Count the Number of Complex Roots

Wenda Li

May 26, 2024

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* $\circ_p$ [:*a*, *b*−*a*:]) (*of-real t*)
⟨*proof*⟩

lemma *path-poly-comp*[*intro*]:
fixes *p*::′*a*::*real-normed-field poly*
shows *path g* ⟹ *path* (*poly p o g*)
⟨*proof*⟩

lemma *cindex-poly-noroot*:
assumes *a*<*b* ∀ *x*. *a*<*x* ∧ *x*<*b* ⟶ *poly p x*≠*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* ⟨*proof*⟩

interpretation *Re-poly-hom*:*map-poly-comm-monoid-add-hom Re*
⟨*proof*⟩

interpretation *Im-poly-hom*:*map-poly-comm-monoid-add-hom Im*
⟨*proof*⟩

1.3 More about *order*

lemma *order-normalize*[*simp*]:*order x* (*normalize p*) = *order x p*
⟨*proof*⟩

lemma *order-gcd*:
assumes *p*≠*0 q*≠*0*
shows *order x* (*gcd p q*) = *min* (*order x p*) (*order x q*)
⟨*proof*⟩

lemma *pderiv-power*: *pderiv* (*p* ^ *n*) = *smult* (*of-nat n*) (*p* ^ (*n*−*1*)) ∗ *pderiv p*
⟨*proof*⟩

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*)) ⟨*proof*⟩

1.4 More about *rsquarefree*

lemma *rsquarefree-0*[*simp*]: ¬ *rsquarefree 0*
⟨*proof*⟩

lemma *rsquarefree-times*:
assumes *rsquarefree* (*p*∗*q*)
shows *rsquarefree q* ⟨*proof*⟩

lemma *rsquarefree-smult-iff*:
assumes *s*≠*0*
shows *rsquarefree* (*smult s p*) ⟷ *rsquarefree p*
⟨*proof*⟩

lemma *card-proots-within-rsquarefree*:
assumes *rsquarefree p*
shows *proots-count p s* = *card* (*proots-within p s*) ⟨*proof*⟩

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*⟩

lemma *poly-gcd-pderiv-iff*:
fixes *p*::*'a*::{*semiring-char-0*,*factorial-ring-gcd*,*semiring-gcd-mult-normalize*} *poly*
shows *poly* (*p div* (*gcd p* (*pderiv p*))) *x* =*0* ⟷ *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 poly*
defines *q1*≡ *fcompose p* [:(*z0*+*r*)∗i,*z0*−*r*:] [:i,*1*:] **and** *q2* ≡ [:i,*1*:] ^ *degree p*
assumes *0*≤*t t*≤*1 t*≠*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 poly* ⇒ *real poly* ⇒ *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*]:
cpoly-of pR pI = *0* ⟷ *pR* = *0* ∧ *pI* = *0*
⟨*proof*⟩

lemma *cpoly-of-decompose*:
p = *cpoly-of* (*map-poly Re p*) (*map-poly Im p*)
⟨*proof*⟩

lemma *cpoly-of-dist-right*:
cpoly-of (*pR*∗*g*) (*pI*∗*g*) = *cpoly-of pR pI* ∗ (*map-poly of-real g*)
⟨*proof*⟩

lemma *poly-cpoly-of-real*:
poly (*cpoly-of pR pI*) (*of-real x*) = *Complex* (*poly pR x*) (*poly pI x*)
⟨*proof*⟩

lemma *poly-cpoly-of-real-iff*:
shows *poly* (*cpoly-of pR pI*) (*of-real t*) =*0* ⟷ *poly pR t* = *0* ∧ *poly pI t*=*0*
⟨*proof*⟩

lemma *order-cpoly-gcd-eq*:
assumes *pR*≠*0* ∨ *pI*≠*0*
shows *order t* (*cpoly-of pR pI*) = *order t* (*gcd pR pI*)
⟨*proof*⟩

lemma *cpoly-of-times*:
shows *cpoly-of pR pI* ∗ *cpoly-of qR qI* = *cpoly-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 ⟦*?a* < *?b*; *poly ?p ?a* ≠ *0*; *poly ?p ?b* ≠ *0*⟧ ⟹ *cindex-poly ?a ?b ?q ?p* = *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* ⇒ '*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* ⟷ (∃ *a*. *a*≠*0* ∧ *p*=[:*a*:])
⟨*proof*⟩

lemma *eventually-poly-nz-at-within*:
fixes *x* :: '*a*::{*idom*,*euclidean-space*}
assumes *p*≠*0*
shows *eventually* (λ*x*. *poly p x*≠*0*) (*at x within S*)

