

Count the Number of Complex Roots

Wenda Li

March 19, 2025

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 op [:a, b-a:]) (of-real t)  
  <proof>
```

```
lemma path-poly-comp[intro]:  
  fixes p::'a::real-normed-field poly  
  shows path g  $\implies$  path (poly p o g)  
  <proof>
```

```
lemma cindex-poly-noroot:  
  assumes a < b  $\forall x. a < x \wedge x < b \implies poly p x \neq 0$   
  shows cindex-poly a b q p = 0  
  <proof>
```

1.2 More polynomial homomorphism interpretations

interpretation *of-real-poly-hom:map-poly-inj-idom-hom of-real* $\langle proof \rangle$

interpretation *Re-poly-hom:map-poly-comm-monoid-add-hom Re*
 $\langle proof \rangle$

interpretation *Im-poly-hom:map-poly-comm-monoid-add-hom Im*
 $\langle proof \rangle$

1.3 More about *order*

lemma *order-normalize[simp]:order x (normalize p) = order x p*
 $\langle proof \rangle$

lemma *order-gcd:*
assumes $p \neq 0 \ q \neq 0$
shows $order\ x\ (gcd\ p\ q) = \min\ (order\ x\ p)\ (order\ x\ q)$
 $\langle proof \rangle$

lemma *pderiv-power: pderiv (p ^ n) = smult (of-nat n) (p ^ (n-1)) * pderiv p*
 $\langle proof \rangle$

lemma *order-pderiv:*
fixes $p::'a::\{idom,semiring-char-0\}$ *poly*
assumes $p \neq 0$ *poly p x=0*
shows $order\ x\ p = Suc\ (order\ x\ (pderiv\ p))$ $\langle proof \rangle$

1.4 More about *rsquarefree*

lemma *rsquarefree-0[simp]: \neg rsquarefree 0*
 $\langle proof \rangle$

lemma *rsquarefree-times:*
assumes *rsquarefree (p*q)*
shows *rsquarefree q* $\langle proof \rangle$

lemma *rsquarefree-smult-iff:*
assumes $s \neq 0$
shows $rsquarefree\ (smult\ s\ p) \longleftrightarrow rsquarefree\ p$
 $\langle proof \rangle$

lemma *card-roots-within-rsquarefree:*
assumes *rsquarefree p*
shows $roots-count\ p\ s = card\ (roots-within\ p\ s)$ $\langle proof \rangle$

lemma *rsquarefree-gcd-pderiv:*
fixes $p::'a::\{factorial-ring-gcd,semiring-gcd-mult-normalize,semiring-char-0\}$ *poly*
assumes $p \neq 0$

shows $rsquarefree (p \text{ div } (gcd p (pderiv p)))$
 ⟨proof⟩

lemma *poly-gcd-pderiv-iff*:
fixes $p::'a::\{semiring-char-0, factorial-ring-gcd, semiring-gcd-mult-normalize\}$ *poly*
shows $poly (p \text{ div } (gcd p (pderiv p))) x = 0 \longleftrightarrow poly p x = 0$
 ⟨proof⟩

1.5 Composition of a polynomial and a circular path

lemma *poly-circlepath-tan-eq*:
fixes $z0::complex$ **and** $r::real$ **and** $p::complex \text{ poly}$
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⟩

1.6 Combining two real polynomials into a complex one

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

lemma *cpoly-of-eq-0-iff*[*iff*]:
 $cpoly\text{-of } pR \ pI = 0 \longleftrightarrow pR = 0 \wedge pI = 0$
 ⟨proof⟩

lemma *cpoly-of-decompose*:
 $p = cpoly\text{-of } (map\text{-poly Re } p) (map\text{-poly Im } p)$
 ⟨proof⟩

lemma *cpoly-of-dist-right*:
 $cpoly\text{-of } (pR*g) (pI*g) = cpoly\text{-of } pR \ pI * (map\text{-poly of-real } g)$
 ⟨proof⟩

lemma *poly-cpoly-of-real*:
 $poly (cpoly\text{-of } pR \ pI) (of\text{-real } x) = Complex (poly pR x) (poly pI x)$
 ⟨proof⟩

lemma *poly-cpoly-of-real-iff*:
shows $poly (cpoly\text{-of } pR \ pI) (of\text{-real } t) = 0 \longleftrightarrow poly pR t = 0 \wedge poly pI t = 0$
 ⟨proof⟩

lemma *order-cpoly-gcd-eq*:
assumes $pR \neq 0 \vee pI \neq 0$
shows $order t (cpoly\text{-of } pR \ pI) = order t (gcd pR \ pI)$
 ⟨proof⟩

lemma *cpoly-of-times*:
shows $cpoly\text{-of } pR \ pI * cpoly\text{-of } qR \ qI = cpoly\text{-of } (pR * qR - pI * qI) (pI*qR + pR*qI)$

<proof>

lemma *map-poly-Re-cpoly[simp]:*
map-poly Re (cpoly-of pR pI) = pR
<proof>

lemma *map-poly-Im-cpoly[simp]:*
map-poly Im (cpoly-of pR pI) = pI
<proof>

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 $\llbracket ?a < ?b; \text{poly } ?p \text{ ?a} \neq 0; \text{poly } ?p \text{ ?b} \neq 0 \rrbracket \implies \text{cindex-poly } ?a \text{ ?b } ?q \text{ ?p} = \text{changes-itv-smods } ?a \text{ ?b } ?p \text{ ?q}$.

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)*
<proof>

lemma *pathfinish-of-real[simp]:pathfinish of-real = 1*
<proof>

lemma *pathstart-of-real[simp]:pathstart of-real = 0*
<proof>

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

lemma *eventually-poly-nz-at-within:*
fixes *x :: 'a::\{idom, euclidean-space\}*
assumes *p $\neq 0$*
shows *eventually ($\lambda x. \text{poly } p \text{ } x \neq 0$) (at x within S)*

<proof>

lemma *sgn-power*:

fixes $x::'a::\text{linordered-idom}$

shows $\text{sgn } (x^n) = (\text{if } n=0 \text{ then } 1 \text{ else if even } n \text{ then } |\text{sgn } x| \text{ else } \text{sgn } x)$

<proof>

lemma *poly-divide-filterlim-at-top*:

fixes $p\ q::\text{real poly}$

defines $ll \equiv (\text{if degree } q < \text{degree } p \text{ then}$

at 0

else if degree } q = \text{degree } p \text{ then}

nhds (lead-coeff } q / \text{lead-coeff } p)

*else if sgn-pos-inf } q * \text{sgn-pos-inf } p > 0 \text{ then}*

at-top

else

at-bot)

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

shows $\text{filterlim } (\lambda x. \text{poly } q\ x / \text{poly } p\ x)\ ll \text{ at-top}$

<proof>

lemma *poly-divide-filterlim-at-bot*:

fixes $p\ q::\text{real poly}$

defines $ll \equiv (\text{if degree } q < \text{degree } p \text{ then}$

at 0

else if degree } q = \text{degree } p \text{ then}

nhds (lead-coeff } q / \text{lead-coeff } p)

*else if sgn-neg-inf } q * \text{sgn-neg-inf } p > 0 \text{ then}*

at-top

else

at-bot)

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

shows $\text{filterlim } (\lambda x. \text{poly } q\ x / \text{poly } p\ x)\ ll \text{ at-bot}$

<proof>

lemma *sgnx-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 $\text{sgnx } (\text{poly } (p * q))\ F = \text{sgnx } (\text{poly } p)\ F * \text{sgnx } (\text{poly } q)\ F$

(**is** $?PQ = ?P * ?Q$)

<proof>

lemma *sgnx-poly-plus*:

