* $s\ r$
shows $cindex\text{-}polyE\ a\ b\ (p * r - q * s)\ (p * s + q * r)$
 $= cindex\text{-}polyE\ a\ b\ p\ q + cindex\text{-}polyE\ a\ b\ r\ s$
 $- cross\text{-}alt\ (p * s + q * r)\ (q * s)\ a\ b / 2\ (\mathbf{is}\ ?L = ?R)$

proof ($cases\ q=0 \vee s=0 \vee p=0 \vee r=0 \vee p * s + q * r = 0$)
case *True*
moreover have *?thesis* **if** $q=0$
proof –
have $p \neq 0$
using *assms(2)* *coprime-poly-0 poly-0 that* **by** *blast*
then show *?thesis* **using** *that cindex-polyE-mult-cancel* **by** *simp*
qed
moreover have *?thesis* **if** $s=0$
proof –
have $r \neq 0$ **using** *assms(3)* *coprime-poly-0 poly-0 that* **by** *blast*
then have $?L = cindex\text{-}polyE\ a\ b\ (r * p)\ (r * q)$

using that by (simp add: algebra-simps)
 also have ... = ?R
 using that cindex-polyE-mult-cancel ⟨r≠0⟩ by simp
 finally show ?thesis .
 qed
 moreover have ?thesis if $p * s + q * r = 0$ $s \neq 0$ $q \neq 0$
 proof -
 have cindex-polyE a b p q = cindex-polyE a b (s*p) (s*q)
 using cindex-polyE-mult-cancel[OF ⟨s≠0⟩] by simp
 also have ... = cindex-polyE a b (-(q * r)) (q * s)
 using that(1)
 by (metis add.inverse-inverse add.inverse-unique mult.commute)
 also have ... = - cindex-polyE a b (q * r) (q * s)
 using cindex-polyE-smult-1[where c=-1,simplified] by simp
 also have ... = - cindex-polyE a b r s
 using cindex-polyE-mult-cancel[OF ⟨q≠0⟩] by simp
 finally have cindex-polyE a b p q = - cindex-polyE a b r s .
 then show ?thesis using that(1) by simp
 qed
 moreover have ?thesis if $p=0$
 proof -
 have poly q a≠0
 using assms(2) coprime-poly-0 order-root that(1) by blast
 have poly q b≠0
 by (metis assms(2) coprime-poly-0 mpoly-base-conv(1) that)
 then have q≠0 using poly-0 by blast

 have ?L = - cindex-polyE a b s r
 using that cindex-polyE-smult-1[where c=-1,simplified]
 cindex-polyE-mult-cancel[OF ⟨q≠0⟩]
 by simp
 also have ... = cindex-polyE a b r s - (cross-alt r s a b) / 2
 apply (subst cindex-polyE-inverse-add-cross[symmetric])
 using ⟨a<b⟩ ⟨coprime s r⟩ by (auto simp: coprime-commute)
 also have ... = ?R
 using ⟨p=0⟩ ⟨poly q a≠0⟩ ⟨poly q b≠0⟩ cross-alt-cancel
 by simp
 finally show ?thesis .
 qed
 moreover have ?thesis if $r=0$
 proof -
 have poly s a≠0
 using assms(3) coprime-poly-0 order-root that by blast
 have poly s b≠0
 using assms(3) coprime-poly-0 order-root that by blast
 then have s≠0 using poly-0 by blast

 have cindex-polyE a b (-(q * s)) (p * s)
 = - cindex-polyE a b (q * s) (p * s)

```

    using cindex-polyE-smult-1 [where c=-1, simplified] by auto
  also have ... = - cindex-polyE a b (s * q) (s * p)
    by (simp add: algebra-simps)
  also have ... = - cindex-polyE a b q p
    using cindex-polyE-mult-cancel[OF ‹s≠0›] by simp
  finally have cindex-polyE a b (- (q * s)) (p * s)
    = - cindex-polyE a b q p .
  moreover have cross-alt (p * s) (q * s) a b / 2
    = cindex-polyE a b q p + cindex-polyE a b p q
  proof -
    have cross-alt (p * s) (q * s) a b
      = cross-alt (s * p) (s * q) a b
    by (simp add: algebra-simps)
    also have ... = cross-alt p q a b
      using cross-alt-cancel by (simp add: ‹poly s a ≠ 0› ‹poly s b ≠ 0›)
    also have ... / 2 = cindex-polyE a b q p + cindex-polyE a b p q
      apply (subst cindex-polyE-inverse-add-cross[symmetric])
      using ‹a < b› ‹coprime q p› coprime-commute by auto
    finally show ?thesis .
  qed
  ultimately show ?thesis using that by simp
qed
ultimately show ?thesis by argo
next
case False
define P where P = (p * s + q * r)
define Q where Q = q * s * P

from False have q≠0 s≠0 p≠0 r≠0 P ≠ 0 Q≠0
  unfolding P-def Q-def by auto
then have finite: finite (proots-within Q {x. a ≤ x ∧ x ≤ b})
  unfolding P-def Q-def
  by (auto intro: finite-proots)

have sign-pos-eq:
  sign-r-pos Q a = (poly Q b > 0)
  poly Q a ≠ 0 ⟹ poly Q a > 0 = (poly Q b > 0)
  if a < b and noroot: ∀ x. a < x ∧ x ≤ b ⟹ poly Q x ≠ 0 for a b Q
proof -
  have sign-r-pos Q a = (sgnx (poly Q) (at-right a) > 0)
    unfolding sign-r-pos-sgnx-iff by simp
  also have ... = (sgnx (poly Q) (at-left b) > 0)
  proof (rule ccontr)
    assume (0 < sgnx (poly Q) (at-right a))
      ≠ (0 < sgnx (poly Q) (at-left b))
    then have ∃ x > a. x < b ∧ poly Q x = 0
      using sgnx-at-left-at-right-IVT[OF - ‹a < b›] by auto
    then show False using that(2) by auto
  qed
qed

```

also have ... = (poly Q b > 0)
apply (subst sgnx-poly-nz)
using that **by** auto
finally show sign-r-pos Q a = (poly Q b > 0) .
show (poly Q a > 0) = (poly Q b > 0) **if** poly Q a ≠ 0
proof (rule ccontr)
assume (0 < poly Q a) ≠ (0 < poly Q b)
then have poly Q a * poly Q b < 0
by (metis ‹sign-r-pos Q a = (0 < poly Q b)› poly-0 sign-r-pos-rec that)
from poly-IVT[OF ‹a < b› this]
have ∃ x > a. x < b ∧ poly Q x = 0 .
then show False **using** noroot **by** auto
qed
qed

define Case **where** Case = (λ a b. cindex-polyE a b (p * r - q * s) P
= cindex-polyE a b p q + cindex-polyE a b r s -
- (cross-alt P (q * s) a b) / 2)

have basic-case: Case a b

if noroot0:proots-within Q {x. a < x ∧ x < b} = {}

and noroot-disj: poly Q a ≠ 0 ∨ poly Q b ≠ 0

and a < b

for a b

proof -

let ?thesis' = λ p r q s a. cindex-polyE a b (p * r - q * s) (p * s + q * r) =
cindex-polyE a b p q + cindex-polyE a b r s -
(cross-alt (p * s + q * r) (q * s) a b) / 2

have base-case: ?thesis' p r q s a

if proots-within (q * s * (p * s + q * r)) {x. a < x ∧ x ≤ b} = {}

and coprime q p coprime s r

q ≠ 0 s ≠ 0 p ≠ 0 r ≠ 0 p * s + q * r ≠ 0

a < b

for p r q s a

proof -

define P **where** P = (p * s + q * r)

have noroot1:proots-within (q * s * P) {x. a < x ∧ x ≤ b} = {}

using that(1) **unfolding** P-def .

have P ≠ 0 **using** ‹p * s + q * r ≠ 0› **unfolding** P-def **by** simp

have cind1: cindex-polyE a b (p * r - q * s) P

= (if poly P a = 0 then jumpF-polyR (p * r - q * s) P a else 0)

proof -

have cindex-poly a b (p * r - q * s) P = 0

apply (rule cindex-poly-noroot)

using noroot1 **by** fastforce

moreover have jumpF-polyL (p * r - q * s) P b = 0

apply (rule jumpF-poly-noroot)

using noroot1 ‹a < b› **by** auto

```

ultimately show ?thesis
  unfolding cindex-polyE-def by (simp add: jumpF-poly-noroot(2))
qed
have cind2:cindex-polyE a b p q
  = (if poly q a = 0 then jumpF-polyR p q a else 0)
proof -
  have cindex-poly a b p q = 0
    apply (rule cindex-poly-noroot)
    using noroot1 ⟨a<b⟩ by auto fastforce
  moreover have jumpF-polyL p q b = 0
    apply (rule jumpF-poly-noroot)
    using noroot1 ⟨a<b⟩ by auto
  ultimately show ?thesis
    unfolding cindex-polyE-def
    by (simp add: jumpF-poly-noroot(2))
qed
have cind3:cindex-polyE a b r s
  = (if poly s a = 0 then jumpF-polyR r s a else 0)
proof -
  have cindex-poly a b r s = 0
    apply (rule cindex-poly-noroot)
    using noroot1 ⟨a<b⟩ by auto fastforce
  moreover have jumpF-polyL r s b = 0
    apply (rule jumpF-poly-noroot)
    using noroot1 ⟨a<b⟩ by auto
  ultimately show ?thesis
    unfolding cindex-polyE-def
    by (simp add: jumpF-poly-noroot(2))
qed

have ?thesis if poly (q * s * P) a ≠ 0
proof -
  have noroot2:proots-within (q * s * P) {x. a ≤ x ∧ x ≤ b} = {}
    using that noroot1 by force
  have cindex-polyE a b (p * r - q * s) P = 0
    apply (rule cindex-polyE-noroot)
    using noroot2 ⟨a < b⟩ by auto
  moreover have cindex-polyE a b p q = 0
    apply (rule cindex-polyE-noroot)
    using noroot2 ⟨a < b⟩ by auto
  moreover have cindex-polyE a b r s = 0
    apply (rule cindex-polyE-noroot)
    using noroot2 ⟨a < b⟩ by auto
  moreover have cross-alt P (q * s) a b = 0
    apply (rule cross-alt-noroot[OF ⟨a<b⟩])
    using noroot2 by auto
  ultimately show ?thesis unfolding P-def by auto
qed
moreover have ?thesis if poly (q * s * P) a = 0

```

```

proof –
  have ?thesis if poly q a = 0 poly s a ≠ 0
proof –
  have poly P a ≠ 0
    using that coprime-poly-0[OF ‹coprime q p›] unfolding P-def
    by simp
  then have cindex-polyE a b (p * r - q * s) P = 0
    using cind1 by auto
  moreover have cindex-polyE a b p q = (cross-alt P (q * s) a b) / 2
proof –
  have cindex-polyE a b p q = jumpF-polyR p q a
    using cind2 that(1) by auto
  also have ... = (cross-alt 1 (q * s * P) a b) / 2
proof –
  have sign-eq:(sign-r-pos q a ‹⟷ poly p a > 0›)
    = (poly (q * s * P) b > 0)
proof –
  have (sign-r-pos q a ‹⟷ poly p a > 0›)
    = (sgnx (poly (q * p)) (at-right a) > 0)
proof –
  have (poly p a > 0) = (sgnx (poly p) (at-right a) > 0)
    apply (subst sgnx-poly-nz)
    using ‹coprime q p› coprime-poly-0 that(1) by auto
  then show ?thesis
    unfolding sign-r-pos-sgnx-iff
    apply (subst sgnx-poly-times[of - a])
    subgoal by simp
    using poly-sgnx-values ‹p ≠ 0› ‹q ≠ 0›
    by (metis (no-types, opaque-lifting) add.inverse-inverse
      mult.right-neutral mult.minus-right zero-less-one)
qed
  also have ... = (sgnx (poly ((q * p) * s2)) (at-right a) > 0)
proof (subst (2) sgnx-poly-times)
  have sgnx (poly (s2)) (at-right a) > 0
    using sgn-zero-iff sgnx-poly-nz(2) that(2) by auto
  then show (0 < sgnx (poly (q * p)) (at-right a)) =
    (0 < sgnx (poly (q * p)) (at-right a)
      * sgnx (poly (s2)) (at-right a))
    by (simp add: zero-less-mult-iff)
qed auto
  also have ... = (sgnx (poly (q * s)) (at-right a)
    * sgnx (poly (p * s)) (at-right a) > 0)
    unfolding power2-eq-square
    apply (subst sgnx-poly-times[where x=a],simp)+
    by (simp add: algebra-simps)
  also have ... = (sgnx (poly (q * s)) (at-right a)
    * sgnx (poly P) (at-right a) > 0)
proof –
  have sgnx (poly P) (at-right a) =

```

```

      sgnx (poly (q * r + p * s)) (at-right a)
    unfolding P-def by (simp add:algebra-simps)
  also have ... = sgnx (poly (p * s)) (at-right a)
    apply (rule sgnx-poly-plus[where x=a])
    subgoal using ⟨poly q a=0⟩ by simp
    subgoal using ⟨coprime q p⟩ coprime-poly-0 poly-mult-zero-iff
      that(1) that(2) by blast
    by simp
  finally show ?thesis by auto
qed
also have ... = (0 < sgnx (poly (q * s * P))) (at-right a)
  apply (subst sgnx-poly-times[where x=a],simp) +
  by (simp add:algebra-simps)
also have ... = (0 < sgnx (poly (q * s * P))) (at-left b)
proof –
  have sgnx (poly (q * s * P)) (at-right a)
    = sgnx (poly (q * s * P)) (at-left b)
  proof (rule ccontr)
    assume sgnx (poly (q * s * P)) (at-right a)
      ≠ sgnx (poly (q * s * P)) (at-left b)
    from sgnx-at-left-at-right-IVT[OF this ⟨a<b⟩]
    have ∃ x>a. x < b ∧ poly (q * s * P) x = 0 .
    then show False using noroot1 by fastforce
  qed
  then show ?thesis by auto
qed
also have ... = (poly (q * s * P) b > 0)
  apply (subst sgnx-poly-nz)
  using noroot1 ⟨a<b⟩ by auto
  finally show ?thesis .
qed
have psign-a:psign-diff 1 (q * s * P) a = 1
  unfolding psign-diff-def using ⟨poly (q * s * P) a=0⟩
  by simp

have poly (q * s * P) b ≠ 0
  using noroot1 ⟨a<b⟩ by blast
moreover have ?thesis if poly (q * s * P) b > 0
proof –
  have psign-diff 1 (q * s * P) b = 0
    using that unfolding psign-diff-def by auto
  moreover have jumpF-polyR p q a = 1/2
    unfolding jumpF-polyR-coprime[OF ⟨coprime q p⟩]
    using ⟨p ≠ 0⟩ ⟨poly q a = 0⟩ ⟨q ≠ 0⟩ sign-eq that by presburger
  ultimately show ?thesis
    unfolding cross-alt-def using psign-a by auto
qed
moreover have ?thesis if poly (q * s * P) b < 0
proof –

```

```

have psign-diff 1 (q * s * P) b = 2
  using that unfolding psign-diff-def by auto
moreover have jumpF-polyR p q a = - 1/2
  unfolding jumpF-polyR-coprime[OF <coprime q p>]
  using <p ≠ 0> <poly q a = 0> <q ≠ 0> sign-eq that by auto
ultimately show ?thesis
  unfolding cross-alt-def using psign-a by auto
qed
ultimately show ?thesis by argo
qed
also have ... = (cross-alt P (q * s) a b) / 2
  apply (subst cross-alt-clear[symmetric])
  using <poly P a ≠ 0> noroot1 <a < b> cross-alt-poly-commute
  by auto
finally show ?thesis .
qed
moreover have cindex-polyE a b r s = 0
  using cind3 that by auto
ultimately show ?thesis using that
  apply (fold P-def)
  by auto
qed
moreover have ?thesis if poly q a ≠ 0 poly s a = 0
proof -
  have poly P a ≠ 0
    using that coprime-poly-0[OF <coprime s r>] unfolding P-def
    by simp
  then have cindex-polyE a b (p * r - q * s) P = 0
    using cind1 by auto
  moreover have cindex-polyE a b r s = (cross-alt P (q * s) a b) / 2
proof -
  have cindex-polyE a b r s = jumpF-polyR r s a
    using cind3 that by auto
  also have ... = (cross-alt 1 (s * q * P) a b) / 2
proof -
  have sign-eq:(sign-r-pos s a ↔ poly r a > 0)
    = (poly (s * q * P) b > 0)
proof -
  have (sign-r-pos s a ↔ poly r a > 0)
    = (sgnx (poly (s*r)) (at-right a) > 0)
proof -
  have (poly r a > 0) = (sgnx (poly r) (at-right a) > 0)
  apply (subst sgnx-poly-nz)
  using <coprime s r> coprime-poly-0 that(2) by auto
then show ?thesis
  unfolding sign-r-pos-sgnx-iff
  apply (subst sgnx-poly-times[of - a])
  subgoal by simp
  subgoal using <r ≠ 0> <s ≠ 0>

```

```

    by (metis (no-types, opaque-lifting) add.inverse-inverse
        mult.right-neutral mult.minus-right poly.sgnx-values(2)
        zero-less-one)
  done
qed
also have ... = (sgnx (poly ((s*r) * q^2)) (at-right a) > 0)
proof (subst (2) sgnx-poly-times)
  have sgnx (poly (q^2)) (at-right a) > 0
by (metis ⟨q ≠ 0⟩ power2-eq-square sign-r-pos-mult sign-r-pos-sgnx-iff)
  then show (0 < sgnx (poly (s * r)) (at-right a)) =
    (0 < sgnx (poly (s * r)) (at-right a)
     * sgnx (poly (q^2)) (at-right a))
    by (simp add: zero-less-mult-iff)
qed auto
also have ... = (sgnx (poly (s * q)) (at-right a)
  * sgnx (poly (r * q)) (at-right a) > 0)
  unfolding power2-eq-square
  apply (subst sgnx-poly-times[where x=a],simp)+
  by (simp add:algebra-simps)
also have ... = (sgnx (poly (s * q)) (at-right a)
  * sgnx (poly P) (at-right a) > 0)
proof -
  have sgnx (poly P) (at-right a) =
    sgnx (poly (p * s + q * r)) (at-right a)
    unfolding P-def by (simp add:algebra-simps)
  also have ... = sgnx (poly (q * r)) (at-right a)
    apply (rule sgnx-poly-plus[where x=a])
    subgoal using ⟨poly s a=0⟩ by simp
    subgoal
      using ⟨coprime s r⟩ coprime-poly-0 poly-mult-zero-iff that(1)
      that(2) by blast
    by simp
  finally show ?thesis by (auto simp:algebra-simps)
qed
also have ... = (0 < sgnx (poly (s * q * P)) (at-right a))
  apply (subst sgnx-poly-times[where x=a],simp)+
  by (simp add:algebra-simps)
also have ... = (0 < sgnx (poly (s * q * P)) (at-left b))
proof -
  have sgnx (poly (s * q * P)) (at-right a)
    = sgnx (poly (s * q * P)) (at-left b)
  proof (rule ccontr)
    assume sgnx (poly (s * q * P)) (at-right a)
      ≠ sgnx (poly (s * q * P)) (at-left b)
    from sgnx-at-left-at-right-IVT[OF this ⟨a < b⟩]
    have ∃ x > a. x < b ∧ poly (s * q * P) x = 0 .
    then show False using noroot1 by fastforce
  qed
then show ?thesis by auto

```

```

qed
also have ... = (poly (s * q * P) b > 0)
  apply (subst sgnx-poly-nz)
  using noroot1 ⟨a<b⟩ by auto
finally show ?thesis .
qed
have psign-a:psign-diff 1 (s * q * P) a = 1
  unfolding psign-diff-def using ⟨poly (q * s * P) a=0⟩
  by (simp add:algebra-simps)

have poly (s * q * P) b ≠ 0
  using noroot1 ⟨a<b⟩ by (auto simp:algebra-simps)
moreover have ?thesis if poly (s * q * P) b > 0
proof -
  have psign-diff 1 (s * q * P) b = 0
    using that unfolding psign-diff-def by auto
  moreover have jumpF-polyR r s a = 1/2
    unfolding jumpF-polyR-coprime[OF ⟨coprime s r⟩]
    using ⟨poly s a = 0⟩ ⟨r ≠ 0⟩ ⟨s ≠ 0⟩ sign-eq that by presburger
  ultimately show ?thesis
    unfolding cross-alt-def using psign-a by auto
qed
moreover have ?thesis if poly (s * q * P) b < 0
proof -
  have psign-diff 1 (s * q * P) b = 2
    using that unfolding psign-diff-def by auto
  moreover have jumpF-polyR r s a = - 1/2
    unfolding jumpF-polyR-coprime[OF ⟨coprime s r⟩]
    using ⟨poly s a = 0⟩ ⟨r ≠ 0⟩ sign-eq that by auto
  ultimately show ?thesis
    unfolding cross-alt-def using psign-a by auto
qed
ultimately show ?thesis by argo
qed
also have ... = (cross-alt P (q * s) a b) / 2
  apply (subst cross-alt-clear[symmetric])
  using ⟨poly P a ≠ 0⟩ noroot1 ⟨a<b⟩ cross-alt-poly-commute
  by (auto simp:algebra-simps)
finally show ?thesis .
qed
moreover have cindex-polyE a b p q = 0
  using cind2 that by auto
ultimately show ?thesis using that
  apply (fold P-def)
  by auto
qed
moreover have ?thesis if poly P a = 0 poly q a ≠ 0 poly s a ≠ 0
proof -
  have cindex-polyE a b (p * r - q * s) P

```

```

    = jumpF-polyR (p * r - q * s) P a
  using cind1 that by auto
  also have ... = (if sign-r-pos P a = (0 < poly (p * r - q * s) a)
    then 1 / 2 else - 1 / 2) (is - = ?R)
  proof (subst jumpF-polyR-coprime')
    let ?C=(P ≠ 0 ∧ p * r - q * s ≠ 0 ∧ poly P a = 0)
    have ?C
      by (smt (verit, del-insts) P-def ⟨P ≠ 0⟩ ⟨p ≠ 0⟩ ⟨q ≠ 0⟩ ⟨r ≠ 0⟩ ⟨s
    ≠ 0⟩ eq-iff-diff-eq-0 no-zero-divisors poly-add
        poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec that(1,2,3))
    then show (if ?C then ?R else 0) = ?R by auto
    show poly P a ≠ 0 ∨ poly (p * r - q * s) a ≠ 0
      by (smt (verit, ccfv-threshold) P-def ⟨p ≠ 0⟩ ⟨q ≠ 0⟩ ⟨r ≠ 0⟩ ⟨s ≠ 0⟩
    no-zero-divisors poly-add poly-diff
        poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec that(2,3))
  qed
  also have ... = - cross-alt P (q * s) a b / 2
  proof -
    have (sign-r-pos P a = (0 < poly (p * r - q * s) a))
      =(¬ (poly (q * s * P) b > 0))
    proof -
      have (poly (q * s * P) b > 0)
        =(sgnx (poly (q * s * P)) (at-left b) > 0)
      apply (subst sgnx-poly-nz)
      using noroot1 ⟨a < b⟩ by auto
      also have ... = (sgnx (poly (q * s * P)) (at-right a) > 0)
      proof (rule ccontr)
        define F where F=(q * s * P)
        assume (0 < sgnx (poly F) (at-left b))
          ≠ (0 < sgnx (poly F) (at-right a))
        then have sgnx (poly F) (at-right a) ≠ sgnx (poly F) (at-left b)
          by auto
        then have ∃ x > a. x < b ∧ poly F x = 0
          using sgnx-at-left-at-right-IVT[OF - ⟨a < b⟩] by auto
        then show False using noroot1 [folded F-def] ⟨a < b⟩ by fastforce
      qed
    also have ... = sign-r-pos (q * s * P) a
      using sign-r-pos-sgnx-iff by simp
    also have ... = (sign-r-pos P a = sign-r-pos (q * s) a)
      apply (subst sign-r-pos-mult[symmetric])
      using ⟨P ≠ 0⟩ ⟨q ≠ 0⟩ ⟨s ≠ 0⟩ by (auto simp add: algebra-simps)
    also have ... = (sign-r-pos P a = (0 ≥ poly (p * r - q * s) a))
    proof -
      have sign-r-pos (q * s) a=(poly (q * s) a > 0)
        by (metis poly-0 poly-mult-zero-iff sign-r-pos-rec
          that(2) that(3))
      also have ... = (0 ≥ poly (p * r - q * s) a)
        using ⟨poly P a = 0⟩ unfolding P-def
          by (smt (verit, ccfv-threshold) ⟨p ≠ 0⟩ ⟨q ≠ 0⟩ ⟨r ≠ 0⟩ ⟨s ≠ 0⟩)

```

divisors-zero

poly-add poly-diff poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec

that(2)

that(3))

finally show *?thesis* **by** *simp*

qed

finally have $(0 < \text{poly } (q * s * P) b)$

$= (\text{sign-r-pos } P a = (\text{poly } (p * r - q * s) a \leq 0))$.

then show *?thesis* **by** *argo*

qed

moreover have *cross-alt* $P (q * s) a b =$

$(\text{if } \text{poly } (q * s * P) b > 0 \text{ then } 1 \text{ else } -1)$

proof –

have *psign-diff* $P (q * s) a = 1$

by (*smt* (*verit*, *ccfv-threshold*) *Sturm-Tarski.sign-def*

dvd-div-mult-self gcd-dvd1 gcd-dvd2 poly-mult-zero-iff

psign-diff-def that(1) that(2) that(3))

moreover have *psign-diff* $P (q * s) b$

$= (\text{if } \text{poly } (q * s * P) b > 0 \text{ then } 0 \text{ else } 2)$

proof –

define F **where** $F = q * s * P$

have *psign-diff* $P (q * s) b = \text{psign-diff } 1 F b$

apply (*subst* *psign-diff-clear*)

using *noroot1* $\langle a < b \rangle$ **unfolding** $F\text{-def}$

by (*auto simp: algebra-simps*)

also have ... $= (\text{if } 0 < \text{poly } F b \text{ then } 0 \text{ else } 2)$

proof –

have $\text{poly } F b \neq 0$

unfolding $F\text{-def}$ **using** $\langle a < b \rangle$ *noroot1* **by** *auto*

then show *?thesis*

unfolding *psign-diff-def* **by** *auto*

qed

finally show *?thesis* **unfolding** $F\text{-def}$.

qed

ultimately show *?thesis* **unfolding** *cross-alt-def* **by** *auto*

qed

ultimately show *?thesis* **by** *auto*

qed

finally have *cindex-polyE* $a b (p * r - q * s) P$

$= - \text{cross-alt } P (q * s) a b / 2$.

moreover have *cindex-polyE* $a b p q = 0$

using *cind2* **that** **by** *auto*

moreover have *cindex-polyE* $a b r s = 0$

using *cind3* **that** **by** *auto*

ultimately show *?thesis*

by (*fold* $P\text{-def}$) *auto*

qed

moreover have *?thesis* **if** $\text{poly } q a = 0$ $\text{poly } s a = 0$

proof –

```

have poly p a ≠ 0
  using ⟨coprime q p⟩ coprime-poly-0 that(1) by blast
have poly r a ≠ 0
  using ⟨coprime s r⟩ coprime-poly-0 that(2) by blast
have poly P a = 0
  unfolding P-def using that by simp

define ff where ff = (λx. if x then 1/(2::real) else -1/2)
define C1 C2 C3 C4 C5 where C1 = (sign-r-pos P a)
  and C2 = (0 < poly p a)
  and C3 = (0 < poly r a)
  and C4 = (sign-r-pos q a)
  and C5 = (sign-r-pos s a)
note CC-def = C1-def C2-def C3-def C4-def C5-def

have cindex-polyE a b (p * r - q * s) P = ff ((C1 = C2) = C3)
proof -
  have cindex-polyE a b (p * r - q * s) P
    = jumpF-polyR (p * r - q * s) P a
  using cind1 ⟨poly P a = 0⟩ by auto
  also have ... = (ff (sign-r-pos P a
    = (0 < poly (p * r - q * s) a)))
  unfolding ff-def
  apply (subst jumpF-polyR-coprime')
  subgoal
    by (simp add: ⟨poly p a ≠ 0⟩ ⟨poly r a ≠ 0⟩ that(1))
  subgoal
    by (smt (verit) ⟨P ≠ 0⟩ ⟨poly P a = 0⟩
      ⟨poly P a ≠ 0 ∨ poly (p * r - q * s) a ≠ 0⟩ poly-0)
  done
  also have ... = (ff (sign-r-pos P a = (0 < poly (p * r) a)))
proof -
  have (0 < poly (p * r - q * s) a) = (0 < poly (p * r) a)
    by (simp add: that(1))
  then show ?thesis by simp
qed
  also have ... = ff ((C1 = C2) = C3)
  unfolding CC-def
    by (smt (verit) ⟨p ≠ 0⟩ ⟨poly p a ≠ 0⟩ ⟨poly r a ≠ 0⟩ ⟨r ≠ 0⟩
no-zero-divisors
  poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec)
  finally show ?thesis .
qed
moreover have cindex-polyE a b p q
  = ff (C4 = C2)
proof -
  have cindex-polyE a b p q = jumpF-polyR p q a
  using cind2 ⟨poly q a = 0⟩ by auto
  also have ... = ff (sign-r-pos q a = (0 < poly p a))

```

```

    apply (subst jumpF-polyR-coprime')
    subgoal using ⟨poly p a ≠ 0⟩ by auto
    subgoal using ⟨p ≠ 0⟩ ⟨q ≠ 0⟩ ff-def that(1) by presburger
    done
  also have ... = ff (C4 = C2)
    using ⟨a<b⟩ noroot1 unfolding CC-def by auto
  finally show ?thesis .
qed
moreover have cindex-polyE a b r s = ff (C5 = C3)
proof -
  have cindex-polyE a b r s = jumpF-polyR r s a
    using cind3 ⟨poly s a=0⟩ by auto
  also have ... = ff (sign-r-pos s a = (0 < poly r a))
    apply (subst jumpF-polyR-coprime')
    subgoal using ⟨poly r a ≠ 0⟩ by auto
    subgoal using ⟨r ≠ 0⟩ ⟨s ≠ 0⟩ ff-def that(2) by presburger
    done
  also have ... = ff (C5 = C3)
    using ⟨a<b⟩ noroot1 unfolding CC-def by auto
  finally show ?thesis .
qed
moreover have cross-alt P (q * s) a b = 2 * ff ((C1 = C4) = C5)
proof -
  have cross-alt P (q * s) a b
    = sign (poly P b * (poly q b * poly s b))
    apply (subst cross-alt-clear)
    apply (subst cross-alt-alt)
    using that by auto
  also have ... = 2 * ff ((C1 = C4) = C5)
proof -
  have sign-r-pos P a = (poly P b > 0)
    apply (rule sign-pos-eq)
    using ⟨a<b⟩ noroot1 by auto
  moreover have sign-r-pos q a = (poly q b > 0)
    apply (rule sign-pos-eq)
    using ⟨a<b⟩ noroot1 by auto
  moreover have sign-r-pos s a = (poly s b > 0)
    apply (rule sign-pos-eq)
    using ⟨a<b⟩ noroot1 by auto
  ultimately show ?thesis
    unfolding CC-def ff-def
    apply (simp add:sign-times)
    using noroot1 ⟨a<b⟩ by (auto simp:sign-def)
qed
finally show ?thesis .
qed
ultimately have ?thesis = (ff ((C1 = C2) = C3) = ff (C4 = C2) +
  ff (C5 = C3) - ff ((C1 = C4) = C5))
  by (fold P-def) auto

```

moreover have $ff ((C1 = C2) = C3) = ff (C4 = C2) + ff (C5 = C3) - ff ((C1 = C4) = C5)$

proof –

have $pp:(0 < poly\ p\ a) = sign-r-pos\ p\ a$
apply $(subst\ sign-r-pos-rec)$
using $\langle poly\ p\ a \neq 0 \rangle$ **by** *auto*

have $rr:(0 < poly\ r\ a) = sign-r-pos\ r\ a$
apply $(subst\ sign-r-pos-rec)$
using $\langle poly\ r\ a \neq 0 \rangle$ **by** *auto*

have $C1\ if\ C2=C5\ C3=C4$

proof –

have $sign-r-pos\ (p * s)\ a$
apply $(subst\ sign-r-pos-mult)$
using $pp\ \langle C2=C5 \rangle\ \langle p \neq 0 \rangle\ \langle s \neq 0 \rangle$ **unfolding** *CC-def* **by** *auto*

moreover have $sign-r-pos\ (q * r)\ a$
apply $(subst\ sign-r-pos-mult)$
using $rr\ \langle C3=C4 \rangle\ \langle q \neq 0 \rangle\ \langle r \neq 0 \rangle$ **unfolding** *CC-def* **by** *auto*

ultimately show *?thesis* **unfolding** *CC-def P-def*
using *sign-r-pos-plus-imp* **by** *auto*

qed

moreover have $foo2:\neg C1\ if\ C2 \neq C5\ C3 \neq C4$

proof –

have $(0 < poly\ p\ a) = sign-r-pos\ (-s)\ a$
apply $(subst\ sign-r-pos-minus)$
using $\langle s \neq 0 \rangle\ \langle C2 \neq C5 \rangle$ **unfolding** *CC-def* **by** *auto*

then have $sign-r-pos\ (p * (-s))\ a$
apply $(subst\ sign-r-pos-mult)$
unfolding *pp* **using** $\langle p \neq 0 \rangle\ \langle s \neq 0 \rangle$ **by** *auto*

moreover have $(0 < poly\ r\ a) = sign-r-pos\ (-q)\ a$
apply $(subst\ sign-r-pos-minus)$
using $\langle q \neq 0 \rangle\ \langle C3 \neq C4 \rangle$ **unfolding** *CC-def* **by** *auto*

then have $sign-r-pos\ (r * (-q))\ a$
apply $(subst\ sign-r-pos-mult)$
unfolding *rr* **using** $\langle r \neq 0 \rangle\ \langle q \neq 0 \rangle$ **by** *auto*

ultimately have $sign-r-pos\ (p * (-s) + r * (-q))\ a$
using *sign-r-pos-plus-imp* **by** *blast*

then have $sign-r-pos\ -(p * s + q * r)\ a$
by $(simp\ add:algebra-simps)$

then have $\neg sign-r-pos\ P\ a$
apply $(subst\ sign-r-pos-minus)$
using $\langle P \neq 0 \rangle$ **unfolding** *P-def* **by** *auto*

then show *?thesis* **unfolding** *CC-def* .

qed

ultimately show *?thesis* **unfolding** *ff-def* **by** *auto*

qed

ultimately show *?thesis* **by** *simp*

qed

ultimately show *?thesis* **using** *that* **by** *auto*

```

qed
ultimately show ?thesis by auto
qed

have ?thesis' p r q s a if poly Q b ≠ 0
  apply (rule base-case[OF - ⟨coprime q p⟩ ⟨coprime s r⟩])
  subgoal using noroot0 that unfolding Q-def P-def by fastforce
  using False ⟨a < b⟩ by auto
moreover have ?thesis' p r q s a if poly Q b = 0
proof -
  have poly Q a ≠ 0 using noroot-disj that by auto

  define h where h = (λp. p ∘p [:a + b, - 1:])

  have h-rw:
    h p - h q = h (p - q)
    h p * h q = h (p * q)
    h p + h q = h (p + q)
    cindex-polyE a b (h q) (h p) = - cindex-polyE a b q p
    cross-alt (h p) (h q) a b = cross-alt p q b a
  for p q
  unfolding h-def pcompose-diff pcompose-mult pcompose-add
    cindex-polyE-linear-comp[OF ⟨a < b⟩, of -1 - a + b, simplified]
    cross-alt-linear-comp[of p a + b - 1 q a b, simplified]
  by simp-all
  have ?thesis' (h p) (h r) (h q) (h s) a
  proof (rule base-case)
    have proots-within (h q * h s * (h p * h s + h q * h r)) {x. a < x ∧ x ≤ b}
      = proots-within (h Q) {x. a < x ∧ x ≤ b}
      unfolding Q-def P-def h-def
      by (simp add: pcompose-diff pcompose-mult pcompose-add)
    also have ... = {}
      unfolding proots-within-def h-def poly-pcompose
      using ⟨a < b⟩ that [folded Q-def] noroot0 [unfolded P-def, folded Q-def] ⟨poly
Q a ≠ 0⟩
      by (auto simp: order.order-iff-strict proots-within-def)
    finally show proots-within (h q * h s * (h p * h s + h q * h r))
      {x. a < x ∧ x ≤ b} = {} .
  show coprime (h q) (h p) unfolding h-def
    apply (rule coprime-linear-comp)
    using ⟨coprime q p⟩ by auto
  show coprime (h s) (h r) unfolding h-def
    apply (rule coprime-linear-comp)
    using ⟨coprime s r⟩ by auto
  show h q ≠ 0 h s ≠ 0 h p ≠ 0 h r ≠ 0
    using False by (auto simp: h-def pcompose-eq-0-iff)
  have h (p * s + q * r) ≠ 0
    using False by (auto simp: h-def pcompose-eq-0-iff)
  then show h p * h s + h q * h r ≠ 0

```

```

    unfolding h-def pcompose-mult pcompose-add by simp
    show  $a < b$  by fact
  qed
  moreover have cross-alt  $(p * s + q * r) (q * s) b a$ 
    = - cross-alt  $(p * s + q * r) (q * s) a b$ 
    unfolding cross-alt-def by auto
  ultimately show ?thesis unfolding h-rw by auto
  qed
  ultimately show ?thesis unfolding Case-def P-def by blast
  qed

show ?thesis using <a<b>
proof (induct card (proots-within  $(q * s * P) \{x. a < x \wedge x \leq b\}$ ) arbitrary:a)
  case 0
  have Case a b
  proof (rule basic-case)
    have *:proots-within  $Q \{x. a < x \wedge x \leq b\} = \{\}$ 
      using 0 <Q≠0> unfolding Q-def by auto
    then show proots-within  $Q \{x. a < x \wedge x < b\} = \{\}$  by force
    show poly  $Q a \neq 0 \vee poly Q b \neq 0$ 
      using * <a<b> by blast
    show  $a < b$  by fact
  qed
  then show ?case unfolding Case-def P-def by simp
next
case (Suc n)

define S where  $S = (\lambda a. \text{proots-within } Q \{x. a < x \wedge x \leq b\})$ 
have Sa-Suc:  $Suc\ n = \text{card } (S\ a)$ 
  using Suc(2) unfolding S-def Q-def by auto

define mroot where  $mroot = \text{Min } (S\ a)$ 
have fin-S:finite  $(S\ a)$  for a
  using Suc(2) unfolding S-def Q-def
  by (simp add: <P ≠ 0> <q ≠ 0> <s ≠ 0>)
have mroot-in:mroot ∈ S a and mroot-min:  $\forall x \in S\ a. mroot \leq x$ 
proof -
  have  $S\ a \neq \{\}$ 
    unfolding S-def Q-def using Suc.hyps(2) by force
  then show mroot ∈ S a unfolding mroot-def
    using Min-in fin-S by auto
  show  $\forall x \in S\ a. mroot \leq x$ 
    using <finite (S a)> Min-le unfolding mroot-def by auto
qed
have mroot-nzero:poly  $Q x \neq 0$  if  $a < x < mroot$  for x
  using mroot-in mroot-min that unfolding S-def
  by (metis (no-types, lifting) dual-order.strict-trans leD
    le-less-linear mem-Collect-eq proots-within-iff )