⟨*proof*⟩

lemma *sgn-power*:
  fixes *x*::*'a*::*linordered-idom*
  shows *sgn* (*x*^*n*) = (*if n*=*0 then 1 else if even n then* |*sgn x*| *else sgn x*)
  ⟨*proof*⟩

lemma *poly-divide-filterlim-at-top*:
  fixes *p q*::*real poly*
  defines *ll*≡(*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*≠*0 q*≠*0*
  shows *filterlim* (λ*x*. *poly q x* / *poly p x*) *ll at-top*
⟨*proof*⟩

lemma *poly-divide-filterlim-at-bot*:
  fixes *p q*::*real poly*
  defines *ll*≡(*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*≠*0 q*≠*0*
  shows *filterlim* (λ*x*. *poly q x* / *poly p x*) *ll at-bot*
⟨*proof*⟩

lemma *sgnx-poly-times*:
  assumes *F*=*at-bot* ∨ *F*=*at-top* ∨ *F*=*at-right x* ∨ *F*=*at-left x*
  shows *sgnx* (*poly* (*p*∗*q*)) *F* = *sgnx* (*poly p*) *F* ∗ *sgnx* (*poly q*) *F*
    (**is** *?PQ* = *?P* ∗ *?Q*)
⟨*proof*⟩

lemma *sgnx-poly-plus*:
  assumes *poly p x*=*0 poly q x*≠*0* **and** *F*:*F*=*at-right x* ∨ *F*=*at-left x*
  shows *sgnx* (*poly* (*p*+*q*)) *F* = *sgnx* (*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*
  ⟨*proof*⟩

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*⟩

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* ($p \circ_p$ *r0*) ($q \circ_p$ *r0*)
⟨*proof*⟩

lemma *finite-ReZ-segments-poly-rectpath*:
    finite-ReZ-segments (*poly p* ∘ *rectpath a b*) *z*
  ⟨*proof*⟩

lemma *valid-path-poly-linepath*:
  fixes *a b*::*'a*::*real-normed-field*
  shows *valid-path* (*poly p o linepath a b*)
⟨*proof*⟩

lemma *valid-path-poly-rectpath*: *valid-path* (*poly p o rectpath a b*)
  ⟨*proof*⟩

2.2 Sign difference

definition *psign-diff* :: *real poly* ⇒*real poly* ⇒ *real* ⇒ *int* **where**
  psign-diff p q x = (*if poly p x* = *0* ∧ *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-diff p q x* = |*sign* (*poly p x*) − *sign* (*poly q x*)|
  ⟨*proof*⟩

lemma *psign-diff-0*[*simp*]:
  psign-diff 0 q x = *1*
  psign-diff p 0 x = *1*
  ⟨*proof*⟩

lemma *psign-diff-poly-commute*:
  psign-diff p q x = *psign-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 *cross-alt* (*q* ∗ *r*) (*q* ∗ *s*) *a b* = *cross-alt r s a b*
⟨*proof*⟩

lemma *cross-alt-noroot*:
assumes *a*<*b* **and** ∀ *x*. *a*≤*x* ∧ *x*≤*b* ⟶ *poly* (*p*∗*q*) *x*≠*0*
shows *cross-alt p q a b* = *0*
⟨*proof*⟩

lemma *cross-alt-linear-comp*:
fixes *b c*::*real*
defines *h* ≡ (λ*p*. *pcompose p* [:*b*,*c*:])
shows *cross-alt* (*h p*) (*h q*) *lb ub* = *cross-alt p q* (*c* ∗ *lb* + *b*) (*c* ∗ *ub* + *b*)
⟨*proof*⟩

2.4 Alternative sign variation sequencse

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* (λ*p*. *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*⟩