assumes $\text{poly } p\ x = 0\ \text{poly } q\ x \neq 0$ **and** $F: F = \text{at-right } x \vee F = \text{at-left } x$

shows $\text{sgnx } (\text{poly } (p + q))\ F = \text{sgnx } (\text{poly } q)\ F$ (**is** $?L = ?R$)

<proof>

lemma *sign-r-pos-plus-imp*:
assumes *sign-r-pos p x sign-r-pos q x*
shows *sign-r-pos (p+q) x*
 \langle *proof* \rangle

lemma *cindex-poly-combine*:
assumes $a < b < c$
shows $cindex\text{-}poly\ a\ b\ q\ p + jump\text{-}poly\ q\ p\ b + cindex\text{-}poly\ b\ c\ q\ p = cindex\text{-}poly\ a\ c\ q\ p$
 \langle *proof* \rangle

lemma *coprime-linear-comp*: — TODO: need to be generalised
fixes $b\ c::real$
defines $r0 \equiv [b,c]$
assumes $coprime\ p\ q\ c \neq 0$
shows $coprime\ (p \circ_p r0)\ (q \circ_p r0)$
 \langle *proof* \rangle

lemma *finite-ReZ-segments-poly-rectpath*:
finite-ReZ-segments (poly p o rectpath a b) z
 \langle *proof* \rangle

lemma *valid-path-poly-linepath*:
fixes $a\ b::'a::real\text{-}normed\text{-}field$
shows $valid\text{-}path\ (poly\ p\ o\ linepath\ a\ b)$
 \langle *proof* \rangle

lemma *valid-path-poly-rectpath*: $valid\text{-}path\ (poly\ p\ o\ rectpath\ a\ b)$
 \langle *proof* \rangle

2.2 Sign difference

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

lemma *psign-diff-alt*:
assumes $coprime\ p\ q$
shows $psign\text{-}diff\ p\ q\ x = |sign\ (poly\ p\ x) - sign\ (poly\ q\ x)|$
 \langle *proof* \rangle

lemma *psign-diff-0[simp]*:
 $psign\text{-}diff\ 0\ q\ x = 1$
 $psign\text{-}diff\ p\ 0\ x = 1$
 \langle *proof* \rangle

lemma *psign-diff-poly-commute*:
 $psign\text{-}diff\ p\ q\ x = psign\text{-}diff\ q\ p\ x$

<proof>

lemma *normalize-real-poly*:

normalize p = smult (1/lead-coeff p) (p::real poly)

<proof>

lemma *psign-diff-cancel*:

assumes *poly r x ≠ 0*

shows *psign-diff (r*p) (r*q) x = psign-diff p q x*

<proof>

lemma *psign-diff-clear*: *psign-diff p q x = psign-diff 1 (p * q) x*

<proof>

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)*

<proof>

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

<proof>

lemma *cross-alt-poly-commute*:

cross-alt p q a b = cross-alt q p a b

<proof>

lemma *cross-alt-clear*:

*cross-alt p q a b = cross-alt 1 (p*q) a b*

<proof>

lemma *cross-alt-alt*:

*cross-alt p q a b = sign (poly (p*q) b) - sign (poly (p*q) a)*

<proof>

lemma *cross-alt-coprime-0*:

assumes *coprime p q p=0 ∨ q=0*

shows *cross-alt p q a b=0*

<proof>

lemma *cross-alt-cancel*:

assumes *poly q a ≠ 0 poly q b ≠ 0*

shows $\text{cross-alt } (q * r) (q * s) a b = \text{cross-alt } r s a b$
 ⟨proof⟩

lemma *cross-alt-noroot*:

assumes $a < b$ **and** $\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } (p * q) x \neq 0$
shows $\text{cross-alt } p q a b = 0$

⟨proof⟩

lemma *cross-alt-linear-comp*:

fixes $b c :: \text{real}$

defines $h \equiv (\lambda p. \text{pcompose } p [:b, c:])$

shows $\text{cross-alt } (h p) (h q) lb ub = \text{cross-alt } p q (c * lb + b) (c * ub + b)$

⟨proof⟩

2.4 Alternative sign variation sequence

fun *changes-alt*:: ($'a :: \text{linordered-idom}$) $\text{list} \Rightarrow \text{int}$ **where**

$\text{changes-alt } [] = 0$

$\text{changes-alt } [-] = 0$ |

$\text{changes-alt } (x1 \# x2 \# xs) = \text{abs}(\text{sign } x1 - \text{sign } x2) + \text{changes-alt } (x2 \# xs)$

definition *changes-alt-poly-at*:: ($'a :: \text{linordered-idom}$) $\text{poly list} \Rightarrow 'a \Rightarrow \text{int}$ **where**

$\text{changes-alt-poly-at } ps a = \text{changes-alt } (\text{map } (\lambda p. \text{poly } p a) ps)$

definition *changes-alt-itv-smods*:: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{int}$
where

$\text{changes-alt-itv-smods } a b p q = (\text{let } ps = \text{smods } p q$
 $\text{in } \text{changes-alt-poly-at } ps a - \text{changes-alt-poly-at } ps b)$

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

assumes $a < b$ *coprime* $p q$

shows $\text{changes-alt-itv-smods } a b p q = \text{cross-alt } p q a b + \text{changes-alt-itv-smods } a b q (-(p \bmod q))$

⟨proof⟩

2.5 jumpF on polynomials

definition *jumpF-polyR*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

$\text{jumpF-polyR } q p a = \text{jumpF } (\lambda x. \text{poly } q x / \text{poly } p x) (\text{at-right } a)$

definition *jumpF-polyL*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**

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

definition *jumpF-poly-top*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real}$ **where**

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

definition *jumpF-poly-bot*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real}$ **where**

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

lemma *jumpF-polyR-0[simp]*: $\text{jumpF-polyR } 0 \ p \ a = 0 \ \text{jumpF-polyR } q \ 0 \ a = 0$
 ⟨proof⟩

lemma *jumpF-polyL-0[simp]*: $\text{jumpF-polyL } 0 \ p \ a = 0 \ \text{jumpF-polyL } q \ 0 \ a = 0$
 ⟨proof⟩

lemma *jumpF-polyR-mult-cancel*:
 assumes $p' \neq 0$
 shows $\text{jumpF-polyR } (p' * q) \ (p' * p) \ a = \text{jumpF-polyR } q \ p \ a$
 ⟨proof⟩

lemma *jumpF-polyL-mult-cancel*:
 assumes $p' \neq 0$
 shows $\text{jumpF-polyL } (p' * q) \ (p' * p) \ a = \text{jumpF-polyL } q \ p \ a$
 ⟨proof⟩

lemma *jumpF-poly-noroot*:
 assumes $\text{poly } p \ a \neq 0$
 shows $\text{jumpF-polyL } q \ p \ a = 0 \ \text{jumpF-polyR } q \ p \ a = 0$
 ⟨proof⟩

lemma *jumpF-polyR-coprime'*:
 assumes $\text{poly } p \ x \neq 0 \ \vee \ \text{poly } q \ x \neq 0$
 shows $\text{jumpF-polyR } q \ p \ x = (\text{if } p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{poly } p \ x = 0 \ \text{then}$
 $\text{if } \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \ \text{then } 1/2 \ \text{else } -1/2$
 else 0)
 ⟨proof⟩

lemma *jumpF-polyR-coprime*:
 assumes $\text{coprime } p \ q$