```

```

define C1 where C1=( $\lambda a b. \text{cindex-polyE } a b (p * r - q * s) P$ )
define C2 where C2=( $\lambda a b. \text{cindex-polyE } a b p q$ )
define C3 where C3=( $\lambda a b. \text{cindex-polyE } a b r s$ )
define C4 where C4=( $\lambda a b. \text{cross-alt } P (q * s) a b$ )
note CC-def = C1-def C2-def C3-def C4-def

```

```

have hyps:C1 mroot b = C2 mroot b + C3 mroot b - C4 mroot b / 2
if mroot < b
  unfolding C1-def C2-def C3-def C4-def P-def
proof (rule Suc.hyps(1)[OF - that])
  have Suc n = card (S a) using Sa-Suc by auto
  also have ... = card (insert mroot (S mroot))
  proof -
    have S a = proots-within Q {x. a < x  $\wedge$  x  $\leq$  b}
      unfolding S-def Q-def by simp
    also have ... = proots-within Q ({x. a < x  $\wedge$  x  $\leq$  mroot}  $\cup$  {x. mroot < x
 $\wedge$  x  $\leq$  b})
      apply (rule arg-cong2[where f=proots-within])
      using mroot-in unfolding S-def by auto
    also have ... = proots-within Q {x. a < x  $\wedge$  x  $\leq$  mroot}  $\cup$  S mroot
      unfolding S-def Q-def
      apply (subst proots-within-union)
      by auto
    also have ... = {mroot}  $\cup$  S mroot
  proof -
    have proots-within Q {x. a < x  $\wedge$  x  $\leq$  mroot} = {mroot}
      using mroot-in mroot-min unfolding S-def
      by auto force
    then show ?thesis by auto
  qed
  finally have S a = insert mroot (S mroot) by auto
  then show ?thesis by auto
qed
also have ... = Suc (card (S mroot))
  apply (rule card-insert-disjoint)
  using fin-S unfolding S-def by auto
finally have Suc n = Suc (card (S mroot)) .
then have n = card (S mroot) by simp
then show n = card (proots-within (q * s * P) {x. mroot < x  $\wedge$  x  $\leq$  b})
  unfolding S-def Q-def by simp
qed

```

```

have ?case if mroot = b
proof -
  have nzero:poly Q x  $\neq$  0 if a < x < b for x
    using mroot-nzero  $\langle$ mroot = b $\rangle$  that by auto

```

```

define m where m=(a+b)/2

```

```

have [simp]:a<m m<b using ‹a<b› unfolding m-def by auto

have Case a m
proof (rule basic-case)
  show roots-within Q {x. a < x ∧ x < m} = {}
    using nzero ‹a<b› unfolding m-def by auto
  have poly Q m ≠ 0 using nzero ‹a<m› ‹m<b› by auto
  then show poly Q a ≠ 0 ∨ poly Q m ≠ 0 by auto
qed simp
moreover have Case m b
proof (rule basic-case)
  show roots-within Q {x. m < x ∧ x < b} = {}
    using nzero ‹a<b› unfolding m-def by auto
  have poly Q m ≠ 0 using nzero ‹a<m› ‹m<b› by auto
  then show poly Q m ≠ 0 ∨ poly Q b ≠ 0 by auto
qed simp
ultimately have C1 a m + C1 m b = (C2 a m + C2 m b)
  + (C3 a m + C3 m b) - (C4 a m + C4 m b)/2
  unfolding Case-def C1-def
  apply simp
  unfolding C2-def C3-def C4-def by (auto simp:algebra-simps)
moreover have
  C1 a m + C1 m b = C1 a b
  C2 a m + C2 m b = C2 a b
  C3 a m + C3 m b = C3 a b
  unfolding CC-def
  by (rule cindex-polyE-combine;auto)+
moreover have C4 a m + C4 m b = C4 a b
  unfolding C4-def cross-alt-def by simp
ultimately have C1 a b = C2 a b + C3 a b - C4 a b/2
  by auto
then show ?thesis unfolding CC-def P-def by auto
qed
moreover have ?case if mroot ≠ b
proof -
  have [simp]:a<mroot mroot < b
    using mroot-in that unfolding S-def by auto

  define m where m=(a+mroot)/2
  have [simp]:a<m m<mroot
    using mroot-in unfolding m-def S-def by auto
  have poly Q m ≠ 0
    by (rule mroot-nzero) auto

  have C1 mroot b = C2 mroot b + C3 mroot b - C4 mroot b / 2
    using hyps ‹mroot<b› by simp
  moreover have Case a m
  apply (rule basic-case)
  subgoal

```

by (smt (verit) Collect-empty-eq ⟨m < mroot⟩ mem-Collect-eq mroot-nzero
 roots-within-def)
 subgoal using ⟨poly Q m ≠ 0⟩ by auto
 by fact
 then have C1 a m = C2 a m + C3 a m - C4 a m / 2
 unfolding Case-def CC-def by auto
 moreover have Case m mroot
 apply (rule basic-case)
 subgoal
 by (smt (verit) Collect-empty-eq ⟨a < m⟩ mem-Collect-eq mroot-nzero
 roots-within-def)
 subgoal using ⟨poly Q m ≠ 0⟩ by auto
 by fact
 then have C1 m mroot = C2 m mroot + C3 m mroot - C4 m mroot / 2
 unfolding Case-def CC-def by auto
 ultimately have C1 a m + C1 m mroot + C1 mroot b =
 (C2 a m + C2 m mroot + C2 mroot b)
 + (C3 a m + C3 m mroot + C3 mroot b)
 - (C4 a m + C4 m mroot + C4 mroot b) / 2
 by simp (simp add: algebra-simps)
 moreover have
 C1 a m + C1 m mroot + C1 mroot b = C1 a b
 C2 a m + C2 m mroot + C2 mroot b = C2 a b
 C3 a m + C3 m mroot + C3 mroot b = C3 a b
 unfolding CC-def
 by (subst cindex-polyE-combine; simp?)+
 moreover have C4 a m + C4 m mroot + C4 mroot b = C4 a b
 unfolding C4-def cross-alt-def by simp
 ultimately have C1 a b = C2 a b + C3 a b - C4 a b / 2
 by auto
 then show ?thesis unfolding CC-def P-def by auto
 qed
 ultimately show ?case by auto
 qed
 qed

lemma cindex-polyE-product:

fixes p r q s :: real poly and a b :: real

assumes a < b

and poly p a ≠ 0 ∨ poly q a ≠ 0 poly p b ≠ 0 ∨ poly q b ≠ 0

and poly r a ≠ 0 ∨ poly s a ≠ 0 poly r b ≠ 0 ∨ poly s b ≠ 0

shows cindex-polyE a b (p * r - q * s) (p * s + q * r)

= cindex-polyE a b p q + cindex-polyE a b r s

- cross-alt (p * s + q * r) (q * s) a b / 2

proof -

define g1 where g1 = gcd p q

obtain p' q' where pq: p = g1 * p' q = g1 * q' and coprime q' p'

unfolding g1-def

by (*metis* *assms*(2) *coprime-commute div-gcd-coprime dvd-mult-div-cancel gcd-dvd1*
gcd-dvd2 order-root)

define *g2* **where** *g2* = *gcd r s*
obtain *r' s'* **where** *rs:r=g2*r' s = g2 * s' coprime s' r'*
unfolding *g2-def* **using** *assms*(4)
by (*metis* *coprime-commute div-gcd-coprime dvd-mult-div-cancel gcd-dvd1 gcd-dvd2*
order-root)

define *g* **where** *g=g1 * g2*
have [*simp*]:*g≠0 g1≠0 g2≠0*
unfolding *g-def g1-def g2-def*
using *assms* **by** *auto*
have [*simp*]:*poly g a ≠ 0 poly g b ≠ 0*
unfolding *g-def g1-def g2-def*
subgoal by (*metis* *assms*(2) *assms*(4) *poly-gcd-0-iff poly-mult-zero-iff*)
subgoal by (*metis* *assms*(3) *assms*(5) *poly-gcd-0-iff poly-mult-zero-iff*)
done

have *cindex-polyE a b (p' * r' - q' * s') (p' * s' + q' * r') =*
cindex-polyE a b p' q' + cindex-polyE a b r' s' -
*(cross-alt (p' * s' + q' * r') (q' * s') a b) / 2*
using *cindex-polyE-product'[OF ‹a<b› ‹coprime q' p'› ‹coprime s' r'›]* .
moreover have *cindex-polyE a b (p * r - q * s) (p * s + q * r)*
= cindex-polyE a b (g(p' * r' - q' * s')) (g*(p' * s' + q' * r'))*
unfolding *pq rs g-def* **by** (*auto simp:algebra-simps*)
then have *cindex-polyE a b (p * r - q * s) (p * s + q * r)*
*= cindex-polyE a b (p' * r' - q' * s') (p' * s' + q' * r')*
apply (*subst (asm) cindex-polyE-mult-cancel*)
by *simp*
moreover have *cindex-polyE a b p q = cindex-polyE a b p' q'*
unfolding *pq* **using** *cindex-polyE-mult-cancel* **by** *simp*
moreover have *cindex-polyE a b r s = cindex-polyE a b r' s'*
unfolding *rs* **using** *cindex-polyE-mult-cancel* **by** *simp*
moreover have *cross-alt (p * s + q * r) (q * s) a b*
= cross-alt (g(p' * s' + q' * r')) (g*(q' * s')) a b*
unfolding *pq rs g-def* **by** (*auto simp:algebra-simps*)
then have *cross-alt (p * s + q * r) (q * s) a b*
*= cross-alt (p' * s' + q' * r') (q' * s') a b*
apply (*subst (asm) cross-alt-cancel*)
by *simp-all*
ultimately show *?thesis* **by** *auto*
qed

lemma *cindex-pathE-linepath-on*:
assumes *z ∈ closed-segment a b*
shows *cindex-pathE (linepath a b) z = 0*
proof –

```

obtain  $u$  where  $0 \leq u \leq 1$ 
  and  $z$ -eq: $z = \text{complex-of-real } (1 - u) * a + \text{complex-of-real } u * b$ 
  using assms unfolding in-segment scaleR-conv-of-real
  by auto

define  $U$  where  $U = [-u, 1:]$ 
have  $U \neq 0$  unfolding  $U$ -def by auto

have  $\text{cindex-pathE } (\text{linepath } a \ b) \ z$ 
  =  $\text{cindexE } 0 \ 1 \ (\lambda t. (\text{Im } a + t * \text{Im } b - (\text{Im } z + t * \text{Im } a))$ 
    /  $(\text{Re } a + t * \text{Re } b - (\text{Re } z + t * \text{Re } a))$ )
  unfolding  $\text{cindex-pathE-def}$ 
  by (simp add:linepath-def algebra-simps)
also have ... =  $\text{cindexE } 0 \ 1$ 
  ( $\lambda t. ((\text{Im } b - \text{Im } a) * (t - u))$ 
    /  $((\text{Re } b - \text{Re } a) * (t - u))$ )
  unfolding  $z$ -eq
  by (simp add:algebra-simps)
also have ... =  $\text{cindex-polyE } 0 \ 1 \ (U * [:\text{Im } b - \text{Im } a:]) \ (U * [:\text{Re } b - \text{Re } a:])$ 
proof (subst cindexE-eq-cindex-polyE[symmetric])
  have  $(\text{Im } b - \text{Im } a) * (t - u) / ((\text{Re } b - \text{Re } a) * (t - u))$ 
    =  $\text{poly } (U * [:\text{Im } b - \text{Im } a:]) \ t / \text{poly } (U * [:\text{Re } b - \text{Re } a:]) \ t$  for  $t$ 
  unfolding  $U$ -def by (simp add:algebra-simps)
  then show  $\text{cindexE } 0 \ 1 \ (\lambda t. (\text{Im } b - \text{Im } a) * (t - u) / ((\text{Re } b - \text{Re } a) * (t - u))) =$ 
     $\text{cindexE } 0 \ 1 \ (\lambda x. \text{poly } (U * [:\text{Im } b - \text{Im } a:]) \ x / \text{poly } (U * [:\text{Re } b - \text{Re } a:]) \ x)$ 
  by auto
qed simp
also have ... =  $\text{cindex-polyE } 0 \ 1 \ [:\text{Im } b - \text{Im } a:] \ [:\text{Re } b - \text{Re } a:]$ 
  apply (rule cindex-polyE-mult-cancel)
  by fact
also have ... =  $\text{cindexE } 0 \ 1 \ (\lambda x. (\text{Im } b - \text{Im } a) / (\text{Re } b - \text{Re } a))$ 
  apply (subst cindexE-eq-cindex-polyE[symmetric])
  by auto
also have ... = 0
  apply (rule cindexE-constI)
  by auto
finally show ?thesis .
qed

```

2.7 More Cauchy indices on polynomials

definition $\text{cindexP-pathE} :: \text{complex poly} \Rightarrow (\text{real} \Rightarrow \text{complex}) \Rightarrow \text{real}$ **where**
 $\text{cindexP-pathE } p \ g = \text{cindex-pathE } (\text{poly } p \ o \ g) \ 0$

definition $\text{cindexP-lineE} :: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{real}$ **where**
 $\text{cindexP-lineE } p \ a \ b = \text{cindexP-pathE } p \ (\text{linepath } a \ b)$

lemma *cindexP-pathE-const:cindexP-pathE* [:c:] $g = 0$
unfolding *cindexP-pathE-def* **by** (*auto intro:cindex-pathE-constI*)

lemma *cindex-poly-pathE-joinpaths*:
assumes *finite-ReZ-segments* (*poly p o g1*) 0
and *finite-ReZ-segments* (*poly p o g2*) 0
and *path g1* **and** *path g2*
and *pathfinish g1 = pathstart g2*
shows *cindexP-pathE p (g1 +++ g2)*
 $= \text{cindexP-pathE } p \text{ } g1 + \text{cindexP-pathE } p \text{ } g2$

proof –

have *path (poly p o g1) path (poly p o g2)*
using $\langle \text{path } g1 \rangle \langle \text{path } g2 \rangle$ **by** *auto*
moreover **have** *pathfinish (poly p o g1) = pathstart (poly p o g2)*
using $\langle \text{pathfinish } g1 = \text{pathstart } g2 \rangle$
by (*simp add: pathfinish-compose pathstart-def*)
ultimately **have**
 $\text{cindex-pathE } ((\text{poly } p \text{ o } g1) \text{ +++ } (\text{poly } p \text{ o } g2)) \text{ } 0 =$
 $\text{cindex-pathE } (\text{poly } p \text{ o } g1) \text{ } 0 + \text{cindex-pathE } (\text{poly } p \text{ o } g2) \text{ } 0$
using *cindex-pathE-joinpaths[OF assms(1,2)]* **by** *auto*
then **show** *?thesis*
unfolding *cindexP-pathE-def*
by (*simp add:path-compose-join*)

qed

lemma *cindexP-lineE-polyE*:
fixes *p::complex poly* **and** *a b::complex*
defines $pp \equiv \text{pcompose } p \text{ } [:a, b-a:]$
defines $pR \equiv \text{map-poly } \text{Re } pp$
and $pI \equiv \text{map-poly } \text{Im } pp$
shows $\text{cindexP-lineE } p \text{ } a \text{ } b = \text{cindex-polyE } 0 \text{ } 1 \text{ } pI \text{ } pR$

proof –

have $\text{cindexP-lineE } p \text{ } a \text{ } b = \text{cindexE } 0 \text{ } 1$
 $(\lambda t. \text{Im } (\text{poly } (p \circ_p [:a, b-a:]) (\text{complex-of-real } t)) /$
 $\text{Re } (\text{poly } (p \circ_p [:a, b-a:]) (\text{complex-of-real } t)))$
unfolding *cindexP-lineE-def cindexP-pathE-def cindex-pathE-def*
by (*simp add:poly-linepath-comp'*)
also **have** $\dots = \text{cindexE } 0 \text{ } 1 (\lambda t. \text{poly } pI \text{ } t / \text{poly } pR \text{ } t)$
unfolding *pI-def pR-def pp-def*
by (*simp add:Im-poly-of-real Re-poly-of-real*)
also **have** $\dots = \text{cindex-polyE } 0 \text{ } 1 \text{ } pI \text{ } pR$
apply (*subst cindexE-eq-cindex-polyE*)
by *simp-all*
finally **show** *?thesis* .

qed

definition *psign-aux* :: *complex poly* \Rightarrow *complex poly* \Rightarrow *complex* \Rightarrow *int* **where**
 $\text{psign-aux } p \text{ } q \text{ } b =$
 $\text{sign } (\text{Im } (\text{poly } p \text{ } b * \text{poly } q \text{ } b)) * (\text{Im } (\text{poly } p \text{ } b) * \text{Im } (\text{poly } q \text{ } b))$

$+ \text{sign } (\text{Re } (\text{poly } p \ b * \text{poly } q \ b) * \text{Im } (\text{poly } p \ b * \text{poly } q \ b))$
 $- \text{sign } (\text{Re } (\text{poly } p \ b) * \text{Im } (\text{poly } p \ b))$
 $- \text{sign } (\text{Re } (\text{poly } q \ b) * \text{Im } (\text{poly } q \ b))$

definition *cdiff-aux* :: *complex poly* \Rightarrow *complex poly* \Rightarrow *complex* \Rightarrow *complex* \Rightarrow *int*
where

cdiff-aux *p q a b* = *psign-aux* *p q b* - *psign-aux* *p q a*

lemma *cindexP-lineE-times*:

fixes *p q*::*complex poly* **and** *a b*::*complex*

assumes *poly p a* $\neq 0$ *poly p b* $\neq 0$ *poly q a* $\neq 0$ *poly q b* $\neq 0$

shows *cindexP-lineE* (*p*q*) *a b* = *cindexP-lineE* *p a b* + *cindexP-lineE* *q a b* + *cdiff-aux* *p q a b* / 2

proof -

define *pR pI* **where** *pR* = *map-poly* *Re* (*p* \circ_p [:*a*, *b* - *a*:])
and *pI* = *map-poly* *Im* (*p* \circ_p [:*a*, *b* - *a*:])

define *qR qI* **where** *qR* = *map-poly* *Re* (*q* \circ_p [:*a*, *b* - *a*:])
and *qI* = *map-poly* *Im* (*q* \circ_p [:*a*, *b* - *a*:])

define *P1 P2* **where** *P1* = *pR* * *qI* + *pI* * *qR* **and** *P2* = *pR* * *qR* - *pI* * *qI*

have *p-poly*:

poly pR 0 = *Re* (*poly p a*)

poly pI 0 = *Im* (*poly p a*)

poly pR 1 = *Re* (*poly p b*)

poly pI 1 = *Im* (*poly p b*)

unfolding *pR-def pI-def*

by (*simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose*) +

have *q-poly*:

poly qR 0 = *Re* (*poly q a*)

poly qI 0 = *Im* (*poly q a*)

poly qR 1 = *Re* (*poly q b*)

poly qI 1 = *Im* (*poly q b*)

unfolding *qR-def qI-def*

by (*simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose*) +

have *P2-poly*:

poly P2 0 = *Re* (*poly (p*q) a*)

poly P2 1 = *Re* (*poly (p*q) b*)

unfolding *P2-def pR-def qI-def pI-def qR-def*

by (*simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose*) +

have *P1-poly*:

poly P1 0 = *Im* (*poly (p*q) a*)

poly P1 1 = *Im* (*poly (p*q) b*)

unfolding *P1-def pR-def qI-def pI-def qR-def*

by (*simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose*) +

have *p-nzero*: *poly pR 0* $\neq 0$ \vee *poly pI 0* $\neq 0$ *poly pR 1* $\neq 0$ \vee *poly pI 1* $\neq 0$

unfolding *p-poly*

using *assms(1,2)* *complex-eqI* **by** *force* +

have $q\text{-nzero:poly } qR \ 0 \neq 0 \vee \text{poly } qI \ 0 \neq 0 \text{ poly } qR \ 1 \neq 0 \vee \text{poly } qI \ 1 \neq 0$
unfolding $q\text{-poly}$ **using** $\text{assms}(3,4)$ complex-eqI **by** force+

have $P12\text{-nzero:poly } P2 \ 0 \neq 0 \vee \text{poly } P1 \ 0 \neq 0 \text{ poly } P2 \ 1 \neq 0 \vee \text{poly } P1 \ 1 \neq 0$
unfolding $P1\text{-poly } P2\text{-poly}$ **using** assms
by $(\text{metis } \text{Im-poly-hom.base.hom-zero } \text{Re-poly-hom.base.hom-zero } \text{complex-eqI } \text{poly-mult-zero-iff})+$

define $C1 \ C2$ **where** $C1 = (\lambda p \ q. \text{cindex-polyE } 0 \ 1 \ p \ q)$
and $C2 = (\lambda p \ q. \text{real-of-int } (\text{cross-alt } p \ q \ 0 \ 1) / 2)$

define CR **where** $CR = C2 \ P1 \ (pI * qI) + C2 \ P2 \ P1 - C2 \ pR \ pI - C2 \ qR \ qI$

have $\text{cindexP-lineE } (p*q) \ a \ b =$
 $\text{cindex-polyE } 0 \ 1 \ (\text{map-poly } \text{Im} \ (\text{cpoly-of } pR \ pI * \text{cpoly-of } qR \ qI))$
 $(\text{map-poly } \text{Re} \ (\text{cpoly-of } pR \ pI * \text{cpoly-of } qR \ qI))$

proof $-$
have $p \circ_p [:a, b - a:] = \text{cpoly-of } pR \ pI$
using $\text{cpoly-of-decompose } pI\text{-def } pR\text{-def}$ **by** blast
moreover **have** $q \circ_p [:a, b - a:] = \text{cpoly-of } qR \ qI$
using $\text{cpoly-of-decompose } qI\text{-def } qR\text{-def}$ **by** blast
ultimately show $?thesis$
apply $(\text{subst } \text{cindexP-lineE-polyE})$
unfolding $p\text{compose-mult}$ **by** simp

qed

also **have** $\dots = \text{cindex-polyE } 0 \ 1 \ (pR * qI + pI * qR) \ (pR * qR - pI * qI)$
unfolding cpoly-of-times **by** $(\text{simp } \text{add:algebra-simps})$

also **have** $\dots = \text{cindex-polyE } 0 \ 1 \ P1 \ P2$
unfolding $P1\text{-def } P2\text{-def}$ **by** simp

also **have** $\dots = \text{cindex-polyE } 0 \ 1 \ pI \ pR + \text{cindex-polyE } 0 \ 1 \ qI \ qR + CR$

proof $-$
have $C1 \ P2 \ P1 = C1 \ pR \ pI + C1 \ qR \ qI - C2 \ P1 \ (pI * qI)$
unfolding $P1\text{-def } P2\text{-def } C1\text{-def } C2\text{-def}$
apply $(\text{rule } \text{cindex-polyE-product})$ **thm** $\text{cindex-polyE-product}$
by simp fact+

moreover **have** $C1 \ P2 \ P1 = C2 \ P2 \ P1 - C1 \ P1 \ P2$
unfolding $C1\text{-def } C2\text{-def}$
apply $(\text{subst } \text{cindex-polyE-inverse-add-cross}'[\text{symmetric}])$
using $P12\text{-nzero}$ **by** simp-all

moreover **have** $C1 \ pR \ pI = C2 \ pR \ pI - C1 \ pI \ pR$
unfolding $C1\text{-def } C2\text{-def}$
apply $(\text{subst } \text{cindex-polyE-inverse-add-cross}'[\text{symmetric}])$
using $p\text{-nzero}$ **by** simp-all

moreover **have** $C1 \ qR \ qI = C2 \ qR \ qI - C1 \ qI \ qR$
unfolding $C1\text{-def } C2\text{-def}$
apply $(\text{subst } \text{cindex-polyE-inverse-add-cross}'[\text{symmetric}])$
using $q\text{-nzero}$ **by** simp-all

ultimately **have** $C2 \ P2 \ P1 - C1 \ P1 \ P2 = (C2 \ pR \ pI - C1 \ pI \ pR)$
 $+ (C2 \ qR \ qI - C1 \ qI \ qR) - C2 \ P1 \ (pI * qI)$

by auto

```

then have  $C1\ P1\ P2 = C1\ pI\ pR + C1\ qI\ qR + CR$ 
  unfolding  $CR\text{-def}$  by (auto simp: algebra-simps)
then show ?thesis unfolding  $C1\text{-def}$  .
qed
also have ... =  $cindexP\text{-lineE}\ p\ a\ b + cindexP\text{-lineE}\ q\ a\ b + CR$ 
  unfolding  $C1\text{-def}\ pI\text{-def}\ pR\text{-def}\ qI\text{-def}\ qR\text{-def}$ 
  apply (subst (1 2)  $cindexP\text{-lineE}\text{-polyE}$ )
  by simp
also have ... =  $cindexP\text{-lineE}\ p\ a\ b + cindexP\text{-lineE}\ q\ a\ b + cdiff\text{-aux}\ p\ q\ a\ b/2$ 
proof -
  have  $CR = cdiff\text{-aux}\ p\ q\ a\ b/2$ 
  unfolding  $CR\text{-def}\ C2\text{-def}\ cross\text{-alt}\text{-alt}\ cdiff\text{-aux}\text{-def}\ psign\text{-aux}\text{-def}$ 
  by (simp add:  $P1\text{-poly}\ P2\text{-poly}\ p\text{-poly}\ q\text{-poly}\ del:\text{times}\text{-complex}\text{-sel}$ )
  then show ?thesis by simp
qed
finally show ?thesis .
qed

lemma  $cindexP\text{-lineE}\text{-changes}$ :
  fixes  $p::\text{complex}\ \text{poly}$  and  $a\ b::\text{complex}$ 
  assumes  $p\neq 0\ a\neq b$ 
  shows  $cindexP\text{-lineE}\ p\ a\ b =$ 
    (let  $p1 = pcompose\ p\ [:a,\ b-a];$ 
       $pR1 = map\text{-poly}\ Re\ p1;$ 
       $pI1 = map\text{-poly}\ Im\ p1;$ 
       $gc1 = gcd\ pR1\ pI1$ 
    in
     $real\text{-of}\text{-int}\ (changes\text{-alt}\text{-itv}\text{-smods}\ 0\ 1$ 
       $(pR1\ div\ gc1)\ (pI1\ div\ gc1)) / 2$ )

proof -
  define  $p1\ pR1\ pI1\ gc1$  where  $p1 = pcompose\ p\ [:a,\ b-a];$ 
  and  $pR1 = map\text{-poly}\ Re\ p1$  and  $pI1 = map\text{-poly}\ Im\ p1$ 
  and  $gc1 = gcd\ pR1\ pI1$ 

  have  $gc1 \neq 0$ 
  proof (rule ccontr)
    assume  $\neg gc1 \neq 0$ 
    then have  $pI1 = 0\ pR1 = 0$  unfolding  $gc1\text{-def}$  by auto
    then have  $p1 = 0$  unfolding  $pI1\text{-def}\ pR1\text{-def}$ 
      by (metis  $cpoly\text{-of}\text{-decompose}\ map\text{-poly}\ 0$ )
    with  $\langle a\neq b \rangle$  have  $p=0$  unfolding  $p1\text{-def}$ 
      by (auto simp:  $pcompose\text{-eq}\ 0\text{-iff}$ )
    then show False using  $\langle p\neq 0 \rangle$  by auto
  qed

  have  $cindexP\text{-lineE}\ p\ a\ b =$ 
     $cindexE\ 0\ 1\ (\lambda t.\ Im\ (poly\ p\ (linepath\ a\ b\ t)))$ 
    /  $Re\ (poly\ p\ (linepath\ a\ b\ t))$ 
  unfolding  $cindexP\text{-lineE}\text{-def}\ cindex\text{-pathE}\text{-def}\ cindexP\text{-pathE}\text{-def}$  by simp

```

```

also have ... = cindexE 0 1 ( $\lambda t. \text{poly } pI1 \ t / \text{poly } pR1 \ t$ )
  unfolding pI1-def pR1-def p1-def poly-linepath-comp'
  by (simp add:Im-poly-of-real Re-poly-of-real)
also have ... = cindex-polyE 0 1 pI1 pR1
  by (simp add:cindexE-eq-cindex-polyE)
also have ... = cindex-polyE 0 1 (pI1 div gc1) (pR1 div gc1)
  using  $\langle gc1 \neq 0 \rangle$ 
  apply (subst (2) cindex-polyE-mult-cancel[of gc1,symmetric])
  by (simp-all add:gc1-def)
also have ... = real-of-int (changes-alt-itv-smods 0 1
    (pR1 div gc1) (pI1 div gc1)) / 2
  apply (rule cindex-polyE-changes-alt-itv-smods)
  apply simp
  by (metis  $\langle gc1 \neq 0 \rangle$  div-gcd-coprime gc1-def gcd-eq-0-iff)
finally show ?thesis
  by (metis gc1-def p1-def pI1-def pR1-def)
qed

```

```

lemma cindexP-lineE-code[code]:
  cindexP-lineE p a b = (if  $p \neq 0 \wedge a \neq b$  then
    (let p1 = pcompose p [:a, b-a];
      pR1 = map-poly Re p1;
      pI1 = map-poly Im p1;
      gc1 = gcd pR1 pI1
    in
      real-of-int (changes-alt-itv-smods 0 1
        (pR1 div gc1) (pI1 div gc1)) / 2)
  else
    Code.abort (STR "cindexP-lineE fails for now")
    ( $\lambda. \text{cindexP-lineE } p \ a \ b$ )
using cindexP-lineE-changes by auto

```

end

```

theory Count-Line imports
  CC-Polynomials-Extra
  Winding-Number-Eval.Winding-Number-Eval
  Extended-Sturm
  Budan-Fourier.Sturm-Multiple-Roots
begin

```

2.8 Misc

```

lemma closed-segment-imp-Re-Im:
  fixes  $x::\text{complex}$ 
  assumes  $x \in \text{closed-segment } lb \ ub$ 
  shows  $\text{Re } lb \leq \text{Re } ub \implies \text{Re } lb \leq \text{Re } x \wedge \text{Re } x \leq \text{Re } ub$ 

```

$Im\ lb \leq Im\ ub \implies Im\ lb \leq Im\ x \wedge Im\ x \leq Im\ ub$

proof –

obtain u **where** $x: x = (1 - u) *_R lb + u *_R ub$ **and** $0 \leq u \leq 1$
using *assms unfolding closed-segment-def* **by** *auto*
have $Re\ lb \leq Re\ x$ **when** $Re\ lb \leq Re\ ub$

proof –

have $Re\ x = Re\ ((1 - u) *_R lb + u *_R ub)$
using $x-u$ **by** *blast*
also have $\dots = Re\ (lb + u *_R (ub - lb))$ **by** *(auto simp add: algebra-simps)*
also have $\dots = Re\ lb + u * (Re\ ub - Re\ lb)$ **by** *auto*
also have $\dots \geq Re\ lb$ **using** $\langle u \geq 0 \rangle \langle Re\ lb \leq Re\ ub \rangle$ **by** *auto*
finally show *?thesis* .

qed

moreover have $Im\ lb \leq Im\ x$ **when** $Im\ lb \leq Im\ ub$

proof –

have $Im\ x = Im\ ((1 - u) *_R lb + u *_R ub)$
using $x-u$ **by** *blast*
also have $\dots = Im\ (lb + u *_R (ub - lb))$ **by** *(auto simp add: algebra-simps)*
also have $\dots = Im\ lb + u * (Im\ ub - Im\ lb)$ **by** *auto*
also have $\dots \geq Im\ lb$ **using** $\langle u \geq 0 \rangle \langle Im\ lb \leq Im\ ub \rangle$ **by** *auto*
finally show *?thesis* .

qed

moreover have $Re\ x \leq Re\ ub$ **when** $Re\ lb \leq Re\ ub$

proof –

have $Re\ x = Re\ ((1 - u) *_R lb + u *_R ub)$
using $x-u$ **by** *blast*
also have $\dots = (1 - u) * Re\ lb + u * Re\ ub$ **by** *auto*
also have $\dots \leq (1 - u) * Re\ ub + u * Re\ ub$
using $\langle u \leq 1 \rangle \langle Re\ lb \leq Re\ ub \rangle$ **by** *(auto simp add: mult-left-mono)*
also have $\dots = Re\ ub$ **by** *(auto simp add: algebra-simps)*
finally show *?thesis* .

qed

moreover have $Im\ x \leq Im\ ub$ **when** $Im\ lb \leq Im\ ub$

proof –

have $Im\ x = Im\ ((1 - u) *_R lb + u *_R ub)$
using $x-u$ **by** *blast*
also have $\dots = (1 - u) * Im\ lb + u * Im\ ub$ **by** *auto*
also have $\dots \leq (1 - u) * Im\ ub + u * Im\ ub$
using $\langle u \leq 1 \rangle \langle Im\ lb \leq Im\ ub \rangle$ **by** *(auto simp add: mult-left-mono)*
also have $\dots = Im\ ub$ **by** *(auto simp add: algebra-simps)*
finally show *?thesis* .

qed

ultimately show

$Re\ lb \leq Re\ ub \implies Re\ lb \leq Re\ x \wedge Re\ x \leq Re\ ub$
 $Im\ lb \leq Im\ ub \implies Im\ lb \leq Im\ x \wedge Im\ x \leq Im\ ub$

by *auto*

qed

lemma *closed-segment-degen-complex*:

```

[[Re lb = Re ub; Im lb ≤ Im ub ]]
  ⇒ x ∈ closed-segment lb ub ⇔ Re x = Re lb ∧ Im lb ≤ Im x ∧ Im x ≤ Im
ub
[[Im lb = Im ub; Re lb ≤ Re ub ]]
  ⇒ x ∈ closed-segment lb ub ⇔ Im x = Im lb ∧ Re lb ≤ Re x ∧ Re x ≤ Re
ub
proof –
  show x ∈ closed-segment lb ub ⇔ Re x = Re lb ∧ Im lb ≤ Im x ∧ Im x ≤ Im
ub
  when Re lb = Re ub Im lb ≤ Im ub
  proof
    show Re x = Re lb ∧ Im lb ≤ Im x ∧ Im x ≤ Im ub when x ∈ closed-segment
lb ub
    using closed-segment-imp-Re-Im[OF that] ⟨Re lb = Re ub⟩ ⟨Im lb ≤ Im ub⟩
by fastforce
  next
    assume asm:Re x = Re lb ∧ Im lb ≤ Im x ∧ Im x ≤ Im ub
    define u where u=(Im x - Im lb)/ (Im ub - Im lb)
    have x = (1 - u) *R lb + u *R ub
    unfolding u-def using asm ⟨Re lb = Re ub⟩ ⟨Im lb ≤ Im ub⟩
    apply (intro complex-eqI)
    apply (auto simp add:field-simps)
    apply (cases Im ub - Im lb =0)
    apply (auto simp add:field-simps)
    done
    moreover have 0 ≤ u ≤ 1 unfolding u-def
    using ⟨Im lb ≤ Im ub⟩ asm
    by (cases Im ub - Im lb =0, auto simp add:field-simps)+
    ultimately show x ∈ closed-segment lb ub unfolding closed-segment-def by
auto
  qed
  show x ∈ closed-segment lb ub ⇔ Im x = Im lb ∧ Re lb ≤ Re x ∧ Re x ≤ Re
ub
  when Im lb = Im ub Re lb ≤ Re ub
  proof
    show Im x = Im lb ∧ Re lb ≤ Re x ∧ Re x ≤ Re ub when x ∈ closed-segment
lb ub
    using closed-segment-imp-Re-Im[OF that] ⟨Im lb = Im ub⟩ ⟨Re lb ≤ Re ub⟩
by fastforce
  next
    assume asm:Im x = Im lb ∧ Re lb ≤ Re x ∧ Re x ≤ Re ub
    define u where u=(Re x - Re lb)/ (Re ub - Re lb)
    have x = (1 - u) *R lb + u *R ub
    unfolding u-def using asm ⟨Im lb = Im ub⟩ ⟨Re lb ≤ Re ub⟩
    apply (intro complex-eqI)
    apply (auto simp add:field-simps)
    apply (cases Re ub - Re lb =0)
    apply (auto simp add:field-simps)
    done

```

moreover have $0 \leq u \leq 1$ **unfolding** *u-def*
using $\langle \text{Re } lb \leq \text{Re } ub \rangle \text{ asm}$
by $(\text{cases } \text{Re } ub - \text{Re } lb = 0, \text{auto simp add: field-simps}) +$
ultimately show $x \in \text{closed-segment } lb \ ub$ **unfolding** *closed-segment-def* **by**
auto
qed
qed

corollary *path-image-part-circlepath-subset*:
assumes $r \geq 0$
shows $\text{path-image}(\text{part-circlepath } z \ r \ st \ tt) \subseteq \text{sphere } z \ r$
proof $(\text{cases } st \leq tt)$
case *True*
then show *?thesis*
by $(\text{auto simp: assms path-image-part-circlepath sphere-def dist-norm algebra-simps norm-mult})$
next
case *False*
then have $\text{path-image}(\text{part-circlepath } z \ r \ tt \ st) \subseteq \text{sphere } z \ r$
by $(\text{auto simp: assms path-image-part-circlepath sphere-def dist-norm algebra-simps norm-mult})$
moreover have $\text{path-image}(\text{part-circlepath } z \ r \ tt \ st) = \text{path-image}(\text{part-circlepath } z \ r \ st \ tt)$
using *path-image-reversepath* **by** *fastforce*
ultimately show *?thesis* **by** *auto*
qed

proposition *in-path-image-part-circlepath*:
assumes $w \in \text{path-image}(\text{part-circlepath } z \ r \ st \ tt)$ $0 \leq r$
shows $\text{norm}(w - z) = r$
proof $-$
have $w \in \{c. \text{dist } z \ c = r\}$
by $(\text{metis } (\text{no-types}) \text{ path-image-part-circlepath-subset sphere-def subset-eq assms})$
thus *?thesis*
by $(\text{simp add: dist-norm norm-minus-commute})$
qed

lemma *infinite-ball*:
fixes $a :: 'a::\text{euclidean-space}$
assumes $r > 0$
shows *infinite* $(\text{ball } a \ r)$
using *uncountable-ball[OF assms, THEN uncountable-infinite]* .

lemma *infinite-cball*:
fixes $a :: 'a::\text{euclidean-space}$
assumes $r > 0$
shows *infinite* $(\text{cball } a \ r)$

using uncountable-cball[OF assms, THEN uncountable-infinite, of a] .