2.5 jumpF on polynomials

definition *jumpF-polyR*:: *real poly* ⇒ *real poly* ⇒ *real* ⇒ *real* **where**
jumpF-polyR q p a = *jumpF* (λ*x*. *poly q x* / *poly p x*) (*at-right a*)

definition *jumpF-polyL*:: *real poly* ⇒ *real poly* ⇒ *real* ⇒ *real* **where**
jumpF-polyL q p a = *jumpF* (λ*x*. *poly q x* / *poly p x*) (*at-left a*)

definition *jumpF-poly-top*:: *real poly* ⇒ *real poly* ⇒ *real* **where**
jumpF-poly-top q p = *jumpF* (λ*x*. *poly q x* / *poly p x*) *at-top*

definition *jumpF-poly-bot*:: *real poly* ⇒ *real poly* ⇒ *real* **where**
jumpF-poly-bot q p = *jumpF* (λ*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*
⟨*proof*⟩

lemma *jumpF-polyL-0*[*simp*]: *jumpF-polyL 0 p a* = *0 jumpF-polyL q 0 a* = *0*
⟨*proof*⟩

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

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

lemma *jumpF-poly-noroot*:
assumes *poly p a*≠*0*
shows *jumpF-polyL q p a* = *0 jumpF-polyR q p a* = *0*
⟨*proof*⟩

lemma *jumpF-polyR-coprime′*:
assumes *poly p x*≠*0* ∨ *poly q x*≠*0*
shows *jumpF-polyR q p x* = (*if p* ≠ *0* ∧ *q* ≠ *0* ∧ *poly p x*=*0 then*
if sign-r-pos p x ⟷ *poly q x*>*0 then 1/2 else* − *1/2*
else 0)
⟨*proof*⟩

lemma *jumpF-polyR-coprime*:
assumes *coprime p q*
shows *jumpF-polyR q p x* = (*if p* ≠ *0* ∧ *q* ≠ *0* ∧ *poly p x*=*0 then*
if sign-r-pos p x ⟷ *poly q x*>*0 then 1/2 else* − *1/2*
else 0)
⟨*proof*⟩

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*⟩

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*)

⟨*proof*⟩

lemma *jumpF-times*:
assumes *tendsto*:$(f \longrightarrow c)$ *F* **and** $c \neq 0$ $F \neq bot$
shows *jumpF* $(\lambda x.\ f\ x * g\ x)$ F = *sgn c* ∗ *jumpF g F*
⟨*proof*⟩

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*⟩

lemma *jumpF-polyL-inverse-add*:
assumes *coprime p q*
shows *jumpF-polyL q p x* + *jumpF-polyL p q x* = *jumpF-polyL 1* $(q*p)$ *x*
⟨*proof*⟩

lemma *jumpF-polyL-smult-1*:
jumpF-polyL (*smult c q*) *p x* = *sgn c* ∗ *jumpF-polyL q p x*
⟨*proof*⟩

lemma *jumpF-polyR-smult-1*:
jumpF-polyR (*smult c q*) *p x* = *sgn c* ∗ *jumpF-polyR q p x*
⟨*proof*⟩

lemma
shows *jumpF-polyR-mod*:*jumpF-polyR q p x* = *jumpF-polyR* (*q mod p*) *p x* **and**
jumpF-polyL-mod:*jumpF-polyL q p x* = *jumpF-polyL* (*q mod p*) *p x*
⟨*proof*⟩

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*⟩

lemma
assumes *order x q* < *order x r* $q \neq 0$
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*⟩

lemma *jumpF-poly-top-0*[*simp*]: *jumpF-poly-top 0 p* = *0* *jumpF-poly-top q 0* = *0*
⟨*proof*⟩

lemma *jumpF-poly-bot-0*[*simp*]: *jumpF-poly-bot 0 p* = *0* *jumpF-poly-bot q 0* = *0*
⟨*proof*⟩

lemma *jumpF-poly-top-code*:
jumpF-poly-top q p = (*if p≠0* ∧ *q≠0* ∧ *degree q>degree p then*
if sgn-pos-inf q ∗ *sgn-pos-inf p* > *0 then 1/2 else* −*1/2 else 0*)
⟨*proof*⟩

lemma *jumpF-poly-bot-code*:
jumpF-poly-bot q p = (*if p≠0* ∧ *q≠0* ∧ *degree q>degree p then*
if sgn-neg-inf q ∗ *sgn-neg-inf p* > *0 then 1/2 else* −*1/2 else 0*)
⟨*proof*⟩

lemma *jump-poly-jumpF-poly*:
shows *jump-poly q p x* = *jumpF-polyR q p x* − *jumpF-polyL q p x*
⟨*proof*⟩