 shows $\text{jumpF-polyR } q \ p \ x = (\text{if } p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{poly } p \ x = 0 \ \text{then}$
 $\text{if } \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \ \text{then } 1/2 \ \text{else } -1/2$
 else 0)
 ⟨proof⟩

lemma *jumpF-polyL-coprime'*:
 assumes $\text{poly } p \ x \neq 0 \ \vee \ \text{poly } q \ x \neq 0$
 shows $\text{jumpF-polyL } q \ p \ x = (\text{if } p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{poly } p \ x = 0 \ \text{then}$
 $\text{if even } (\text{order } x \ p) \longleftrightarrow \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \ \text{then } 1/2 \ \text{else}$
 $-1/2 \ \text{else } 0)$
 ⟨proof⟩

lemma *jumpF-polyL-coprime*:
 assumes $\text{coprime } p \ q$
 shows $\text{jumpF-polyL } q \ p \ x = (\text{if } p \neq 0 \ \wedge \ q \neq 0 \ \wedge \ \text{poly } p \ x = 0 \ \text{then}$
 $\text{if even } (\text{order } x \ p) \longleftrightarrow \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \ \text{then } 1/2 \ \text{else}$
 $-1/2 \ \text{else } 0)$

<proof>

lemma *jumpF-times:*

assumes *tendsto*:($f \longrightarrow c$) F **and** $c \neq 0$ $F \neq \text{bot}$

shows $\text{jumpF } (\lambda x. f x * g x) F = \text{sgn } c * \text{jumpF } g F$

<proof>

lemma *jumpF-polyR-inverse-add:*

assumes *coprime* p q

shows $\text{jumpF-polyR } q p x + \text{jumpF-polyR } p q x = \text{jumpF-polyR } 1 (q*p) x$

<proof>

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>

lemma *jumpF-polyL-smult-1:*

$\text{jumpF-polyL } (\text{smult } c q) p x = \text{sgn } c * \text{jumpF-polyL } q p x$

<proof>

lemma *jumpF-polyR-smult-1:*

$\text{jumpF-polyR } (\text{smult } c q) p x = \text{sgn } c * \text{jumpF-polyR } q p x$

<proof>

lemma

shows *jumpF-polyR-mod:jumpF-polyR* $q p x = \text{jumpF-polyR } (q \text{ mod } p) p x$ **and**

jumpF-polyL-mod:jumpF-polyL $q p x = \text{jumpF-polyL } (q \text{ mod } p) p x$

<proof>

lemma

assumes $\text{order } x p \leq \text{order } x r$

shows *jumpF-polyR-order-leq:jumpF-polyR* $(r+q) p x = \text{jumpF-polyR } q p x$

and *jumpF-polyL-order-leq:jumpF-polyL* $(r+q) p x = \text{jumpF-polyL } q p x$

<proof>

lemma

assumes $\text{order } x q < \text{order } x r$ $q \neq 0$

shows *jumpF-polyR-order-le:jumpF-polyR* $(r+q) p x = \text{jumpF-polyR } q p x$

and *jumpF-polyL-order-le:jumpF-polyL* $(r+q) p x = \text{jumpF-polyL } q p x$

<proof>

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

<proof>

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

<proof>

lemma *jumpF-poly-top-code*:

jumpF-poly-top $q\ p = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{degree } q > \text{degree } p \text{ then}$
 $\text{if } \text{sgn-pos-inf } q * \text{sgn-pos-inf } p > 0 \text{ then } 1/2 \text{ else } -1/2 \text{ else } 0)$

<proof>

lemma *jumpF-poly-bot-code*:

jumpF-poly-bot $q\ p = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{degree } q > \text{degree } p \text{ then}$
 $\text{if } \text{sgn-neg-inf } q * \text{sgn-neg-inf } p > 0 \text{ then } 1/2 \text{ else } -1/2 \text{ else } 0)$

<proof>

lemma *jump-poly-jumpF-poly*:

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

<proof>

2.6 The extended Cauchy index on polynomials

definition *cindex-polyE*:: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real}$ **where**

cindex-polyE $a\ b\ q\ p = \text{jumpF-polyR } q\ p\ a + \text{cindex-poly } a\ b\ q\ p - \text{jumpF-polyL } q\ p\ b$

definition *cindex-poly-ubd*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{int}$ **where**

cindex-poly-ubd $q\ p = (\text{THE } l. (\forall_F r \text{ in at-top. } \text{cindexE } (-r)\ r (\lambda x. \text{poly } q\ x / \text{poly } p\ x) = \text{of-int } l))$

lemma *cindex-polyE-0[simp]*: $\text{cindex-polyE } a\ b\ 0\ p = 0$ $\text{cindex-polyE } a\ b\ q\ 0 = 0$

<proof>

lemma *cindex-polyE-mult-cancel*:

fixes $p\ q\ p'::\text{real poly}$

assumes $p' \neq 0$

shows $\text{cindex-polyE } a\ b\ (p' * q) (p' * p) = \text{cindex-polyE } a\ b\ q\ p$

<proof>

lemma *cindexE-eq-cindex-polyE*:

assumes $a < b$

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

<proof>

lemma *cindex-polyE-cross*:

fixes $p::\text{real poly}$ **and** $a\ b::\text{real}$

assumes $a < b$

shows $\text{cindex-polyE } a\ b\ 1\ p = \text{cross-alt } 1\ p\ a\ b / 2$

<proof>

lemma *cindex-polyE-inverse-add*:

fixes $p\ q::\text{real poly}$

assumes $cp:\text{coprime } p\ q$

shows $\text{cindex-polyE } a\ b\ q\ p + \text{cindex-polyE } a\ b\ p\ q = \text{cindex-polyE } a\ b\ 1\ (q * p)$

<proof>

lemma *cindex-polyE-inverse-add-cross*:

fixes $p\ q::\text{real poly}$

assumes $a < b$ *coprime* $p\ q$

shows $\text{cindex-polyE } a\ b\ q\ p + \text{cindex-polyE } a\ b\ p\ q = \text{cross-alt } p\ q\ a\ b / 2$

<proof>

lemma *cindex-polyE-inverse-add-cross'*:

fixes $p\ q::\text{real poly}$

assumes $a < b$ $\text{poly } p\ a \neq 0 \vee \text{poly } q\ a \neq 0$ $\text{poly } p\ b \neq 0 \vee \text{poly } q\ b \neq 0$

shows $\text{cindex-polyE } a\ b\ q\ p + \text{cindex-polyE } a\ b\ p\ q = \text{cross-alt } p\ q\ a\ b / 2$

<proof>

lemma *cindex-polyE-smult-1*:

fixes $p\ q::\text{real poly}$ **and** $c::\text{real}$

shows $\text{cindex-polyE } a\ b\ (\text{smult } c\ q)\ p = (\text{sgn } c) * \text{cindex-polyE } a\ b\ q\ p$

<proof>

lemma *cindex-polyE-smult-2*:

fixes $p\ q::\text{real poly}$ **and** $c::\text{real}$

shows $\text{cindex-polyE } a\ b\ q\ (\text{smult } c\ p) = (\text{sgn } c) * \text{cindex-polyE } a\ b\ q\ p$

<proof>

lemma *cindex-polyE-mod*:

fixes $p\ q::\text{real poly}$

shows $\text{cindex-polyE } a\ b\ q\ p = \text{cindex-polyE } a\ b\ (q \bmod p)\ p$

<proof>

lemma *cindex-polyE-rec*:

fixes $p\ q::\text{real poly}$

assumes $a < b$ *coprime* $p\ q$

shows $\text{cindex-polyE } a\ b\ q\ p = \text{cross-alt } q\ p\ a\ b / 2 + \text{cindex-polyE } a\ b\ (- (p \bmod q))\ q$