lemma *infinite-sphere*:

fixes $a :: \text{complex}$

assumes $r > 0$

shows *infinite* (sphere a r)

proof –

have uncountable (path-image (circlepath a r))

 apply (rule simple-path-image-uncountable)

 using simple-path-circlepath assms by simp

then have uncountable (sphere a r)

 using assms by simp

from uncountable-infinite[OF this] show ?thesis .

qed

lemma *infinite-halfspace-Im-gt*: *infinite* { x . $\text{Im } x > b$ }

 apply (rule connected-uncountable[THEN uncountable-infinite, of - (b+1)* i (b+2)*i])

 by (auto intro!: convex-connected simp add: convex-halfspace-Im-gt)

lemma (in ring-1) *Ints-minus2*: $- a \in \mathbb{Z} \implies a \in \mathbb{Z}$

 using Ints-minus[of -a] by auto

lemma *dvd-divide-Ints-iff*:

$b \text{ dvd } a \vee b=0 \iff \text{of-int } a / \text{of-int } b \in (\mathbb{Z} :: 'a :: \{\text{field}, \text{ring-char-0}\} \text{ set})$

proof

 assume asm: $b \text{ dvd } a \vee b=0$

 let ?thesis = $\text{of-int } a / \text{of-int } b \in (\mathbb{Z} :: 'a :: \{\text{field}, \text{ring-char-0}\} \text{ set})$

 have ?thesis when $b \text{ dvd } a$

proof –

 obtain c where $a=b * c$ using $\langle b \text{ dvd } a \rangle$ unfolding dvd-def by auto

 then show ?thesis by (auto simp add: field-simps)

qed

 moreover have ?thesis when $b=0$

 using that by auto

 ultimately show ?thesis using asm by auto

next

 assume $\text{of-int } a / \text{of-int } b \in (\mathbb{Z} :: 'a :: \{\text{field}, \text{ring-char-0}\} \text{ set})$

 from Ints-cases[OF this] obtain c where $*(\text{of-int}::- \Rightarrow 'a) c = \text{of-int } a / \text{of-int } b$

 by metis

 have $b \text{ dvd } a$ when $b \neq 0$

proof –

 have $(\text{of-int}::- \Rightarrow 'a) a = \text{of-int } b * \text{of-int } c$ using that * by auto

 then have $a = b * c$ using of-int-eq-iff by fastforce

 then show ?thesis unfolding dvd-def by auto

qed

 then show $b \text{ dvd } a \vee b = 0$ by auto

qed

lemma *of-int-div-field*:
assumes $d \text{ dvd } n$
shows $(\text{of-int}::\Rightarrow 'a::\text{field-char-0}) (n \text{ div } d) = \text{of-int } n / \text{of-int } d$
apply (*subst* (2) *dvd-mult-div-cancel*[*OF assms,symmetric*])
by (*auto simp add:field-simps*)

lemma *powr-eq-1-iff*:
assumes $a > 0$
shows $(a::\text{real}) \text{ powr } b = 1 \iff a = 1 \vee b = 0$
proof
assume $a \text{ powr } b = 1$
have $b * \ln a = 0$
using $\langle a \text{ powr } b = 1 \rangle \text{ ln-powr}$ [*of a b*] *assms* **by** *auto*
then have $b = 0 \vee \ln a = 0$ **by** *auto*
then show $a = 1 \vee b = 0$ **using** *assms* **by** *auto*
qed (*insert assms, auto*)

lemma *tan-inj-pi*:
 $-(\pi/2) < x \implies x < \pi/2 \implies -(\pi/2) < y \implies y < \pi/2 \implies \tan x = \tan y$
 $\implies x = y$
by (*metis arctan-tan*)

lemma *finite-ReZ-segments-poly-circlepath*:
 $\text{finite-ReZ-segments } (\text{poly } p \circ \text{circlepath } z0 \text{ } r) \text{ } 0$
proof (*cases* $\forall t \in \{0..1\} - \{1/2\}. \text{Re } ((\text{poly } p \circ \text{circlepath } z0 \text{ } r) \text{ } t) = 0$)
case *True*
have *isCont* $(\text{Re} \circ \text{poly } p \circ \text{circlepath } z0 \text{ } r) (1/2)$
by (*auto intro!:continuous-intros simp:circlepath*)
moreover have $(\text{Re} \circ \text{poly } p \circ \text{circlepath } z0 \text{ } r) - 1/2 \rightarrow 0$
proof -
have $\forall_F x \text{ in at } (1 / 2). (\text{Re} \circ \text{poly } p \circ \text{circlepath } z0 \text{ } r) \text{ } x = 0$
unfolding *eventually-at-le*
apply (*rule exI*[*where* $x = 1/2$])
unfolding *dist-real-def abs-diff-le-iff*
by (*auto intro!:True*[*rule-format, unfolded comp-def*])
then show *?thesis* **by** (*rule tendsto-eventually*)
qed
ultimately have $\text{Re } ((\text{poly } p \circ \text{circlepath } z0 \text{ } r) (1/2)) = 0$
unfolding *comp-def* **by** (*simp add: LIM-unique continuous-within*)
then have $\forall t \in \{0..1\}. \text{Re } ((\text{poly } p \circ \text{circlepath } z0 \text{ } r) \text{ } t) = 0$
using *True* **by** *blast*
then show *?thesis*
apply (*rule-tac finite-ReZ-segments-constI*[*THEN finite-ReZ-segments-congE*])
by *auto*
next
case *False*
define $q1 \text{ } q2$ **where** $q1 = \text{fcompose } p [:(z0+r)*i, z0-r:] [:(i,1:)]$ **and**

```

      q2=([i, 1:] ^ degree p)
define q1R q1I where q1R=map-poly Re q1 and q1I=map-poly Im q1
define q2R q2I where q2R=map-poly Re q2 and q2I=map-poly Im q2
define qq where qq=q1R*q2R + q1I*q2I

have poly-eq:Re ((poly p o circlepath z0 r) t) = 0  $\longleftrightarrow$  poly qq (tan (pi * t)) = 0
  when 0 ≤ t ≤ 1 t ≠ 1/2 for t
proof -
  define tt where tt=tan (pi * t)
  have Re ((poly p o circlepath z0 r) t) = 0  $\longleftrightarrow$  Re (poly q1 tt / poly q2 tt) = 0
    unfolding comp-def
    apply (subst poly-circlepath-tan-eq[of t p z0 r, folded q1-def q2-def tt-def])
    using that by simp-all
  also have ...  $\longleftrightarrow$  poly q1R tt * poly q2R tt + poly q1I tt * poly q2I tt = 0
    unfolding q1I-def q1R-def q2R-def q2I-def
    by (simp add:Re-complex-div-eq-0 Re-poly-of-real Im-poly-of-real)
  also have ...  $\longleftrightarrow$  poly qq tt = 0
    unfolding qq-def by simp
  finally show ?thesis unfolding tt-def .
qed

```

```

have finite {t. Re ((poly p o circlepath z0 r) t) = 0 ∧ 0 ≤ t ∧ t ≤ 1}
proof -
  define P where P=(λt. Re ((poly p o circlepath z0 r) t) = 0)
  define A where A= ({0..1}::real set)
  define S where S={t∈A-{1,1/2}. P t}
  have finite {t. poly qq (tan (pi * t)) = 0 ∧ 0 ≤ t ∧ t < 1 ∧ t ≠ 1/2}
  proof -
    define A where A={t::real. 0 ≤ t ∧ t < 1 ∧ t ≠ 1 / 2}
    have finite ((λt. tan (pi * t)) -' {x. poly qq x=0} ∩ A)
    proof (rule finite-vimage-IntI)
      have x = y when tan (pi * x) = tan (pi * y) x∈A y∈A for x y
      proof -
        define x' where x'=(if x<1/2 then x else x-1)
        define y' where y'=(if y<1/2 then y else y-1)
        have x'*pi = y'*pi
        proof (rule tan-inj-pi)
          have *: - 1 / 2 < x' x' < 1 / 2 - 1 / 2 < y' y' < 1 / 2
            using that(2,3) unfolding x'-def y'-def A-def by simp-all
          show - (pi / 2) < x' * pi x' * pi < pi / 2 - (pi / 2) < y' * pi
            y'*pi < pi / 2
            using mult-strict-right-mono[OF *(1), of pi]
              mult-strict-right-mono[OF *(2), of pi]
              mult-strict-right-mono[OF *(3), of pi]
              mult-strict-right-mono[OF *(4), of pi]
          by auto
        next
          have tan (x' * pi) = tan (x * pi)
            unfolding x'-def using tan-periodic-int[of - - 1, simplified]

```

```

      by (auto simp add: algebra-simps)
    also have ... = tan (y * pi)
      using ⟨tan (pi * x) = tan (pi * y)⟩ by (auto simp: algebra-simps)
    also have ... = tan (y' * pi)
      unfolding y'-def using tan-periodic-int[of - - 1, simplified]
      by (auto simp add: algebra-simps)
    finally show tan (x' * pi) = tan (y' * pi) .
  qed
  then have x'=y' by auto
  then show ?thesis
    using that(2,3) unfolding x'-def y'-def A-def by (auto split: if-splits)
  qed
  then show inj-on (λt. tan (pi * t)) A
    unfolding inj-on-def by blast
next
  have qq≠0
  proof (rule ccontr)
    assume ¬ qq ≠ 0
    then have Re ((poly p ∘ circlepath z0 r) t) = 0 when t ∈ {0..1} - {1/2}
for t
    apply (subst poly-eq)
    using that by auto
    then show False using False by blast
  qed
  then show finite {x. poly qq x = 0} by (simp add: poly-roots-finite)
  qed
  then show ?thesis by (elim rev-finite-subset) (auto simp: A-def)
  qed
  moreover have {t. poly qq (tan (pi * t)) = 0 ∧ 0 ≤ t ∧ t < 1 ∧ t ≠ 1/2} = S
    unfolding S-def P-def A-def using poly-eq by force
  ultimately have finite S by blast
  then have finite (S ∪ (if P 1 then {1} else {}) ∪ (if P (1/2) then {1/2} else
{}))
    by auto
  moreover have (S ∪ (if P 1 then {1} else {}) ∪ (if P (1/2) then {1/2} else
{}))
    = {t. P t ∧ 0 ≤ t ∧ t ≤ 1}
  proof -
    have 1 ∈ A 1/2 ∈ A unfolding A-def by auto
    then have (S ∪ (if P 1 then {1} else {}) ∪ (if P (1/2) then {1/2} else {}))
      = {t ∈ A. P t}
      unfolding S-def
      apply auto
      by (metis eq-divide-eq-numeral1(1) zero-neq-numeral)+
    also have ... = {t. P t ∧ 0 ≤ t ∧ t ≤ 1}
      unfolding A-def by auto
    finally show ?thesis .
  qed
  ultimately have finite {t. P t ∧ 0 ≤ t ∧ t ≤ 1} by auto

```

then show *?thesis unfolding P-def by simp*
qed
then show *?thesis*
apply (*rule-tac finite-imp-finite-ReZ-segments*)
by *auto*
qed

lemma *changes-itv-smods-ext-geq-0:*
assumes *a < b poly p a ≠ 0 poly p b ≠ 0*
shows *changes-itv-smods-ext a b p (pderiv p) ≥ 0*
using *sturm-ext-interval[OF assms]* **by** *auto*

2.9 Some useful conformal/*bij-betw* properties

lemma *bij-betw-plane-ball:bij-betw* ($\lambda x. (i-x)/(i+x)$) $\{x. \text{Im } x > 0\}$ (*ball 0 1*)
proof (*rule bij-betw-imageI*)
have *neq:i + x ≠ 0 when Im x > 0 for x*
using *that*
by (*metis add-less-same-cancel2 add-uminus-conv-diff diff-0 diff-add-cancel*
imaginary-unit.simps(2) not-one-less-zero uminus-complex.sel(2))
then show *inj-on* ($\lambda x. (i-x)/(i+x)$) $\{x. 0 < \text{Im } x\}$
unfolding *inj-on-def* **by** (*auto simp add:divide-simps algebra-simps*)
have *cmod* ($(i-x)/(i+x)$) < 1 **when** $0 < \text{Im } x$ **for** x
proof –
have *cmod* ($i-x$) $<$ *cmod* ($i+x$)
unfolding *norm-lt inner-complex-def* **using** *that*
by (*auto simp add:algebra-simps*)
then show *?thesis*
unfolding *norm-divide* **using** *neq[OF that]* **by** *auto*
qed
moreover **have** $x \in (\lambda x. (i-x)/(i+x))^{-1} \{x. 0 < \text{Im } x\}$ **when** *cmod* $x < 1$
for x
proof (*rule rev-image-eqI[of i*(1-x)/(1+x)]*)
have $1 + x \neq 0$ $i * 2 + i * (x * 2) \neq 0$
subgoal **using** *that* **by** (*metis complex-mod-triangle-sub norm-one norm-zero*
not-le pth-7(1))
subgoal **using** *that* **by** (*metis* $\langle 1 + x \neq 0 \rangle$ *complex-i-not-zero div-mult-self4*
mult-2
mult-zero-right nonzero-mult-div-cancel-left nonzero-mult-div-cancel-right
one-add-one zero-neq-numeral)
done
then show $x = (i - i * (1-x) / (1+x)) / (i + i * (1-x) / (1+x))$
by (*auto simp add:field-simps*)
show $i * (1-x) / (1+x) \in \{x. 0 < \text{Im } x\}$
apply (*auto simp:Im-complex-div-gt-0 algebra-simps*)
using *that* **unfolding** *cmod-def* **by** (*auto simp:power2-eq-square*)
qed
ultimately show ($\lambda x. (i-x)/(i+x)$) $^{-1} \{x. 0 < \text{Im } x\} = \text{ball } 0 \ 1$
by *auto*

qed

lemma *bij-betw-axis-sphere*: $\text{bij-betw } (\lambda x. (i-x)/(i+x)) \{x. \text{Im } x=0\} (\text{sphere } 0 \ 1 - \{-1\})$

proof (rule *bij-betw-imageI*)

have $\text{neq}: i + x \neq 0$ **when** $\text{Im } x=0$ **for** x

using *that*

by (*metis add-diff-cancel-left' imaginary-unit.simps(2) minus-complex.simps(2)*)

right-minus-eq zero-complex.simps(2) zero-neq-one)

then show $\text{inj-on } (\lambda x. (i-x)/(i+x)) \{x. \text{Im } x = 0\}$

unfolding *inj-on-def* **by** (*auto simp add:divide-simps algebra-simps*)

have $\text{cmod } ((i-x)/(i+x)) = 1$ $(i-x)/(i+x) \neq -1$ **when** $\text{Im } x = 0$ **for** x

proof -

have $\text{cmod } (i+x) = \text{cmod } (i-x)$

using *that* **unfolding** *cmod-def* **by** *auto*

then show $\text{cmod } ((i-x)/(i+x)) = 1$

unfolding *norm-divide* **using** *neq[OF that]* **by** *auto*

show $(i-x)/(i+x) \neq -1$ **using** *neq[OF that]* **by** (*auto simp add:divide-simps*)

qed

moreover have $x \in (\lambda x. (i-x)/(i+x))^{-1} \{x. \text{Im } x = 0\}$

when $\text{cmod } x = 1$ $x \neq -1$ **for** x

proof (rule *rev-image-eqI*[of $i*(1-x)/(1+x)$])

have $1 + x \neq 0$ $i * 2 + i * (x * 2) \neq 0$

subgoal using *that(2)* **by** *algebra*

subgoal using *that* **by** (*metis <1 + x ≠ 0> complex-i-not-zero div-mult-self4 mult-2*)

mult-zero-right nonzero-mult-div-cancel-left nonzero-mult-div-cancel-right one-add-one zero-neq-numeral)

done

then show $x = (i - i * (1-x)/(1+x)) / (i + i * (1-x)/(1+x))$

by (*auto simp add:field-simps*)

show $i * (1-x)/(1+x) \in \{x. \text{Im } x = 0\}$

apply (*auto simp:algebra-simps Im-complex-div-eq-0*)

using *that(1)* **unfolding** *cmod-def* **by** (*auto simp:power2-eq-square*)

qed

ultimately show $(\lambda x. (i-x)/(i+x))^{-1} \{x. \text{Im } x = 0\} = \text{sphere } 0 \ 1 - \{-1\}$

by *force*

qed

lemma *bij-betw-ball-uball*:

assumes $r > 0$

shows *bij-betw* $(\lambda x. \text{complex-of-real } r*x + z0)$ (*ball* 0 1) (*ball* z0 r)

proof (rule *bij-betw-imageI*)

show *inj-on* $(\lambda x. \text{complex-of-real } r * x + z0)$ (*ball* 0 1)

unfolding *inj-on-def* **using** *assms* **by** *simp*

have *dist* z0 (*complex-of-real* $r * x + z0$) < r **when** $\text{cmod } x < 1$ **for** x

using *that* *assms* **by** (*auto simp:dist-norm norm-mult abs-of-pos*)

moreover have $x \in (\lambda x. \text{complex-of-real } r * x + z0)$ ‘ball 0 1’ **when** $\text{dist } z0 \ x < r$ **for** x
apply (rule rev-image-eqI[of (x-z0)/r])
using that assms by (auto simp add: dist-norm norm-divide norm-minus-commute)
ultimately show $(\lambda x. \text{complex-of-real } r * x + z0)$ ‘ball 0 1 = ball z0 r’
by auto
qed

lemma *bij-betw-sphere-usphere*:

assumes $r > 0$
shows *bij-betw* $(\lambda x. \text{complex-of-real } r * x + z0)$ (sphere 0 1) (sphere z0 r)
proof (rule *bij-betw-imageI*)
show *inj-on* $(\lambda x. \text{complex-of-real } r * x + z0)$ (sphere 0 1)
unfolding *inj-on-def* **using assms by** *simp*
have $\text{dist } z0 (\text{complex-of-real } r * x + z0) = r$ **when** $\text{cmod } x = 1$ **for** x
using that assms by (auto simp: dist-norm norm-mult abs-of-pos)
moreover have $x \in (\lambda x. \text{complex-of-real } r * x + z0)$ ‘sphere 0 1’ **when** $\text{dist } z0 \ x = r$ **for** x
apply (rule rev-image-eqI[of (x-z0)/r])
using that assms by (auto simp add: dist-norm norm-divide norm-minus-commute)
ultimately show $(\lambda x. \text{complex-of-real } r * x + z0)$ ‘sphere 0 1 = sphere z0 r’
by auto
qed

lemma *proots-ball-plane-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $\text{proots-count } p (\text{ball } 0 \ 1) = \text{proots-count } (fcompose \ p \ q1 \ q2) \{x. \ 0 < \text{Im } x\}$
unfolding *q1-def q2-def*
proof (rule *proots-fcompose-bij-eq[OF - ⟨p≠0⟩]*)
show $\forall x \in \{x. \ 0 < \text{Im } x\}. \ \text{poly } [i, 1:] \ x \neq 0$
apply *simp*
by (metis *add-less-same-cancel2 imaginary-unit.simps(2) not-one-less-zero plus-complex.simps(2) zero-complex.simps(2)*)
show *infinite* (UNIV::complex set) **by** (simp add: *infinite-UNIV-char-0*)
qed (use *bij-betw-plane-ball in auto*)

lemma *proots-sphere-axis-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $\text{proots-count } p (\text{sphere } 0 \ 1 - \{-1\}) = \text{proots-count } (fcompose \ p \ q1 \ q2) \{x. \ 0 = \text{Im } x\}$
unfolding *q1-def q2-def*
proof (rule *proots-fcompose-bij-eq[OF - ⟨p≠0⟩]*)
show $\forall x \in \{x. \ 0 = \text{Im } x\}. \ \text{poly } [i, 1:] \ x \neq 0$ **by** (simp add: *Complex-eq-0 plus-complex.code*)
show *infinite* (UNIV::complex set) **by** (simp add: *infinite-UNIV-char-0*)
qed (use *bij-betw-axis-sphere in auto*)

lemma *roots-card-ball-plane-eq*:
defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $\text{card} (\text{roots-within } p (\text{ball } 0 \ 1)) = \text{card} (\text{roots-within } (fcompose \ p \ q1 \ q2) \{x. \ 0 < \text{Im } x\})$
unfolding $q1\text{-def}$ $q2\text{-def}$
proof (*rule roots-card-fcompose-bij-eq*[*OF* - $\langle p \neq 0 \rangle$])
show $\forall x \in \{x. \ 0 < \text{Im } x\}. \ \text{poly } [i, 1:] \ x \neq 0$
apply *simp*
by (*metis add-less-same-cancel2 imaginary-unit.simps*(2) *not-one-less-zero plus-complex.simps*(2) *zero-complex.simps*(2))
qed (*use bij-betw-plane-ball infinite-UNIV-char-0 in auto*)

lemma *roots-card-sphere-axis-eq*:
defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $\text{card} (\text{roots-within } p (\text{sphere } 0 \ 1 - \{-1\})) = \text{card} (\text{roots-within } (fcompose \ p \ q1 \ q2) \{x. \ 0 = \text{Im } x\})$
unfolding $q1\text{-def}$ $q2\text{-def}$
proof (*rule roots-card-fcompose-bij-eq*[*OF* - $\langle p \neq 0 \rangle$])
show $\forall x \in \{x. \ 0 = \text{Im } x\}. \ \text{poly } [i, 1:] \ x \neq 0$ **by** (*simp add: Complex-eq-0 plus-complex.code*)
qed (*use bij-betw-axis-sphere infinite-UNIV-char-0 in auto*)

lemma *roots-uball-eq*:
fixes $z0::\text{complex}$ **and** $r::\text{real}$
defines $q \equiv [z0, \text{of-real } r:]$
assumes $p \neq 0$ **and** $r > 0$
shows $\text{roots-count } p (\text{ball } z0 \ r) = \text{roots-count } (p \circ_p \ q) (\text{ball } 0 \ 1)$
proof –
show *?thesis*
apply (*rule roots-pcompose-bij-eq*[*OF* - $\langle p \neq 0 \rangle$])
subgoal unfolding $q\text{-def}$ **using** *bij-betw-ball-uball*[*OF* $\langle r > 0 \rangle, \text{of } z0$] **by** (*auto simp: algebra-simps*)
subgoal unfolding $q\text{-def}$ **using** $\langle r > 0 \rangle$ **by** *auto*
done
qed

lemma *roots-card-uball-eq*:
fixes $z0::\text{complex}$ **and** $r::\text{real}$
defines $q \equiv [z0, \text{of-real } r:]$
assumes $r > 0$
shows $\text{card} (\text{roots-within } p (\text{ball } z0 \ r)) = \text{card} (\text{roots-within } (p \circ_p \ q) (\text{ball } 0 \ 1))$
proof –
have *?thesis*
when $p = 0$
proof –

```

have card (ball z0 r) = 0 card (ball (0::complex) 1) = 0
  using infinite-ball[OF ‹r>0›,of z0] infinite-ball[of 1 0::complex] by auto
then show ?thesis using that by auto
qed
moreover have ?thesis
  when p≠0
  apply (rule proots-card-pcompose-bij-eq[OF - ‹p≠0›])
  subgoal unfolding q-def using bij-betw-ball-uball[OF ‹r>0›,of z0] by (auto
simp:algebra-simps)
  subgoal unfolding q-def using ‹r>0› by auto
  done
ultimately show ?thesis
  by blast
qed

```

```

lemma proots-card-usphere-eq:
  fixes z0::complex and r::real
  defines q≡[:z0, of-real r:]
  assumes r>0
  shows card (proots-within p (sphere z0 r)) = card (proots-within (p ∘p q) (sphere
0 1))
proof –
  have ?thesis
    when p=0
  proof –
    have card (sphere z0 r) = 0 card (sphere (0::complex) 1) = 0
      using infinite-sphere[OF ‹r>0›,of z0] infinite-sphere[of 1 0::complex] by auto

```

```

  then show ?thesis using that by auto
qed
moreover have ?thesis
  when p≠0
  apply (rule proots-card-pcompose-bij-eq[OF - ‹p≠0›])
  subgoal unfolding q-def using bij-betw-sphere-usphere[OF ‹r>0›,of z0]
    by (auto simp:algebra-simps)
  subgoal unfolding q-def using ‹r>0› by auto
  done
  ultimately show card (proots-within p (sphere z0 r)) = card (proots-within (p
∘p q) (sphere 0 1))
    by blast
qed

```

2.10 Number of roots on a (bounded or unbounded) segment

definition unbounded-line::'a::real-vector ⇒ 'a ⇒ 'a **set** **where**
unbounded-line a b = ({x. ∃ u::real. x = (1 - u) *_R a + u *_R b})

definition proots-line-card:: complex poly ⇒ complex ⇒ complex ⇒ nat **where**
proots-line-card p st tt = card (proots-within p (open-segment st tt))

definition *roots-unbounded-line-card*:: $\text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**

roots-unbounded-line-card p st tt = $\text{card} (\text{roots-within } p (\text{unbounded-line } st \ tt))$

definition *roots-unbounded-line* :: $\text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**

roots-unbounded-line p st tt = $\text{roots-count } p (\text{unbounded-line } st \ tt)$

lemma *card-roots-open-segments*:

assumes $\text{poly } p \ st \neq 0$ $\text{poly } p \ tt \neq 0$

shows $\text{card} (\text{roots-within } p (\text{open-segment } st \ tt)) =$

$(\text{let } pc = \text{pcompose } p \ [:st, \ tt - st :];$

$pR = \text{map-poly } \text{Re } pc;$

$pI = \text{map-poly } \text{Im } pc;$

$g = \text{gcd } pR \ pI$

$\text{in } \text{changes-itv-smods } 0 \ 1 \ g \ (\text{pderiv } g))$ (**is** $?L = ?R$)

proof –

define pc pR pI g **where**

$pc = \text{pcompose } p \ [:st, \ tt - st :]$ **and**

$pR = \text{map-poly } \text{Re } pc$ **and**

$pI = \text{map-poly } \text{Im } pc$ **and**

$g = \text{gcd } pR \ pI$

have $\text{poly-iff}:\text{poly } g \ t = 0 \longleftrightarrow \text{poly } pc \ t = 0$ **for** t

proof –

have $\text{poly } g \ t = 0 \longleftrightarrow \text{poly } pR \ t = 0 \wedge \text{poly } pI \ t = 0$

unfolding $g\text{-def}$ **using** poly-gcd-0-iff **by** *auto*

also have $\dots \longleftrightarrow \text{poly } pc \ t = 0$

proof –

have $\text{cpoly-of } pR \ pI = pc$

unfolding $pc\text{-def}$ $pR\text{-def}$ $pI\text{-def}$ **using** $\text{cpoly-of-decompose}$ **by** *auto*

then show $?thesis$ **using** $\text{poly-cpoly-of-real-iff}$ **by** *blast*

qed

finally show $?thesis$ **by** *auto*

qed

have $?R = \text{changes-itv-smods } 0 \ 1 \ g \ (\text{pderiv } g)$

unfolding $pc\text{-def}$ $g\text{-def}$ $pI\text{-def}$ $pR\text{-def}$ **by** $(\text{auto } \text{simp } \text{add:Let-def})$

also have $\dots = \text{card } \{t. \text{poly } g \ t = 0 \wedge 0 < t \wedge t < 1\}$

proof –

have $\text{poly } g \ 0 \neq 0$

using $\text{poly-iff}[of \ 0]$ assms **unfolding** $pc\text{-def}$ **by** $(\text{auto } \text{simp } \text{add:poly-pcompose})$

moreover have $\text{poly } g \ 1 \neq 0$

using $\text{poly-iff}[of \ 1]$ assms **unfolding** $pc\text{-def}$ **by** $(\text{auto } \text{simp } \text{add:poly-pcompose})$

ultimately show $?thesis$ **using** $\text{sturm-interval}[of \ 0 \ 1 \ g]$ **by** *auto*

qed

also have $\dots = \text{card } \{t::\text{real}. \text{poly } pc \ (\text{of-real } t) = 0 \wedge 0 < t \wedge t < 1\}$

unfolding poly-iff **by** *simp*

also have $\dots = ?L$

```

proof (cases st=tt)
  case True
    then show ?thesis unfolding pc-def poly-pcompose using ⟨poly p tt ≠ 0⟩
    by auto
  next
    case False
    define ff where ff = (λt::real. st + t*(tt-st))
    define ll where ll = {t. poly pc (complex-of-real t) = 0 ∧ 0 < t ∧ t < 1}
    have ff ‘ ll = roots-within p (open-segment st tt)
    proof (rule equalityI)
      show ff ‘ ll ⊆ roots-within p (open-segment st tt)
        unfolding ll-def ff-def pc-def poly-pcompose
        by (auto simp add:in-segment False scaleR-conv-of-real algebra-simps)
    next
      show roots-within p (open-segment st tt) ⊆ ff ‘ ll
      proof clarify
        fix x assume asm:x ∈ roots-within p (open-segment st tt)
        then obtain u where 0 < u and u < 1 and u:x = (1 - u) *R st + u *R tt
          by (auto simp add:in-segment)
        then have poly p ((1 - u) *R st + u *R tt) = 0 using asm by simp
        then have u ∈ ll
          unfolding ll-def pc-def poly-pcompose
          by (simp add:scaleR-conv-of-real algebra-simps ⟨0 < u⟩ ⟨u < 1⟩)
        moreover have x = ff u
        unfolding ff-def using u by (auto simp add:algebra-simps scaleR-conv-of-real)
        ultimately show x ∈ ff ‘ ll by (rule rev-image-eqI[of u])
      qed
    moreover have inj-on ff ll
      unfolding ff-def using False inj-on-def by fastforce
    ultimately show ?thesis unfolding ll-def
      using card-image[of ff] by fastforce
    qed
  finally show ?thesis by simp
qed

```

lemma unbounded-line-closed-segment: closed-segment a b ⊆ unbounded-line a b
unfolding unbounded-line-def closed-segment-def **by** auto

lemma card-roots-unbounded-line:

```

assumes st≠tt
shows card (roots-within p (unbounded-line st tt)) =
  (let pc = pcompose p [:st, tt - st];
    pR = map-poly Re pc;
    pI = map-poly Im pc;
    g = gcd pR pI
    in nat (changes-R-smods g (pderiv g))) (is ?L = ?R)

```

proof –
define pc pR pI g **where**

```

    pc = pcompose p [:st, tt-st:] and
    pR = map-poly Re pc and
    pI = map-poly Im pc and
    g = gcd pR pI
have poly-iff:poly g t=0  $\longleftrightarrow$  poly pc t = 0 for t
proof -
  have poly g t = 0  $\longleftrightarrow$  poly pR t = 0  $\wedge$  poly pI t = 0
    unfolding g-def using poly-gcd-0-iff by auto
  also have ...  $\longleftrightarrow$  poly pc t = 0
  proof -
    have cpoly-of pR pI = pc
      unfolding pc-def pR-def pI-def using cpoly-of-decompose by auto
    then show ?thesis using poly-cpoly-of-real-iff by blast
  qed
finally show ?thesis by auto
qed

have ?R = nat (changes-R-smods g (pderiv g))
  unfolding pc-def g-def pI-def pR-def by (auto simp add:Let-def)
also have ... = card {t. poly g t = 0}
  using sturm-R[of g] by simp
also have ... = card {t::real. poly pc t = 0}
  unfolding poly-iff by simp
also have ... = ?L
proof (cases st=tt)
  case True
  then show ?thesis unfolding pc-def poly-pcompose unbounded-line-def using
assms
  by (auto simp add:roots-within-def)
next
case False
define ff where ff = ( $\lambda t::real. st + t*(tt-st)$ )
define ll where ll = {t. poly pc (complex-of-real t) = 0}
have ff ' ll = roots-within p (unbounded-line st tt)
proof (rule equalityI)
  show ff ' ll  $\subseteq$  roots-within p (unbounded-line st tt)
    unfolding ll-def ff-def pc-def poly-pcompose
  by (auto simp add:unbounded-line-def False scaleR-conv-of-real algebra-simps)
next
show roots-within p (unbounded-line st tt)  $\subseteq$  ff ' ll
proof clarify
  fix x assume asm:x  $\in$  roots-within p (unbounded-line st tt)
  then obtain u where u:x = (1 - u) *R st + u *R tt
    by (auto simp add:unbounded-line-def)
  then have poly p ((1 - u) *R st + u *R tt) = 0 using asm by simp
  then have u  $\in$  ll
    unfolding ll-def pc-def poly-pcompose
  by (simp add:scaleR-conv-of-real algebra-simps unbounded-line-def)
  moreover have x = ff u

```

```

    unfolding ff-def using u by (auto simp add:algebra-simps scaleR-conv-of-real)
    ultimately show  $x \in ff \text{ ' } ll$  by (rule rev-image-eqI[of u])
  qed
  qed
  moreover have inj-on ff ll
    unfolding ff-def using False inj-on-def by fastforce
  ultimately show ?thesis unfolding ll-def
    using card-image[of ff] by metis
  qed
  finally show ?thesis by simp
  qed

lemma roots-count-gcd-eq:
  fixes  $p::\text{complex poly}$  and  $st\ tt::\text{complex}$ 
  and  $g::\text{real poly}$ 
  defines  $pc \equiv pcompose\ p\ [:st, tt - st:]$ 
  defines  $pR \equiv map\text{-poly}\ Re\ pc$  and  $pI \equiv map\text{-poly}\ Im\ pc$ 
  defines  $g \equiv gcd\ pR\ pI$ 
  assumes  $st \neq tt$   $p \neq 0$ 
    and  $s1\text{-def}:s1 = (\lambda x. poly\ [:st, tt - st:] (of\text{-real}\ x)) \text{ ' } s2$ 
  shows  $roots\text{-count}\ p\ s1 = roots\text{-count}\ g\ s2$ 
  proof -
    have [simp]:  $g \neq 0$   $pc \neq 0$ 
    proof -
      show  $pc \neq 0$  using assms  $pc\text{-def}$   $pcompose\text{-eq}\ 0$ 
      by (metis cancel-comm-monoid-add-class.diff-cancel degree-pCons-eq-if
        diff-eq-diff-eq less-nat-zero-code pCons-eq-0-iff zero-less-Suc)
      then have  $pR \neq 0 \vee pI \neq 0$  unfolding  $pR\text{-def}\ pI\text{-def}$  by (metis cpoly-of-decompose
        map-poly-0)
      then show  $g \neq 0$  unfolding  $g\text{-def}$  by simp
    qed
    have  $order\text{-eq}:order\ t\ g = order\ t\ pc$  for  $t$ 
      apply (subst order-cpoly-gcd-eq[of pR pI, folded g-def, symmetric])
      subgoal using  $\langle g \neq 0 \rangle$  unfolding  $g\text{-def}$  by simp
      subgoal unfolding  $pR\text{-def}\ pI\text{-def}$  by (simp add:cpoly-of-decompose[symmetric])
      done

    have  $roots\text{-count}\ g\ s2 = roots\text{-count}\ (map\text{-poly}\ complex\text{-of}\text{-real}\ g)$ 
      (of-real '  $s2$ )
      apply (subst roots-count-of-real)
      by auto
    also have  $\dots = roots\text{-count}\ pc$  (of-real '  $s2$ )
      apply (rule roots-count-cong)
      by (auto simp add: map-poly-order-of-real order-eq)
    also have  $\dots = roots\text{-count}\ p\ s1$ 
      unfolding  $pc\text{-def}\ s1\text{-def}$ 
      apply (subst roots-pcompose)
      using  $\langle st \neq tt \rangle$   $\langle p \neq 0 \rangle$  by (simp-all add:image-image)
    finally show ?thesis by simp

```

qed

lemma *proots-unbounded-line*:

assumes $st \neq tt$ $p \neq 0$

shows $(\text{proots-count } p \ (\text{unbounded-line } st \ tt)) =$

$(\text{let } pc = \text{pcompose } p \ [:st, tt - st:];$

$pR = \text{map-poly } Re \ pc;$

$pI = \text{map-poly } Im \ pc;$

$g = \text{gcd } pR \ pI$

$\text{in nat } (\text{changes-R-smods-ext } g \ (\text{pderiv } g))) \ (\text{is } ?L = ?R)$

proof –

define $pc \ pR \ pI \ g$ **where**

$pc = \text{pcompose } p \ [:st, tt - st:]$ **and**

$pR = \text{map-poly } Re \ pc$ **and**

$pI = \text{map-poly } Im \ pc$ **and**

$g = \text{gcd } pR \ pI$

have $[simp]: g \neq 0 \ pc \neq 0$

proof –

show $pc \neq 0$ **using** $assms(1) \ assms(2) \ pc\text{-def} \ pcompose\text{-eq-0}$

by $(\text{metis } \text{cancel-comm-monoid-add-class.diff-cancel } \text{degree-pCons-eq-if}$

$\text{diff-eq-diff-eq } \text{less-nat-zero-code } \text{pCons-eq-0-iff } \text{zero-less-Suc})$

then have $pR \neq 0 \vee pI \neq 0$ **unfolding** $pR\text{-def} \ pI\text{-def}$ **by** $(\text{metis } \text{cpoly-of-decompose}$
 $\text{map-poly-0})$

then show $g \neq 0$ **unfolding** $g\text{-def}$ **by** $simp$

qed

have $\text{order-eq:order } t \ g = \text{order } t \ pc$ **for** t

apply $(\text{subst } \text{order-cpoly-gcd-eq}[of \ pR \ pI, \text{folded } g\text{-def}, \text{symmetric}])$

subgoal using $\langle g \neq 0 \rangle$ **unfolding** $g\text{-def}$ **by** $simp$

subgoal unfolding $pR\text{-def} \ pI\text{-def}$ **by** $(\text{simp } \text{add:cpoly-of-decompose}[\text{symmetric}])$

done

have $?R = \text{nat } (\text{changes-R-smods-ext } g \ (\text{pderiv } g))$

unfolding $pc\text{-def} \ g\text{-def} \ pI\text{-def} \ pR\text{-def}$ **by** $(\text{auto } \text{simp } \text{add:Let-def})$

also have $\dots = \text{proots-count } g \ UNIV$

using $\text{sturm-ext-R}[OF \ \langle g \neq 0 \rangle]$ **by** $auto$

also have $\dots = \text{proots-count } (\text{map-poly } \text{complex-of-real } g) \ \{x. \text{Im } x = 0\}$

apply $(\text{subst } \text{proots-count-of-real})$

by $auto$

also have $\dots = \text{proots-count } (\text{map-poly } \text{complex-of-real } g) \ \{x. \text{Im } x = 0\}$

apply $(\text{rule } \text{arg-cong2}[\text{where } f = \text{proots-count}])$

using $\text{Reals-def } \text{complex-is-Real-iff}$ **by** $auto$

also have $\dots = \text{proots-count } pc \ \{x. \text{Im } x = 0\}$

apply $(\text{rule } \text{proots-count-cong})$

apply $(\text{metis } (\text{mono-tags}) \ \text{Im-complex-of-real } \text{Re-complex-of-real } \langle g \neq 0 \rangle \ \text{complex-surj}$

$\text{map-poly-order-of-real } \text{mem-Collect-eq } \text{order-eq})$

by $auto$

also have $\dots = \text{proots-count } p \ (\text{unbounded-line } st \ tt)$

proof –

```

have poly [:st, tt - st:] ‘ {x. Im x = 0} = unbounded-line st tt
unfolding unbounded-line-def
apply safe
subgoal for - x
  apply (rule-tac x=Re x in exI)
  apply (simp add:algebra-simps)
  by (simp add: mult.commute scaleR-complex.code times-complex.code)
subgoal for - u
  apply (rule rev-image-eqI[of of-real u])
  by (auto simp:scaleR-conv-of-real algebra-simps)
done
then show ?thesis
unfolding pc-def
apply (subst proots-pcompose)
using ⟨p≠0⟩ ⟨st≠tt⟩ by auto
qed
finally show ?thesis by simp
qed

lemma proots-unbounded-line-card-code[code]:
  proots-unbounded-line-card p st tt =
    (if st≠tt then
      (let pc = pcompose p [:st, tt - st:];
        pR = map-poly Re pc;
        pI = map-poly Im pc;
        g = gcd pR pI
      in nat (changes-R-smods g (pderiv g)))
    else
      Code.abort (STR "proots-unbounded-line-card fails due to invalid
hyperplanes."))
    (λ-. proots-unbounded-line-card p st tt)
unfolding proots-unbounded-line-card-def using card-proots-unbounded-line[of st
tt p] by auto

lemma proots-unbounded-line-code[code]:
  proots-unbounded-line p st tt =
    ( if st≠tt then
      if p≠0 then
        (let pc = pcompose p [:st, tt - st:];
          pR = map-poly Re pc;
          pI = map-poly Im pc;
          g = gcd pR pI
        in nat (changes-R-smods-ext g (pderiv g)))
      else
        Code.abort (STR "proots-unbounded-line fails due to p=0")
        (λ-. proots-unbounded-line p st tt)
      else
        Code.abort (STR "proots-unbounded-line fails due to invalid
hyperplanes."))