2.6 The extended Cauchy index on polynomials

definition *cindex-polyE*:: *real* ⇒ *real* ⇒ *real poly* ⇒ *real poly* ⇒ *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* ⇒ *real poly* ⇒ *int* **where**
cindex-poly-ubd q p = (*THE l.* (∀ $_F$ *r in at-top. cindexE* (−*r*) *r* (λ*x. 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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *cindexE-eq-cindex-polyE*:
assumes *a<b*
shows *cindexE a b* (λ*x. poly q x/poly p x*) = *cindex-polyE a b q p*
⟨*proof*⟩

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*⟩

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*)

⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *cindex-polyE-mod*:
fixes *p q*::*real poly*
shows *cindex-polyE a b q p* = *cindex-polyE a b* (*q mod p*) *p*
⟨*proof*⟩

lemma *cindex-polyE-rec*:
fixes *p q*::*real poly*
assumes $a < b$ *coprime p q*
shows *cindex-polyE a b q p* = *cross-alt q p a b*/*2* + *cindex-polyE a b* (− (*p mod q*)) *q*
⟨*proof*⟩

lemma *cindex-polyE-changes-alt-itv-mods*:
assumes $a<b$ *coprime p q*
shows *cindex-polyE a b q p* = *changes-alt-itv-smods a b p q* / *2* ⟨*proof*⟩

lemma *cindex-poly-ubd-eventually*:
shows $\forall_F$ *r in at-top*. *cindexE* (−*r*) *r* (λ*x*. *poly q x*/*poly p x*) = *of-int* (*cindex-poly-ubd q p*)
⟨*proof*⟩

lemma *cindex-poly-ubd-0*:
assumes *p*=*0* ∨ *q*=*0*
shows *cindex-poly-ubd q p* = *0*
⟨*proof*⟩

lemma *cindex-poly-ubd-code*:
  shows *cindex-poly-ubd q p* = *changes-R-smods p q*
⟨*proof*⟩

lemma *cindexE-ubd-poly*: *cindexE-ubd* (λ*x*. *poly q x*/*poly p x*) = *cindex-poly-ubd q p*
⟨*proof*⟩

lemma *cindex-polyE-noroot*:
  assumes *a*<*b* ∀ *x*. *a*≤*x* ∧ *x*≤*b* ⟶ *poly p x*≠*0*
  shows *cindex-polyE a b q p* = *0*
⟨*proof*⟩

lemma *cindex-polyE-combine*:
  assumes *a*<*b* *b*<*c*
  shows *cindex-polyE a b q p* + *cindex-polyE b c q p* = *cindex-polyE a c q p*
⟨*proof*⟩

lemma *cindex-polyE-linear-comp*:
  fixes *b c*::*real*
  defines *h* ≡ (λ*p*. *pcompose p* [:*b*,*c*:])
  assumes *lb*<*ub* *c*≠*0*
  shows *cindex-polyE lb ub* (*h q*) (*h p*) =
        (*if 0* < *c then cindex-polyE* (*c* ∗ *lb* + *b*) (*c* ∗ *ub* + *b*) *q p*
         else − *cindex-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-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* (**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*≠*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*⟩