<proof>

lemma *cindex-polyE-changes-alt-itv-mods*:

assumes $a < b$ *coprime* $p\ q$

shows $\text{cindex-polyE } a\ b\ q\ p = \text{changes-alt-itv-smods } a\ b\ p\ q / 2$ *<proof>*

lemma *cindex-poly-ubd-eventually*:

shows $\forall_F r$ *in at-top*. $\text{cindexE } (-r)\ r\ (\lambda x. \text{poly } q\ x / \text{poly } p\ x) = \text{of-int } (\text{cindex-poly-ubd } q\ p)$

<proof>

lemma *cindex-poly-ubd-0*:

assumes $p=0 \vee q=0$

shows $\text{cindex-poly-ubd } q\ p = 0$

<proof>

lemma *cindex-poly-ubd-code*:

shows $cindex\text{-}poly\text{-}ubd\ q\ p = changes\text{-}R\text{-}smods\ p\ q$
<proof>

lemma *cindexE-ubd-poly*: $cindexE\text{-}ubd\ (\lambda x. poly\ q\ x / poly\ p\ x) = cindex\text{-}poly\text{-}ubd\ q\ p$
<proof>

lemma *cindex-polyE-noroot*:

assumes $a < b\ \forall x. a \leq x \wedge x \leq b \longrightarrow poly\ p\ x \neq 0$
shows $cindex\text{-}polyE\ a\ b\ q\ p = 0$
<proof>

lemma *cindex-polyE-combine*:

assumes $a < b\ b < c$
shows $cindex\text{-}polyE\ a\ b\ q\ p + cindex\text{-}polyE\ b\ c\ q\ p = cindex\text{-}polyE\ a\ c\ q\ p$
<proof>

lemma *cindex-polyE-linear-comp*:

fixes $b\ c :: real$
defines $h \equiv (\lambda p. pcompose\ p\ [:b, c:])$
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>

lemma *cindex-polyE-product'*:

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

lemma *cindex-polyE-product*:

fixes $p\ r\ q\ s :: real\ poly$ **and** $a\ b :: real$
assumes $a < b$
and $poly\ p\ a \neq 0 \vee poly\ q\ a \neq 0\ poly\ p\ b \neq 0 \vee poly\ q\ b \neq 0$
and $poly\ r\ a \neq 0 \vee poly\ s\ a \neq 0\ poly\ r\ b \neq 0 \vee poly\ s\ b \neq 0$
shows $cindex\text{-}polyE\ a\ b\ (p * r - q * s)\ (p * s + q * r)$
 $= cindex\text{-}polyE\ a\ b\ p\ q + cindex\text{-}polyE\ a\ b\ r\ s$
 $- cross\text{-}alt\ (p * s + q * r)\ (q * s)\ a\ b / 2$
<proof>

lemma *cindex-pathE-linepath-on*:
assumes $z \in \text{closed-segment } a \ b$
shows $\text{cindex-pathE } (\text{linepath } a \ b) \ z = 0$
<proof>

2.7 More Cauchy indices on polynomials

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

definition *cindexP-lineE :: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow 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*
<proof>

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

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

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

definition *cdiff-aux :: complex poly \Rightarrow complex poly \Rightarrow complex \Rightarrow complex \Rightarrow int*
where
 $\text{cdiff-aux } p \ q \ a \ b = \text{psign-aux } p \ q \ b - \text{psign-aux } p \ q \ a$

lemma *cindexP-lineE-times*:
fixes $p \ q::\text{complex poly}$ **and** $a \ b::\text{complex}$
assumes $\text{poly } p \ a \neq 0$ $\text{poly } p \ b \neq 0$ $\text{poly } q \ a \neq 0$ $\text{poly } q \ b \neq 0$
shows $\text{cindexP-lineE } (p*q) \ a \ b = \text{cindexP-lineE } p \ a \ b + \text{cindexP-lineE } q \ a \ b + \text{cdiff-aux } p \ q \ a \ b / 2$

<proof>

lemma *cindexP-lineE-changes*:

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

assumes *p≠0 a≠b*

shows *cindexP-lineE p a b =*

(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)

<proof>

lemma *cindexP-lineE-code[code]*:

cindexP-lineE p a b = (if p≠0 ∧ a≠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")

(λ-. cindexP-lineE p a b))

<proof>

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::complex*

assumes *x∈closed-segment lb ub*

shows *Re lb ≤ Re ub ⇒ Re lb ≤ Re x ∧ Re x ≤ Re ub*

Im lb ≤ Im ub ⇒ Im lb ≤ Im x ∧ Im x ≤ Im ub

<proof>

lemma *closed-segment-degen-complex:*

[[$Re\ lb = Re\ ub; Im\ lb \leq Im\ ub$]]
 $\implies x \in \text{closed-segment } lb\ ub \iff Re\ x = Re\ lb \wedge Im\ lb \leq Im\ x \wedge Im\ x \leq Im\ ub$
[[$Im\ lb = Im\ ub; Re\ lb \leq Re\ ub$]]
 $\implies x \in \text{closed-segment } lb\ ub \iff Im\ x = Im\ lb \wedge Re\ lb \leq Re\ x \wedge Re\ x \leq Re\ ub$
(*proof*)

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*)

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*)

lemma *infinite-ball:*

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

lemma *infinite-cball:*

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

lemma *infinite-sphere:*

fixes $a :: \text{complex}$
assumes $r > 0$
shows *infinite* ($\text{sphere } a\ r$)
(*proof*)

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

(*proof*)

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

(*proof*)

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*)

lemma *of-int-div-field*:

assumes $d \text{ dvd } n$

shows $(\text{of-int}::\Rightarrow'a::\text{field-char-0}) (n \text{ div } d) = \text{of-int } n / \text{of-int } d$

$\langle \text{proof} \rangle$

lemma *powr-eq-1-iff*:

assumes $a > 0$

shows $(a::\text{real}) \text{ powr } b = 1 \iff a = 1 \vee b = 0$

$\langle \text{proof} \rangle$

lemma *tan-inj-pi*:

$-(\pi/2) < x \implies x < \pi/2 \implies -(\pi/2) < y \implies y < \pi/2 \implies \tan x = \tan y$
 $\implies x = y$

$\langle \text{proof} \rangle$

lemma *finite-ReZ-segments-poly-circlepath*:

finite-ReZ-segments (poly p o circlepath z0 r) 0

$\langle \text{proof} \rangle$

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*

$\langle \text{proof} \rangle$

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)*

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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)*

$\langle \text{proof} \rangle$

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)*

$\langle \text{proof} \rangle$

lemma *proots-ball-plane-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$

assumes $p \neq 0$

shows *proots-count p (ball 0 1) = proots-count (fcompose p q1 q2) {x. 0 < Im*

$x\}$
 $\langle proof \rangle$