```

(λ -. *proots-unbounded-line* p st tt))
unfolding *proots-unbounded-line-def* **using** *proots-unbounded-line* **by** *auto*

2.11 Checking if there a polynomial root on a closed segment

definition *no-proots-line::complex poly \Rightarrow complex \Rightarrow complex \Rightarrow bool* **where**
no-proots-line p st tt = (*proots-within* p (*closed-segment* st tt) = { })

lemma *no-proots-line-code[code]: no-proots-line* p st tt = (*if* *poly* p st $\neq 0$ \wedge *poly* p tt $\neq 0$ *then*

(*let* pc = *pcompose* p [st , $tt - st$];
 pR = *map-poly* *Re* pc ;
 pI = *map-poly* *Im* pc ;
 g = *gcd* pR pI
in if *changes-itv-smods* 0 1 g (*pderiv* g) = 0 *then* *True* *else* *False*)

else False)

(*is* $?L$ = $?R$)

proof (*cases* *poly* p st $\neq 0$ \wedge *poly* p tt $\neq 0$)

case *False*

thus *?thesis* **unfolding** *no-proots-line-def* **by** *auto*

next

case *True*

then *have* *poly* p st $\neq 0$ *poly* p tt $\neq 0$ **by** *auto*

define pc pR pI g **where**

pc = *pcompose* p [st , $tt-st$] **and**
 pR = *map-poly* *Re* pc **and**
 pI = *map-poly* *Im* pc **and**
 g = *gcd* pR pI

have *poly-iff:poly* g $t=0$ \longleftrightarrow *poly* pc $t=0$ **for** t

proof –

have *poly* g $t=0$ \longleftrightarrow *poly* pR $t=0$ \wedge *poly* pI $t=0$

unfolding *g-def* **using** *poly-gcd-0-iff* **by** *auto*

also *have* ... \longleftrightarrow *poly* pc $t=0$

proof –

have *cpoly-of* pR pI = pc

unfolding *pc-def* *pR-def* *pI-def* **using** *cpoly-of-decompose* **by** *auto*

then *show* *?thesis* **using** *poly-cpoly-of-real-iff* **by** *blast*

qed

finally *show* *?thesis* **by** *auto*

qed

have $?R$ = (*changes-itv-smods* 0 1 g (*pderiv* g) = 0)

using *True* **unfolding** *pc-def* *g-def* *pI-def* *pR-def*

by (*auto simp add:Let-def*)

also *have* ... = (*card* { x . *poly* g $x=0$ \wedge $0 < x \wedge x < 1$ } = 0)

proof –

have *poly* g 0 $\neq 0$

using *poly-iff[of 0]* *True* **unfolding** *pc-def* **by** (*auto simp add:poly-pcompose*)

moreover *have* *poly* g 1 $\neq 0$

```

    using poly-iff[of 1] True unfolding pc-def by (auto simp add:poly-pcompose)
    ultimately show ?thesis using sturm-interval[of 0 1 g] by auto
qed
also have ... = ({x. poly g (of-real x) = 0 ∧ 0 < x ∧ x < 1} = {})
proof -
  have g≠0
  proof (rule ccontr)
    assume ¬ g ≠ 0
    then have poly pc 0 = 0
      using poly-iff[of 0] by auto
    then show False using True unfolding pc-def by (auto simp add:poly-pcompose)
  qed
  from poly-roots-finite[OF this] have finite {x. poly g x = 0 ∧ 0 < x ∧ x < 1}
    by auto
  then show ?thesis using card-eq-0-iff by auto
qed
also have ... = ?L
proof -
  have (∃ t. poly g (of-real t) = 0 ∧ 0 < t ∧ t < 1) ↔
    (∃ t::real. poly pc (of-real t) = 0 ∧ 0 < t ∧ t < 1)
    using poly-iff by auto
  also have ... ↔ (∃ x. x ∈ closed-segment st tt ∧ poly p x = 0)
  proof
    assume ∃ t. poly pc (complex-of-real t) = 0 ∧ 0 < t ∧ t < 1
    then obtain t where *:poly pc (of-real t) = 0 and 0 < t t < 1 by auto
    define x where x=poly [:st, tt - st:] t
    have x ∈ closed-segment st tt using ‹0 < t› ‹t < 1› unfolding x-def in-segment
      by (intro exI[where x=t], auto simp add: algebra-simps scaleR-conv-of-real)
    moreover have poly p x=0 using * unfolding pc-def x-def
      by (auto simp add:poly-pcompose)
    ultimately show ∃ x. x ∈ closed-segment st tt ∧ poly p x = 0 by auto
  next
    assume ∃ x. x ∈ closed-segment st tt ∧ poly p x = 0
    then obtain x where x ∈ closed-segment st tt poly p x = 0 by auto
    then obtain t::real where *:x = (1 - t) *R st + t *R tt and 0 ≤ t ≤ 1
      unfolding in-segment by auto
    then have x=poly [:st, tt - st:] t by (auto simp add: algebra-simps scaleR-conv-of-real)
    then have poly pc (complex-of-real t) = 0
      using ‹poly p x=0› unfolding pc-def by (auto simp add:poly-pcompose)
    moreover have t≠0 t≠1 using True * ‹poly p x=0› by auto
    then have 0 < t t < 1 using ‹0 ≤ t› ‹t ≤ 1› by auto
    ultimately show ∃ t. poly pc (complex-of-real t) = 0 ∧ 0 < t ∧ t < 1 by
auto
  qed
  finally show ?thesis
    unfolding no-roots-line-def proots-within-def
    by blast
qed
finally show ?thesis by simp

```

qed

2.12 Number of roots on a bounded open segment

definition *roots-line*:: *complex poly* \Rightarrow *complex* \Rightarrow *complex* \Rightarrow *nat* **where**
roots-line *p st tt* = *roots-count* *p* (*open-segment* *st tt*)

lemma *roots-line-commute*:

roots-line *p st tt* = *roots-line* *p tt st*

unfolding *roots-line-def* **by** (*simp* *add*: *open-segment-commute*)

lemma *roots-line-smods*:

assumes *poly* *p st* $\neq 0$ *poly* *p tt* $\neq 0$ *st* \neq *tt*

shows *roots-line* *p st tt* =

(*let* *pc* = *pcompose* *p* [*st*, *tt* - *st*];
pR = *map-poly* *Re* *pc*;
pI = *map-poly* *Im* *pc*;
g = *gcd* *pR* *pI*
in *nat* (*changes-itv-smods-ext* 0 1 *g* (*pderiv* *g*)))

(*is* $\neq ?R$)

proof -

have *p* $\neq 0$ **using** *assms*(2) *poly-0* **by** *blast*

define *pc pR pI g* **where**

pc = *pcompose* *p* [*st*, *tt* - *st*] **and**

pR = *map-poly* *Re* *pc* **and**

pI = *map-poly* *Im* *pc* **and**

g = *gcd* *pR* *pI*

have [*simp*]: *g* $\neq 0$ *pc* $\neq 0$

proof -

show *pc* $\neq 0$

by (*metis* *assms*(1) *coeff-pCons-0* *pCons-0-0* *pc-def* *pcompose-coeff-0*)

then have *pR* $\neq 0$ \vee *pI* $\neq 0$ **unfolding** *pR-def* *pI-def*

by (*metis* *cpoly-of-decompose* *map-poly-0*)

then show *g* $\neq 0$ **unfolding** *g-def* **by** *simp*

qed

have *order-eq*:*order* *t g* = *order* *t pc* **for** *t*

apply (*subst* *order-cpoly-gcd-eq*[*of* *pR* *pI*,*folded* *g-def*,*symmetric*])

subgoal using $\langle g \neq 0 \rangle$ **unfolding** *g-def* **by** *simp*

subgoal unfolding *pR-def* *pI-def* **by** (*simp* *add*:*cpoly-of-decompose*[*symmetric*])

done

have *poly-iff*:*poly* *g t* = 0 \longleftrightarrow *poly* *pc t* = 0 **for** *t*

using *order-eq* **by** (*simp* *add*: *order-root*)

have *poly* *g* 0 $\neq 0$ *poly* *g* 1 $\neq 0$

unfolding *poly-iff* *pc-def*

using *assms* **by** (*simp*-*all* *add*:*poly-pcompose*)

have $?R$ = *changes-itv-smods-ext* 0 1 *g* (*pderiv* *g*)

```

unfolding Let-def
apply (fold pc-def g-def pI-def pR-def)
using assms changes-itu-smods-ext-geq-0[OF - ⟨poly g 0≠0⟩ ⟨poly g 1≠0⟩]
by auto
also have ... = int (roots-count g {x. 0 < x ∧ x < 1})
apply (rule sturm-ext-interval[symmetric])
by simp fact+
also have ... = int (roots-count p (open-segment st tt))
proof -
define f where f = ( $\lambda x. \text{poly } [:st, tt - st:] (\text{complex-of-real } x)$ )
have  $x \in f^{-1} \{x. 0 < x \wedge x < 1\}$  if  $x \in \text{open-segment } st \text{ } tt$  for x
proof -
obtain u where  $u:u>0 \ u < 1 \ x = (1 - u) *_R st + u *_R tt$ 
using  $\langle x \in \text{open-segment } st \text{ } tt \rangle$  unfolding in-segment by auto
show ?thesis
apply (rule rev-image-eqI[where  $x=u$ ])
using u unfolding f-def
by (auto simp: algebra-simps scaleR-conv-of-real)
qed
moreover have  $x \in \text{open-segment } st \text{ } tt$  if  $x \in f^{-1} \{x. 0 < x \wedge x < 1\}$  for x
using that ⟨st≠tt⟩ in-segment f-def
by (auto simp: scaleR-conv-of-real algebra-simps)
ultimately have  $\text{open-segment } st \text{ } tt = f^{-1} \{x. 0 < x \wedge x < 1\}$ 
by auto
then have  $\text{roots-count } p (\text{open-segment } st \text{ } tt)$ 
=  $\text{roots-count } g \{x. 0 < x \wedge x < 1\}$ 
using roots-count-gcd-eq[OF ⟨st≠tt⟩ ⟨p≠0⟩,
folded pc-def pR-def pI-def g-def] unfolding f-def
by auto
then show ?thesis by auto
qed
also have ... = roots-line p st tt
unfolding roots-line-def by simp
finally show ?thesis by simp
qed

```

lemma *roots-line-code*[*code*]:

```

roots-line p st tt =
  (if poly p st ≠ 0 ∧ poly p tt ≠ 0 then
    (if st≠tt then
      (let pc = pcompose p [:st, tt - st];
       pR = map-poly Re pc;
       pI = map-poly Im pc;
       g = gcd pR pI
       in nat (changes-itu-smods-ext 0 1 g (pderiv g)))
    else 0)

```

else Code.abort (STR "rootsline does not handle vanishing endpoints for now")

```

      ( $\lambda$ -. proots-line p st tt) (is ?L = ?R)
proof (cases poly p st  $\neq 0 \wedge$  poly p tt  $\neq 0 \wedge$  st $\neq$ tt)
  case False
    moreover have ?thesis if st=tt p $\neq 0$ 
      using that unfolding proots-line-def by auto
    ultimately show ?thesis by fastforce
  next
    case True
    then show ?thesis using proots-line-smods by auto
qed

end

```

```

theory Count-Half-Plane imports
  Count-Line
begin

```

2.13 Polynomial roots on the upper half-plane

```

definition proots-upper :: complex poly  $\Rightarrow$  nat where
  proots-upper p = proots-count p {z. Im z > 0}

```

— Roots counted WITHOUT multiplicity

```

definition proots-upper-card :: complex poly  $\Rightarrow$  nat where
  proots-upper-card p = card (proots-within p {x. Im x > 0})

```

```

lemma Im-Ln-tendsto-at-top: (( $\lambda$ x. Im (Ln (Complex a x)))  $\longrightarrow$  pi/2 ) at-top

```

```

proof (cases a=0)
  case False
    define f where f=( $\lambda$ x. if a>0 then arctan (x/a) else arctan (x/a) + pi)
    define g where g=( $\lambda$ x. Im (Ln (Complex a x)))
    have (f  $\longrightarrow$  pi / 2) at-top
    proof (cases a>0)
      case True
        then have (f  $\longrightarrow$  pi / 2) at-top  $\longleftrightarrow$  (( $\lambda$ x. arctan (x * inverse a))  $\longrightarrow$  pi
          / 2) at-top
          unfolding f-def field-class.field-divide-inverse by auto
          also have ...  $\longleftrightarrow$  (arctan  $\longrightarrow$  pi / 2) at-top
          apply (subst filterlim-at-top-linear-iff[of inverse a arctan 0 nhds (pi/2),simplified])
          using True by auto
          also have ... using tendsto-arctan-at-top .
          finally show ?thesis .
        next
          case False
          then have (f  $\longrightarrow$  pi / 2) at-top  $\longleftrightarrow$  (( $\lambda$ x. arctan (x * inverse a) + pi)  $\longrightarrow$ 
            pi / 2) at-top
            unfolding f-def field-class.field-divide-inverse by auto
            also have ...  $\longleftrightarrow$  (( $\lambda$ x. arctan (x * inverse a))  $\longrightarrow$  - pi / 2) at-top
            apply (subst tendsto-add-const-iff[of -pi,symmetric])

```

```

    by auto
  also have ...  $\longleftrightarrow$  (arctan  $\longrightarrow$   $-\pi / 2$ ) at-bot
  apply (subst filterlim-at-top-linear-iff[of inverse a arctan 0,simplified])
  using False  $\langle a \neq 0 \rangle$  by auto
  also have ... using tendsto-arctan-at-bot by simp
  finally show ?thesis .
qed
moreover have  $\forall_F x$  in at-top.  $f x = g x$ 
  unfolding f-def g-def using  $\langle a \neq 0 \rangle$ 
  apply (subst Im-Ln-eq)
  subgoal for x using Complex-eq-0 by blast
  subgoal unfolding eventually-at-top-linorder by auto
  done
ultimately show ?thesis
  using tendsto-cong[of f g at-top] unfolding g-def by auto
next
case True
show ?thesis
  apply (rule tendsto-eventually)
  apply (rule eventually-at-top-linorderI[of 1])
  using True by (subst Im-Ln-eq,auto simp add:Complex-eq-0)
qed

lemma Im-Ln-tendsto-at-bot: (( $\lambda x.$  Im (Ln (Complex a x)))  $\longrightarrow$   $-\pi / 2$ ) at-bot

proof (cases a=0)
case False
define f where f=( $\lambda x.$  if  $a > 0$  then arctan (x/a) else arctan (x/a) -  $\pi$ )
define g where g=( $\lambda x.$  Im (Ln (Complex a x)))
have (f  $\longrightarrow$   $-\pi / 2$ ) at-bot
proof (cases a>0)
case True
then have (f  $\longrightarrow$   $-\pi / 2$ ) at-bot  $\longleftrightarrow$  (( $\lambda x.$  arctan (x * inverse a))  $\longrightarrow$ 
 $-\pi / 2$ ) at-bot
  unfolding f-def field-class.field-divide-inverse by auto
  also have ...  $\longleftrightarrow$  (arctan  $\longrightarrow$   $-\pi / 2$ ) at-bot
  apply (subst filterlim-at-bot-linear-iff[of inverse a arctan 0,simplified])
  using True by auto
  also have ... using tendsto-arctan-at-bot by simp
  finally show ?thesis .
next
case False
then have (f  $\longrightarrow$   $-\pi / 2$ ) at-bot  $\longleftrightarrow$  (( $\lambda x.$  arctan (x * inverse a) -  $\pi$ )
 $\longrightarrow$   $-\pi / 2$ ) at-bot
  unfolding f-def field-class.field-divide-inverse by auto
  also have ...  $\longleftrightarrow$  (( $\lambda x.$  arctan (x * inverse a))  $\longrightarrow$   $\pi / 2$ ) at-bot
  apply (subst tendsto-add-const-iff[of  $\pi$ ,symmetric])
  by auto
  also have ...  $\longleftrightarrow$  (arctan  $\longrightarrow$   $\pi / 2$ ) at-top

```

```

    apply (subst filterlim-at-bot-linear-iff[of inverse a arctan 0,simplified])
    using False ⟨a≠0⟩ by auto
    also have ... using tendsto-arctan-at-top by simp
    finally show ?thesis .
qed
moreover have ∀F x in at-bot. f x = g x
  unfolding f-def g-def using ⟨a≠0⟩
  apply (subst Im-Ln-eq)
  subgoal for x using Complex-eq-0 by blast
  subgoal unfolding eventually-at-bot-linorder by (auto intro:exI[where x=-1])
  done
ultimately show ?thesis
  using tendsto-cong[of f g at-bot] unfolding g-def by auto
next
case True
show ?thesis
  apply (rule tendsto-eventually)
  apply (rule eventually-at-bot-linorderI[of -1])
  using True by (subst Im-Ln-eq,auto simp add:Complex-eq-0)
qed

lemma Re-winding-number-tendsto-part-circlepath:
  shows ((λr. Re (winding-number (part-circlepath z0 r 0 pi ) a)) → 1/2 )
  at-top
proof (cases Im z0 ≤ Im a)
case True
  define g1 where g1=(λr. part-circlepath z0 r 0 pi)
  define g2 where g2=(λr. part-circlepath z0 r pi (2*pi))
  define f1 where f1=(λr. Re (winding-number (g1 r ) a))
  define f2 where f2=(λr. Re (winding-number (g2 r ) a))
  have (f2 → 1/2 ) at-top
  proof -
    define h1 where h1 = (λr. Im (Ln (Complex ( Im a - Im z0 ) (Re z0 - Re a
+ r))))
    define h2 where h2 = (λr. Im (Ln (Complex ( Im a - Im z0 ) (Re z0 - Re a
- r))))
    have ∀F x in at-top. f2 x = (h1 x - h2 x) / (2 * pi)
    proof (rule eventually-at-top-linorderI[of cmod (a-z0) + 1])
      fix r assume asm:r ≥ cmod (a - z0) + 1
      have Im p ≤ Im a when p∈path-image (g2 r) for p
      proof -
        obtain t where p-def:p=z0 + of-real r * exp (i * of-real t) and pi ≤ t ≤ 2*pi
          using ⟨p∈path-image (g2 r)⟩
          unfolding g2-def path-image-part-circlepath[of pi 2*pi,simplified]
          by auto
        then have Im p=Im z0 + sin t * r by (auto simp add:Im-exp)
        also have ... ≤ Im z0
      proof -
        have sin t ≤ 0 using ⟨pi ≤ t⟩ ⟨t ≤ 2*pi⟩ sin-le-zero by fastforce

```

```

    moreover have  $r \geq 0$ 
  using asm by (metis add.inverse-inverse add.left-neutral add.uminus-conv-diff
    diff-ge-0-iff-ge norm-ge-zero order-trans zero-le-one)
    ultimately have  $\sin t * r \leq 0$  using mult-le-0-iff by blast
    then show ?thesis by auto
  qed
  also have  $\dots \leq \text{Im } a$  using True .
  finally show ?thesis .
  qed
  moreover have valid-path (g2 r) unfolding g2-def by auto
  moreover have  $a \notin \text{path-image } (g2 \ r)$ 
    unfolding g2-def
    apply (rule not-on-circlepathI)
    using asm by auto
  moreover have [symmetric]: $\text{Im } (Ln \ (i * \text{pathfinish } (g2 \ r) - i * a)) = h1 \ r$ 
    unfolding h1-def g2-def
    apply (simp only: pathfinish-pathstart-partcirclepath-simps)
    apply (subst (4 10) complex-eq)
    by (auto simp add: algebra-simps Complex-eq)
  moreover have [symmetric]: $\text{Im } (Ln \ (i * \text{pathstart } (g2 \ r) - i * a)) = h2 \ r$ 
    unfolding h2-def g2-def
    apply (simp only: pathfinish-pathstart-partcirclepath-simps)
    apply (subst (4 10) complex-eq)
    by (auto simp add: algebra-simps Complex-eq)
  ultimately show  $f2 \ r = (h1 \ r - h2 \ r) / (2 * \pi)$ 
    unfolding f2-def
    apply (subst Re-winding-number-half-lower)
    by (auto simp add: exp-Euler algebra-simps)
  qed
  moreover have  $((\lambda x. (h1 \ x - h2 \ x) / (2 * \pi)) \longrightarrow 1/2) \text{ at-top}$ 
  proof -
    have  $(h1 \longrightarrow \pi/2) \text{ at-top}$ 
      unfolding h1-def
    apply (subst filterlim-at-top-linear-iff[of  $1 - \text{Re } a - \text{Re } z0$  ,simplified,symmetric])

      using Im-Ln-tendsto-at-top by (simp del: Complex-eq)
    moreover have  $(h2 \longrightarrow -\pi/2) \text{ at-top}$ 
      unfolding h2-def
    apply (subst filterlim-at-bot-linear-iff[of  $-1 - \text{Re } a + \text{Re } z0$  ,simplified,symmetric])

      using Im-Ln-tendsto-at-bot by (simp del: Complex-eq)
    ultimately have  $((\lambda x. h1 \ x - h2 \ x) \longrightarrow \pi) \text{ at-top}$ 
      by (auto intro: tendsto-eq-intros)
    then show ?thesis
      by (auto intro: tendsto-eq-intros)
  qed
  ultimately show ?thesis by (auto dest: tendsto-cong)
  qed
  moreover have  $\forall_F \ r \text{ in at-top. } f2 \ r = 1 - f1 \ r$ 

```

```

proof (rule eventually-at-top-linorderI[of cmod (a-z0) + 1])
  fix r assume asm:r ≥ cmod (a - z0) + 1
  have f1 r + f2 r = Re(winding-number (g1 r +++ g2 r) a)
    unfolding f1-def f2-def g1-def g2-def
    apply (subst winding-number-join)
    using asm by (auto intro!:not-on-circlepathI)
  also have ... = Re(winding-number (circlepath z0 r) a)
  proof -
    have g1 r +++ g2 r = circlepath z0 r
      unfolding circlepath-def g1-def g2-def joinpaths-def part-circlepath-def
linepath-def
    by (auto simp add:field-simps)
    then show ?thesis by auto
  qed
  also have ... = 1
  proof -
    have winding-number (circlepath z0 r) a = 1
      apply (rule winding-number-circlepath)
      using asm by auto
    then show ?thesis by auto
  qed
  finally have f1 r+f2 r=1 .
  then show f2 r = 1 - f1 r by auto
  qed
  ultimately have ((λr. 1 - f1 r) → 1/2) at-top
    using tendsto-cong[of f2 λr. 1 - f1 r at-top] by auto
  then have (f1 → 1/2) at-top
    apply (rule-tac tendsto-minus-cancel)
    apply (subst tendsto-add-const-iff[of 1,symmetric])
    by auto
  then show ?thesis unfolding f1-def g1-def by auto
next
  case False
  define g where g=(λr. part-circlepath z0 r 0 pi)
  define f where f=(λr. Re (winding-number (g r) a))
  have (f → 1/2) at-top
  proof -
    define h1 where h1 = (λr. Im (Ln (Complex ( Im z0-Im a) (Re a - Re z0
+ r))))
    define h2 where h2= (λr. Im (Ln (Complex ( Im z0 -Im a) (Re a - Re
z0 - r))))
    have ∀F x in at-top. f x = (h1 x - h2 x) / (2 * pi)
  proof (rule eventually-at-top-linorderI[of cmod (a-z0) + 1])
    fix r assume asm:r ≥ cmod (a - z0) + 1
    have Im p ≥ Im a when p∈path-image (g r) for p
    proof -
      obtain t where p-def:p=z0 + of-real r * exp (i * of-real t) and 0≤t t≤pi
      using ⟨p∈path-image (g r)⟩
      unfolding g-def path-image-part-circlepath[of 0 pi,simplified]

```

```

    by auto
  then have  $Im\ p = Im\ z0 + sin\ t * r$  by (auto simp add:Im-exp)
  moreover have  $sin\ t * r \geq 0$ 
  proof -
    have  $sin\ t \geq 0$  using  $\langle 0 \leq t \rangle \langle t \leq pi \rangle$  sin-ge-zero by fastforce
    moreover have  $r \geq 0$ 
  using asm by (metis add.inverse-inverse add.left-neutral add-uminus-conv-diff
    diff-ge-0-iff-ge norm-ge-zero order-trans zero-le-one)
    ultimately have  $sin\ t * r \geq 0$  by simp
    then show ?thesis by auto
  qed
  ultimately show ?thesis using False by auto
qed
moreover have valid-path (g r) unfolding g-def by auto
moreover have  $a \notin path\ image\ (g\ r)$ 
  unfolding g-def
  apply (rule not-on-circlepathI)
  using asm by auto
moreover have [symmetric]: $Im\ (Ln\ (i * a - i * pathfinish\ (g\ r))) = h1\ r$ 
  unfolding h1-def g-def
  apply (simp only:pathfinish-pathstart-partcirclepath-simps)
  apply (subst (4 9) complex-eq)
  by (auto simp add:algebra-simps Complex-eq)
moreover have [symmetric]: $Im\ (Ln\ (i * a - i * pathstart\ (g\ r))) = h2\ r$ 
  unfolding h2-def g-def
  apply (simp only:pathfinish-pathstart-partcirclepath-simps)
  apply (subst (4 9) complex-eq)
  by (auto simp add:algebra-simps Complex-eq)
ultimately show  $f\ r = (h1\ r - h2\ r) / (2 * pi)$ 
  unfolding f-def
  apply (subst Re-winding-number-half-upper)
  by (auto simp add:exp-Euler algebra-simps)
qed
moreover have  $((\lambda x. (h1\ x - h2\ x) / (2 * pi)) \longrightarrow 1/2)$  at-top
proof -
  have  $(h1 \longrightarrow pi/2)$  at-top
  unfolding h1-def
  apply (subst filterlim-at-top-linear-iff[of 1 - Re a + Re z0 ,simplified,symmetric])

    using Im-Ln-tendsto-at-top by (simp del:Complex-eq)
  moreover have  $(h2 \longrightarrow - pi/2)$  at-top
  unfolding h2-def
  apply (subst filterlim-at-bot-linear-iff[of - 1 - Re a - Re z0 ,simplified,symmetric])

    using Im-Ln-tendsto-at-bot by (simp del:Complex-eq)
  ultimately have  $((\lambda x. h1\ x - h2\ x) \longrightarrow pi)$  at-top
  by (auto intro:tendsto-eq-intros)
  then show ?thesis
  by (auto intro:tendsto-eq-intros)

```

```

qed
ultimately show ?thesis by (auto dest:tendsto-cong)
qed
then show ?thesis unfolding f-def g-def by auto
qed

lemma not-image-at-top-poly-part-circlepath:
  assumes degree  $p > 0$ 
  shows  $\forall_F r$  in at-top.  $b \notin \text{path-image } (\text{poly } p \text{ o part-circlepath } z0 \ r \ st \ tt)$ 
proof -
  have finite (proots (p-[b:]))
  apply (rule finite-proots)
  using assms by auto
  from finite-ball-include[OF this]
  obtain  $R::\text{real}$  where  $R > 0$  and  $R\text{-ball:proots } (p-[b:]) \subseteq \text{ball } z0 \ R$  by auto
  show ?thesis
  proof (rule eventually-at-top-linorderI[of R])
    fix  $r$  assume  $r \geq R$ 
    show  $b \notin \text{path-image } (\text{poly } p \text{ o part-circlepath } z0 \ r \ st \ tt)$ 
      unfolding path-image-compose
    proof clarify
      fix  $x$  assume  $asm:b = \text{poly } p \ x \ x \in \text{path-image } (\text{part-circlepath } z0 \ r \ st \ tt)$ 
      then have  $x \in \text{proots } (p-[b:])$  unfolding proots-def by auto
      then have  $x \in \text{ball } z0 \ r$  using  $R\text{-ball } \langle r \geq R \rangle$  by auto
      then have  $cmod (x - z0) < r$ 
        by (simp add: dist-commute dist-norm)
      moreover have  $cmod (x - z0) = r$ 
        using  $asm(2)$  in-path-image-part-circlepath  $\langle R > 0 \rangle \ \langle r \geq R \rangle$  by auto
      ultimately show False by auto
    qed
  qed
qed

```

```

lemma not-image-poly-part-circlepath:
  assumes degree  $p > 0$ 
  shows  $\exists r > 0$ .  $b \notin \text{path-image } (\text{poly } p \text{ o part-circlepath } z0 \ r \ st \ tt)$ 
proof -
  have finite (proots (p-[b:]))
  apply (rule finite-proots)
  using assms by auto
  from finite-ball-include[OF this]
  obtain  $r::\text{real}$  where  $r > 0$  and  $r\text{-ball:proots } (p-[b:]) \subseteq \text{ball } z0 \ r$  by auto
  have  $b \notin \text{path-image } (\text{poly } p \text{ o part-circlepath } z0 \ r \ st \ tt)$ 
    unfolding path-image-compose
  proof clarify
    fix  $x$  assume  $asm:b = \text{poly } p \ x \ x \in \text{path-image } (\text{part-circlepath } z0 \ r \ st \ tt)$ 
    then have  $x \in \text{proots } (p-[b:])$  unfolding proots-def by auto
    then have  $x \in \text{ball } z0 \ r$  using  $r\text{-ball}$  by auto
    then have  $cmod (x - z0) < r$ 

```

```

    by (simp add: dist-commute dist-norm)
  moreover have cmod (x - z0) = r
    using asm(2) in-path-image-part-circlepath ⟨r>0 by auto
  ultimately show False by auto
qed
then show ?thesis using ⟨r>0 by blast
qed

lemma Re-winding-number-poly-part-circlepath:
  assumes degree p > 0
  shows ((λr. Re (winding-number (poly p o part-circlepath z0 r 0 pi) 0)) →
  degree p/2 ) at-top
  using assms
  proof (induct rule:poly-root-induct-alt)
    case 0
    then show ?case by auto
  next
    case (no-roots p)
    then have False
      using Fundamental-Theorem-Algebra.fundamental-theorem-of-algebra constant-degree
      neq0-conv
      by blast
    then show ?case by auto
  next
    case (root a p)
    define g where g = (λr. part-circlepath z0 r 0 pi)
    define q where q = [- a, 1:] * p
    define w where w = (λr. winding-number (poly q o g r) 0)
    have ?case when degree p = 0
    proof -
      obtain pc where pc-def: p = [:pc:] using ⟨degree p = 0⟩ degree-eq-zeroE by blast
      then have pc ≠ 0 using root(2) by auto
      have ∀ F r in at-top. Re (w r) = Re (winding-number (g r) a)
      proof (rule eventually-at-top-linorderI[of cmod ((pc * a) / pc - z0) + 1])
        fix r::real assume asm:cmod ((pc * a) / pc - z0) + 1 ≤ r
        have w r = winding-number ((λx. pc*x - pc*a) o (g r)) 0
          unfolding w-def pc-def g-def q-def
          apply auto
        by (metis (no-types, opaque-lifting) add.right-neutral mult.commute mult-zero-right

          poly-0 poly-pCons uminus-add-conv-diff)
      also have ... = winding-number (g r) a
      apply (subst winding-number-comp-linear[where b = -pc*a, simplified])
      subgoal using ⟨pc ≠ 0⟩ .
      subgoal unfolding g-def by auto
      subgoal unfolding g-def
        apply (rule not-on-circlepathI)
        using asm by auto
      subgoal using ⟨pc ≠ 0⟩ by (auto simp add:field-simps)
    end
  end
end

```

done
finally have $w r = \text{winding-number } (g r) a$.
then show $\text{Re } (w r) = \text{Re } (\text{winding-number } (g r) a)$ **by simp**
qed
moreover have $((\lambda r. \text{Re } (\text{winding-number } (g r) a)) \longrightarrow 1/2)$ *at-top*
using *Re-winding-number-tendsto-part-circlepath* **unfolding** *g-def* **by auto**
ultimately have $((\lambda r. \text{Re } (w r)) \longrightarrow 1/2)$ *at-top*
by *(auto dest!:tendsto-cong)*
moreover have $\text{degree } ([: - a, 1:] * p) = 1$ **unfolding** *pc-def* **using** $\langle pc \neq 0 \rangle$
by auto
ultimately show *?thesis* **unfolding** *w-def g-def comp-def q-def* **by simp**
qed
moreover have *?case* **when** $\text{degree } p > 0$
proof –
have $\forall_F r$ *in at-top*. $0 \notin \text{path-image } (\text{poly } q \circ g r)$
unfolding *g-def*
apply *(rule not-image-at-top-poly-part-circlepath)*
unfolding *q-def* **using** *root.premis* **by blast**
then have $\forall_F r$ *in at-top*. $\text{Re } (w r) = \text{Re } (\text{winding-number } (g r) a)$
 $+ \text{Re } (\text{winding-number } (\text{poly } p \circ g r) 0)$
proof *(rule eventually-mono)*
fix r **assume** $asm: 0 \notin \text{path-image } (\text{poly } q \circ g r)$
define cc **where** $cc = 1 / (\text{of-real } (2 * \pi) * i)$
define pf **where** $pf = (\lambda w. \text{deriv } (\text{poly } p) w / \text{poly } p w)$
define af **where** $af = (\lambda w. 1 / (w - a))$
have $w r = cc * \text{contour-integral } (g r) (\lambda w. \text{deriv } (\text{poly } q) w / \text{poly } q w)$
unfolding *w-def*
apply *(subst winding-number-comp[of UNIV,simplified])*
using asm **unfolding** *g-def cc-def* **by auto**
also have $\dots = cc * \text{contour-integral } (g r) (\lambda w. \text{deriv } (\text{poly } p) w / \text{poly } p w$
 $+ 1 / (w - a))$
proof –
have $\text{contour-integral } (g r) (\lambda w. \text{deriv } (\text{poly } q) w / \text{poly } q w)$
 $= \text{contour-integral } (g r) (\lambda w. \text{deriv } (\text{poly } p) w / \text{poly } p w + 1 / (w - a))$
proof *(rule contour-integral-eq)*
fix x **assume** $x \in \text{path-image } (g r)$
have $\text{deriv } (\text{poly } q) x = \text{deriv } (\text{poly } p) x * (x - a) + \text{poly } p x$
proof –
have $\text{poly } q = (\lambda x. (x - a) * \text{poly } p x)$
apply *(rule ext)*
unfolding *q-def* **by** *(auto simp add: algebra-simps)*
then show *?thesis*
apply *simp*
apply *(subst deriv-mult[of $\lambda x. x - a - \text{poly } p$])*
by *(auto intro: derivative-intros)*
qed
moreover have $\text{poly } p x \neq 0 \wedge x - a \neq 0$
proof *(rule ccontr)*
assume $\neg (\text{poly } p x \neq 0 \wedge x - a \neq 0)$

```

    then have  $\text{poly } q \ x=0$  unfolding  $q\text{-def}$  by auto
    then have  $0 \in \text{path-image } (g \ r)$ 
      using  $\langle x \in \text{path-image } (g \ r) \rangle$  by auto
    then show False using  $\langle 0 \notin \text{path-image } (\text{poly } q \circ g \ r) \rangle$ 
      unfolding  $\text{path-image-compose}$  by auto
  qed
  ultimately show  $\text{deriv } (\text{poly } q) \ x / \text{poly } q \ x = \text{deriv } (\text{poly } p) \ x / \text{poly } p \ x$ 
+  $1 / (x - a)$ 
    unfolding  $q\text{-def}$  by (auto simp add:field-simps)
  qed
  then show ?thesis by auto
qed
also have  $\dots = cc * \text{contour-integral } (g \ r) \ (\lambda w. \text{deriv } (\text{poly } p) \ w / \text{poly } p \ w)$ 
+  $cc * \text{contour-integral } (g \ r) \ (\lambda w. 1/(w-a))$ 
proof (subst contour-integral-add)
  have  $\text{continuous-on } (\text{path-image } (g \ r)) \ (\lambda w. \text{deriv } (\text{poly } p) \ w)$ 
    unfolding  $\text{deriv-pderiv}$  by (intro continuous-intros)
  moreover have  $\forall w \in \text{path-image } (g \ r). \text{poly } p \ w \neq 0$ 
    using asm unfolding  $q\text{-def}$   $\text{path-image-compose}$  by auto
  ultimately show  $(\lambda w. \text{deriv } (\text{poly } p) \ w / \text{poly } p \ w)$   $\text{contour-integrable-on } g$ 
r
    unfolding  $g\text{-def}$ 
      by (auto intro!: contour-integrable-continuous-part-circlepath continuous-intros)
    show  $(\lambda w. 1 / (w - a))$   $\text{contour-integrable-on } g \ r$ 
      apply (rule contour-integrable-inversediff)
      subgoal unfolding  $g\text{-def}$  by auto
      subgoal using asm unfolding  $q\text{-def}$   $\text{path-image-compose}$  by auto
    done
  qed (auto simp add:algebra-simps)
  also have  $\dots = \text{winding-number } (g \ r) \ a + \text{winding-number } (\text{poly } p \circ g \ r) \ 0$ 
proof -
    have  $\text{winding-number } (\text{poly } p \circ g \ r) \ 0$ 
      =  $cc * \text{contour-integral } (g \ r) \ (\lambda w. \text{deriv } (\text{poly } p) \ w / \text{poly } p \ w)$ 
      apply (subst winding-number-comp[of UNIV,simplified])
      using  $\langle 0 \notin \text{path-image } (\text{poly } q \circ g \ r) \rangle$  unfolding  $\text{path-image-compose}$   $q\text{-def}$ 
g-def cc-def
      by auto
    moreover have  $\text{winding-number } (g \ r) \ a = cc * \text{contour-integral } (g \ r) \ (\lambda w.$ 
 $1/(w-a))$ 
      apply (subst winding-number-valid-path)
      using  $\langle 0 \notin \text{path-image } (\text{poly } q \circ g \ r) \rangle$  unfolding  $\text{path-image-compose}$   $q\text{-def}$ 
g-def cc-def
      by auto
    ultimately show ?thesis by auto
  qed
  finally show  $\text{Re } (w \ r) = \text{Re } (\text{winding-number } (g \ r) \ a) + \text{Re } (\text{winding-number}$ 
 $(\text{poly } p \circ g \ r) \ 0)$ 
    by auto