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

2.7 More Cauchy indices on polynomials

definition *cindexP-pathE*::*complex poly* ⇒ (*real* ⇒ *complex*) ⇒ *real* **where**
cindexP-pathE p g = *cindex-pathE* (*poly p o g*) *0*

definition *cindexP-lineE* :: *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *real* **where**
cindexP-lineE p a b = *cindexP-pathE p* (*linepath a b*)

lemma *cindexP-pathE-const*:*cindexP-pathE* [:*c*:] *g* = *0*
⟨*proof*⟩

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*)
= *cindexP-pathE p g1* + *cindexP-pathE p g2*
⟨*proof*⟩

lemma *cindexP-lineE-polyE*:
fixes *p*::*complex poly* **and** *a b*::*complex*
defines *pp* ≡ *pcompose p* [:*a*, *b*−*a*:]
defines *pR* ≡ *map-poly Re pp*
and *pI* ≡ *map-poly Im pp*
shows *cindexP-lineE p a b* = *cindex-polyE 0 1 pI pR*
⟨*proof*⟩

definition *psign-aux* :: *complex poly* ⇒ *complex poly* ⇒ *complex* ⇒ *int* **where**
psign-aux p q b =
sign (*Im* (*poly p b* ∗ *poly q b*) ∗ (*Im* (*poly p b*) ∗ *Im* (*poly q b*)))
+ *sign* (*Re* (*poly p b* ∗ *poly q b*) ∗ *Im* (*poly p b* ∗ *poly q b*))
− *sign* (*Re* (*poly p b*) ∗ *Im* (*poly p b*))
− *sign* (*Re* (*poly q b*) ∗ *Im* (*poly q b*))

definition *cdiff-aux* :: *complex poly* ⇒ *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *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*≠*0 poly p b*≠*0 poly q a*≠*0 poly q b*≠*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*⟩

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*:
  $[\![Re\ lb = Re\ ub;\ Im\ lb \leq Im\ ub\ ]\!]$
    $\Longrightarrow x \in closed\text{-}segment\ lb\ ub \longleftrightarrow Re\ x = Re\ lb \wedge Im\ lb \leq Im\ x \wedge Im\ x \leq Im\ ub$
  $[\![Im\ lb = Im\ ub;\ Re\ lb \leq Re\ ub\ ]\!]$
    $\Longrightarrow x \in closed\text{-}segment\ lb\ ub \longleftrightarrow Im\ x = Im\ lb \wedge Re\ lb \leq Re\ x \wedge Re\ x \leq Re\ ub$
$\langle proof \rangle$

corollary *path-image-part-circlepath-subset*:
  assumes $r \geq 0$
  shows $path\text{-}image(part\text{-}circlepath\ z\ r\ st\ tt) \subseteq sphere\ z\ r$
$\langle proof \rangle$

proposition *in-path-image-part-circlepath*:
  assumes $w \in path\text{-}image(part\text{-}circlepath\ z\ r\ st\ tt)\ \ 0 \leq r$
  shows $norm(w - z) = r$
$\langle proof \rangle$

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

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

lemma *infinite-sphere*:
  fixes $a :: complex$
  assumes $r > 0$
  shows $infinite\ (sphere\ a\ r)$
$\langle proof \rangle$

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

lemma (**in** *ring-1*) *Ints-minus2*: $-\ a \in \mathbb{Z} \Longrightarrow a \in \mathbb{Z}$
  $\langle proof \rangle$

lemma *dvd-divide-Ints-iff*:
  $b\ dvd\ a \vee b{=}0 \longleftrightarrow of\text{-}int\ a\ /\ of\text{-}int\ b \in (\mathbb{Z} :: 'a :: \{field,ring\text{-}char\text{-}0\}\ set)$
$\langle proof \rangle$

lemma *of-int-div-field*:
assumes *d dvd n*
shows (*of-int*::-⇒′*a*::*field-char-0*) (*n div d*) = *of-int n* / *of-int d*
⟨*proof*⟩

lemma *powr-eq-1-iff*:
assumes *a*>*0*
shows (*a*::*real*) *powr b* =*1* ⟷ *a*=*1* ∨ *b*=*0*
⟨*proof*⟩

lemma *tan-inj-pi*:
− (*pi*/*2*) < *x* ⟹ *x* < *pi*/*2* ⟹ − (*pi*/*2*) < *y* ⟹ *y* < *pi*/*2* ⟹ *tan x* = *tan y* ⟹ *x* = *y*
⟨*proof*⟩

lemma *finite-ReZ-segments-poly-circlepath*:
finite-ReZ-segments (*poly p* ∘ *circlepath z0 r*) *0*
⟨*proof*⟩