lemma *proots-sphere-axis-eq:*

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $proots-count\ p\ (sphere\ 0\ 1\ -\ \{-1\}) = proots-count\ (fcompose\ p\ q1\ q2)$
 $\{x.\ 0 = Im\ x\}$
 $\langle proof \rangle$

lemma *proots-card-ball-plane-eq:*

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $card\ (proots-within\ p\ (ball\ 0\ 1)) = card\ (proots-within\ (fcompose\ p\ q1\ q2)$
 $\{x.\ 0 < Im\ x\})$
 $\langle proof \rangle$

lemma *proots-card-sphere-axis-eq:*

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$
assumes $p \neq 0$
shows $card\ (proots-within\ p\ (sphere\ 0\ 1\ -\ \{-1\}))$
 $= card\ (proots-within\ (fcompose\ p\ q1\ q2)\ \{x.\ 0 = Im\ x\})$
 $\langle proof \rangle$

lemma *proots-uball-eq:*

fixes $z0::complex$ **and** $r::real$
defines $q \equiv [z0, of-real\ r:]$
assumes $p \neq 0$ **and** $r > 0$
shows $proots-count\ p\ (ball\ z0\ r) = proots-count\ (p\ \circ_p\ q)\ (ball\ 0\ 1)$
 $\langle proof \rangle$

lemma *proots-card-uball-eq:*

fixes $z0::complex$ **and** $r::real$
defines $q \equiv [z0, of-real\ r:]$
assumes $r > 0$
shows $card\ (proots-within\ p\ (ball\ z0\ r)) = card\ (proots-within\ (p\ \circ_p\ q)\ (ball\ 0$
 $1))$
 $\langle proof \rangle$

lemma *proots-card-usphere-eq:*

fixes $z0::complex$ **and** $r::real$
defines $q \equiv [z0, of-real\ r:]$
assumes $r > 0$
shows $card\ (proots-within\ p\ (sphere\ z0\ r)) = card\ (proots-within\ (p\ \circ_p\ q)\ (sphere$
 $0\ 1))$
 $\langle proof \rangle$

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

definition *unbounded-line*:: 'a::real-vector \Rightarrow 'a \Rightarrow 'a set **where**

$$\text{unbounded-line } a \ b = (\{x. \exists u::\text{real. } x = (1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b\})$$

definition *proots-line-card*:: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat **where**

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

definition *proots-unbounded-line-card*:: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat **where**

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

definition *proots-unbounded-line* :: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat **where**

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

lemma *card-proots-open-segments*:

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

shows card (proots-within p (open-segment st tt)) =

$$\begin{aligned} & (\text{let } pc = \text{pcompose } p \ [:\text{st}, \text{tt} - \text{st}:]; \\ & \quad pR = \text{map-poly } \text{Re } pc; \\ & \quad pI = \text{map-poly } \text{Im } pc; \\ & \quad g = \text{gcd } pR \ pI \\ & \text{in changes-itv-smods } 0 \ 1 \ g \ (\text{pderiv } g)) \ (\text{is } ?L = ?R) \end{aligned}$$

<proof>

lemma *unbounded-line-closed-segment*: closed-segment a b \subseteq unbounded-line a b

<proof>

lemma *card-proots-unbounded-line*:

assumes st \neq tt

shows card (proots-within p (unbounded-line st tt)) =

$$\begin{aligned} & (\text{let } pc = \text{pcompose } p \ [:\text{st}, \text{tt} - \text{st}:]; \\ & \quad pR = \text{map-poly } \text{Re } pc; \\ & \quad pI = \text{map-poly } \text{Im } pc; \\ & \quad g = \text{gcd } pR \ pI \\ & \text{in nat } (\text{changes-R-smods } g \ (\text{pderiv } g)) \ (\text{is } ?L = ?R) \end{aligned}$$

<proof>

lemma *proots-count-gcd-eq*:

fixes p::complex poly **and** st tt::complex

and g::real poly

defines pc \equiv pcompose p [:\text{st}, \text{tt} - \text{st}:]

defines pR \equiv map-poly Re pc **and** pI \equiv map-poly Im pc

defines g \equiv gcd pR pI

assumes st \neq tt p $\neq 0$

and s1-def:s1 = ($\lambda x. \text{poly } [:\text{st}, \text{tt} - \text{st}:] \ (\text{of-real } x)$) ' s2

shows proots-count p s1 = proots-count g s2

<proof>

lemma *roots-unbounded-line*:

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

shows $(\text{roots-count } p \ (\text{unbounded-line } st \ tt)) =$
 $(\text{let } pc = \text{pcompose } p \ [:st, tt - st:];$
 $\quad pR = \text{map-poly } Re \ pc;$
 $\quad pI = \text{map-poly } Im \ pc;$
 $\quad g = \text{gcd } pR \ pI$
 $\text{in nat } (\text{changes-R-smods-ext } g \ (\text{pderiv } g))) \ (\text{is } ?L = ?R)$

<proof>

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

roots-unbounded-line-card $p \ st \ tt =$

(if $st \neq tt$ *then*
 $(\text{let } pc = \text{pcompose } p \ [:st, tt - st:];$
 $\quad pR = \text{map-poly } Re \ pc;$
 $\quad pI = \text{map-poly } Im \ pc;$
 $\quad g = \text{gcd } pR \ pI$
 $\text{in nat } (\text{changes-R-smods } g \ (\text{pderiv } g)))$

else

$\text{Code.abort } (\text{STR } \text{"roots-unbounded-line-card fails due to invalid$

hyperplanes."})

$(\lambda-. \text{roots-unbounded-line-card } p \ st \ tt))$

<proof>

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

roots-unbounded-line $p \ st \ tt =$

(if $st \neq tt$ *then*
 $\text{if } p \neq 0 \text{ then}$
 $(\text{let } pc = \text{pcompose } p \ [:st, tt - st:];$
 $\quad pR = \text{map-poly } Re \ pc;$
 $\quad pI = \text{map-poly } Im \ pc;$
 $\quad g = \text{gcd } pR \ pI$
 $\text{in nat } (\text{changes-R-smods-ext } g \ (\text{pderiv } g)))$

else

$\text{Code.abort } (\text{STR } \text{"roots-unbounded-line fails due to } p=0\text{"})$

$(\lambda-. \text{roots-unbounded-line } p \ st \ tt)$

else

$\text{Code.abort } (\text{STR } \text{"roots-unbounded-line fails due to invalid$

hyperplanes."})

$(\lambda-. \text{roots-unbounded-line } p \ st \ tt))$

<proof>

2.11 Checking if there a polynomial root on a closed segment

definition *no-roots-line*:: $\text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{bool}$ **where**

$\text{no-roots-line } p \ st \ tt = (\text{roots-within } p \ (\text{closed-segment } st \ tt) = \{\})$

lemma *no-roots-line-code*[code]: $\text{no-roots-line } p \ st \ tt = (\text{if } \text{poly } p \ st \neq 0 \wedge \text{poly } p$