```

```

qed
moreover have (( $\lambda r$ .  $Re$  ( $winding\_number$  ( $g$   $r$ )  $a$ )
  +  $Re$  ( $winding\_number$  ( $poly$   $p \circ g$   $r$ )  $0$ ))  $\longrightarrow$   $degree$   $q / 2$ ) at-top
proof -
  have (( $\lambda r$ .  $Re$  ( $winding\_number$  ( $g$   $r$ )  $a$ ))  $\longrightarrow$   $1 / 2$ ) at-top
    unfolding  $g$ -def by ( $rule$   $Re$ - $winding\_number$ - $tendsto$ - $part$ - $circlepath$ )
    moreover have (( $\lambda r$ .  $Re$  ( $winding\_number$  ( $poly$   $p \circ g$   $r$ )  $0$ ))  $\longrightarrow$   $degree$   $p$ 
/  $2$ ) at-top
      unfolding  $g$ -def by ( $rule$   $root(1)$ [ $OF$   $that$ ])
      moreover have  $degree$   $q = degree$   $p + 1$ 
        unfolding  $q$ -def
        apply ( $subst$   $degree$ - $mult$ - $eq$ )
        using  $that$  by  $auto$ 
      ultimately show  $?thesis$ 
        by ( $simp$   $add$ :  $tendsto$ - $add$   $add$ - $divide$ - $distrib$ )
    qed
  ultimately have (( $\lambda r$ .  $Re$  ( $w$   $r$ ))  $\longrightarrow$   $degree$   $q/2$ ) at-top
    by ( $auto$   $dest!$ : $tendsto$ - $cong$ )
  then show  $?thesis$  unfolding  $w$ -def  $q$ -def  $g$ -def by  $blast$ 
qed
ultimately show  $?case$  by  $blast$ 
qed

```

```

lemma  $Re$ - $winding\_number$ - $poly$ - $linepth$ :
  fixes  $pp::complex$   $poly$ 
  defines  $g \equiv (\lambda r$ .  $poly$   $pp \circ linepath$  ( $-r$ ) ( $of$ - $real$   $r$ ))
  assumes  $lead$ - $coeff$   $pp=1$  and  $no$ - $real$ - $zero$ : $\forall x \in roots$   $pp$ .  $Im$   $x \neq 0$ 
  shows (( $\lambda r$ .  $2 * Re$  ( $winding\_number$  ( $g$   $r$ )  $0$ ) +  $cindex$ - $pathE$  ( $g$   $r$ )  $0$ )  $\longrightarrow$   $0$ 
) at-top
proof -
  define  $p$  where  $p=map$ - $poly$   $Re$   $pp$ 
  define  $q$  where  $q=map$ - $poly$   $Im$   $pp$ 
  define  $f$  where  $f=(\lambda t$ .  $poly$   $q$   $t / poly$   $p$   $t$ )
  have  $sgnx$ - $top$ : $sgnx$  ( $poly$   $p$ ) at-top =  $1$ 
    unfolding  $sgnx$ - $poly$ - $at$ - $top$   $sgn$ - $pos$ - $inf$ - $def$   $p$ -def using  $\langle lead$ - $coeff$   $pp=1 \rangle$ 
    by ( $subst$   $lead$ - $coeff$ - $map$ - $poly$ - $nz$ , $auto$ )
  have  $not$ - $g$ - $image$ : $0 \notin path$ - $image$  ( $g$   $r$ ) for  $r$ 
  proof ( $rule$   $ccontr$ )
    assume  $\neg 0 \notin path$ - $image$  ( $g$   $r$ )
    then obtain  $x$  where  $poly$   $pp$   $x=0$   $x \in closed$ - $segment$  ( $-$   $of$ - $real$   $r$ ) ( $of$ - $real$   $r$ )
      unfolding  $g$ -def  $path$ - $image$ - $compose$   $of$ - $real$ - $linepath$  by  $auto$ 
    then have  $Im$   $x=0$   $x \in roots$   $pp$ 
      using  $closed$ - $segment$ - $imp$ - $Re$ - $Im(2)$  unfolding  $roots$ -def by  $fastforce+$ 
    then show  $False$  using  $\langle \forall x \in roots$   $pp$ .  $Im$   $x \neq 0 \rangle$  by  $auto$ 
  qed
  have  $arctan$ - $f$ - $tendsto$ :(( $\lambda r$ . ( $arctan$  ( $f$   $r$ ) -  $arctan$  ( $f$  ( $-r$ ))) /  $\pi$ )  $\longrightarrow$   $0$ )
at-top
  proof ( $cases$   $degree$   $p > 0$ )
    case  $True$ 

```

```

have degree p > degree q
proof -
  have degree p = degree pp
    unfolding p-def using ‹lead-coeff pp = 1›
    by (auto intro: map-poly-degree-eq)
  moreover then have degree q < degree pp
    unfolding q-def using ‹lead-coeff pp = 1› True
    by (auto intro!: map-poly-degree-less)
  ultimately show ?thesis by auto
qed
then have (f ⟶ 0) at-infinity
  unfolding f-def using poly-divide-tendsto-0-at-infinity by auto
then have 0: (f ⟶ 0) at-bot (f ⟶ 0) at-top
  by (auto elim!: filterlim-mono simp add: at-top-le-at-infinity at-bot-le-at-infinity)
have ((λr. arctan (f r)) ⟶ 0) at-top
  by (auto intro: 0 tendsto-eq-intros)
moreover have ((λr. arctan (f (-r))) ⟶ 0) at-top
  using 0 filterlim-at-bot-mirror tendsto-arctan by fastforce
ultimately show ?thesis
  by (auto intro: tendsto-eq-intros)
next
case False
obtain c where f = (λr. c)
proof -
  have degree p = 0 using False by auto
  moreover have degree q ≤ degree p
  proof -
    have degree p = degree pp
      unfolding p-def using ‹lead-coeff pp = 1›
      by (auto intro: map-poly-degree-eq)
    moreover have degree q ≤ degree pp
      by (simp add: degree-map-poly-le q-def)
    ultimately show ?thesis by auto
  qed
  ultimately have degree q = 0 by simp
  then obtain pa qa where p = [:pa:] q = [:qa:]
    using ‹degree p = 0› by (meson degree-eq-zeroE)
  then show ?thesis using that unfolding f-def by auto
qed
then show ?thesis by auto
qed
have [simp]: valid-path (g r) path (g r) finite-ReZ-segments (g r) 0 for r
proof -
  show valid-path (g r) unfolding g-def
    apply (rule valid-path-compose-holomorphic[where S = UNIV])
    by (auto simp add: of-real-linepath)
  then show path (g r) using valid-path-imp-path by auto
  show finite-ReZ-segments (g r) 0
    unfolding g-def of-real-linepath using finite-ReZ-segments-poly-linepath by

```

```

simp
qed
have g-f-eq:Im (g r t) / Re (g r t) = (f o (λx. 2*r*x - r)) t for r t
proof -
  have Im (g r t) / Re (g r t) = Im ((poly pp o of-real o (λx. 2*r*x - r)) t)
    / Re ((poly pp o of-real o (λx. 2*r*x - r)) t)
  unfolding g-def linepath-def comp-def
  by (auto simp add:algebra-simps)
  also have ... = (f o (λx. 2*r*x - r)) t
  unfolding comp-def
  by (simp only:Im-poly-of-real diff-0-right Re-poly-of-real f-def q-def p-def)
  finally show ?thesis .
qed

have ?thesis when proots p={ }
proof -
  have ∀ Fr in at-top. 2 * Re (winding-number (g r) 0) + cindex-pathE (g r) 0
    = (arctan (f r) - arctan (f (-r))) / pi
  proof (rule eventually-at-top-linorderI[of 1])
    fix r::real assume r ≥ 1
    have image-pos:∀ p ∈ path-image (g r). 0 < Re p
    proof (rule ccontr)
      assume ¬ (∀ p ∈ path-image (g r). 0 < Re p)
      then obtain t where poly p t ≤ 0
      unfolding g-def path-image-compose of-real-linepath p-def
      using Re-poly-of-real
      apply (simp add:closed-segment-def)
      by (metis not-less of-real-def real-vector.scale-scale scaleR-left-diff-distrib)

    moreover have False when poly p t < 0
    proof -
      have sgnx (poly p) (at-right t) = -1
      using sgnx-poly-nz that by auto
      then obtain x where x > t poly p x = 0
      using sgnx-at-top-IVT[of p t] sgnx-top by auto
      then have x ∈ proots p unfolding proots-def by auto
      then show False using ⟨proots p = { }⟩ by auto
    qed

    moreover have False when poly p t = 0
    using ⟨proots p = { }⟩ that unfolding proots-def by auto
    ultimately show False by linarith
  qed

  have Re (winding-number (g r) 0) = (Im (Ln (pathfinish (g r))) - Im (Ln
(pathstart (g r))))
    / (2 * pi)
  apply (rule Re-winding-number-half-right[of g r 0,simplified])
  subgoal using image-pos by auto
  subgoal by (auto simp add:not-g-image)
  done

```

```

also have ... = (arctan (f r) - arctan (f (-r)))/(2*pi)
proof -
  have Im (Ln (pathfinish (g r))) = arctan (f r)
proof -
  have Re (pathfinish (g r)) > 0
  by (auto intro: image-pos[rule-format])
  then have Im (Ln (pathfinish (g r)))
    = arctan (Im (pathfinish (g r)) / Re (pathfinish (g r)))
  by (subst Im-Ln-eq,auto)
  also have ... = arctan (f r)
  unfolding path-defs by (subst g-f-eq,auto)
  finally show ?thesis .
qed
moreover have Im (Ln (pathstart (g r))) = arctan (f (-r))
proof -
  have Re (pathstart (g r)) > 0
  by (auto intro: image-pos[rule-format])
  then have Im (Ln (pathstart (g r)))
    = arctan (Im (pathstart (g r)) / Re (pathstart (g r)))
  by (subst Im-Ln-eq,auto)
  also have ... = arctan (f (-r))
  unfolding path-defs by (subst g-f-eq,auto)
  finally show ?thesis .
qed
ultimately show ?thesis by auto
qed
finally have Re (winding-number (g r) 0) = (arctan (f r) - arctan (f
(-r)))/(2*pi) .
moreover have cindex-pathE (g r) 0 = 0
proof -
  have cindex-pathE (g r) 0 = cindex-pathE (poly pp o of-real o (λx. 2*r*x
- r)) 0
  unfolding g-def linepath-def comp-def
  by (auto simp add: algebra-simps)
  also have ... = cindexE 0 1 (f o (λx. 2*r*x - r))
  unfolding cindex-pathE-def comp-def
  by (simp only: Im-poly-of-real diff-0-right Re-poly-of-real f-def q-def p-def)
  also have ... = cindexE (-r) r f
  apply (subst cindexE-linear-comp[of 2*r 0 1 - -r,simplified])
  using ⟨r ≥ 1⟩ by auto
  also have ... = 0
proof -
  have jumpF f (at-left x) = 0 jumpF f (at-right x) = 0 when x ∈ {-r..r}
for x
proof -
  have poly p x ≠ 0 using ⟨roots p = {}⟩ unfolding roots-def by auto
  then show jumpF f (at-left x) = 0 jumpF f (at-right x) = 0
  unfolding f-def by (auto intro!: jumpF-not-infinity continuous-intros)
qed

```

```

    then show ?thesis unfolding cindexE-def by auto
  qed
  finally show ?thesis .
  qed
  ultimately show 2 * Re (winding-number (g r) 0) + cindex-pathE (g r) 0
    = (arctan (f r) - arctan (f (-r))) / pi
    unfolding path-defs by (auto simp add:field-simps)
  qed
  with arctan-f-tendsto show ?thesis by (auto dest:tendsto-cong)
  qed
  moreover have ?thesis when roots p≠{}
  proof -
    define max-r where max-r=Max (roots p)
    define min-r where min-r=Min (roots p)
    have max-r ∈proots p min-r ∈proots p min-r≤max-r and
      min-max-bound:∀ p∈proots p. p∈{min-r..max-r}
    proof -
      have p≠0
    proof -
      have (0::real) ≠ 1
        by simp
      then show ?thesis
        by (metis (full-types) ⟨p ≡ map-poly Re pp⟩ assms(2) coeff-0 coeff-map-poly
one-complex.simps(1) zero-complex.sel(1))
    qed
    then have finite (roots p) by auto
    then show max-r ∈proots p min-r ∈proots p
      using Min-in Max-in that unfolding max-r-def min-r-def by fast+
    then show ∀ p∈proots p. p∈{min-r..max-r}
      using Min-le Max-ge ⟨finite (roots p)⟩ unfolding max-r-def min-r-def by
auto
    then show min-r≤max-r using ⟨max-r∈proots p⟩ by auto
  qed
  have ∀ Fr in at-top. 2 * Re (winding-number (g r) 0) + cindex-pathE (g r) 0
    = (arctan (f r) - arctan (f (-r))) / pi
  proof (rule eventually-at-top-linorderI[of max (norm max-r) (norm min-r) +
1])
    fix r assume r-asm:max (norm max-r) (norm min-r) + 1 ≤ r
    then have r≠0 min-r>-r max-r<r by auto
    define u where u=(min-r + r)/(2*r)
    define v where v=(max-r + r)/(2*r)
    have uv:u∈{0..1} v∈{0..1} u≤v
      unfolding u-def v-def using r-asm ⟨min-r≤max-r⟩
      by (auto simp add:field-simps)
    define g1 where g1=subpath 0 u (g r)
    define g2 where g2=subpath u v (g r)
    define g3 where g3=subpath v 1 (g r)
    have path g1 path g2 path g3 valid-path g1 valid-path g2 valid-path g3
      unfolding g1-def g2-def g3-def using uv

```

```

    by (auto intro!:path-subpath valid-path-subpath)
    define wc-add where wc-add = ( $\lambda g. 2 * Re (winding-number g 0) + cindex-pathE g 0$ )
    have wc-add (g r) = wc-add g1 + wc-add g2 + wc-add g3
    proof -
      have winding-number (g r) 0 = winding-number g1 0 + winding-number g2 0 + winding-number g3 0
      unfolding g1-def g2-def g3-def using  $\langle u \in \{0..1\} \rangle \langle v \in \{0..1\} \rangle not-g-image$ 
      by (subst winding-number-subpath-combine,simp-all)+
      moreover have cindex-pathE (g r) 0 = cindex-pathE g1 0 + cindex-pathE g2 0 + cindex-pathE g3 0
      unfolding g1-def g2-def g3-def using  $\langle u \in \{0..1\} \rangle \langle v \in \{0..1\} \rangle \langle u \leq v \rangle not-g-image$ 
      by (subst cindex-pathE-subpath-combine,simp-all)+
      ultimately show ?thesis unfolding wc-add-def by auto
    qed
    moreover have wc-add g2=0
    proof -
      have  $2 * Re (winding-number g2 0) = - cindex-pathE g2 0$ 
      unfolding g2-def
      apply (rule winding-number-cindex-pathE-aux)
      subgoal using uv by (auto intro:finite-ReZ-segments-subpath)
      subgoal using uv by (auto intro:valid-path-subpath)
      subgoal using Path-Connected.path-image-subpath-subset  $\langle \bigwedge r. path (g r) \rangle not-g-image uv$ 
      by blast
      subgoal unfolding subpath-def v-def g-def linepath-def using r-asm  $\langle max-r \in roots p \rangle$ 
      by (auto simp add:field-simps Re-poly-of-real p-def)
      subgoal unfolding subpath-def u-def g-def linepath-def using r-asm  $\langle min-r \in roots p \rangle$ 
      by (auto simp add:field-simps Re-poly-of-real p-def)
      done
      then show ?thesis unfolding wc-add-def by auto
    qed
    moreover have wc-add g1=- arctan (f (-r)) / pi
    proof -
      have g1-pq:
         $Re (g1 t) = poly p (min-r*t+r*t-r)$ 
         $Im (g1 t) = poly q (min-r*t+r*t-r)$ 
         $Im (g1 t)/Re (g1 t) = (f o (\lambda x. (min-r+r)*x - r)) t$ 
      for t
    proof -
      have g1 t = poly pp (of-real (min-r*t+r*t-r))
      using  $\langle r \neq 0 \rangle$  unfolding g1-def g-def linepath-def subpath-def u-def p-def

      by (auto simp add:field-simps)
      then show
         $Re (g1 t) = poly p (min-r*t+r*t-r)$ 

```

```

       $Im (g1 t) = poly q (min-r*t+r*t-r)$ 
      unfolding  $p-def q-def$ 
      by (simp only:Re-poly-of-real Im-poly-of-real) +
      then show  $Im (g1 t)/Re (g1 t) = (f o (\lambda x. (min-r+r)*x - r)) t$ 
      unfolding  $f-def$  by (auto simp add:algebra-simps)
qed
have  $Re(g1 1)=0$ 
  using  $\langle r \neq 0 \rangle$  Re-poly-of-real  $\langle min-r \in roots p \rangle$ 
  unfolding  $g1-def subpath-def u-def g-def linepath-def$ 
  by (auto simp add:field-simps p-def)
have  $0 \notin path-image g1$ 
  by (metis (full-types) path-image-subpath-subset  $\langle \bigwedge r. path (g r) \rangle$ 
    atLeastAtMost-iff g1-def le-less not-g-image subsetCE uv(1) zero-le-one)

have  $wc-add-pos:wc-add h = - arctan (poly q (- r) / poly p (-r)) / pi$ 
when
   $Re-pos:\forall x \in \{0..<1\}. 0 < (Re \circ h) x$ 
  and  $hp:\forall t. Re (h t) = c*poly p (min-r*t+r*t-r)$ 
  and  $hq:\forall t. Im (h t) = c*poly q (min-r*t+r*t-r)$ 
  and  $[simp]:c \neq 0$ 

  and  $Re (h 1) = 0$ 
  and valid-path h
  and  $h-img:0 \notin path-image h$ 
  for  $h c$ 
proof -
  define  $f$  where  $f = (\lambda t. c*poly q t / (c*poly p t))$ 
  define  $farg$  where  $farg = (if 0 < Im (h 1) then pi / 2 else - pi / 2)$ 
  have  $Re (winding-number h 0) = (Im (Ln (pathfinish h))$ 
     $- Im (Ln (pathstart h))) / (2 * pi)$ 
  apply (rule Re-winding-number-half-right[of h 0,simplified])
  subgoal using  $\langle Re (h 1) = 0 \rangle$  unfolding path-image-def
    by (auto simp add:le-less)
  subgoal using  $\langle valid-path h \rangle$  .
  subgoal using  $h-img$  .
  done
also have  $\dots = (farg - arctan (f (-r))) / (2 * pi)$ 
proof -
  have  $Im (Ln (pathfinish h)) = farg$ 
  using  $\langle Re(h 1)=0 \rangle$  unfolding farg-def path-defs
  apply (subst Im-Ln-eq)
  subgoal using  $h-img$  unfolding path-defs by fastforce
  subgoal by simp
  done
moreover have  $Im (Ln (pathstart h)) = arctan (f (-r))$ 
proof -
  have  $pathstart h \neq 0$ 
  using  $h-img$ 
  by (metis pathstart-in-path-image)

```

```

    then have  $Im (Ln (pathstart h)) = arctan (Im (pathstart h) / Re$ 
(pathstart h))
    using  $Re-pos[rule-format, of 0]$ 
    by ( $simp add: Im-Ln-eq path-defs$ )
    also have  $\dots = arctan (f (-r))$ 
    unfolding  $f-def path-defs hp[rule-format] hq[rule-format]$ 
    by  $simp$ 
    finally show ?thesis .
  qed
  ultimately show ?thesis by auto
  qed
  finally have  $Re (winding-number h 0) = (farg - arctan (f (-r))) / (2 *$ 
pi) .
  moreover have  $cindex-pathE h 0 = (-farg/pi)$ 
  proof -
    have  $cindex-pathE h 0 = cindexE 0 1 (f \circ (\lambda x. (min-r + r) * x - r))$ 
    unfolding  $cindex-pathE-def$  using  $\langle c \neq 0 \rangle$ 
    by ( $auto simp add: hp hq f-def comp-def algebra-simps$ )
    also have  $\dots = cindexE (-r) min-r f$ 
    apply ( $subst cindexE-linear-comp[where b=-r, simplified]$ )
    using  $r-asm$  by auto
    also have  $\dots = - jumpF f (at-left min-r)$ 
    proof -
      define right where  $right = \{x. jumpF f (at-right x) \neq 0 \wedge -r \leq x$ 
 $\wedge x < min-r\}$ 
      define left where  $left = \{x. jumpF f (at-left x) \neq 0 \wedge -r < x \wedge x$ 
 $\leq min-r\}$ 
      have  $*: jumpF f (at-right x) = 0 \wedge jumpF f (at-left x) = 0$  when
 $x \in \{-r..<min-r\}$  for x
      proof -
        have False when  $poly p x = 0$ 
        proof -
          have  $x \geq min-r$ 
          using  $min-max-bound[rule-format, of x]$  that by auto
          then show False using  $\langle x \in \{-r..<min-r\} \rangle$  by auto
        qed
        then show  $jumpF f (at-right x) = 0 \wedge jumpF f (at-left x) = 0$ 
        unfolding  $f-def$  by ( $auto intro!: jumpF-not-infinity continuous-intros$ )
      qed
      then have right = {}
      unfolding right-def by force
      moreover have left = (if  $jumpF f (at-left min-r) = 0$  then {} else
{min-r})
      unfolding left-def le-less using  $* r-asm$  by force
      ultimately show ?thesis
      unfolding  $cindexE-def$  by ( $fold left-def right-def, auto$ )
    qed
    also have  $\dots = (-farg/pi)$ 

```

```

proof –
  have  $p\text{-pos}:c*\text{poly } p \ x > 0$  when  $x \in \{-r < \dots < \text{min-}r\}$  for  $x$ 
proof –
  define  $hh$  where  $hh = (\lambda t. \text{min-}r*t + r*t - r)$ 
  have  $(x+r)/(min-r+r) \in \{0..<1\}$ 
    using that r-asm by (auto simp add:field-simps)
  then have  $0 < c*\text{poly } p \ (hh \ ((x+r)/(min-r+r)))$ 
    apply (drule-tac Re-pos[rule-format])
    unfolding comp-def hp[rule-format] hq[rule-format] hh-def .
  moreover have  $hh \ ((x+r)/(min-r+r)) = x$ 
    unfolding hh-def using  $\langle \text{min-}r > -r \rangle$ 
    apply (auto simp add:divide-simps)
    by (auto simp add:algebra-simps)
  ultimately show ?thesis by simp
qed

  have  $c*\text{poly } q \ \text{min-}r \neq 0$ 
    using no-real-zero  $\langle c \neq 0 \rangle$ 
by (metis Im-complex-of-real UNIV-I  $\langle \text{min-}r \in \text{roots } p \rangle$  cpoly-of-decompose)

     $\text{mult-eq-0-iff } p\text{-def } \text{poly-cpoly-of-real-iff } \text{roots-within-iff } q\text{-def}$ 

  moreover have ?thesis when  $c*\text{poly } q \ \text{min-}r > 0$ 
proof –
  have  $0 < \text{Im } (h \ 1)$  unfolding hq[rule-format] hp[rule-format] using
that by auto
  moreover have  $\text{jumpF } f \ (\text{at-left } \text{min-}r) = 1/2$ 
proof –
  have  $((\lambda t. c*\text{poly } p \ t) \ \text{has-sgnx } 1) \ (\text{at-left } \text{min-}r)$ 
    unfolding has-sgnx-def
    apply (rule eventually-at-leftI[of -r])
    using  $p\text{-pos}$   $\langle \text{min-}r > -r \rangle$  by auto
  then have filterlim f at-top (at-left min-r)
    unfolding f-def
    apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q min-r])
    using that  $\langle \text{min-}r \in \text{roots } p \rangle$  by (auto intro!:tendsto-eq-intros)
  then show ?thesis unfolding jumpF-def by auto
qed
  ultimately show ?thesis unfolding farg-def by auto
qed
  moreover have ?thesis when  $c*\text{poly } q \ \text{min-}r < 0$ 
proof –
  have  $0 > \text{Im } (h \ 1)$  unfolding hq[rule-format] hp[rule-format] using
that by auto
  moreover have  $\text{jumpF } f \ (\text{at-left } \text{min-}r) = -1/2$ 
proof –
  have  $((\lambda t. c*\text{poly } p \ t) \ \text{has-sgnx } 1) \ (\text{at-left } \text{min-}r)$ 
    unfolding has-sgnx-def
    apply (rule eventually-at-leftI[of -r])

```

```

    using p-pos ⟨min-r⟩-r by auto
  then have filterlim f at-bot (at-left min-r)
    unfolding f-def
    apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q min-r])
    using that ⟨min-r∈proots p⟩ by (auto intro!:tendsto-eq-intros)
  then show ?thesis unfolding jumpF-def by auto
qed
ultimately show ?thesis unfolding farg-def by auto
qed
ultimately show ?thesis by linarith
qed
finally show ?thesis .
qed
ultimately show ?thesis unfolding wc-add-def f-def by (auto simp
add:field-simps)
qed

have ∀x∈{0..<1}. (Re ∘ g1) x ≠ 0
proof (rule ccontr)
  assume ¬ (∀x∈{0..<1}. (Re ∘ g1) x ≠ 0)
  then obtain t where t-def:Re (g1 t) = 0 t∈{0..<1}
    unfolding path-image-def by fastforce
  define m where m=min-r*t+r*t-r
  have poly p m=0
  proof -
    have Re (g1 t) = Re (poly pp (of-real m))
      unfolding m-def g1-def g-def linepath-def subpath-def u-def using
⟨r≠0⟩
    by (auto simp add:field-simps)
  then show ?thesis using t-def unfolding Re-poly-of-real p-def by auto
qed
moreover have m<min-r
proof -
  have min-r+r>0 using r-asm by simp
  then have (min-r + r)*(t-1)<0 using ⟨t∈{0..<1}⟩
    by (simp add: mult-pos-neg)
  then show ?thesis unfolding m-def by (auto simp add:algebra-simps)
qed
ultimately show False using min-max-bound unfolding proots-def by
auto
qed
then have (∀x∈{0..<1}. 0 < (Re ∘ g1) x) ∨ (∀x∈{0..<1}. (Re ∘ g1) x
< 0)
  apply (elim continuous-on-neq-split)
  using ⟨path g1⟩ unfolding path-def
  by (auto intro!:continuous-intros elim:continuous-on-subset)
moreover have ?thesis when ∀x∈{0..<1}. (Re ∘ g1) x < 0
proof -
  have wc-add (uminus o g1) = - arctan (f (- r)) / pi

```

```

    unfolding f-def
    apply (rule wc-add-pos[of - -1])
    using g1-pq that ⟨min-r ∈proots p⟩ ⟨valid-path g1⟩ ⟨0 ∉ path-image g1⟩
    by (auto simp add:path-image-compose)
    moreover have wc-add (uminus ∘ g1) = wc-add g1
    unfolding wc-add-def cindex-pathE-def
    apply (subst winding-number-uminus-comp)
    using ⟨valid-path g1⟩ ⟨0 ∉ path-image g1⟩ by auto
    ultimately show ?thesis by auto
qed
moreover have ?thesis when ∀x∈{0..<1}. (Re ∘ g1) x > 0
    unfolding f-def
    apply (rule wc-add-pos[of - 1])
    using g1-pq that ⟨min-r ∈proots p⟩ ⟨valid-path g1⟩ ⟨0 ∉ path-image g1⟩
    by (auto simp add:path-image-compose)
    ultimately show ?thesis by blast
qed
moreover have wc-add g3 = arctan (f r) / pi
proof -
  have g3-pq:
    Re (g3 t) = poly p ((r-max-r)*t + max-r)
    Im (g3 t) = poly q ((r-max-r)*t + max-r)
    Im (g3 t)/Re (g3 t) = (f o (λx. (r-max-r)*x + max-r)) t
  for t
proof -
  have g3 t = poly pp (of-real ((r-max-r)*t + max-r))
  using ⟨r≠0⟩ ⟨max-r<r⟩ unfolding g3-def g-def linepath-def subpath-def
v-def p-def
  by (auto simp add:algebra-simps)
  then show
    Re (g3 t) = poly p ((r-max-r)*t + max-r)
    Im (g3 t) = poly q ((r-max-r)*t + max-r)
  unfolding p-def q-def
  by (simp only:Re-poly-of-real Im-poly-of-real)+
  then show Im (g3 t)/Re (g3 t) = (f o (λx. (r-max-r)*x + max-r)) t
  unfolding f-def by (auto simp add:algebra-simps)
qed
have Re(g3 0)=0
  using ⟨r≠0⟩ Re-poly-of-real ⟨max-r∈proots p⟩
  unfolding g3-def subpath-def v-def g-def linepath-def
  by (auto simp add:field-simps p-def)
have 0 ∉ path-image g3
proof -
  have (1::real) ∈ {0..1}
  by auto
  then show ?thesis
  using Path-Connected.path-image-subpath-subset ⟨∧r. path (g r)⟩ g3-def
not-g-image wv(2) by blast
qed

```

```

have wc-add-pos:wc-add  $h = \arctan (\text{poly } q \ r / \text{poly } p \ r) / \pi$  when
  Re-pos: $\forall x \in \{0 <.. 1\}. 0 < (\text{Re } \circ h) \ x$ 
  and hp: $\forall t. \text{Re } (h \ t) = c * \text{poly } p ((r - \text{max-r}) * t + \text{max-r})$ 
  and hq: $\forall t. \text{Im } (h \ t) = c * \text{poly } q ((r - \text{max-r}) * t + \text{max-r})$ 
  and [simp]: $c \neq 0$ 

  and  $\text{Re } (h \ 0) = 0$ 
  and valid-path  $h$ 
  and h-img: $0 \notin \text{path-image } h$ 
  for  $h \ c$ 
proof -
  define  $f$  where  $f = (\lambda t. c * \text{poly } q \ t / (c * \text{poly } p \ t))$ 
  define  $farg$  where  $farg = (\text{if } 0 < \text{Im } (h \ 0) \text{ then } \pi / 2 \text{ else } - \pi / 2)$ 
  have  $\text{Re } (\text{winding-number } h \ 0) = (\text{Im } (\text{Ln } (\text{pathfinish } h))$ 
     $- \text{Im } (\text{Ln } (\text{pathstart } h))) / (2 * \pi)$ 
  apply (rule Re-winding-number-half-right[of h 0,simplified])
  subgoal using that  $\langle \text{Re } (h \ 0) = 0 \rangle$  unfolding path-image-def
    by (auto simp add:le-less)
  subgoal using  $\langle \text{valid-path } h \rangle$  .
  subgoal using h-img .
  done
also have  $\dots = (\arctan (f \ r) - farg) / (2 * \pi)$ 
proof -
  have  $\text{Im } (\text{Ln } (\text{pathstart } h)) = farg$ 
  using  $\langle \text{Re}(h \ 0)=0 \rangle$  unfolding farg-def path-defs
  apply (subst Im-Ln-eq)
  subgoal using h-img unfolding path-defs by fastforce
  subgoal by simp
  done
moreover have  $\text{Im } (\text{Ln } (\text{pathfinish } h)) = \arctan (f \ r)$ 
proof -
  have  $\text{pathfinish } h \neq 0$ 
  using h-img
  by (metis pathfinish-in-path-image)
  then have  $\text{Im } (\text{Ln } (\text{pathfinish } h)) = \arctan (\text{Im } (\text{pathfinish } h) / \text{Re}$ 
(pathfinish h))
  using Re-pos[rule-format,of 1]
  by (simp add: Im-Ln-eq path-defs)
  also have  $\dots = \arctan (f \ r)$ 
  unfolding f-def path-defs hp[rule-format] hq[rule-format]
  by simp
  finally show ?thesis .
qed
ultimately show ?thesis by auto
qed
finally have  $\text{Re } (\text{winding-number } h \ 0) = (\arctan (f \ r) - farg) / (2 * \pi)$  .
moreover have cindex-pathE  $h \ 0 = farg / \pi$ 
proof -

```

```

have cindex-pathE h 0 = cindexE 0 1 (f ∘ (λx. (r-max-r)*x + max-r))
  unfolding cindex-pathE-def using ⟨c≠0⟩
  by (auto simp add:hp hq f-def comp-def algebra-simps)
also have ... = cindexE max-r r f
  apply (subst cindexE-linear-comp)
  using r-asm by auto
also have ... = jumpF f (at-right max-r)
proof -
  define right where right = {x. jumpF f (at-right x) ≠ 0 ∧ max-r ≤ x
∧ x < r}
  define left where left = {x. jumpF f (at-left x) ≠ 0 ∧ max-r < x ∧ x
≤ r}
  have *:jumpF f (at-right x) = 0 jumpF f (at-left x) = 0 when
x∈{max-r<..r} for x
  proof -
    have False when poly p x = 0
    proof -
      have x < max-r
      using min-max-bound[rule-format, of x] that by auto
      then show False using ⟨x∈{max-r<..r}⟩ by auto
    qed
    then show jumpF f (at-right x) = 0 jumpF f (at-left x) = 0
    unfolding f-def by (auto intro!:jumpF-not-infinity continuous-intros)

  qed
  then have left = {}
  unfolding left-def by force
  moreover have right = (if jumpF f (at-right max-r) = 0 then {} else
{max-r})
  unfolding right-def le-less using * r-asm by force
  ultimately show ?thesis
  unfolding cindexE-def by (fold left-def right-def, auto)
qed
also have ... = farg/pi
proof -
  have p-pos:c*poly p x > 0 when x ∈ {max-r<..<r} for x
  proof -
    define hh where hh=(λt. (r-max-r)*t + max-r)
    have (x-max-r)/(r-max-r) ∈ {0<..1}
    using that r-asm by (auto simp add:field-simps)
    then have 0 < c*poly p (hh ((x-max-r)/(r-max-r)))
    apply (drule-tac Re-pos[rule-format])
    unfolding comp-def hp[rule-format] hq[rule-format] hh-def .
    moreover have hh ((x-max-r)/(r-max-r)) = x
    unfolding hh-def using ⟨max-r<r⟩
    by (auto simp add:divide-simps)
    ultimately show ?thesis by simp
  qed

```

```

    have c*poly q max-r ≠ 0
      using no-real-zero ⟨c≠0⟩
  by (metis Im-complex-of-real UNIV-I ⟨max-r ∈ roots p⟩ cpoly-of-decompose

      mult-eq-0-iff p-def poly-cpoly-of-real-iff roots-within-iff q-def)

  moreover have ?thesis when c*poly q max-r > 0
  proof -
    have 0 < Im (h 0) unfolding hq[rule-format] hp[rule-format] using
that by auto
    moreover have jumpF f (at-right max-r) = 1/2
    proof -
      have ((λt. c*poly p t) has-sgnx 1) (at-right max-r)
        unfolding has-sgnx-def
        apply (rule eventually-at-rightI[of - r])
        using p-pos ⟨max-r < r⟩ by auto
      then have filterlim f at-top (at-right max-r)
        unfolding f-def
        apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q max-r])
        using that ⟨max-r ∈ roots p⟩ by (auto intro!: tendsto-eq-intros)
      then show ?thesis unfolding jumpF-def by auto
    qed
    ultimately show ?thesis unfolding farg-def by auto
  qed
  moreover have ?thesis when c*poly q max-r < 0
  proof -
    have 0 > Im (h 0) unfolding hq[rule-format] hp[rule-format] using
that by auto
    moreover have jumpF f (at-right max-r) = - 1/2
    proof -
      have ((λt. c*poly p t) has-sgnx 1) (at-right max-r)
        unfolding has-sgnx-def
        apply (rule eventually-at-rightI[of - r])
        using p-pos ⟨max-r < r⟩ by auto
      then have filterlim f at-bot (at-right max-r)
        unfolding f-def
        apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q max-r])
        using that ⟨max-r ∈ roots p⟩ by (auto intro!: tendsto-eq-intros)
      then show ?thesis unfolding jumpF-def by auto
    qed
    ultimately show ?thesis unfolding farg-def by auto
  qed
  ultimately show ?thesis by linarith
  qed
  finally show ?thesis .
  qed
  ultimately show ?thesis unfolding wc-add-def f-def by (auto simp
add:field-simps)
  qed

```

```

have  $\forall x \in \{0 <..1\}. (Re \circ g\beta) x \neq 0$ 
proof (rule ccontr)
  assume  $\neg (\forall x \in \{0 <..1\}. (Re \circ g\beta) x \neq 0)$ 
  then obtain  $t$  where  $t\text{-def}: Re (g\beta t) = 0$   $t \in \{0 <..1\}$ 
    unfolding path-image-def by fastforce
  define  $m$  where  $m = (r - \max\text{-}r) * t + \max\text{-}r$ 
  have poly p m = 0
  proof -
    have  $Re (g\beta t) = Re (poly\ pp (of\text{-}real\ m))$ 
    unfolding m-def g\beta-def g-def linepath-def subpath-def v-def using  $\langle r \neq 0 \rangle$ 
      by (auto simp add: algebra-simps)
    then show ?thesis using t-def unfolding Re-poly-of-real p-def by auto
  qed
  moreover have  $m > \max\text{-}r$ 
  proof -
    have  $r - \max\text{-}r > 0$  using r-asm by simp
    then have  $(r - \max\text{-}r) * t > 0$  using  $\langle t \in \{0 <..1\} \rangle$ 
      by (simp add: mult-pos-neg)
    then show ?thesis unfolding m-def by (auto simp add: algebra-simps)
  qed
  ultimately show False using min-max-bound unfolding proots-def by
auto
qed
then have  $(\forall x \in \{0 <..1\}. 0 < (Re \circ g\beta) x) \vee (\forall x \in \{0 <..1\}. (Re \circ g\beta) x < 0)$ 
  apply (elim continuous-on-neq-split)
  using  $\langle path\ g\beta \rangle$  unfolding path-def
  by (auto intro!: continuous-intros elim: continuous-on-subset)
moreover have ?thesis when  $\forall x \in \{0 <..1\}. (Re \circ g\beta) x < 0$ 
proof -
  have  $wc\text{-}add (u\text{-}minus \circ g\beta) = arctan (f\ r) / \pi$ 
  unfolding f-def
  apply (rule wc-add-pos[of - -1])
  using g\beta-pq that  $\langle \max\text{-}r \in \text{proots } p \rangle \langle \text{valid-path } g\beta \rangle \langle 0 \notin \text{path-image } g\beta \rangle$ 
    by (auto simp add: path-image-compose)
  moreover have  $wc\text{-}add (u\text{-}minus \circ g\beta) = wc\text{-}add\ g\beta$ 
  unfolding wc-add-def cindex-pathE-def
  apply (subst winding-number-u\text{-}minus-comp)
  using  $\langle \text{valid-path } g\beta \rangle \langle 0 \notin \text{path-image } g\beta \rangle$  by auto
  ultimately show ?thesis by auto
qed
moreover have ?thesis when  $\forall x \in \{0 <..1\}. (Re \circ g\beta) x > 0$ 
  unfolding f-def
  apply (rule wc-add-pos[of - 1])
  using g\beta-pq that  $\langle \max\text{-}r \in \text{proots } p \rangle \langle \text{valid-path } g\beta \rangle \langle 0 \notin \text{path-image } g\beta \rangle$ 
    by (auto simp add: path-image-compose)
  ultimately show ?thesis by blast
qed

```

```

ultimately have wc-add (g r) = (arctan (f r) - arctan (f (-r))) / pi
  by (auto simp add:field-simps)
then show 2 * Re (winding-number (g r) 0) + cindex-pathE (g r) 0
  = (arctan (f r) - arctan (f (-r))) / pi
  unfolding wc-add-def .
qed
with arctan-f-tendsto show ?thesis by (auto dest:tendsto-cong)
qed
ultimately show ?thesis by auto
qed

lemma proots-upper-cindex-eg:
  assumes lead-coeff p=1 and no-real-roots:∀ x∈proots p. Im x≠0
  shows proots-upper p =
    (degree p - cindex-poly-ubd (map-poly Im p) (map-poly Re p)) / 2
proof (cases degree p = 0)
  case True
  then obtain c where p=[:c:] using degree-eg-zeroE by blast
  then have p-def:p=[:1:] using ⟨lead-coeff p=1⟩ by simp
  have proots-count p {x. Im x>0} = 0 unfolding p-def proots-count-def by auto

  moreover have cindex-poly-ubd (map-poly Im p) (map-poly Re p) = 0
    apply (subst cindex-poly-ubd-code)
    unfolding p-def
  by (auto simp add:map-poly-pCons changes-R-smods-def changes-poly-neg-inf-def
    changes-poly-pos-inf-def)
  ultimately show ?thesis using True unfolding proots-upper-def by auto
next
  case False
  then have degree p>0 p≠0 by auto
  define w1 where w1=(λr. Re (winding-number (poly p ∘
    (λx. complex-of-real (linepath (- r) (of-real r) x))) 0))
  define w2 where w2=(λr. Re (winding-number (poly p ∘ part-circlepath 0 r 0
    pi) 0))
  define cp where cp=(λr. cindex-pathE (poly p ∘ (λx.
    of-real (linepath (- r) (of-real r) x))) 0)
  define ci where ci=(λr. cindexE (-r) r (λx. poly (map-poly Im p) x/poly
    (map-poly Re p) x))
  define cubd where cubd =cindex-poly-ubd (map-poly Im p) (map-poly Re p)
  obtain R where proots p ⊆ ball 0 R and R>0
    using ⟨p≠0⟩ finite-ball-include[of proots p 0] by auto
  have ((λr. w1 r + w2 r + cp r / 2 - ci r/2)
    → real (degree p) / 2 - of-int cubd / 2) at-top
  proof -
    have t1:(λr. 2 * w1 r + cp r) → 0 at-top
      using Re-winding-number-poly-linepth[OF assms] unfolding w1-def cp-def
    by auto
    have t2:(w2 → real (degree p) / 2) at-top

```

```

    using Re-winding-number-poly-part-circlepath[OF ‹degree p>0›,of 0] unfolding
    w2-def by auto
    have t3:(ci ⟶ of-int cubd) at-top
    apply (rule tendsto-eventually)
    using cindex-poly-ubd-eventually[of map-poly Im p map-poly Re p]
    unfolding ci-def cubd-def by auto
    from tendsto-add[OF tendsto-add[OF tendsto-mult-left[OF t3,of -1/2,simplified]
        tendsto-mult-left[OF t1,of 1/2,simplified]
        t2]]
    show ?thesis by (simp add:algebra-simps)
qed
moreover have  $\forall_F r$  in at-top.  $w1\ r + w2\ r + cp\ r / 2 - ci\ r / 2 = \text{roots-count}$ 
p {x. Im x>0}
proof (rule eventually-at-top-linorderI[of R])
    fix r assume  $r \geq R$ 
    then have r-ball:roots p  $\subseteq$  ball 0 r and  $r > 0$ 
    using ‹R>0› ‹roots p  $\subseteq$  ball 0 R› by auto
    define ll where ll=linepath (- complex-of-real r) r
    define rr where rr=part-circlepath 0 r 0 pi
    define lr where lr = ll +++ rr
    have img-ll:path-image ll  $\subseteq$  - roots p and img-rr: path-image rr  $\subseteq$  - roots
p
    subgoal unfolding ll-def using ‹0 < r› closed-segment-degen-complex(2)
no-real-roots by auto
    subgoal unfolding rr-def using in-path-image-part-circlepath ‹0 < r› r-ball
by fastforce
    done
    have [simp]:valid-path (poly p o ll) valid-path (poly p o rr)
    valid-path ll valid-path rr
    pathfinish rr=pathstart ll pathfinish ll = pathstart rr
proof -
    show valid-path (poly p o ll) valid-path (poly p o rr)
    unfolding ll-def rr-def by (auto intro:valid-path-compose-holomorphic)
    then show valid-path ll valid-path rr unfolding ll-def rr-def by auto
    show pathfinish rr=pathstart ll pathfinish ll = pathstart rr
    unfolding ll-def rr-def by auto
qed
    have roots-count p {x. Im x>0} =  $(\sum_{x \in \text{roots } p} \text{winding-number } lr\ x\ * \text{of-nat } (\text{order } x\ p))$ 
unfolding roots-count-def of-nat-sum
proof (rule sum.mono-neutral-cong-left)
    show finite (roots p) roots-within p {x. 0 < Im x}  $\subseteq$  roots p
    using ‹p≠0› by auto
next
    have winding-number lr x=0 when  $x \in \text{roots } p - \text{roots-within } p\ \{x.\ 0 < \text{Im } x\}$ 
for x
    unfolding lr-def ll-def rr-def
proof (eval-winding,simp-all)

```

```

show *:x ∉ closed-segment (− complex-of-real r) (complex-of-real r)
  using img-ll that unfolding ll-def by auto
show x ∉ path-image (part-circlepath 0 r 0 pi)
  using img-rr that unfolding rr-def by auto
have xr-facts:0 > Im x −r < Re x Re x < r cmod x < r
proof −
  have Im x ≤ 0 using that by auto
  moreover have Im x ≠ 0 using no-real-roots that by blast
  ultimately show 0 > Im x by auto
next
show cmod x < r using that r-ball by auto
then have |Re x| < r
  using abs-Re-le-cmod[of x] by argo
then show −r < Re x Re x < r by linarith+
qed
then have cindex-pathE ll x = 1
  using ⟨r>0⟩ unfolding cindex-pathE-linepath[OF *] ll-def
  by (auto simp add: mult-pos-neg)
moreover have cindex-pathE rr x = −1
  unfolding rr-def using r-ball that
  by (auto intro!: cindex-pathE-circlepath-upper)
ultimately show −cindex-pathE (linepath (− of-real r) (of-real r)) x =
  cindex-pathE (part-circlepath 0 r 0 pi) x
  unfolding ll-def rr-def by auto
qed
then show ∀ i ∈ roots p − roots-within p {x. 0 < Im x}.
  winding-number lr i * of-nat (order i p) = 0
  by auto
next
fix x assume x-asm:x ∈ roots-within p {x. 0 < Im x}
have winding-number lr x = 1 unfolding lr-def ll-def rr-def
proof (eval-winding,simp-all)
show *:x ∉ closed-segment (− complex-of-real r) (complex-of-real r)
  using img-ll x-asm unfolding ll-def by auto
show x ∉ path-image (part-circlepath 0 r 0 pi)
  using img-rr x-asm unfolding rr-def by auto
have xr-facts:0 < Im x −r < Re x Re x < r cmod x < r
proof −
  show 0 < Im x using x-asm by auto
next
show cmod x < r using x-asm r-ball by auto
then have |Re x| < r
  using abs-Re-le-cmod[of x] by argo
then show −r < Re x Re x < r by linarith+
qed
then have cindex-pathE ll x = −1
  using ⟨r>0⟩ unfolding cindex-pathE-linepath[OF *] ll-def
  by (auto simp add: mult-less-0-iff)
moreover have cindex-pathE rr x = −1

```

```

    unfolding rr-def using r-ball x-asm
    by (auto intro!: cindex-pathE-circlepath-upper)
    ultimately show – of-real (cindex-pathE (linepath (– of-real r) (of-real
r)) x) –
      of-real (cindex-pathE (part-circlepath 0 r 0 pi) x) = 2
    unfolding ll-def rr-def by auto
  qed
  then show of-nat (order x p) = winding-number lr x * of-nat (order x p) by
auto
  qed
  also have ... = 1/(2*pi*i) * contour-integral lr (λx. deriv (poly p) x / poly p
x)
    apply (subst argument-principle-poly[of p lr])
    using ⟨p≠0⟩ img-ll img-rr unfolding lr-def ll-def rr-def
    by (auto simp add:path-image-join)
  also have ... = winding-number (poly p ∘ lr) 0
    apply (subst winding-number-comp[of UNIV poly p lr 0])
    using ⟨p≠0⟩ img-ll img-rr unfolding lr-def ll-def rr-def
    by (auto simp add:path-image-join path-image-compose)
  also have ... = Re (winding-number (poly p ∘ lr) 0)
  proof –
    have winding-number (poly p ∘ lr) 0 ∈ Ints
      apply (rule integer-winding-number)
      using ⟨p≠0⟩ img-ll img-rr unfolding lr-def
      by (auto simp add:path-image-join path-image-compose path-compose-join
pathstart-compose pathfinish-compose valid-path-imp-path)
    then show ?thesis by (simp add: complex-eqI complex-is-Int-iff)
  qed
  also have ... = Re (winding-number (poly p ∘ ll) 0) + Re (winding-number
(poly p ∘ rr) 0)
    unfolding lr-def path-compose-join using img-ll img-rr
    apply (subst winding-number-join)
    by (auto simp add:valid-path-imp-path path-image-compose pathstart-compose
pathfinish-compose)
  also have ... = w1 r + w2 r
    unfolding w1-def w2-def ll-def rr-def of-real-linepath by auto
  finally have of-nat (proots-count p {x. 0 < Im x}) = complex-of-real (w1 r +
w2 r) .
  then have proots-count p {x. 0 < Im x} = w1 r + w2 r
    using of-real-eq-iff by fastforce
  moreover have cp r = ci r
  proof –
    define f where f=(λx. Im (poly p (of-real x)) / Re (poly p x))
    have cp r = cindex-pathE (poly p ∘ (λx. 2*r*x – r)) 0
      unfolding cp-def linepath-def by (auto simp add:algebra-simps)
    also have ... = cindexE 0 1 (f o (λx. 2*r*x – r))
      unfolding cp-def ci-def cindex-pathE-def f-def comp-def by auto
    also have ... = cindexE (–r) r f
      apply (subst cindexE-linear-comp[of 2*r 0 1 f –r,simplified])

```

using $\langle r > 0 \rangle$ **by** *auto*
also have $\dots = ci\ r$
unfolding *ci-def f-def Im-poly-of-real Re-poly-of-real* **by** *simp*
finally show *?thesis* .
qed
ultimately show $w1\ r + w2\ r + cp\ r / 2 - ci\ r / 2 = real\ (proots-count\ p\ \{x.\ 0 < Im\ x\})$
by *auto*
qed
ultimately have $((\lambda r::real.\ real\ (proots-count\ p\ \{x.\ 0 < Im\ x\})) \longrightarrow real\ (degree\ p) / 2 - of-int\ cubd / 2)$ *at-top*
by *(auto dest: tendsto-cong)*
then show *?thesis*
apply *(subst (asm) tendsto-const-iff)*
unfolding *cubd-def proots-upper-def* **by** *auto*
qed

lemma *cindexE-roots-on-horizontal-border:*

fixes $a::complex$ **and** $s::real$
defines $g \equiv linepath\ a\ (a + of-real\ s)$
assumes $pqr:p = q * r$ **and** $r\text{-monic}:lead-coeff\ r=1$ **and** $r\text{-proots}:\forall x \in proots\ r.\ Im\ x = Im\ a$
shows $cindexE\ lb\ ub\ (\lambda t.\ Im\ ((poly\ p \circ g)\ t) / Re\ ((poly\ p \circ g)\ t)) = cindexE\ lb\ ub\ (\lambda t.\ Im\ ((poly\ q \circ g)\ t) / Re\ ((poly\ q \circ g)\ t))$
using *assms*
proof *(induct r arbitrary:p rule:poly-root-induct-alt)*
case 0
then have *False*
by *(metis Im-complex-of-real UNIV-I imaginary-unit.simps(2) proots-within-0 zero-neq-one)*
then show *?case* **by** *simp*
next
case *(no-proots r)*
then obtain b **where** $b \neq 0\ r = [:b:]$
using *fundamental-theorem-of-algebra-alt* **by** *blast*
then have $r=1$ **using** $\langle lead-coeff\ r = 1 \rangle$ **by** *simp*
with $\langle p = q * r \rangle$ **show** *?case* **by** *simp*
next
case *(root b r)*
then have *?case* **when** $s=0$
using *that(1)* **unfolding** *cindex-pathE-def* **by** *(simp add:cindexE-constI)*
moreover have *?case* **when** $s \neq 0$
proof –
define qrg **where** $qrg = poly\ (q*r) \circ g$
have $cindexE\ lb\ ub\ (\lambda t.\ Im\ ((poly\ p \circ g)\ t) / Re\ ((poly\ p \circ g)\ t)) = cindexE\ lb\ ub\ (\lambda t.\ Im\ (qrg\ t * (g\ t - b)) / Re\ (qrg\ t * (g\ t - b)))$
unfolding *qrg-def* $\langle p = q * ([: - b, 1:] * r) \rangle$ *comp-def*
by *(simp add:algebra-simps)*
also have $\dots = cindexE\ lb\ ub$

```

      (λt. ((Re a + t * s - Re b) * Im (qrg t)) /
        ((Re a + t * s - Re b) * Re (qrg t)))
    proof -
      have Im b = Im a
        using ⟨∀ x ∈ roots ([:- b, 1:] * r). Im x = Im a⟩ by auto
      then show ?thesis
        unfolding cindex-pathE-def g-def linepath-def
        by (simp add: algebra-simps)
    qed
  also have ... = cindexE lb ub (λt. Im (qrg t) / Re (qrg t))
  proof (rule cindexE-cong[of {t. Re a + t * s - Re b = 0}])
    show finite {t. Re a + t * s - Re b = 0}
    proof (cases Re a = Re b)
      case True
      then have {t. Re a + t * s - Re b = 0} = {0}
        using ⟨s ≠ 0⟩ by auto
      then show ?thesis by auto
    next
      case False
      then have {t. Re a + t * s - Re b = 0} = {(Re b - Re a) / s}
        using ⟨s ≠ 0⟩ by (auto simp add: field-simps)
      then show ?thesis by auto
    qed
  next
    fix x assume asm: x ∉ {t. Re a + t * s - Re b = 0}
    define tt where tt = Re a + x * s - Re b
    have tt ≠ 0 using asm unfolding tt-def by auto
    then show tt * Im (qrg x) / (tt * Re (qrg x)) = Im (qrg x) / Re (qrg x)
      by auto
    qed
  also have ... = cindexE lb ub (λt. Im ((poly q ∘ g) t) / Re ((poly q ∘ g) t))
    unfolding qrg-def
  proof (rule root(1))
    show lead-coeff r = 1
    by (metis lead-coeff-mult lead-coeff-pCons(1) mult-cancel-left2 one-poly-eq-simps(2)
      root.premis(2) zero-neq-one)
    qed (use root in simp-all)
  finally show ?thesis .
  qed
  ultimately show ?case by auto
  qed

lemma poly-decompose-by-roots:
  fixes p :: 'a :: idom poly
  assumes p ≠ 0
  shows ∃ q r. p = q * r ∧ lead-coeff q = 1 ∧ (∀ x ∈ roots q. P x) ∧ (∀ x ∈ roots r.
  ¬P x) using assms

```

```

proof (induct p rule:poly-root-induct-alt)
  case 0
  then show ?case by simp
next
  case (no-roots p)
  then show ?case
    by (metis lead-coeff-1 mult-1 poly-1 roots-within-iff zero-neq-one)
next
  case (root a p)
  then obtain q r where pqr:p = q * r and leadq:lead-coeff q=1
    and qball:∀ a∈proots q. P a and rball:∀ x∈proots r. ¬ P x
    using mult-zero-right by metis
  have ?case when P a
    apply (rule-tac x=[:- a, 1:] * q in exI)
    apply (rule-tac x=r in exI)
    using pqr qball rball that leadq unfolding lead-coeff-mult
    by (auto simp add:algebra-simps)
  moreover have ?case when ¬ P a
    apply (rule-tac x=q in exI)
    apply (rule-tac x=[:- a, 1:] * r in exI)
    using pqr qball rball that leadq unfolding lead-coeff-mult
    by (auto simp add:algebra-simps)
  ultimately show ?case by blast
qed

```

lemma *proots-upper-cindex-eq'*:

assumes lead-coeff p=1

shows $\text{proots-upper } p = (\text{degree } p - \text{proots-count } p \{x. \text{Im } x=0\} - \text{cindex-poly-ubd } (\text{map-poly } \text{Im } p) (\text{map-poly } \text{Re } p)) / 2$

proof –

have $p \neq 0$ **using** *assms* **by** auto

from *poly-decompose-by-roots*[OF *this*, of $\lambda x. \text{Im } x \neq 0$]

obtain q r **where** pqr:p = q * r **and** leadq:lead-coeff q=1

and qball: $\forall x \in \text{proots } q. \text{Im } x \neq 0$ **and** rball: $\forall x \in \text{proots } r. \text{Im } x = 0$

by auto

have $\text{real-of-int } (\text{proots-upper } p) = \text{proots-upper } q + \text{proots-upper } r$

using $\langle p \neq 0 \rangle$ **unfolding** *proots-upper-def* pqr **by** (auto simp add:proots-count-times)

also have $\dots = \text{proots-upper } q$

proof –

have $\text{proots-within } r \{z. 0 < \text{Im } z\} = \{\}$

using rball **by** auto

then have $\text{proots-upper } r = 0$

unfolding *proots-upper-def* *proots-count-def* **by** simp

then show ?thesis **by** auto

qed

also have $\dots = (\text{degree } q - \text{cindex-poly-ubd } (\text{map-poly } \text{Im } q) (\text{map-poly } \text{Re } q))$

/ 2

by (rule *proots-upper-cindex-eq*[OF leadq qball])

also have $\dots = (\text{degree } p - \text{proots-count } p \{x. \text{Im } x=0\})$

```

      - cindex-poly-ubd (map-poly Im p) (map-poly Re p)) / 2
proof -
  have degree q = degree p - roots-count p {x. Im x=0}
proof -
  have degree p = degree q + degree r
    unfolding pqr
    using ⟨p ≠ 0⟩ degree-monic-mult leadq pqr by auto
  moreover have degree r = roots-count p {x. Im x=0}
    unfolding degree-roots-count roots-count-def
  proof (rule sum.cong)
    fix x assume x ∈ roots-within p {x. Im x = 0}
    then have Im x=0 by auto
    then have order x q = 0
      using qball order-0I by blast
    then show order x r = order x p
      using ⟨p≠0⟩ unfolding pqr by (simp add: order-mult)
  next
    show roots r = roots-within p {x. Im x = 0}
      unfolding pqr roots-within-times using qball rball by auto
  qed
  ultimately show ?thesis by auto
qed
moreover have cindex-poly-ubd (map-poly Im q) (map-poly Re q)
  = cindex-poly-ubd (map-poly Im p) (map-poly Re p)
proof -
  define iq rq ip rp where iq = map-poly Im q and rq = map-poly Re q
    and ip = map-poly Im p and rp = map-poly Re p
  have cindexE (- x) x (λx. poly iq x / poly rq x)
    = cindexE (- x) x (λx. poly ip x / poly rp x) for x
  proof -
    have lead-coeff r = 1
      using pqr leadq ⟨lead-coeff p=1⟩ by (simp add: coeff-degree-mult)
    then have cindexE (- x) x (λt. Im (poly p (t *R 1)) / Re (poly p (t *R
1))) =
      cindexE (- x) x (λt. Im (poly q (t *R 1)) / Re (poly q (t *R 1)))
    using cindexE-roots-on-horizontal-border[OF pqr, of 0 -x x 1
      ,unfolded linepath-def comp-def, simplified] rball by simp
    then show ?thesis
      unfolding scaleR-conv-of-real iq-def ip-def rq-def rp-def
      by (simp add: Im-poly-of-real Re-poly-of-real)
  qed
  then have ∀F r::real in at-top. (cindex-poly-ubd iq rq) = cindex-poly-ubd ip
rp
    using eventually-conj[OF cindex-poly-ubd-eventually[of iq rq]
      cindex-poly-ubd-eventually[of ip rp]] cindex-poly-ubd-def by auto
  then show ?thesis
    by (simp flip: iq-def rq-def ip-def rp-def)
qed
ultimately show ?thesis by auto

```

qed
 finally show ?thesis by simp
 qed

lemma *proots-within-upper-squarefree*:

assumes *rsquarefree p*

shows $\text{card} (\text{proots-within } p \{x. \text{Im } x > 0\}) = (\text{let}$
 $\text{pp} = \text{smult} (\text{inverse} (\text{lead-coeff } p)) p;$
 $\text{pI} = \text{map-poly } \text{Im } \text{pp};$
 $\text{pR} = \text{map-poly } \text{Re } \text{pp};$
 $g = \text{gcd } \text{pR } \text{pI}$

in

$\text{nat} ((\text{degree } p - \text{changes-R-smods } g (\text{pderiv } g) - \text{changes-R-smods } \text{pR}$
 $\text{pI}) \text{div } 2)$
 $)$

proof –

define *pp* where $\text{pp} = \text{smult} (\text{inverse} (\text{lead-coeff } p)) p$

define *pI* where $\text{pI} = \text{map-poly } \text{Im } \text{pp}$

define *pR* where $\text{pR} = \text{map-poly } \text{Re } \text{pp}$

define *g* where $g = \text{gcd } \text{pR } \text{pI}$

have $\text{card} (\text{proots-within } p \{x. \text{Im } x > 0\}) = \text{proots-upper } p$

unfolding *proots-upper-def* **using** *card-proots-within-rsquarefree[OF assms]* **by**
auto

also have $\dots = \text{proots-upper } \text{pp}$

unfolding *proots-upper-def pp-def*

apply (*subst proots-count-smult*)

using *assms* **by** *auto*

also have $\dots = (\text{degree } \text{pp} - \text{proots-count } \text{pp} \{x. \text{Im } x = 0\} - \text{cindex-poly-ubd}$
 $\text{pI } \text{pR}) \text{div } 2$

proof –

define *rr* where $\text{rr} = \text{proots-count } \text{pp} \{x. \text{Im } x = 0\}$

define *cpp* where $\text{cpp} = \text{cindex-poly-ubd } \text{pI } \text{pR}$

have $*: \text{proots-upper } \text{pp} = (\text{degree } \text{pp} - \text{rr} - \text{cpp}) / 2$

using *assms cpp-def pI-def pR-def pp-def proots-upper-cindex-eq' rr-def* **by**

auto

also have $\dots = (\text{degree } \text{pp} - \text{rr} - \text{cpp}) \text{div } 2$

proof (*subst real-of-int-div*)

define *tt* where $\text{tt} = \text{int} (\text{degree } \text{pp} - \text{rr}) - \text{cpp}$

have *real-of-int tt=2*proots-upper pp*

by (*simp add:*[folded tt-def]*)

then show *even tt* **by** (*metis dvd-triv-left even-of-nat of-int-eq-iff of-int-of-nat-eq*)

qed *simp*

finally show ?thesis **unfolding** *rr-def cpp-def* **by** *simp*

qed

also have $\dots = (\text{degree } \text{pp} - \text{changes-R-smods } g (\text{pderiv } g)$
 $- \text{cindex-poly-ubd } \text{pI } \text{pR}) \text{div } 2$

proof –

have *rsquarefree pp*

```

    using assms rsquarefree-smult-iff unfolding pp-def
    by (metis inverse-eq-imp-eq inverse-zero leading-coeff-neq-0 rsquarefree-0)
from card-roots-within-rsquarefree[OF this]
have roots-count pp {x. Im x = 0} = card (roots-within pp {x. Im x = 0})
    by simp
also have ... = card (roots-within pp (unbounded-line 0 1))
proof -
    have {x. Im x = 0} = unbounded-line 0 1
        unfolding unbounded-line-def
        by (simp add: complex-eq-iff)
    then show ?thesis by simp
qed
also have ... = changes-R-smods g (pderiv g)
    unfolding card-roots-unbounded-line[of 0 1 pp,simplified,folded pI-def pR-def]
g-def
    by (auto simp add:Let-def sturm-R[symmetric])
finally have roots-count pp {x. Im x = 0} = changes-R-smods g (pderiv g) .
moreover have degree pp ≥ roots-count pp {x. Im x = 0}
    by (metis <rsquarefree pp> roots-count-leq-degree rsquarefree-0)
ultimately show ?thesis
    by auto
qed
also have ... = (degree p - changes-R-smods g (pderiv g) - changes-R-smods
pR pI) div 2
    using cindex-poly-ubd-code unfolding pp-def by simp
finally have card (roots-within p {x. 0 < Im x}) = (degree p - changes-R-smods
g (pderiv g) -
        changes-R-smods pR pI) div 2 .
then show ?thesis unfolding Let-def
    by (simp add: pp-def flip: pp-def pR-def pI-def g-def)
qed

lemma roots-upper-code1[code]:
    roots-upper p =
      (if p ≠ 0 then
        (let pp=smult (inverse (lead-coeff p)) p;
          pI=map-poly Im pp;
          pR=map-poly Re pp;
          g = gcd pI pR
        in
          nat ((degree p - nat (changes-R-smods-ext g (pderiv g)) - changes-R-smods
pR pI) div 2)
        )
      else
        Code.abort (STR "roots-upper fails when p=0.") (λ-. roots-upper p))
proof -
define pp where pp = smult (inverse (lead-coeff p)) p
define pI where pI = map-poly Im pp
define pR where pR=map-poly Re pp

```

```

define g where  $g = \text{gcd } pI \ pR$ 
have ?thesis when  $p=0$ 
  using that by auto
moreover have ?thesis when  $p \neq 0$ 
proof –
  have  $pp \neq 0$  unfolding pp-def using that by auto
  define rr where  $rr = \text{int } (\text{degree } pp - \text{roots-count } pp \ \{x. \text{Im } x = 0\}) -$ 
cindex-poly-ubd pI pR
  have lead-coeff  $p \neq 0$  using  $\langle p \neq 0 \rangle$  by simp
  then have roots-upper  $pp = rr / 2$  unfolding rr-def
  by (simp add: pI-def pR-def pp-def roots-upper-cindex-eq)
  then have roots-upper  $pp = \text{nat } (rr \ \text{div } 2)$  by linarith
  moreover have
     $rr = \text{degree } p - \text{nat } (\text{changes-R-smods-ext } g \ (\text{pderiv } g)) - \text{changes-R-smods}$ 
pR pI
  proof –
    have  $\text{degree } pp = \text{degree } p$  unfolding pp-def by auto
    moreover have roots-count  $pp \ \{x. \text{Im } x = 0\} = \text{nat } (\text{changes-R-smods-ext}$ 
g (pderiv g))
    proof –
      have  $\{x. \text{Im } x = 0\} = \text{unbounded-line } 0 \ 1$ 
      unfolding unbounded-line-def by (simp add: complex-eq-iff)
      then show ?thesis
      using roots-unbounded-line[of  $0 \ 1$  pp,simplified, folded pI-def pR-def]
\langle pp \neq 0 \rangle
      by (auto simp:Let-def g-def gcd.commute)
    qed
  moreover have cindex-poly-ubd pI pR = changes-R-smods pR pI
  using cindex-poly-ubd-code by auto
  ultimately show ?thesis unfolding rr-def by auto
qed
moreover have roots-upper  $pp = \text{roots-upper } p$ 
  unfolding pp-def roots-upper-def
  by (meson  $\langle \text{lead-coeff } p \neq 0 \rangle$  inverse-nonzero-iff-nonzero roots-count-smult)
ultimately show ?thesis
  unfolding Let-def using that
  by (simp flip: pp-def pI-def pR-def g-def)
qed
ultimately show ?thesis by blast
qed

lemma roots-upper-card-code[code]:
  roots-upper-card  $p = (\text{if } p=0 \ \text{then } 0 \ \text{else}$ 
    (let
       $pf = p \ \text{div } (\text{gcd } p \ (\text{pderiv } p));$ 
       $pp = \text{smult } (\text{inverse } (\text{lead-coeff } pf)) \ pf;$ 
       $pI = \text{map-poly } \text{Im } pp;$ 
       $pR = \text{map-poly } \text{Re } pp;$ 
       $g = \text{gcd } pR \ pI$ 

```

```

      in
        nat ((degree pf - changes-R-smods g (pderiv g) - changes-R-smods pR
pI) div 2)
      ))
proof (cases p=0)
  case True
  then show ?thesis unfolding proots-upper-card-def using infinite-halfspace-Im-gt
by auto
next
  case False
  define pf pp pI pR g where
    pf = p div (gcd p (pderiv p))
  and pp = smult (inverse (lead-coeff pf)) pf
  and pI = map-poly Im pp
  and pR = map-poly Re pp
  and g = gcd pR pI
  have proots-upper-card p = proots-upper-card pf
  proof -
    have proots-within p {x. 0 < Im x} = proots-within pf {x. 0 < Im x}
    unfolding proots-within-def using poly-gcd-pderiv-iff[of p,folded pf-def]
    by auto
    then show ?thesis unfolding proots-upper-card-def by auto
  qed
  also have ... = nat ((degree pf - changes-R-smods g (pderiv g) - changes-R-smods
pR pI) div 2)
  using proots-within-upper-squarefree[OF rsquarefree-gcd-pderiv[OF ‹p≠0›
,unfolded Let-def,folded pf-def,folded pp-def pI-def pR-def g-def]
  unfolding proots-upper-card-def by blast
  finally show ?thesis unfolding Let-def
  using False
  by (simp flip: pf-def pp-def pI-def pR-def g-def)
qed

```

2.14 Polynomial roots on a general half-plane

the number of roots of polynomial p , counted with multiplicity, on the left half plane of the vector $b - a$.

definition *proots-half* :: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat **where**
proots-half p a b = proots-count p {w. Im ((w-a) / (b-a)) > 0}

lemma *proots-half-empty*:

assumes a=b

shows *proots-half* p a b = 0

unfolding *proots-half-def* **using** assms **by auto**

lemma *proots-half-proots-upper*:

assumes a≠b p≠0

shows *proots-half* p a b = *proots-upper* (pcompose p [:a, (b-a):])

```

proof –
  define  $q$  where  $q \equiv [ :a, (b - a) : ]$ 
  define  $f$  where  $f \equiv (\lambda x. (b-a)*x + a)$ 
  have  $(\sum_{r \in \text{roots-within } p} \{w. \text{Im } ((w-a) / (b-a)) > 0\}. \text{order } r \ p)$ 
     $= (\sum_{r \in \text{roots-within } (p \circ_p q)} \{z. 0 < \text{Im } z\}. \text{order } r \ (p \circ_p q))$ 
  proof (rule sum.reindex-cong[of f])
    have inj f
      using assms unfolding f-def inj-on-def by fastforce
    then show inj-on f (roots-within (p \circ_p q) {z. 0 < Im z})
      using inj-on-subset by blast
  next
  show  $\text{roots-within } p \ \{w. \text{Im } ((w-a) / (b-a)) > 0\} = f \ ' \ \text{roots-within } (p \circ_p q) \ \{z. 0 < \text{Im } z\}$ 
  proof safe
    fix  $x$  assume  $x\text{-asm}: x \in \text{roots-within } p \ \{w. \text{Im } ((w-a) / (b-a)) > 0\}$ 
    define  $xx$  where  $xx = (x - a) / (b - a)$ 
    have poly (p \circ_p q) xx = 0
      unfolding q-def xx-def poly-pcompose using assms x-asm by auto
    moreover have  $\text{Im } xx > 0$ 
      unfolding xx-def using x-asm by auto
    ultimately have  $xx \in \text{roots-within } (p \circ_p q) \ \{z. 0 < \text{Im } z\}$  by auto
    then show  $x \in f \ ' \ \text{roots-within } (p \circ_p q) \ \{z. 0 < \text{Im } z\}$ 
      unfolding f-def xx-def using assms by (force intro: image-eqI)
  next
  fix  $x$  assume  $x \in \text{roots-within } (p \circ_p q) \ \{z. 0 < \text{Im } z\}$ 
  then show  $f \ x \in \text{roots-within } p \ \{w. 0 < \text{Im } ((w-a) / (b - a))\}$ 
    unfolding f-def q-def using assms
    apply (simp add: poly-pcompose)
    by (auto simp add: algebra-simps)
  qed
next
  fix  $x$  assume  $x \in \text{roots-within } (p \circ_p q) \ \{z. 0 < \text{Im } z\}$ 
  show  $\text{order } (f \ x) \ p = \text{order } x \ (p \circ_p q)$  using  $\langle p \neq 0 \rangle$ 
  proof (induct p rule: poly-root-induct-alt)
    case  $0$ 
      then show ?case by simp
  next
  case (no-roots p)
  have  $\text{order } (f \ x) \ p = 0$ 
    by (simp add: no-roots.hyps order-0I)
  moreover have  $\text{order } x \ (p \circ_p q) = 0$ 
    by (simp add: no-roots.hyps order-0I poly-pcompose)
  ultimately show ?case by auto
next
  case (root c p)
  have  $\text{order } (f \ x) \ ([ : -c, 1 : ] * p) = \text{order } (f \ x) \ [ : -c, 1 : ] + \text{order } (f \ x) \ p$ 
    using order-mult root.premis by blast
  also have  $\dots = \text{order } x \ ([ : -c, 1 : ] \circ_p q) + \text{order } x \ (p \circ_p q)$ 
  proof –

```

```

have order (f x) [: - c, 1:] = order x ([: - c, 1:] ∘p q)
proof (cases f x=c)
  case True
    have [: - c, 1:] ∘p q = smult (b-a) [: -x, 1:]
      using True unfolding q-def f-def pcompose-pCons by auto
    then have order x ([: - c, 1:] ∘p q) = order x (smult (b-a) [: -x, 1:])
      by auto
    then have order x ([: - c, 1:] ∘p q) = 1
      by (metis assms(1) eq-iff-diff-eq-0 order-linear order-smult)
    moreover have order (f x) [: - c, 1:] = 1
      using True order-power-n-n[of - 1, simplified] by auto
    ultimately show ?thesis by auto
  next
    case False
      have order (f x) [: - c, 1:] = 0
        by (simp add: False)
      moreover have poly ([: - c, 1:] ∘p q) x ≠ 0
        using False unfolding f-def q-def poly-pcompose by auto
      ultimately show ?thesis
        by (metis order-0I)
    qed
  moreover have order (f x) p = order x (p ∘p q)
    using root by fastforce
  ultimately show ?thesis by auto
qed
also have ... = order x (([: - c, 1:] * p) ∘p q)
  unfolding pcompose-mult q-def
  by (metis One-nat-def assms(1) degree-pCons-eq-if zero-less-one order-mult
    pCons-eq-0-iff pcompose-eq-0-iff pcompose-hom.hom-mult right-minus-eq
    root.prem)
  finally show ?case .
qed
qed
then show ?thesis unfolding proots-half-def proots-upper-def proots-count-def
q-def
  by auto
qed

lemma proots-half-code1[code]:
  proots-half p a b = (if a≠b then
    if p≠0 then proots-upper (p ∘p [:a, b - a:])
    else Code.abort (STR "proots-half fails when p=0.")
    (λ-. proots-half p a b)
  else 0)

using proots-half-empty proots-half-proots-upper by auto

end

theory Count-Circle imports

```

Count-Half-Plane

begin

2.15 Polynomial roots within a circle (open ball)

definition *proots-ball* :: *complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-ball *p z0 r* = *proots-count* *p* (*ball* *z0 r*)

— Roots counted WITHOUT multiplicity

definition *proots-ball-card* :: *complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-ball-card *p z0 r* = *card* (*proots-within* *p* (*ball* *z0 r*))

lemma *proots-ball-code1*[*code*]:

proots-ball *p z0 r* = (*if* $r \leq 0$ *then*
 0
 else if $p \neq 0$ *then*
 proots-upper (*fcompose* (*p* \circ_p [*z0*, *of-real* *r*:]) [*i*, -1:] [*i*, 1:])
 else
 Code.abort (*STR* "proots-ball fails when p=0."
 (λ -. *proots-ball* *p z0 r*)