```

tt ≠ 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⟩

```

2.12 Number of roots on a bounded open segment

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

lemma *proots-line-commute*:

```

proots-line p st tt = proots-line p tt st
⟨proof⟩

```

lemma *proots-line-smods*:

assumes *poly* p st ≠ 0 *poly* p tt ≠ 0 st ≠ tt

shows *proots-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 = ?R)

⟨proof⟩

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

proots-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-itv-smods-ext 0 1 g (pderiv g)))

```

else 0)

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

(λ-. *proots-line* p st tt) (is ?L = ?R)

⟨proof⟩

end

theory *Count-Half-Plane* **imports**

Count-Line
begin

2.13 Polynomial roots on the upper half-plane

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

— Roots counted WITHOUT multiplicity

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

lemma *Im-Ln-tendsto-at-top*: ((λx . *Im* (*Ln* (*Complex a x*))) \longrightarrow *pi/2*) *at-top*
 <*proof*>

lemma *Im-Ln-tendsto-at-bot*: ((λx . *Im* (*Ln* (*Complex a x*))) \longrightarrow $-$ *pi/2*) *at-bot*
 <*proof*>

lemma *Re-winding-number-tendsto-part-circlepath*:

shows ((λr . *Re* (*winding-number* (*part-circlepath* *z0 r 0 pi*) *a*)) \longrightarrow *1/2*)
at-top
 <*proof*>

lemma *not-image-at-top-poly-part-circlepath*:

assumes *degree p* > 0
shows $\forall_F r$ *in at-top*. $b \notin$ *path-image* (*poly p o part-circlepath* *z0 r st tt*)
 <*proof*>

lemma *not-image-poly-part-circlepath*:

assumes *degree p* > 0
shows $\exists r > 0$. $b \notin$ *path-image* (*poly p o part-circlepath* *z0 r st tt*)
 <*proof*>

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*)) \longrightarrow
degree p/2) *at-top*
 <*proof*>

lemma *Re-winding-number-poly-linepth*:

fixes *pp* :: *complex poly*
defines *g* \equiv (λr . *poly pp o linepath* ($-r$) (*of-real r*))
assumes *lead-coeff pp* = 1 **and** *no-real-zero*: $\forall x \in$ *roots pp*. *Im x* \neq 0
shows ((λr . $2 * \text{Re}$ (*winding-number* (*g r*) *0*) + *cindex-pathE* (*g r*) *0*) \longrightarrow 0)
at-top
 <*proof*>

lemma *roots-upper-cindex-eq*:

assumes *lead-coeff p=1* **and** *no-real-roots:∀ x∈roots p. Im x≠0*
shows *roots-upper p =*
 $(\text{degree } p - \text{cindex-poly-ubd } (\text{map-poly } \text{Im } p) (\text{map-poly } \text{Re } p)) / 2$
⟨*proof*⟩

lemma *cindexE-roots-on-horizontal-border:*

fixes *a::complex and s::real*
defines *g≡linepath a (a + of-real s)*
assumes *pqr:p = q * r and r-monic:lead-coeff r=1 and r-roots:∀ x∈roots r. Im x=Im a*
shows *cindexE lb ub (λt. Im ((poly p ∘ g) t) / Re ((poly p ∘ g) t)) =*
cindexE lb ub (λt. Im ((poly q ∘ g) t) / Re ((poly q ∘ g) t))
⟨*proof*⟩

lemma *poly-decompose-by-roots:*

fixes *p ::'a::idom poly*
assumes *p≠0*
shows $\exists q r. p = q * r \wedge \text{lead-coeff } q=1 \wedge (\forall x \in \text{roots } q. P x) \wedge (\forall x \in \text{roots } r. \neg P x)$ ⟨*proof*⟩

lemma *roots-upper-cindex-eq':*

assumes *lead-coeff p=1*
shows *roots-upper p = (degree p - roots-count p {x. Im x=0})*
 $- \text{cindex-poly-ubd } (\text{map-poly } \text{Im } p) (\text{map-poly } \text{Re } p) / 2$
⟨*proof*⟩

lemma *roots-within-upper-squarefree:*

assumes *rsquarefree p*
shows *card (roots-within p {x. Im x > 0}) = (let*
pp = smult (inverse (lead-coeff p)) p;
pI = map-poly Im pp;
pR = map-poly Re pp;
g = gcd pR pI
in
 $\text{nat } ((\text{degree } p - \text{changes-R-smods } g (\text{pderiv } g) - \text{changes-R-smods } pR$
 $pI) \text{ div } 2)$
 $)$
⟨*proof*⟩

lemma *roots-upper-code1[code]:*

roots-upper p =
(if p ≠ 0 then
 $(\text{let } pp = \text{smult } (\text{inverse } (\text{lead-coeff } p)) p;$
 $pI = \text{map-poly } \text{Im } pp;$
 $pR = \text{map-poly } \text{Re } pp;$
 $g = \text{gcd } pI pR$
 $)$
 $)$

```

    in
      nat ((degree p - nat (changes-R-smods-ext g (pderiv g)) - changes-R-smods
pR pI) div 2)
    )
  else
    Code.abort (STR "proots-upper fails when p=0.") (λ-. proots-upper p)
<proof>

```

lemma *proots-upper-card-code*[code]:

```

proots-upper-card p = (if p=0 then 0 else
  (let
    pf = p div (gcd p (pderiv p));
    pp = smult (inverse (lead-coeff pf)) pf;
    pI = map-poly Im pp;
    pR = map-poly Re pp;
    g = gcd pR pI
  in
    nat ((degree pf - changes-R-smods g (pderiv g) - changes-R-smods pR
pI) div 2)
  ))
<proof>

```

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
<proof>

```

lemma *proots-half-proots-upper*:

```

  assumes a≠b p≠0
  shows proots-half p a b = proots-upper (pcompose p [:a, (b-a):])
<proof>

```

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)
<proof>

```

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>

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>

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 = proots-count p (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 = card (proots-within p (sphere z0 r))

lemma *proots-sphere-card-code1[code]:*

proots-sphere-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

```

      (if poly p (z0-r) =0 then 1 else 0) +
      roots-unbounded-line-card (fcompose (p o_p [:z0, of-real r:]) [:i,-1:]
[:i,1:])
      0 1
    )
  <proof>

```

2.17 Polynomial roots on a closed ball

definition *roots-cball::complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat where*
roots-cball p z0 r = roots-count p (cball z0 r)

— Roots counted WITHOUT multiplicity

definition *roots-cball-card ::complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat where*
roots-cball-card p z0 r = card (roots-within p (cball z0 r))

lemma *roots-cball-card-code1 [code]:*

```

  roots-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 o_p [:z0, of-real r:]) [:i,-1:] [:i,1:]
          in
            (if poly p (z0-r) =0 then 1 else 0)
            + roots-unbounded-line-card pp 0 1
            + roots-upper-card pp
          )
        )
    )
  <proof>

```

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 $\text{proots-count } [:c:] s = 0$
 <proof>

lemma *proots-count-nzero*:
 assumes $\bigwedge x. x \in s \implies \text{poly } p x \neq 0$
 shows $\text{proots-count } p s = 0$
 <proof>

lemma *complex-box-ne-empty*:
 fixes $a b :: \text{complex}$
 shows
 $\text{cbox } a b \neq \{\} \iff (\text{Re } a \leq \text{Re } b \wedge \text{Im } a \leq \text{Im } b)$
 $\text{box } a b \neq \{\} \iff (\text{Re } a < \text{Re } b \wedge \text{Im } a < \text{Im } b)$
 <proof>

2.19 Counting roots in a rectangle

definition *proots-rect* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
 $\text{proots-rect } p \text{ lb ub} = \text{proots-count } p (\text{box } \text{lb } \text{ub})$

definition *proots-crect* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
 $\text{proots-crect } p \text{ lb ub} = \text{proots-count } p (\text{cbox } \text{lb } \text{ub})$

definition *proots-rect-ll* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
 $\text{proots-rect-ll } p \text{ lb ub} = \text{proots-count } p (\text{box } \text{lb } \text{ub} \cup \{\text{lb}\})$
 $\cup \text{open-segment } \text{lb} (\text{Complex } (\text{Re } \text{ub}) (\text{Im } \text{lb}))$
 $\cup \text{open-segment } \text{lb} (\text{Complex } (\text{Re } \text{lb}) (\text{Im } \text{ub}))$

definition *proots-rect-border* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{nat}$ **where**
 $\text{proots-rect-border } p a b = \text{proots-count } p (\text{path-image } (\text{rectpath } a b))$

definition *not-rect-vertex* $:: \text{complex} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{bool}$ **where**
 $\text{not-rect-vertex } r a b = (r \neq a \wedge r \neq \text{Complex } (\text{Re } b) (\text{Im } a) \wedge r \neq b \wedge r \neq \text{Complex } (\text{Re } a) (\text{Im } b))$

definition *not-rect-vanishing* $:: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{complex} \Rightarrow \text{bool}$ **where**
 $\text{not-rect-vanishing } p a b = (\text{poly } p a \neq 0 \wedge \text{poly } p (\text{Complex } (\text{Re } b) (\text{Im } a)) \neq 0$
 $\wedge \text{poly } p b \neq 0 \wedge \text{poly } p (\text{Complex } (\text{Re } a) (\text{Im } b)) \neq 0)$

lemma *cindexP-rectpath-edge-base*:
 assumes $\text{Re } a < \text{Re } b \text{ Im } a < \text{Im } b$
 and *not-rect-vertex* $r a b$
 and $r \in \text{path-image } (\text{rectpath } a b)$

shows $cindexP\text{-}pathE [-r,1:] (rectpath\ a\ b) = -1$
 $\langle proof \rangle$

lemma $cindexP\text{-}rectpath\text{-}vertex\text{-}base$:
assumes $Re\ a < Re\ b\ Im\ a < Im\ b$
and $\neg\ not\text{-}rect\text{-}vertex\ r\ a\ b$
shows $cindexP\text{-}pathE [-r,1:] (rectpath\ a\ b) = -1/2$
 $\langle proof \rangle$

lemma $cindexP\text{-}rectpath\text{-}interior\text{-}base$:
assumes $r \in box\ a\ b$
shows $cindexP\text{-}pathE [-r,1:] (rectpath\ a\ b) = -2$
 $\langle proof \rangle$