)
proof (*cases* $p=0 \vee r \leq 0$)

case *False*

have *proots-ball* *p z0 r* = *proots-count* (*p* \circ_p [*z0*, *of-real* *r*:]) (*ball* 0 1)

unfolding *proots-ball-def*

using *False* *proots-uball-eq* **by** *auto*

also have ... = *proots-upper* (*fcompose* (*p* \circ_p [*z0*, *of-real* *r*:]) [*i*, -1:] [*i*, 1:])

unfolding *proots-upper-def*

apply (*rule* *proots-ball-plane-eq*[*THEN* *arg-cong*])

using *False* *pcompose-eq-0*[*of* *p* [*z0*, *of-real* *r*:]]

by (*simp* *add*: *pcompose-eq-0-iff*)

finally show *?thesis* **using** *False* **by** *auto*

qed (*auto* *simp*:*proots-ball-def* *ball-empty*)

lemma *proots-ball-card-code1*[*code*]:

proots-ball-card *p z0 r* =
 (*if* $r \leq 0 \vee p=0$ *then*
 0
 else
 proots-upper-card (*fcompose* (*p* \circ_p [*z0*, *of-real* *r*:]) [*i*, -1:] [*i*, 1:])
)

proof (*cases* $p=0 \vee r \leq 0$)

case *True*

moreover have *?thesis* **when** $r \leq 0$

proof —

have *proots-within* *p* (*ball* *z0 r*) = {}

by (*simp* *add*: *ball-empty* *that*)

then show *?thesis* **unfolding** *proots-ball-card-def* **using** *that* **by** *auto*

qed

moreover have *?thesis when* $r > 0 \ p = 0$
unfolding *proots-ball-card-def using that infinite-ball[*of r z0*]*
by *auto*
ultimately show *?thesis by argo*
next
case *False*
then have $p \neq 0 \ r > 0$ **by** *auto*

have *proots-ball-card* $p \ z0 \ r = \text{card} (\text{proots-within} (p \circ_p [:z0, \text{of-real } r:])) (\text{ball } 0 \ 1))$
unfolding *proots-ball-card-def*
by (*rule proots-card-uball-eq[OF <r>0>, THEN arg-cong]*)
also have $\dots = \text{proots-upper-card} (\text{fcompose} (p \circ_p [:z0, \text{of-real } r:])) [i, -1:] [i, 1:]$
unfolding *proots-upper-card-def*
apply (*rule proots-card-ball-plane-eq[THEN arg-cong]*)
using *False pcompose-eq-0[*of p [:z0, of-real r:]*] by (simp add: pcompose-eq-0-iff)*
finally show *?thesis using False by auto*
qed

2.16 Polynomial roots on a circle (sphere)

definition *proots-sphere::complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat where*
proots-sphere $p \ z0 \ r = \text{proots-count } p (\text{sphere } z0 \ r)$

— Roots counted WITHOUT multiplicity

definition *proots-sphere-card ::complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat where*
proots-sphere-card $p \ z0 \ r = \text{card} (\text{proots-within } p (\text{sphere } z0 \ r))$

lemma *proots-sphere-card-code1[code]:*

proots-sphere-card $p \ z0 \ r =$
 $(\text{if } r = 0 \ \text{then}$
 $(\text{if } \text{poly } p \ z0 = 0 \ \text{then } 1 \ \text{else } 0)$
 $\text{else if } r < 0 \ \vee \ p = 0 \ \text{then}$
 0
 else
 $(\text{if } \text{poly } p (z0 - r) = 0 \ \text{then } 1 \ \text{else } 0) +$
 $\text{proots-unbounded-line-card} (\text{fcompose} (p \circ_p [:z0, \text{of-real } r:])) [i, -1:]$
 $[i, 1:]$
 $0 \ 1$
 $)$

proof —

have *?thesis when* $r = 0$

proof —

have *proots-within* $p \ \{z0\} = (\text{if } \text{poly } p \ z0 = 0 \ \text{then } \{z0\} \ \text{else } \{\})$

by *auto*

then show *?thesis unfolding proots-sphere-card-def using that by simp*

qed

moreover have *?thesis when* $r \neq 0 \ r < 0 \ \vee \ p = 0$

proof —

```

have ?thesis when  $r < 0$ 
proof -
  have  $\text{proots-within } p \text{ (sphere } z0 \ r) = \{\}$ 
  by (auto simp add: ball-empty that)
  then show ?thesis unfolding proots-sphere-card-def using that by auto
qed
moreover have ?thesis when  $r > 0 \ p = 0$ 
  unfolding proots-sphere-card-def using that infinite-sphere[of  $r \ z0$ ]
  by auto
ultimately show ?thesis using that by argo
qed
moreover have ?thesis when  $r > 0 \ p \neq 0$ 
proof -
  define  $pp$  where  $pp = p \circ_p \[:z0, \text{ of-real } r\:]$ 
  define  $ppp$  where  $ppp = fcompose \ pp \ \[:i, -1\:] \ \[:i, 1\:]$ 

  have  $pp \neq 0$  unfolding pp-def using that pcompose-eq-0
  by force

  have  $\text{proots-sphere-card } p \ z0 \ r = \text{card} \ (\text{proots-within } pp \ (\text{sphere } 0 \ 1))$ 
  unfolding proots-sphere-card-def pp-def
  by (rule proots-card-usphere-eq[OF  $\langle r > 0 \rangle$ , THEN arg-cong])
  also have  $\dots = \text{card} \ (\text{proots-within } pp \ \{-1\} \cup \text{proots-within } pp \ (\text{sphere } 0 \ 1 - \{-1\}))$ 
  by (simp add: insert-absorb proots-within-union)
  also have  $\dots = \text{card} \ (\text{proots-within } pp \ \{-1\}) + \text{card} \ (\text{proots-within } pp \ (\text{sphere } 0 \ 1 - \{-1\}))$ 
  apply (rule card-Un-disjoint)
  using  $\langle pp \neq 0 \rangle$  by auto
  also have  $\dots = \text{card} \ (\text{proots-within } pp \ \{-1\}) + \text{card} \ (\text{proots-within } ppp \ \{x. \ 0 = \text{Im } x\})$ 
  using proots-card-sphere-axis-eq[OF  $\langle pp \neq 0 \rangle$ , folded ppp-def] by simp
  also have  $\dots = (\text{if } \text{poly } p \ (z0 - r) = 0 \ \text{then } 1 \ \text{else } 0) + \text{proots-unbounded-line-card } ppp \ 0 \ 1$ 
  proof -
    have  $\text{proots-within } pp \ \{-1\} = (\text{if } \text{poly } p \ (z0 - r) = 0 \ \text{then } \{-1\} \ \text{else } \{\})$ 
    unfolding pp-def by (auto simp: poly-pcompose)
    then have  $\text{card} \ (\text{proots-within } pp \ \{-1\}) = (\text{if } \text{poly } p \ (z0 - r) = 0 \ \text{then } 1 \ \text{else } 0)$ 
    by auto
    moreover have  $\{x. \ \text{Im } x = 0\} = \text{unbounded-line } 0 \ 1$ 
    unfolding unbounded-line-def
    apply auto
    by (metis complex-is-Real-iff of-real-Re of-real-def)
    then have  $\text{card} \ (\text{proots-within } ppp \ \{x. \ 0 = \text{Im } x\}) = \text{proots-unbounded-line-card } ppp \ 0 \ 1$ 
    unfolding proots-unbounded-line-card-def by simp
  ultimately show ?thesis by auto
qed

```

```

finally show ?thesis
  apply (fold pp-def, fold ppp-def)
  using that by auto
qed
ultimately show ?thesis by auto
qed

```

2.17 Polynomial roots on a closed ball

definition *proots-cball*::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-cball *p* *z0* *r* = *proots-count* *p* (*cball* *z0* *r*)

— Roots counted WITHOUT multiplicity

definition *proots-cball-card* ::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-cball-card *p* *z0* *r* = *card* (*proots-within* *p* (*cball* *z0* *r*))

lemma *proots-cball-card-code1* [*code*]:

```

proots-cball-card p z0 r =
  ( if r=0 then
    ( if poly p z0=0 then 1 else 0)
    else if r < 0  $\vee$  p=0 then
      0
    else
      ( let pp=fcompose (p  $\circ_p$  [z0, of-real r:]) [i, -1:] [i, 1:]
        in
          ( if poly p (z0-r) =0 then 1 else 0)
          + proots-unbounded-line-card pp 0 1
          + proots-upper-card pp
        )
      )
  )

```

proof —

have ?thesis **when** *r*=0

proof —

have *proots-within* *p* {*z0*} = (*if* *poly* *p* *z0* = 0 *then* {*z0*} *else* {}) **by** *auto*

then show ?thesis **unfolding** *proots-cball-card-def* **using** that **by** *simp*
qed

moreover have ?thesis **when** *r*≠0 *r* < 0 \vee *p*=0

proof —

have ?thesis **when** *r*<0

proof —

have *proots-within* *p* (*cball* *z0* *r*) = {} **by** (*auto simp add: ball-empty that*)

then show ?thesis **unfolding** *proots-cball-card-def* **using** that **by** *auto*
qed

moreover have ?thesis **when** *r*>0 *p*=0

unfolding *proots-cball-card-def* **using** that *infinite-cball*[*of* *r* *z0*] **by** *auto*

```

ultimately show ?thesis using that by argo
qed
moreover have ?thesis when  $p \neq 0$   $r > 0$ 
proof -
  define pp where  $pp = fcompose (p \circ_p [:z0, of-real r:]) [:i, -1:] [:i, 1:]$ 

  have  $proots-cball-card\ p\ z0\ r = card (proots-within\ p (sphere\ z0\ r) \cup proots-within\ p (ball\ z0\ r))$ 
    unfolding  $proots-cball-card-def$ 
    apply (simp add:proots-within-union)
    by (metis Diff-partition cball-diff-sphere sphere-cball)
  also have ... =  $card (proots-within\ p (sphere\ z0\ r)) + card (proots-within\ p (ball\ z0\ r))$ 
    apply (rule card-Un-disjoint)
    using  $\langle p \neq 0 \rangle$  by auto
  also have ... =  $(if\ poly\ p\ (z0 - r) = 0\ then\ 1\ else\ 0) + proots-unbounded-line-card\ pp\ 0\ 1 + proots-upper-card\ pp$ 
    using  $proots-sphere-card-code1 [of\ p\ z0\ r, folded\ pp-def, unfolded\ proots-sphere-card-def]$ 
       $proots-ball-card-code1 [of\ p\ z0\ r, folded\ pp-def, unfolded\ proots-ball-card-def]$ 
      that
    by simp
  finally show ?thesis
    apply (fold pp-def)
    using that by auto
qed
ultimately show ?thesis by auto
qed
end

```

```

theory Count-Rectangle imports Count-Line
begin

```

Counting roots in a rectangular area can be in a purely algebraic approach without introducing (analytic) winding number (*winding-number*) nor the argument principle ($\llbracket open\ ?S; connected\ ?S; ?f\ holomorphic-on\ ?S - ?poles; ?h\ holomorphic-on\ ?S; valid-path\ ?g; pathfinish\ ?g = pathstart\ ?g; path-image\ ?g \subseteq ?S - \{w \in ?S. ?f\ w = 0 \vee w \in ?poles\}; \forall z. z \notin ?S \longrightarrow winding-number\ ?g\ z = 0; finite\ \{w \in ?S. ?f\ w = 0 \vee w \in ?poles\}; \forall p \in ?S \cap ?poles. is-pole\ ?f\ p \rrbracket \implies contour-integral\ ?g\ (\lambda x. deriv\ ?f\ x * ?h\ x / ?f\ x) = complex-of-real\ (2 * pi) * i * (\sum p \in \{w \in ?S. ?f\ w = 0 \vee w \in ?poles\}. winding-number\ ?g\ p * ?h\ p * complex-of-int\ (zorder\ ?f\ p))$). This has been illustrated by Michael Eisermann [1]. We lightly make use of *winding-number* here only to shorten the proof of one of the technical lemmas.

2.18 Misc

lemma *proots-count-const*:
assumes $c \neq 0$
shows *proots-count* $[:c:] s = 0$
unfolding *proots-count-def* **using** *assms* **by** *auto*

lemma *proots-count-nzero*:
assumes $\bigwedge x. x \in s \implies \text{poly } p \ x \neq 0$
shows *proots-count* $p \ s = 0$
unfolding *proots-count-def*
by (*rule sum.neutral*) (*use assms in auto*)

lemma *complex-box-ne-empty*:
fixes $a \ b :: \text{complex}$
shows
 $\text{cbox } a \ b \neq \{\}$ $\longleftrightarrow (Re \ a \leq Re \ b \wedge Im \ a \leq Im \ b)$
 $\text{box } a \ b \neq \{\}$ $\longleftrightarrow (Re \ a < Re \ b \wedge Im \ a < Im \ b)$
by (*auto simp add: box-ne-empty Basis-complex-def*)

2.19 Counting roots in a rectangle

definition *proots-rect* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
proots-rect $p \ lb \ ub = \text{proots-count } p \ (\text{box } lb \ ub)$

definition *proots-crect* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
proots-crect $p \ lb \ ub = \text{proots-count } p \ (\text{cbox } lb \ ub)$

definition *proots-rect-ll* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
proots-rect-ll $p \ lb \ ub = \text{proots-count } p \ (\text{box } lb \ ub \cup \{lb\}$
 $\cup \text{open-segment } lb \ (\text{Complex } (Re \ ub) \ (Im \ lb))$
 $\cup \text{open-segment } lb \ (\text{Complex } (Re \ lb) \ (Im \ ub)))$

definition *proots-rect-border* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
proots-rect-border $p \ a \ b = \text{proots-count } p \ (\text{path-image } (\text{rectpath } a \ b))$

definition *not-rect-vertex* $:: \text{complex} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{bool}$ **where**
not-rect-vertex $r \ a \ b = (r \neq a \wedge r \neq \text{Complex } (Re \ b) \ (Im \ a) \wedge r \neq b \wedge r \neq \text{Complex } (Re \ a) \ (Im \ b))$

definition *not-rect-vanishing* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{bool}$ **where**
not-rect-vanishing $p \ a \ b = (\text{poly } p \ a \neq 0 \wedge \text{poly } p \ (\text{Complex } (Re \ b) \ (Im \ a)) \neq 0$
 $\wedge \text{poly } p \ b \neq 0 \wedge \text{poly } p \ (\text{Complex } (Re \ a) \ (Im \ b)) \neq 0)$

lemma *cindexP-rectpath-edge-base*:
assumes $Re \ a < Re \ b \ Im \ a < Im \ b$
and *not-rect-vertex* $r \ a \ b$
and $r \in \text{path-image } (\text{rectpath } a \ b)$
shows *cindexP-pathE* $[-r, 1:] (\text{rectpath } a \ b) = -1$
proof –

```

have r-nzero: $r \neq a$   $r \neq \text{Complex } (\text{Re } b) (\text{Im } a)$   $r \neq b$   $r \neq \text{Complex } (\text{Re } a) (\text{Im } b)$ 
  using  $\langle \text{not-rect-vertex } r \ a \ b \rangle$  unfolding not-rect-vertex-def by auto

define rr where  $rr = [-r, 1:]$ 
have rr-linepath:cindexP-pathE rr (linepath a b)
  = cindex-pathE (linepath ( $a - r$ ) ( $b - r$ )) 0 for a b
  unfolding rr-def
  unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
  by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real algebra-simps)

have cindexP-pathE-eq:cindexP-pathE rr (rectpath a b) =
  cindexP-pathE rr (linepath a ( $\text{Complex } (\text{Re } b) (\text{Im } a)$ ))
  + cindexP-pathE rr (linepath ( $\text{Complex } (\text{Re } b) (\text{Im } a)$ ) b)
  + cindexP-pathE rr (linepath b ( $\text{Complex } (\text{Re } a) (\text{Im } b)$ ))
  + cindexP-pathE rr (linepath ( $\text{Complex } (\text{Re } a) (\text{Im } b)$ ) a)
  unfolding rectpath-def Let-def
  by ((subst cindex-poly-pathE-joinpaths
    |subst finite-ReZ-segments-joinpaths
    |intro path-poly-comp conjI);
    (simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
      pathfinish-compose pathstart-compose poly-pcompose)?)+)

have ( $\text{Im } r = \text{Im } a \wedge \text{Re } a < \text{Re } r \wedge \text{Re } r < \text{Re } b$ )
   $\vee (\text{Re } r = \text{Re } b \wedge \text{Im } a < \text{Im } r \wedge \text{Im } r < \text{Im } b)$ 
   $\vee (\text{Im } r = \text{Im } b \wedge \text{Re } a < \text{Re } r \wedge \text{Re } r < \text{Re } b)$ 
   $\vee (\text{Re } r = \text{Re } a \wedge \text{Im } a < \text{Im } r \wedge \text{Im } r < \text{Im } b)$ 
proof –
  have  $r \in \text{closed-segment } a \ (\text{Complex } (\text{Re } b) (\text{Im } a))$ 
     $\vee r \in \text{closed-segment } (\text{Complex } (\text{Re } b) (\text{Im } a)) \ b$ 
     $\vee r \in \text{closed-segment } b \ (\text{Complex } (\text{Re } a) (\text{Im } b))$ 
     $\vee r \in \text{closed-segment } (\text{Complex } (\text{Re } a) (\text{Im } b)) \ a$ 
  using  $\langle r \in \text{path-image } (\text{rectpath } a \ b) \rangle$ 
  unfolding rectpath-def Let-def
  by (subst (asm) path-image-join; simp) +
  then show ?thesis
  by (smt (verit, del-insts) assms(1) assms(2) r-nzero
    closed-segment-commute closed-segment-imp-Re-Im(1) closed-segment-imp-Re-Im(2)
    complex.sel(1) complex.sel(2) complex-eq-iff)
qed
moreover have cindexP-pathE rr (rectpath a b) = -1
  if  $\text{Im } r = \text{Im } a$   $\text{Re } a < \text{Re } r$   $\text{Re } r < \text{Re } b$ 
proof –
  have cindexP-pathE rr (linepath a ( $\text{Complex } (\text{Re } b) (\text{Im } a)$ )) = 0
  unfolding rr-linepath
  apply (rule cindex-pathE-linepath-on)
  using closed-segment-degen-complex(2) that(1) that(2) that(3) by auto

```

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ b)\ (Im\ a))\ b) = 0$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(1)\ that(3)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ b\ (Complex\ (Re\ a)\ (Im\ b))) = -1$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $assms(2)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(2)\ that(1)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ a)\ (Im\ b))\ a) = 0$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(1)\ that(2)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

ultimately show $?thesis$ **unfolding** $cindexP\text{-}pathE\text{-}eq$ **by** $auto$
qed

moreover have $cindexP\text{-}pathE\ rr\ (rectpath\ a\ b) = -1$
if $Re\ r = Re\ b\ Im\ a < Im\ r\ Im\ r < Im\ b$
proof $-$

have $cindexP\text{-}pathE\ rr\ (linepath\ a\ (Complex\ (Re\ b)\ (Im\ a))) = -1/2$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(2)\ that(2)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ b)\ (Im\ a))\ b) = 0$
unfolding $rr\text{-}linepath$
apply $(rule\ cindex\text{-}pathE\text{-}linepath\text{-}on)$
using $closed\text{-}segment\text{-}degen\text{-}complex(1)\ that(1)\ that(2)\ that(3)$ **by** $auto$

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ b\ (Complex\ (Re\ a)\ (Im\ b))) = -1/2$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(2)\ that(3)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ a)\ (Im\ b))\ a) = 0$
unfolding $rr\text{-}linepath$
apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
subgoal using $assms(1)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(1)\ that(1)$ **by** $fastforce$
subgoal using $that\ assms$ **unfolding** $Let\text{-}def$ **by** $auto$
done

ultimately show $?thesis$ **unfolding** $cindexP\text{-}pathE\text{-}eq$ **by** $auto$
qed

moreover have $\text{cindexP-pathE } rr \text{ (rectpath } a \ b) = -1$
if $Im \ r = Im \ b \ Re \ a < Re \ r \ Re \ r < Re \ b$
proof –
have $\text{cindexP-pathE } rr \text{ (linepath } a \ (\text{Complex } (Re \ b) \ (Im \ a))) = -1$
unfolding $rr\text{-linepath}$
apply $(subst \ \text{cindex-pathE-linepath})$
subgoal using $assms(2) \ \text{closed-segment-imp-Re-Im}(2) \ \text{that}(1)$ **by** $fastforce$
subgoal using $that \ \text{assms}$ **unfolding** $Let\text{-def}$ **by** $auto$
done
moreover have $\text{cindexP-pathE } rr \text{ (linepath } (\text{Complex } (Re \ b) \ (Im \ a)) \ b) = 0$
unfolding $rr\text{-linepath}$
apply $(subst \ \text{cindex-pathE-linepath})$
subgoal using $\text{closed-segment-imp-Re-Im}(1) \ \text{that}(3)$ **by** $force$
subgoal using $that \ \text{assms}$ **unfolding** $Let\text{-def}$ **by** $auto$
done
moreover have $\text{cindexP-pathE } rr \text{ (linepath } b \ (\text{Complex } (Re \ a) \ (Im \ b))) = 0$
unfolding $rr\text{-linepath}$
apply $(rule \ \text{cindex-pathE-linepath-on})$
by $(smt \ (verit, \ del\text{-insts}) \ \text{Im-poly-hom.base.hom-zero} \ \text{Re-poly-hom.base.hom-zero}$

 $\text{closed-segment-commute} \ \text{closed-segment-degen-complex}(2) \ \text{complex.sel}(1)$
 $\text{complex.sel}(2) \ \text{minus-complex.simps}(1) \ \text{minus-complex.simps}(2) \ \text{that}(1)$
 $\text{that}(2) \ \text{that}(3))$
moreover have $\text{cindexP-pathE } rr \text{ (linepath } (\text{Complex } (Re \ a) \ (Im \ b)) \ a) = 0$
unfolding $rr\text{-linepath}$
apply $(subst \ \text{cindex-pathE-linepath})$
subgoal using $\text{closed-segment-imp-Re-Im}(1) \ \text{that}(2)$ **by** $fastforce$
subgoal using $that \ \text{assms}$ **unfolding** $Let\text{-def}$ **by** $auto$
done
ultimately show $?thesis$ **unfolding** cindexP-pathE-eq **by** $auto$
qed
moreover have $\text{cindexP-pathE } rr \text{ (rectpath } a \ b) = -1$
if $Re \ r = Re \ a \ Im \ a < Im \ r \ Im \ r < Im \ b$
proof –
have $\text{cindexP-pathE } rr \text{ (linepath } a \ (\text{Complex } (Re \ b) \ (Im \ a))) = -1/2$
unfolding $rr\text{-linepath}$
apply $(subst \ \text{cindex-pathE-linepath})$
subgoal using $\text{closed-segment-imp-Re-Im}(2) \ \text{that}(2)$ **by** $fastforce$
subgoal using $that \ \text{assms}$ **unfolding** $Let\text{-def}$ **by** $auto$
done
moreover have $\text{cindexP-pathE } rr \text{ (linepath } (\text{Complex } (Re \ b) \ (Im \ a)) \ b) = 0$
unfolding $rr\text{-linepath}$
apply $(subst \ \text{cindex-pathE-linepath})$
subgoal using $assms(1) \ \text{closed-segment-imp-Re-Im}(1) \ \text{that}(1)$ **by** $fastforce$
subgoal using $that \ \text{assms}$ **unfolding** $Let\text{-def}$ **by** $auto$
done
moreover have $\text{cindexP-pathE } rr \text{ (linepath } b \ (\text{Complex } (Re \ a) \ (Im \ b))) =$
 $-1/2$
unfolding $rr\text{-linepath}$

apply (*subst cindex-pathE-linepath*)
subgoal using *closed-segment-imp-Re-Im(2) that(3)* **by** *fastforce*
subgoal using *that assms unfolding Let-def* **by** *auto*
done
moreover have *cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a) = 0*
unfolding *rr-linepath*
apply (*rule cindex-pathE-linepath-on*)
by (*smt (verit) Im-poly-hom.base.hom-zero Re-poly-hom.base.hom-zero*
closed-segment-commute closed-segment-degen-complex(1) complex.sel(1)
complex.sel(2) minus-complex.simps(1) minus-complex.simps(2) that(1)
that(2) that(3))
ultimately show *?thesis unfolding cindexP-pathE-eq* **by** *auto*
qed
ultimately show *?thesis unfolding rr-def* **by** *auto*
qed

lemma *cindexP-rectpath-vertex-base:*
assumes *Re a < Re b Im a < Im b*
and \neg *not-rect-vertex r a b*
shows *cindexP-pathE [-r,1:] (rectpath a b) = -1/2*
proof –
have *r-cases:r=a \vee r=Complex (Re b) (Im a) \vee r=b \vee r=Complex (Re a) (Im b)*
using $\langle \neg$ *not-rect-vertex r a b* **unfolding** *not-rect-vertex-def* **by** *auto*
define *rr* **where** *rr = [-r,1:]*
have *rr-linepath:cindexP-pathE rr (linepath a b)*
= cindex-pathE (linepath (a - r) (b-r)) 0 **for** *a b*
unfolding *rr-def*
unfolding *cindexP-lineE-def cindexP-pathE-def poly-linepath-comp*
by (*simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-*
bra-simps)

have *cindexP-pathE-eq:cindexP-pathE rr (rectpath a b) =*
cindexP-pathE rr (linepath a (Complex (Re b) (Im a)))
+ cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b)
+ cindexP-pathE rr (linepath b (Complex (Re a) (Im b)))
+ cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)
unfolding *rectpath-def Let-def*
by (*(subst cindex-poly-pathE-joinpaths*
|subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?)**+**

have *cindexP-pathE rr (rectpath a b) = -1/2*
if *r=a*
proof –
have *cindexP-pathE rr (linepath a (Complex (Re b) (Im a))) = 0*

```

    unfolding rr-linepath
    apply (rule cindex-pathE-linepath-on)
    by (simp add: that)
  moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) = 0
    unfolding rr-linepath
    apply (subst cindex-pathE-linepath)
    subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce
    subgoal using that assms unfolding Let-def by auto
    done
  moreover have cindexP-pathE rr (linepath b (Complex (Re a) (Im b))) =
-1/2
    unfolding rr-linepath
    apply (subst cindex-pathE-linepath)
    subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
    subgoal using that assms unfolding Let-def by auto
    done
  moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a) = 0
    unfolding rr-linepath
    apply (rule cindex-pathE-linepath-on)
    by (simp add: that)
  ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
moreover have cindexP-pathE rr (rectpath a b) = -1/2
  if r=Complex (Re b) (Im a)
proof -
  have cindexP-pathE rr (linepath a (Complex (Re b) (Im a))) = 0
    unfolding rr-linepath
    apply (rule cindex-pathE-linepath-on)
    by (simp add: that)
  moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) = 0
    unfolding rr-linepath
    apply (rule cindex-pathE-linepath-on)
    by (simp add: that)
  moreover have cindexP-pathE rr (linepath b (Complex (Re a) (Im b))) =
-1/2
    unfolding rr-linepath
    apply (subst cindex-pathE-linepath)
    subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
    subgoal using that assms unfolding Let-def by auto
    done
  moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a) = 0
    unfolding rr-linepath
    apply (subst cindex-pathE-linepath)
    subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce
    subgoal by (smt (verit) complex.sel(1) minus-complex.simps(1))
    done
  ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
moreover have cindexP-pathE rr (rectpath a b) = -1/2

```

if $r=b$
proof –
 have $cindexP\text{-}pathE\ rr\ (linepath\ a\ (Complex\ (Re\ b)\ (Im\ a))) = -1/2$
 unfolding $rr\text{-}linepath$
 apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
 subgoal using $assms(2)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(2)$ **that by fastforce**
 subgoal using $assms(1)\ assms(2)$ **that by auto**
 done
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ b)\ (Im\ a))\ b) = 0$
 unfolding $rr\text{-}linepath$
 apply $(rule\ cindex\text{-}pathE\text{-}linepath\text{-}on)$
 by $(simp\ add:\ that)$
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ b\ (Complex\ (Re\ a)\ (Im\ b))) = 0$
 unfolding $rr\text{-}linepath$
 apply $(rule\ cindex\text{-}pathE\text{-}linepath\text{-}on)$
 by $(simp\ add:\ that)$
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ a)\ (Im\ b))\ a) = 0$
 unfolding $rr\text{-}linepath$
 apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
 subgoal using $assms(1)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(1)$ **that by fastforce**
 subgoal by $(smt\ (verit)\ complex.sel(1)\ minus\text{-}complex.simps(1))$
 done
 ultimately show *?thesis* **unfolding** $cindexP\text{-}pathE\text{-}eq$ **by auto**
qed
 moreover have $cindexP\text{-}pathE\ rr\ (rectpath\ a\ b) = -1/2$
 if $r=Complex\ (Re\ a)\ (Im\ b)$
proof –
 have $cindexP\text{-}pathE\ rr\ (linepath\ a\ (Complex\ (Re\ b)\ (Im\ a))) = -1/2$
 unfolding $rr\text{-}linepath$
 apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
 subgoal using $assms(2)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(2)$ **that by fastforce**
 subgoal using $assms(1)\ assms(2)$ **that by auto**
 done
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ b)\ (Im\ a))\ b) = 0$
 unfolding $rr\text{-}linepath$
 apply $(subst\ cindex\text{-}pathE\text{-}linepath)$
 subgoal using $assms(1)\ closed\text{-}segment\text{-}imp\text{-}Re\text{-}Im(1)$ **that by fastforce**
 subgoal by $(smt\ (verit)\ complex.sel(1)\ minus\text{-}complex.simps(1))$
 done
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ b\ (Complex\ (Re\ a)\ (Im\ b))) = 0$
 unfolding $rr\text{-}linepath$
 apply $(rule\ cindex\text{-}pathE\text{-}linepath\text{-}on)$
 by $(simp\ add:\ that)$
 moreover have $cindexP\text{-}pathE\ rr\ (linepath\ (Complex\ (Re\ a)\ (Im\ b))\ a) = 0$
 unfolding $rr\text{-}linepath$
 apply $(rule\ cindex\text{-}pathE\text{-}linepath\text{-}on)$
 by $(simp\ add:\ that)$
 ultimately show *?thesis* **unfolding** $cindexP\text{-}pathE\text{-}eq$ **by auto**
qed

ultimately show *?thesis* **using** *r-cases* **unfolding** *rr-def* **by** *auto*
qed

lemma *cindexP-rectpath-interior-base*:

assumes *r∈box a b*

shows *cindexP-pathE* $[-r,1:]$ (*rectpath a b*) = -2

proof –

have *inbox:Re r* $\in \{Re\ a <..<Re\ b\} \wedge Im\ r \in \{Im\ a <..<Im\ b\}$

using $\langle r \in box\ a\ b \rangle$ **unfolding** *in-box-complex-iff* **by** *auto*

then have *r-zero:r≠a r≠Complex (Re b) (Im a) r≠b r≠Complex (Re a) (Im b)*

by *auto*

have *Re a < Re b Im a < Im b*

using $\langle r \in box\ a\ b \rangle$ *complex-box-ne-empty* **by** *blast+*

define *rr* **where** *rr* = $[-r,1:]$

have *rr-linepath:cindexP-pathE rr (linepath a b)*

= *cindex-pathE* (*linepath (a - r) (b - r)*) 0 **for** *a b*

unfolding *rr-def*

unfolding *cindexP-lineE-def cindexP-pathE-def poly-linepath-comp*

by (*simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real algebra-simps*)

have *cindexP-pathE rr (rectpath a b)* =

cindexP-pathE rr (linepath a (Complex (Re b) (Im a)))

+ *cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b)*

+ *cindexP-pathE rr (linepath b (Complex (Re a) (Im b)))*

+ *cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)*

unfolding *rectpath-def Let-def*

by (*subst cindex-poly-pathE-joinpaths*

|*subst finite-ReZ-segments-joinpaths*

|*intro path-poly-comp conjI*);

(*simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join*

pathfinish-compose pathstart-compose poly-pcompose)?) +

also have ... = -2

proof –

have *cindexP-pathE rr (linepath a (Complex (Re b) (Im a)))* = -1

unfolding *rr-linepath*

apply (*subst cindex-pathE-linepath*)

subgoal using *closed-segment-imp-Re-Im(2) inbox* **by** *fastforce*

using *inbox* **by** *auto*

moreover have *cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b)* = 0

unfolding *rr-linepath*

apply (*subst cindex-pathE-linepath*)

subgoal using *closed-segment-imp-Re-Im(1) inbox* **by** *fastforce*

using *inbox* **by** *auto*

moreover have *cindexP-pathE rr (linepath b (Complex (Re a) (Im b)))* = -1

unfolding *rr-linepath*

```

    apply (subst cindex-pathE-linepath)
    subgoal using closed-segment-imp-Re-Im(2) inbox by fastforce
    using inbox by auto
  moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a) = 0
    unfolding rr-linepath
    apply (subst cindex-pathE-linepath)
    subgoal using closed-segment-imp-Re-Im(1) inbox by fastforce
    using inbox by auto
  ultimately show ?thesis by auto
qed
finally show ?thesis unfolding rr-def .
qed

```

lemma *cindexP-rectpath-outside-base*:

```

  assumes Re a < Re b Im a < Im b
    and rnotin_cbox a b
  shows cindexP-pathE [:-r,1:] (rectpath a b) = 0
proof -
  have not_cbox:¬ (Re r ∈ {Re a..Re b} ∧ Im r ∈ {Im a..Im b})
    using ⟨rnotin_cbox a b⟩ unfolding in_cbox_complex_iff by auto
  then have r_nzero:r≠a r≠Complex (Re b) (Im a) r≠b r≠Complex (Re a) (Im
b)
    using assms by auto

```

```

define rr where rr = [:-r,1:]
have rr-linepath:cindexP-pathE rr (linepath a b)
  = cindex-pathE (linepath (a - r) (b-r)) 0 for a b
  unfolding rr-def
  unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
  by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-
bra-simps)

```

```

have cindexP-pathE rr (rectpath a b) =
  cindexP-pathE rr (linepath a (Complex (Re b) (Im a)))
  + cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b)
  + cindexP-pathE rr (linepath b (Complex (Re a) (Im b)))
  + cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)
unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths
  |subst finite-ReZ-segments-joinpaths
  |intro path-poly-comp conjI);
  (simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join

```

```

  pathfinish-compose pathstart-compose poly-pcompose)?)+
have cindexP-pathE rr (rectpath a b) = cindex-pathE (poly rr ∘ rectpath a b) 0
  unfolding cindexP-pathE-def by simp
also have ... = - 2 * winding-number (poly rr ∘ rectpath a b) 0
  — We don't need winding-number to finish the proof, but thanks to Cauchy's

```

Index theorem (i.e., $\llbracket \text{finite-ReZ-segments } ?g \text{ } ?z; \text{ valid-path } ?g; ?z \notin \text{path-image } ?g; \text{ pathfinish } ?g = \text{pathstart } ?g \rrbracket \implies \text{winding-number } ?g \text{ } ?z = \text{complex-of-real } (-\text{cindex-pathE } ?g \text{ } ?z / 2)$) we can make the proof shorter.

```

proof –
  have winding-number (poly rr ◦ rectpath a b) 0
    = - cindex-pathE (poly rr ◦ rectpath a b) 0 / 2
proof (rule winding-number-cindex-pathE)
  show finite-ReZ-segments (poly rr ◦ rectpath a b) 0
    using finite-ReZ-segments-poly-rectpath .
  show valid-path (poly rr ◦ rectpath a b)
    using valid-path-poly-rectpath .
  show  $0 \notin \text{path-image}$  (poly rr ◦ rectpath a b)
  by (smt (verit) DiffE add.right-neutral add-diff-cancel-left' add-uminus-conv-diff

      assms(1) assms(2) assms(3) basic-cqe-conv1(1) diff-add-cancel imageE
mult.right-neutral
      mult-zero-right path-image-compose path-image-rectpath-cbox-minus-box
poly-pCons rr-def)
  show pathfinish (poly rr ◦ rectpath a b) = pathstart (poly rr ◦ rectpath a b)
    by (simp add: pathfinish-compose pathstart-compose)
  qed
  then show ?thesis by auto
qed
also have ... = 0
proof –
  have winding-number (poly rr ◦ rectpath a b) 0 = 0
proof (rule winding-number-zero-outside)
  have path-image (poly rr ◦ rectpath a b) = poly rr ‘ path-image (rectpath a b)
    using path-image-compose by simp
  also have ... = poly rr ‘ (cbox a b - box a b)
    apply (subst path-image-rectpath-cbox-minus-box)
    using assms(1,2) by (simp|blast)+
  also have ...  $\subseteq (\lambda x. x - r)$  ‘ cbox a b
    unfolding rr-def by (simp add: image-subset-iff)
  finally show path-image (poly rr ◦ rectpath a b)  $\subseteq (\lambda x. x - r)$  ‘ cbox a b .
  show  $0 \notin (\lambda x. x - r)$  ‘ cbox a b using assms(3) by force
  show path (poly rr ◦ rectpath a b) by (simp add: path-poly-comp)
  show convex (( $\lambda x. x - r$ ) ‘ cbox a b)
    using convex-box(1) convex-translation-subtract-eq by blast
  show pathfinish (poly rr ◦ rectpath a b) = pathstart (poly rr ◦ rectpath a b)
    by (simp add: pathfinish-compose pathstart-compose)
  qed
  then show ?thesis by simp
qed
finally show ?thesis unfolding rr-def by simp
qed

```

lemma *cindexP-rectpath-add-one-root*:
assumes $\text{Re } a < \text{Re } b$ $\text{Im } a < \text{Im } b$

```

and not-rect-vertex  $r$   $a$   $b$ 
and not-rect-vanishing  $p$   $a$   $b$ 
shows cindexP-pathE ( $[-r,1:] * p$ ) (rectpath  $a$   $b$ ) =
      cindexP-pathE  $p$  (rectpath  $a$   $b$ )
      + (if  $r \in \text{box } a$   $b$  then  $-2$  else if  $r \in \text{path-image } (\text{rectpath } a$   $b$ ) then  $-1$  else
0)
proof –
define  $rr$  where  $rr = [-r,1:]$ 
have  $rr$ -nzero:poly  $rr$   $a \neq 0$  poly  $rr$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $\neq 0$ 
      poly  $rr$   $b \neq 0$  poly  $rr$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $\neq 0$ 
using  $\langle \text{not-rect-vertex } r$   $a$   $b \rangle$  unfolding rr-def not-rect-vertex-def by auto

have  $p$ -nzero:poly  $p$   $a \neq 0$  poly  $p$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $\neq 0$ 
      poly  $p$   $b \neq 0$  poly  $p$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $\neq 0$ 
using  $\langle \text{not-rect-vanishing } p$   $a$   $b \rangle$  unfolding not-rect-vanishing-def by auto

define  $cindp$  where  $cindp = (\lambda p$   $a$   $b$ .
      cindexP-lineE  $p$   $a$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))
      + cindexP-lineE  $p$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $b$ 
      + cindexP-lineE  $p$   $b$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))
      + cindexP-lineE  $p$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $a$ 
      )
define  $cdiff$  where  $cdiff = (\lambda rr$   $p$   $a$   $b$ .
      cdiff-aux  $rr$   $p$   $a$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))
      + cdiff-aux  $rr$   $p$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $b$ 
      + cdiff-aux  $rr$   $p$   $b$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))
      + cdiff-aux  $rr$   $p$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $a$ 
      )

have cindexP-pathE ( $rr * p$ ) (rectpath  $a$   $b$ ) =
      cindexP-pathE ( $rr * p$ ) (linepath  $a$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ )))
      + cindexP-pathE ( $rr * p$ ) (linepath (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $b$ )
      + cindexP-pathE ( $rr * p$ ) (linepath  $b$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ )))
      + cindexP-pathE ( $rr * p$ ) (linepath (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $a$ )
unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths
      |subst finite-ReZ-segments-joinpaths
      |intro path-poly-comp conjI);
      (simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
      pathfinish-compose pathstart-compose poly-pcompose)?) +
also have ... = cindexP-lineE ( $rr * p$ )  $a$  (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))
      + cindexP-lineE ( $rr * p$ ) (Complex ( $Re$   $b$ ) ( $Im$   $a$ ))  $b$ 
      + cindexP-lineE ( $rr * p$ )  $b$  (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))
      + cindexP-lineE ( $rr * p$ ) (Complex ( $Re$   $a$ ) ( $Im$   $b$ ))  $a$ 
unfolding cindexP-lineE-def by simp
also have ... =  $cindp$   $rr$   $a$   $b$  +  $cindp$   $p$   $a$   $b$  +  $cdiff$   $rr$   $p$   $a$   $b$  / 2
unfolding cindp-def cdiff-def
by (subst cindexP-lineE-times;

```

(use *rr-nzero p-nzero one-complex.code imaginary-unit.code in simp*)?) +
also have ... = *cindexP-pathE p (rectpath a b) + (if r ∈ box a b then -2 else*
if r ∈ path-image (rectpath a b) then -1 else 0)
proof –
have *cindp rr a b = cindexP-pathE rr (rectpath a b)*
unfolding *rectpath-def Let-def cindp-def cindexP-lineE-def*
by ((*subst cindex-poly-pathE-joinpaths*
| *subst finite-ReZ-segments-joinpaths*
| *intro path-poly-comp conjI*);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?) +
also have ... = *(if r ∈ box a b then -2 else*
if r ∈ path-image (rectpath a b) then -1 else 0)
proof –
have *?thesis if r ∈ box a b*
using *cindexP-rectpath-interior-base rr-def that by presburger*
moreover have *?thesis if r ∉ box a b r ∈ path-image (rectpath a b)*
using *cindexP-rectpath-edge-base[OF assms(1,2,3)] that unfolding rr-def*
by auto
moreover have *?thesis if r ∉ box a b r ∉ path-image (rectpath a b)*
proof –
have *r ∉ cbox a b*
using *that assms(1) assms(2) path-image-rectpath-cbox-minus-box by auto*
then show *?thesis unfolding rr-def*
using *assms(1) assms(2) cindexP-rectpath-outside-base that(1) that(2)*
by presburger
qed
ultimately show *?thesis by auto*
qed
finally have *cindp rr a b = (if r ∈ box a b then -2 else*
if r ∈ path-image (rectpath a b) then -1 else 0) .
moreover have *cindp p a b = cindexP-pathE p (rectpath a b)*
unfolding *rectpath-def Let-def cindp-def cindexP-lineE-def*
by ((*subst cindex-poly-pathE-joinpaths*
| *subst finite-ReZ-segments-joinpaths*
| *intro path-poly-comp conjI*);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?) +
moreover have *cdiff rr p a b = 0*
unfolding *cdiff-def cdiff-aux-def by simp*
ultimately show *?thesis by auto*
qed
finally show *?thesis unfolding rr-def .*
qed
lemma *proots-rect-cindexP-pathE:*
assumes *Re a < Re b Im a < Im b*

```

    and not-rect-vanishing p a b
  shows roots-rect p a b = -(roots-rect-border p a b + cindexP-pathE p (rectpath
a b)) / 2
  using ⟨not-rect-vanishing p a b⟩
proof (induct p rule:poly-root-induct-alt)
  case 0
  then have False unfolding not-rect-vanishing-def by auto
  then show ?case by simp
next
  case (no-roots p)
  then obtain c where pc:p=[:c:] c≠0
    by (meson fundamental-theorem-of-algebra-alt)
  have cindexP-pathE p (rectpath a b) = 0
    using pc by (auto intro:cindexP-pathE-const)
  moreover have roots-rect p a b = 0 roots-rect-border p a b = 0
    using pc roots-count-const
    unfolding roots-rect-def roots-rect-border-def by auto
  ultimately show ?case by auto
next
  case (root r p)
  define rr where rr=[:-r,1:]

  have hyps:real (roots-rect p a b) =
    -(roots-rect-border p a b + cindexP-pathE p (rectpath a b)) / 2
  apply (rule root(1))
  by (meson not-rect-vanishing-def poly-mult-zero-iff root.premis)

  have cind-eq:cindexP-pathE (rr * p) (rectpath a b) =
    cindexP-pathE p (rectpath a b) +
    (if r ∈ box a b then - 2 else if r ∈ path-image (rectpath a b) then - 1
else 0)
  proof (rule cindexP-rectpath-add-one-root[OF assms(1,2),of r p,folded rr-def])
    show not-rect-vertex r a b
      using not-rect-vanishing-def not-rect-vertex-def root.premis by auto
    show not-rect-vanishing p a b
      using not-rect-vanishing-def root.premis by force
  qed

  have rect-eq:roots-rect (rr * p) a b = roots-rect p a b
    + (if r∈box a b then 1 else 0)
  proof -
    have roots-rect (rr * p) a b
      = roots-count rr (box a b) + roots-rect p a b
      unfolding roots-rect-def
      apply (rule roots-count-times)
      by (metis not-rect-vanishing-def poly-0 root.premis rr-def)
    moreover have roots-count rr (box a b) = (if r∈box a b then 1 else 0)
      using roots-count-pCons-1-iff rr-def by blast
    ultimately show ?thesis by auto

```

qed

have *border-eq:proots-rect-border* ($rr * p$) $a b =$
 proots-rect-border $p a b$
 + (if $r \in \text{path-image } (\text{rectpath } a b)$ then 1 else 0)

proof –

have *proots-rect-border* ($rr * p$) $a b = \text{proots-count } rr (\text{path-image } (\text{rectpath } a b))$
 + *proots-rect-border* $p a b$

unfolding *proots-rect-border-def*

apply (*rule proots-count-times*)

by (*metis not-rect-vanishing-def poly-0 root.premis rr-def*)

moreover have *proots-count* $rr (\text{path-image } (\text{rectpath } a b))$
 = (if $r \in \text{path-image } (\text{rectpath } a b)$ then 1 else 0)

using *proots-count-pCons-1-iff rr-def* **by** *blast*

ultimately show *?thesis* **by** *auto*

qed

have *?case* **if** $r \in \text{box } a b$

proof –

have *proots-rect* ($rr * p$) $a b = \text{proots-rect } p a b + 1$

unfolding *rect-eq* **using** *that* **by** *auto*

moreover have *proots-rect-border* ($rr * p$) $a b = \text{proots-rect-border } p a b$

unfolding *border-eq* **using** *that*

using *assms(1) assms(2) path-image-rectpath-cbox-minus-box* **by** *auto*

moreover have *cindexP-pathE* ($rr * p$) ($\text{rectpath } a b$) = *cindexP-pathE* p
 ($\text{rectpath } a b$) – 2

using *cind-eq* **that** **by** *auto*

ultimately show *?thesis* **using** *hyps*

by (*fold rr-def*) *simp*

qed

moreover have *?case* **if** $r \notin \text{box } a b$ $r \in \text{path-image } (\text{rectpath } a b)$

proof –

have *proots-rect* ($rr * p$) $a b = \text{proots-rect } p a b$

unfolding *rect-eq* **using** *that* **by** *auto*

moreover have *proots-rect-border* ($rr * p$) $a b = \text{proots-rect-border } p a b + 1$

unfolding *border-eq* **using** *that*

using *assms(1) assms(2) path-image-rectpath-cbox-minus-box* **by** *auto*

moreover have *cindexP-pathE* ($rr * p$) ($\text{rectpath } a b$) = *cindexP-pathE* p
 ($\text{rectpath } a b$) – 1

using *cind-eq* **that** **by** *auto*

ultimately show *?thesis* **using** *hyps*

by (*fold rr-def*) *auto*

qed

moreover have *?case* **if** $r \notin \text{box } a b$ $r \notin \text{path-image } (\text{rectpath } a b)$

proof –

have *proots-rect* ($rr * p$) $a b = \text{proots-rect } p a b$

unfolding *rect-eq* **using** *that* **by** *auto*

moreover have *proots-rect-border* ($rr * p$) $a b = \text{proots-rect-border } p a b$

unfolding *border-eq* **using** *that*
using *assms(1) assms(2) path-image-rectpath-cbox-minus-box* **by** *auto*
moreover **have** *cindexP-pathE (rr * p) (rectpath a b) = cindexP-pathE p*
(rectpath a b)
using *cind-eq* **that** **by** *auto*
ultimately **show** *?thesis* **using** *hyp*
by *(fold rr-def) auto*
qed
ultimately **show** *?case* **by** *auto*
qed

2.20 Code generation

lemmas *Complex-minus-eq = minus-complex.code*

lemma *cindexP-pathE-rect-smods:*

fixes *p::complex poly* **and** *lb ub::complex*
assumes *ab-le:Re lb < Re ub Im lb < Im ub*
and *not-rect-vanishing p lb ub*
shows *cindexP-pathE p (rectpath lb ub) =*
(let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb):];
pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub):];
pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
(changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
) / 2) (is ?L=?R)

proof –

have *cindexP-pathE p (rectpath lb ub) =*
cindexP-lineE p lb (Complex (Re ub) (Im lb))
+ cindexP-lineE (p) (Complex (Re ub) (Im lb)) ub
+ cindexP-lineE (p) ub (Complex (Re lb) (Im ub))
+ cindexP-lineE (p) (Complex (Re lb) (Im ub)) lb

unfolding *rectpath-def Let-def cindexP-lineE-def*

by *((subst cindex-poly-pathE-joinpaths*

|subst finite-ReZ-segments-joinpaths

|intro path-poly-comp conjI);

(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join

pathfinish-compose pathstart-compose poly-pcompose)?)+

```

also have ... = ?R
  apply (subst (1 2 3 4) cindexP-lineE-changes)
  subgoal using assms(3) not-rect-vanishing-def by fastforce
  subgoal by (smt (verit) assms(2) complex.sel(2))
  subgoal by (metis assms(1) complex.sel(1) order-less-irrefl)
  subgoal by (smt (verit) assms(2) complex.sel(2))
  subgoal by (metis assms(1) complex.sel(1) order-less-irrefl)
  subgoal unfolding Let-def by (simp-all add: Complex-minus-eq)
done
finally show ?thesis .
qed

lemma open-segment-Im-equal:
  assumes Re x ≠ Re y Im x = Im y
  shows open-segment x y = {z. Im z = Im x
    ∧ Re z ∈ open-segment (Re x) (Re y)}

proof -
  have open-segment x y = (λu. (1 - u) *R x + u *R y) ‘ {0 <..< 1}
  unfolding open-segment-image-interval
  using assms by auto
  also have ... = (λu. Complex (Re x + u * (Re y - Re x))
    (Im y)) ‘ {0 <..< 1}
  apply (subst (1 2 3 4) complex-surj[symmetric])
  using assms by (simp add: scaleR-conv-of-real algebra-simps)
  also have ... = {z. Im z = Im x ∧ Re z ∈ open-segment (Re x) (Re y)}
proof -
  have Re x + u * (Re y - Re x) ∈ open-segment (Re x) (Re y)
  if Re x ≠ Re y Im x = Im y 0 < u u < 1 for u
proof -
  define yx where yx = Re y - Re x
  have Re y = yx + Re x yx > 0 ∨ yx < 0
  unfolding yx-def using that by auto
  then show ?thesis
  unfolding open-segment-eq-real-ivl
  using that mult-pos-neg by auto
qed
moreover have z ∈ (λxa. Complex (Re x + xa * (Re y - Re x)) (Im y))
  ‘ {0 <..< 1}
  if Im x = Im y Im z = Im y Re z ∈ open-segment (Re x) (Re y) for z
  apply (rule rev-image-eqI[of (Re z - Re x)/(Re y - Re x)])
  subgoal
  using that unfolding open-segment-eq-real-ivl
  by (auto simp: divide-simps)
  subgoal using ⟨Re x ≠ Re y⟩ complex-eq-iff that(2) by auto
  done
  ultimately show ?thesis using assms by auto
qed
finally show ?thesis .
qed

```

lemma *open-segment-Re-equal*:
assumes $Re\ x = Re\ y$ $Im\ x \neq Im\ y$
shows $open\text{-}segment\ x\ y = \{z. Re\ z = Re\ x$
 $\wedge Im\ z \in open\text{-}segment\ (Im\ x)\ (Im\ y)\}$

proof –
have $open\text{-}segment\ x\ y = (\lambda u. (1 - u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y) \text{ ‘ } \{0 <..< 1\}$
unfolding *open-segment-image-interval*
using *assms* **by** *auto*
also have $\dots = (\lambda u. Complex\ (Re\ y)\ (Im\ x + u * (Im\ y - Im\ x))$
 $\text{ ‘ } \{0 <..< 1\}$
apply (*subst* (1 2 3 4) *complex-surj[symmetric]*)
using *assms* **by** (*simp add:scaleR-conv-of-real-algebra-simps*)
also have $\dots = \{z. Re\ z = Re\ x \wedge Im\ z \in open\text{-}segment\ (Im\ x)\ (Im\ y)\}$
proof –
have $Im\ x + u * (Im\ y - Im\ x) \in open\text{-}segment\ (Im\ x)\ (Im\ y)$
if $Im\ x \neq Im\ y$ $Re\ x = Re\ y$ $0 < u$ $u < 1$ **for** u
proof –
define yx **where** $yx = Im\ y - Im\ x$
have $Im\ y = yx + Im\ x$ $yx > 0 \vee yx < 0$
unfolding *yx-def* **using** *that* **by** *auto*
then show *?thesis*
unfolding *open-segment-eq-real-ivl*
using *that mult-pos-neg* **by** *auto*
qed
moreover have $z \in (\lambda xa. Complex\ (Re\ y)\ (Im\ x + xa * (Im\ y - Im\ x)))$
 $\text{ ‘ } \{0 <..< 1\}$
if $Re\ x = Re\ y$ $Re\ z = Re\ y$ $Im\ z \in open\text{-}segment\ (Im\ x)\ (Im\ y)$ **for** z
apply (*rule rev-image-eqI[of (Im z - Im x)/(Im y - Im x)]*)
subgoal
using *that* **unfolding** *open-segment-eq-real-ivl*
by (*auto simp:divide-simps*)
subgoal using $\langle Im\ x \neq Im\ y \rangle$ *complex-eq-iff that(2)* **by** *auto*
done
ultimately show *?thesis* **using** *assms* **by** *auto*
qed
finally show *?thesis* .
qed

lemma *Complex-eq-iff*:
 $x = Complex\ y\ z \iff Re\ x = y \wedge Im\ x = z$
 $Complex\ y\ z = x \iff Re\ x = y \wedge Im\ x = z$
by *auto*

lemma *proots-rect-border-eq-lines*:
fixes $p::complex$ **poly** **and** $lb\ ub::complex$
assumes $ab\text{-}le: Re\ lb < Re\ ub$ $Im\ lb < Im\ ub$
and $not\text{-}van: not\text{-}rect\text{-}vanishing\ p\ lb\ ub$
shows $proots\text{-}rect\text{-}border\ p\ lb\ ub =$

```

    roots-line p lb (Complex (Re ub) (Im lb))
      + roots-line p (Complex (Re ub) (Im lb)) ub
      + roots-line p ub (Complex (Re lb) (Im ub))
      + roots-line p (Complex (Re lb) (Im ub)) lb
proof –
  have p≠0
    using not-rect-vanishing-def not-van order-root by blast

define l1 l2 l3 l4 where l1 = open-segment lb (Complex (Re ub) (Im lb))
  and l2 = open-segment (Complex (Re ub) (Im lb)) ub
  and l3 = open-segment ub (Complex (Re lb) (Im ub))
  and l4 = open-segment (Complex (Re lb) (Im ub)) lb

have ll-eq:
  l1 = {z. Im z ∈ {Im lb} ∧ Re z ∈ {Re lb<..subgoal unfolding l1-def
    apply (subst open-segment-Im-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  subgoal unfolding l2-def
    apply (subst open-segment-Re-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  subgoal unfolding l3-def
    apply (subst open-segment-Im-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  subgoal unfolding l4-def
    apply (subst open-segment-Re-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  done

have ll-disj: l1 ∩ l2 = {} l1 ∩ l3 = {} l1 ∩ l4 = {}
  l2 ∩ l3 = {} l2 ∩ l4 = {} l3 ∩ l4 = {}
  using assms unfolding ll-eq by auto

have roots-rect-border p lb ub = roots-count p
  ({z. Re z ∈ {Re lb, Re ub} ∧ Im z ∈ {Im lb..Im ub}} ∪
   {z. Im z ∈ {Im lb, Im ub} ∧ Re z ∈ {Re lb..Re ub}})
  unfolding roots-rect-border-def
  apply (subst path-image-rectpath)
  using assms(1,2) by auto
also have ... = roots-count p
  ({z. Re z ∈ {Re lb, Re ub} ∧ Im z ∈ {Im lb<..apply (rule arg-cong2[where f=roots-count])
  unfolding not-rect-vanishing-def using assms(1,2) complex.exhaust-sel
  by (auto simp add:order.order-iff-strict intro:complex-eqI)
also have ... = roots-count p

```

```

      ({z. Re z ∈ {Re lb, Re ub} ∧ Im z ∈ {Im lb<..apply (subst roots-count-union-disjoint)
using ⟨p≠0⟩ by auto
also have ... = roots-count p
      ({z. Re z ∈ {Re lb, Re ub} ∧ Im z ∈ {Im lb<..proof -
  have roots-count p
      ({lb,Complex (Re ub) (Im lb), ub,Complex (Re lb) (Im ub)}) = 0
  apply (rule roots-count-nzero)
  using not-van unfolding not-rect-vanishing-def by auto
  then show ?thesis by auto
qed
also have ... = roots-count p (l1 ∪ l2 ∪ l3 ∪ l4)
  apply (rule arg-cong2[where f=roots-count])
  unfolding ll-eq by auto
also have ... = roots-count p l1
      + roots-count p l2
      + roots-count p l3
      + roots-count p l4
  using ll-disj ⟨p≠0⟩
  by (subst roots-count-union-disjoint;
      (simp add:Int-Un-distrib Int-Un-distrib2 )?)
also have ... = roots-line p lb (Complex (Re ub) (Im lb))
      + roots-line p (Complex (Re ub) (Im lb)) ub
      + roots-line p ub (Complex (Re lb) (Im ub))
      + roots-line p (Complex (Re lb) (Im ub)) lb
  unfolding roots-line-def l1-def l2-def l3-def l4-def by simp-all
  finally show ?thesis .
qed

lemma roots-rect-border-smods:
  fixes p::complex poly and lb ub::complex
  assumes ab-le:Re lb < Re ub Im lb < Im ub
  and not-van:not-rect-vanishing p lb ub
  shows roots-rect-border p lb ub =
    (let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
      pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
      p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb):];
      pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
      p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
      pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
      p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub):];
      pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4

```

```

    in
      nat ( changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
          + changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
          + changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
          + changes-itv-smods-ext 0 1 gc4 (pderiv gc4)
          ) ) (is ?L=?R)
proof -
  have roots-rect-border p lb ub = roots-line p lb (Complex (Re ub) (Im lb))
    + roots-line p (Complex (Re ub) (Im lb)) ub
    + roots-line p ub (Complex (Re lb) (Im ub))
    + roots-line p (Complex (Re lb) (Im ub)) lb
  apply (rule roots-rect-border-eq-lines)
  by fact+
  also have ... = ?R
proof -
  define p1 pR1 pI1 gc1 C1 where pp1:
    p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:]
    pR1 = map-poly Re p1
    pI1 = map-poly Im p1
    gc1 = gcd pR1 pI1
  and
    C1=changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
  define p2 pR2 pI2 gc2 C2 where pp2:
    p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im lb):]
    pR2 = map-poly Re p2
    pI2 = map-poly Im p2
    gc2 = gcd pR2 pI2
  and
    C2=changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
  define p3 pR3 pI3 gc3 C3 where pp3:
    p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:]
    pR3 = map-poly Re p3
    pI3 = map-poly Im p3
    gc3 = gcd pR3 pI3
  and
    C3=changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
  define p4 pR4 pI4 gc4 C4 where pp4:
    p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im ub):]
    pR4 = map-poly Re p4
    pI4 = map-poly Im p4
    gc4 = gcd pR4 pI4
  and
    C4=changes-itv-smods-ext 0 1 gc4 (pderiv gc4)

  have poly gc1 0 ≠ 0 poly gc1 1 ≠ 0
    poly gc2 0 ≠ 0 poly gc2 1 ≠ 0
    poly gc3 0 ≠ 0 poly gc3 1 ≠ 0
    poly gc4 0 ≠ 0 poly gc4 1 ≠ 0
  unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff

```

```

using not-van[unfolded not-rect-vanishing-def]
by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
      ; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+

have roots-line p lb (Complex (Re ub) (Im lb)) = nat C1
apply (subst roots-line-smods)
using not-van assms(1,2)
unfolding not-rect-vanishing-def C1-def pp1 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have roots-line p (Complex (Re ub) (Im lb)) ub = nat C2
apply (subst roots-line-smods)
using not-van assms(1,2)
unfolding not-rect-vanishing-def C2-def pp2 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have roots-line p ub (Complex (Re lb) (Im ub)) = nat C3
apply (subst roots-line-smods)
using not-van assms(1,2)
unfolding not-rect-vanishing-def C3-def pp3 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have roots-line p (Complex (Re lb) (Im ub)) lb = nat C4
apply (subst roots-line-smods)
using not-van assms(1,2)
unfolding not-rect-vanishing-def C4-def pp4 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have C1 ≥ 0 C2 ≥ 0 C3 ≥ 0 C4 ≥ 0
unfolding C1-def C2-def C3-def C4-def
by (rule changes-itv-smods-ext-geq-0;(fact|simp))+
ultimately have roots-line p lb (Complex (Re ub) (Im lb))
      + roots-line p (Complex (Re ub) (Im lb)) ub
      + roots-line p ub (Complex (Re lb) (Im ub))
      + roots-line p (Complex (Re lb) (Im ub)) lb
      = nat (C1+C2+C3+C4)

by linarith
also have ... = ?R
unfolding C1-def C2-def C3-def C4-def pp1 pp2 pp3 pp4 Let-def
by simp
finally show ?thesis .
qed
finally show ?thesis .
qed

lemma roots-rect-smods:
assumes Re lb < Re ub Im lb < Im ub
and not-van:not-rect-vanishing p lb ub
shows roots-rect p lb ub = (
  let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
      pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
      p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb):];

```

```

    pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
    p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
    pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
    p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub):];
    pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
    nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
+ 2*changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
+ 2*changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
+ 2*changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
+ 2*changes-itv-smods-ext 0 1 gc4 (pderiv gc4)) div 4)
)
proof -
define p1 pR1 pI1 gc1 C1 D1 where pp1:
    p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:]
    pR1 = map-poly Re p1
    pI1 = map-poly Im p1
    gc1 = gcd pR1 pI1
    and C1=changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
    and D1=changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
define p2 pR2 pI2 gc2 C2 D2 where pp2:
    p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im lb):]
    pR2 = map-poly Re p2
    pI2 = map-poly Im p2
    gc2 = gcd pR2 pI2
    and C2=changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
    and D2=changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
define p3 pR3 pI3 gc3 C3 D3 where pp3:
    p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:]
    pR3 = map-poly Re p3
    pI3 = map-poly Im p3
    gc3 = gcd pR3 pI3
    and C3=changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
    and D3=changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
define p4 pR4 pI4 gc4 C4 D4 where pp4:
    p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im ub):]
    pR4 = map-poly Re p4
    pI4 = map-poly Im p4
    gc4 = gcd pR4 pI4
    and C4=changes-itv-smods-ext 0 1 gc4 (pderiv gc4)
    and D4=changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
have poly gc1 0 ≠ 0 poly gc1 1 ≠ 0
    poly gc2 0 ≠ 0 poly gc2 1 ≠ 0
    poly gc3 0 ≠ 0 poly gc3 1 ≠ 0
    poly gc4 0 ≠ 0 poly gc4 1 ≠ 0

```

```

unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff
using not-van[unfolded not-rect-vanishing-def]
by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
      ; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+
have C1≥0 C2≥0 C3≥0 C4≥0
unfolding C1-def C2-def C3-def C4-def
by (rule changes-itv-smods-ext-geq-0;(fact|simp))+

define CC DD where CC=C1 + C2 + C3 + C4
      and DD=D1 + D2 + D3 + D4

have real (proots-rect p lb ub) = - (real (proots-rect-border p lb ub)
      + cindexP-pathE p (rectpath lb ub)) / 2
apply (rule proots-rect-cindexP-pathE)
by fact+
also have ... = -(nat CC + DD / 2) / 2
proof -
have proots-rect-border p lb ub = nat CC
apply (rule proots-rect-border-smods[
      of lb ub p,
      unfolded Let-def,
      folded pp1 pp2 pp3 pp4,
      folded C1-def C2-def C3-def C4-def,
      folded CC-def])
by fact+
moreover have cindexP-pathE p (rectpath lb ub) = (real-of-int DD) / 2
apply (rule cindexP-pathE-rect-smods[
      of lb ub p,
      unfolded Let-def,
      folded pp1 pp2 pp3 pp4,
      folded D1-def D2-def D3-def D4-def,
      folded DD-def])
by fact+
ultimately show ?thesis by auto
qed
also have ... = - (DD + 2*CC) / 4
by (simp add: CC-def ⟨0 ≤ C1⟩ ⟨0 ≤ C2⟩ ⟨0 ≤ C3⟩ ⟨0 ≤ C4⟩)
finally have real (proots-rect p lb ub)
      = real-of-int (- (DD + 2 * CC)) / 4 .
then have proots-rect p lb ub = nat (- (DD + 2 * CC) div 4)
by simp
then show ?thesis unfolding Let-def
apply (fold pp1 pp2 pp3 pp4)
apply (fold C1-def C2-def C3-def C4-def D1-def D2-def D3-def D4-def)
by (simp add:CC-def DD-def)
qed

```

lemma proots-rect-code[code]:

```

proots-rect p lb ub =
  (if Re lb < Re ub ∧ Im lb < Im ub then
    if not-rect-vanishing p lb ub then
      (
        let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
            pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
            p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb):];
            pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
            p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
            pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
            p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub):];
            pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
          nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
+ 2*changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
+ 2*changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
+ 2*changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
+ 2*changes-itv-smods-ext 0 1 gc4 (pderiv gc4)) div 4)
        )
      else Code.abort (STR "proots-rect: the polynomial should not vanish
at the four vertices for now") (λ-. proots-rect p lb ub)
    else 0)
proof (cases Re lb < Re ub ∧ Im lb < Im ub ∧ not-rect-vanishing p lb ub)
case False
have ?thesis if ¬ (Re lb < Re ub) ∨ ¬ (Im lb < Im ub)
proof -
  have box lb ub = {} using that by (metis complex-box-ne-empty(2))
  then show ?thesis
  unfolding proots-rect-def
  using proots-count-empty that by fastforce
qed
then show ?thesis using False by auto
next
case True
then show ?thesis
  apply (subst proots-rect-smods)
  unfolding Let-def by simp-all
qed

lemma proots-rect-ll-rect:
assumes Re lb < Re ub Im lb < Im ub
and not-van:not-rect-vanishing p lb ub
shows proots-rect-ll p lb ub = proots-rect p lb ub
+ proots-line p lb (Complex (Re ub) (Im lb))

```

+ *proots-line* p lb (*Complex* (Re lb) (Im ub))

proof –

have $p \neq 0$

using *not-rect-vanishing-def not-van order-root* **by** *blast*

define $l1$ $l4$ **where** $l1 = \text{open-segment } lb$ (*Complex* (Re ub) (Im lb))
and $l4 = \text{open-segment } lb$ (*Complex* (Re lb) (Im ub))

have *ll-eq*:

$l1 = \{z. Im\ z \in \{Im\ lb\} \wedge Re\ z \in \{Re\ lb < .. < Re\ ub\}\}$

$l4 = \{z. Re\ z \in \{Re\ lb\} \wedge Im\ z \in \{Im\ lb < .. < Im\ ub\}\}$

subgoal unfolding *l1-def*

apply (*subst open-segment-Im-equal*)

using *assms unfolding open-segment-eq-real-ivl* **by** *auto*

subgoal unfolding *l4-def*

apply (*subst open-segment-Re-equal*)

using *assms unfolding open-segment-eq-real-ivl* **by** *auto*

done

have *ll-disj*: $l1 \cap l4 = \{\}$ *box* lb $ub \cap \{lb\} = \{\}$

$box\ lb\ ub \cap l1 = \{\}$ *box* lb $ub \cap l4 = \{\}$

$l1 \cap \{lb\} = \{\}$ $l4 \cap \{lb\} = \{\}$

using *assms unfolding ll-eq*

by (*auto simp: in-box-complex-iff*)

have *proots-rect-ll* p lb $ub = \text{proots-count } p$ (*box* lb ub)

+ *proots-count* p $\{lb\}$

+ *proots-count* p $l1$

+ *proots-count* p $l4$

unfolding *proots-rect-ll-def* **using** *ll-disj* $\langle p \neq 0 \rangle$

apply (*fold l1-def l4-def*)

by (*subst proots-count-union-disjoint*

; (simp add: Int-Un-distrib Int-Un-distrib2 del: Un-insert-right) ?) +

also have ... = *proots-rect* p lb ub

+ *proots-line* p lb (*Complex* (Re ub) (Im lb))

+ *proots-line* p lb (*Complex* (Re lb) (Im ub))

proof –

have *proots-count* p $\{lb\} = 0$

by (*metis not-rect-vanishing-def not-van proots-count-nzero singleton-iff*)

then show *?thesis*

unfolding *proots-rect-def l1-def l4-def proots-line-def* **by** *simp*

qed

finally show *?thesis* .

qed

lemma *proots-rect-ll-smods*:

assumes $Re\ lb < Re\ ub$ $Im\ lb < Im\ ub$

and *not-van: not-rect-vanishing* p lb ub

shows *proots-rect-ll* p lb $ub =$ (

```

    let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
        pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
        p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb):];
        pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
        p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
        pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
        p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub):];
        pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
    nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
- 2*changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
+ 2*changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
+ 2*changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
- 2*changes-itv-smods-ext 0 1 gc4 (pderiv gc4)) div 4))

proof -
  have p≠0
    using not-rect-vanishing-def not-van order-root by blast

define l1 l4 where l1 = open-segment lb (Complex (Re ub) (Im lb))
    and l4 = open-segment lb (Complex (Re lb) (Im ub))
have l4-alt:l4 = open-segment (Complex (Re lb) (Im ub)) lb
    unfolding l4-def by (simp add: open-segment-commute)

have ll-eq:
  l1 = {z. Im z ∈ {Im lb} ∧ Re z ∈ {Re lb<..subgoal unfolding l1-def
    apply (subst open-segment-Im-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  subgoal unfolding l4-def
    apply (subst open-segment-Re-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
  done

have ll-disj: l1 ∩ l4 = {} box lb ub ∩ {lb} = {}
  box lb ub ∩ l1 = {} box lb ub ∩ l4 = {}
  l1 ∩ {lb} = {} l4 ∩ {lb} = {}
  using assms unfolding ll-eq
  by (auto simp:in-box-complex-iff)

define p1 pR1 pI1 gc1 C1 D1 where pp1:
  p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:]
  pR1 = map-poly Re p1
  pI1 = map-poly Im p1

```

```

    gc1 = gcd pR1 pI1
  and C1=changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
  and D1=changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
define p2 pR2 pI2 gc2 C2 D2 where pp2:
  p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im lb):]
  pR2 = map-poly Re p2
  pI2 = map-poly Im p2
  gc2 = gcd pR2 pI2
  and C2=changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
  and D2=changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
define p3 pR3 pI3 gc3 C3 D3 where pp3:
  p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:]
  pR3 = map-poly Re p3
  pI3 = map-poly Im p3
  gc3 = gcd pR3 pI3
  and C3=changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
  and D3=changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
define p4 pR4 pI4 gc4 C4 D4 where pp4:
  p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im ub):]
  pR4 = map-poly Re p4
  pI4 = map-poly Im p4
  gc4 = gcd pR4 pI4
  and C4=changes-itv-smods-ext 0 1 gc4 (pderiv gc4)
  and D4=changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
have poly gc1 0 ≠ 0 poly gc1 1 ≠ 0
  poly gc2 0 ≠ 0 poly gc2 1 ≠ 0
  poly gc3 0 ≠ 0 poly gc3 1 ≠ 0
  poly gc4 0 ≠ 0 poly gc4 1 ≠ 0
  unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff
  using not-van[unfolded not-rect-vanishing-def]
  by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
      ; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+
have CC-pos:C1 ≥ 0 C2 ≥ 0 C3 ≥ 0 C4 ≥ 0
  unfolding C1-def C2-def C3-def C4-def
  by (rule changes-itv-smods-ext-geq-0;(fact|simp))+

define CC DD where CC= C2 + C3 - C4 - C1
  and DD=D1 + D2 + D3 + D4

define p1 p2 p3 p4 where pp:p1=proots-line p lb (Complex (Re ub) (Im lb))
  p2 = proots-line p (Complex (Re ub) (Im lb)) ub
  p3 = proots-line p ub (Complex (Re lb) (Im ub))
  p4 = proots-line p (Complex (Re lb) (Im ub)) lb
have p4-alt:p4 = proots-line p lb (Complex (Re lb) (Im ub))
  unfolding pp by (simp add: proots-line-commute)

have real (proots-rect-ll p lb ub) = real (proots-rect p lb ub) + p1 + p4
  unfolding pp by (simp add: proots-rect-ll-rect[OF assms] proots-line-commute)

```

also have ... = $(p1 + p4 - \text{real } p2 - \text{real } p3 - \text{cindexP-pathE } p (\text{rectpath } lb \text{ ub})) / 2$
proof –
have $\text{real } (\text{proots-rect } p \text{ lb } ub) = - (\text{real } (\text{proots-rect-border } p \text{ lb } ub) + \text{cindexP-pathE } p (\text{rectpath } lb \text{ ub})) / 2$
apply (rule *proots-rect-cindexP-pathE*)
by *fact+*
also have ... = $-(p1 + p2 + p3 + p4 + \text{cindexP-pathE } p (\text{rectpath } lb \text{ ub})) / 2$
using *proots-rect-border-eq-lines*[*OF assms, folded pp*] **by** *simp*
finally have $\text{real } (\text{proots-rect } p \text{ lb } ub) = - (\text{real } (p1 + p2 + p3 + p4) + \text{cindexP-pathE } p (\text{rectpath } lb \text{ ub})) / 2 .$
then show *?thesis* **by** *auto*
qed
also have ... = $(\text{nat } C1 + \text{nat } C4 - \text{real } (\text{nat } C2) - \text{real } (\text{nat } C3) - ((\text{real-of-int } DD) / 2)) / 2$
proof –
have $p1 = \text{nat } C1 \text{ } p2 = \text{nat } C2 \text{ } p3 = \text{nat } C3 \text{ } p4 = \text{nat } C4$
using *not-van*[*unfolded not-rect-vanishing-def*] *assms(1,2)*
unfolding *pp C1-def pp1 C2-def pp2 C3-def pp3 C4-def pp4*
by (*subst proots-line-smods* ; *simp-all add:Complex-eq-iff Let-def Complex-minus-eq*)+
moreover have $\text{cindexP-pathE } p (\text{rectpath } lb \text{ ub}) = (\text{real-of-int } DD) / 2$
apply (rule *cindexP-pathE-rect-smods*[*of lb ub p, unfolded Let-def, folded pp1 pp2 pp3 pp4, folded D1-def D2-def D3-def D4-def, folded DD-def*])
by *fact+*
ultimately show *?thesis* **by** *presburger*
qed
also have ... = $-(DD + 2 * CC) / 4$
unfolding *CC-def* **using** *CC-pos* **by** (*auto simp add:divide-simps algebra-simps*)
finally have $\text{real } (\text{proots-rect-ll } p \text{ lb } ub) = \text{real-of-int } (- (DD + 2 * CC)) / 4 .$
then have $\text{proots-rect-ll } p \text{ lb } ub = \text{nat } (- (DD + 2 * CC) \text{ div } 4)$
by *simp*
then show *?thesis*
unfolding *Let-def*
apply (*fold pp1 pp2 pp3 pp4*)
apply (*fold C1-def C2-def C3-def C4-def D1-def D2-def D3-def D4-def*)
by (*simp add:CC-def DD-def*)
qed
lemma *proots-rect-ll-code*[*code*]:
proots-rect-ll p lb ub =

```

(if Re lb < Re ub ∧ Im lb < Im ub then
  if not-rect-vanishing p lb ub then
    (
      let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0];
          pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
          p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb)];
          pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
          p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0];
          pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
          p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub)];
          pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
      nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)
- 2*changes-itv-smods-ext 0 1 gc1 (pderiv gc1)
+ 2*changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
+ 2*changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
- 2*changes-itv-smods-ext 0 1 gc4 (pderiv gc4)) div 4)
    )
  else Code.abort (STR "proots-rect-ll: the polynomial should not vanish
at the four vertices for now") (λ-. proots-rect-ll p lb ub)
else Code.abort (STR "proots-rect-ll: the box is improper")
(λ-. proots-rect-ll p lb ub))
proof (cases Re lb < Re ub ∧ Im lb < Im ub ∧ not-rect-vanishing p lb ub)
  case False
  then show ?thesis using False by auto
next
  case True
  then show ?thesis
  apply (subst proots-rect-ll-smods)
  unfolding Let-def by simp-all
qed
end

```

3 Procedures to count the number of complex roots in various areas

```

theory Count-Complex-Roots imports
  Count-Half-Plane
  Count-Line
  Count-Circle
  Count-Rectangle
begin

```

end

4 Some examples for complex root counting

```
theory Count-Complex-Roots-Examples
  imports Count-Complex-Roots
begin
```

```
value proots-rect [2*i,0,i] (Complex (-1) 0) (Complex 2 2)
```

```
value proots-rect [-1,-2*i,1:]
  (Complex (-1) 0) (Complex 2 2)
```

```
value proots-rect-ll [-1,1:]
  (Complex (-1) 0) (Complex 2 2)
```

```
value proots-half [1-i,2-i,1:]
  0 (Complex 0 1)
```

```
value proots-half [1-i,2-i,1:] (Complex 0 1) 0
```

```
value [code] proots-ball ([: -2,1:]*[: -2,1:]*[: -3,1:]) 0 4
```

end

5 Acknowledgements

The work was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council and led by Professor Lawrence Paulson at the University of Cambridge, UK.

References

- [1] M. Eisermann. The fundamental theorem of algebra made effective: An elementary real-algebraic proof via Sturm chains. *American Mathematical Monthly*, 119(9):715–752, 2012.
- [2] Q. I. Rahman and G. Schmeisser. *Analytic theory of polynomials*. Number 26. Oxford University Press, 2002.