lemma $cindexP\text{-}rectpath\text{-}outside\text{-}base$:
assumes $Re\ a < Re\ b\ Im\ a < Im\ b$
and $r \notin cbox\ a\ b$
shows $cindexP\text{-}pathE [-r,1:] (rectpath\ a\ b) = 0$
 $\langle proof \rangle$

lemma $cindexP\text{-}rectpath\text{-}add\text{-}one\text{-}root$:
assumes $Re\ a < Re\ b\ Im\ a < Im\ b$
and $not\text{-}rect\text{-}vertex\ r\ a\ b$
and $not\text{-}rect\text{-}vanishing\ p\ a\ b$
shows $cindexP\text{-}pathE ([:-r,1:] * p) (rectpath\ a\ b) =$
 $cindexP\text{-}pathE\ p\ (rectpath\ a\ b)$
 $+ (if\ r \in box\ a\ b\ then\ -2\ else\ if\ r \in path\text{-}image\ (rectpath\ a\ b)\ then\ -1\ else$
 $0)$
 $\langle proof \rangle$

lemma $proots\text{-}rect\text{-}cindexP\text{-}pathE$:
assumes $Re\ a < Re\ b\ Im\ a < Im\ b$
and $not\text{-}rect\text{-}vanishing\ p\ a\ b$
shows $proots\text{-}rect\ p\ a\ b = -(proots\text{-}rect\text{-}border\ p\ a\ b + cindexP\text{-}pathE\ p\ (rectpath$
 $a\ b)) / 2$
 $\langle proof \rangle$

2.20 Code generation

lemmas $Complex\text{-}minus\text{-}eq = minus\text{-}complex.code$

lemma $cindexP\text{-}pathE\text{-}rect\text{-}smods$:
fixes $p::complex\ poly$ **and** $lb\ ub::complex$
assumes $ab\text{-}le:Re\ lb < Re\ ub\ Im\ lb < Im\ ub$
and $not\text{-}rect\text{-}vanishing\ p\ lb\ ub$
shows $cindexP\text{-}pathE\ p\ (rectpath\ lb\ ub) =$
 $(let\ p1 = pcompose\ p\ [:-lb,\ Complex\ (Re\ ub - Re\ lb)\ 0:];$
 $pR1 = map\text{-}poly\ Re\ p1; pI1 = map\text{-}poly\ Im\ p1; gc1 = gcd\ pR1\ pI1;$

lb);];
 $p2 = pcompose\ p\ [:\text{Complex}\ (Re\ ub)\ (Im\ lb),\ \text{Complex}\ 0\ (Im\ ub - Im$
 $pR2 = map-poly\ Re\ p2; pI2 = map-poly\ Im\ p2; gc2 = gcd\ pR2\ pI2;$
 $p3 = pcompose\ p\ [:\text{Complex}\ (Re\ lb - Re\ ub)\ 0];$
 $pR3 = map-poly\ Re\ p3; pI3 = map-poly\ Im\ p3; gc3 = gcd\ pR3\ pI3;$
 ub);];
 $p4 = pcompose\ p\ [:\text{Complex}\ (Re\ lb)\ (Im\ ub),\ \text{Complex}\ 0\ (Im\ lb - Im$
 $pR4 = map-poly\ Re\ p4; pI4 = map-poly\ Im\ p4; gc4 = gcd\ pR4\ pI4$
in
 $(changes-alt-itv-smods\ 0\ 1\ (pR1\ div\ gc1)\ (pI1\ div\ gc1)$
 $+ changes-alt-itv-smods\ 0\ 1\ (pR2\ div\ gc2)\ (pI2\ div\ gc2)$
 $+ changes-alt-itv-smods\ 0\ 1\ (pR3\ div\ gc3)\ (pI3\ div\ gc3)$
 $+ changes-alt-itv-smods\ 0\ 1\ (pR4\ div\ gc4)\ (pI4\ div\ gc4)$
 $) / 2) (is\ ?L=?R)$
 $\langle proof \rangle$

lemma *open-segment-Im-equal*:
assumes $Re\ x \neq Re\ y\ Im\ x = Im\ y$
shows $open-segment\ x\ y = \{z.\ Im\ z = Im\ x$
 $\wedge Re\ z \in open-segment\ (Re\ x)\ (Re\ y)\}$
 $\langle proof \rangle$

lemma *open-segment-Re-equal*:
assumes $Re\ x = Re\ y\ Im\ x \neq Im\ y$
shows $open-segment\ x\ y = \{z.\ Re\ z = Re\ x$
 $\wedge Im\ z \in open-segment\ (Im\ x)\ (Im\ y)\}$
 $\langle proof \rangle$

lemma *Complex-eq-iff*:
 $x = \text{Complex}\ y\ z \iff Re\ x = y \wedge Im\ x = z$
 $\text{Complex}\ y\ z = x \iff Re\ x = y \wedge Im\ x = z$
 $\langle proof \rangle$

lemma *proots-rect-border-eq-lines*:
fixes $p::\text{complex}\ poly$ **and** $lb\ ub::\text{complex}$
assumes $ab-le:Re\ lb < Re\ ub\ Im\ lb < Im\ ub$
and $not-van:not-rect-vanishing\ p\ lb\ ub$
shows $proots-rect-border\ p\ lb\ ub =$
 $proots-line\ p\ lb\ (\text{Complex}\ (Re\ ub)\ (Im\ lb))$
 $+ proots-line\ p\ (\text{Complex}\ (Re\ ub)\ (Im\ lb))\ ub$
 $+ proots-line\ p\ ub\ (\text{Complex}\ (Re\ lb)\ (Im\ ub))$
 $+ proots-line\ p\ (\text{Complex}\ (Re\ lb)\ (Im\ ub))\ lb$
 $\langle proof \rangle$

lemma *proots-rect-border-smods*:
fixes $p::\text{complex}\ poly$ **and** $lb\ ub::\text{complex}$
assumes $ab-le:Re\ lb < Re\ ub\ Im\ lb < Im\ ub$
and $not-van:not-rect-vanishing\ p\ lb\ ub$
shows $proots-rect-border\ p\ lb\ ub =$

```

    (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⟩

```

lemma *proots-rect-smods*:

```

  assumes Re lb < Re ub Im lb < Im ub
  and not-van: not-rect-vanishing p lb ub
  shows proots-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⟩

```

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⟩

```

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⟩

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⟩

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;
          lb);];
            p2 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
            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;
          ub);];
            p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
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⟩

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 Mathemat-*

ical Monthly, 119(9):715–752, 2012.

- [2] Q. I. Rahman and G. Schmeisser. *Analytic theory of polynomials*. Number 26. Oxford University Press, 2002.