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*
⟨*proof*⟩

2.9 Some useful conformal/*bij-betw* properties

lemma *bij-betw-plane-ball*:*bij-betw* (λ*x*. (i−*x*)/(i+*x*)) {*x*. *Im x*>*0*} (*ball 0 1*)
⟨*proof*⟩

lemma *bij-betw-axis-sphere*:*bij-betw* (λ*x*. (i−*x*)/(i+*x*)) {*x*. *Im x*=*0*} (*sphere 0 1* − {−*1*})
⟨*proof*⟩

lemma *bij-betw-ball-uball*:
assumes *r*>*0*
shows *bij-betw* (λ*x*. *complex-of-real r*∗*x* + *z0*) (*ball 0 1*) (*ball z0 r*)
⟨*proof*⟩

lemma *bij-betw-sphere-usphere*:
assumes *r*>*0*
shows *bij-betw* (λ*x*. *complex-of-real r*∗*x* + *z0*) (*sphere 0 1*) (*sphere z0 r*)
⟨*proof*⟩

lemma *proots-ball-plane-eq*:
defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
assumes *p*≠*0*
shows *proots-count p* (*ball 0 1*) = *proots-count* (*fcompose p q1 q2*) {*x*. *0* < *Im*

x}
⟨*proof*⟩

lemma *proots-sphere-axis-eq*:
defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
assumes *p*≠*0*
shows *proots-count p* (*sphere 0 1* − {− *1*}) = *proots-count* (*fcompose p q1 q2*) {*x*. *0* = *Im x*}
⟨*proof*⟩

lemma *proots-card-ball-plane-eq*:
defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
assumes *p*≠*0*
shows *card* (*proots-within p* (*ball 0 1*)) = *card* (*proots-within* (*fcompose p q1 q2*) {*x*. *0* < *Im x*})
⟨*proof*⟩

lemma *proots-card-sphere-axis-eq*:
defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
assumes *p*≠*0*
shows *card* (*proots-within p* (*sphere 0 1* − {− *1*}))
= *card* (*proots-within* (*fcompose p q1 q2*) {*x*. *0* = *Im x*})
⟨*proof*⟩

lemma *proots-uball-eq*:
fixes *z0*::*complex* **and** *r*::*real*
defines *q*≡[:*z0*, *of-real r*:]
assumes *p*≠*0* **and** *r*>*0*
shows *proots-count p* (*ball z0 r*) = *proots-count* ($p \circ_p q$) (*ball 0 1*)
⟨*proof*⟩

lemma *proots-card-uball-eq*:
fixes *z0*::*complex* **and** *r*::*real*
defines *q*≡[:*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*))
⟨*proof*⟩

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* ($p \circ_p q$) (*sphere 0 1*))
⟨*proof*⟩

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

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

lemma *card-proots-open-segments*:
  assumes *poly p st* ≠*0 poly p tt* ≠ *0*
  shows *card* (*proots-within p* (*open-segment st tt*)) =
        (*let pc* = *pcompose p* [:*st*, *tt* − *st*:];
          pR = *map-poly Re pc*;
          pI = *map-poly Im pc*;
          g = *gcd pR pI*
        in changes-itv-smods 0 1 g (*pderiv g*)) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *unbounded-line-closed-segment*: *closed-segment a b* ⊆ *unbounded-line a b*
  ⟨*proof*⟩

lemma *card-proots-unbounded-line*:
  assumes *st*≠*tt*
  shows *card* (*proots-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*⟩

lemma *proots-count-gcd-eq*:
  fixes *p*::*complex poly* **and** *st tt*::*complex*
    and *g*::*real poly*
  defines *pc* ≡ *pcompose p* [:*st*, *tt* − *st*:]
  defines *pR* ≡ *map-poly Re pc* **and** *pI* ≡ *map-poly Im pc*
  defines *g* ≡ *gcd pR pI*
  assumes *st*≠*tt p*≠*0*
    and *s1-def*:*s1* = (λ*x*. *poly* [:*st*, *tt* − *st*:] (*of-real x*)) ‘ *s2*
  shows *proots-count p s1* = *proots-count g s2*
⟨*proof*⟩

lemma *proots-unbounded-line*:
  assumes *st*≠*tt p*≠*0*
  shows (*proots-count 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-ext g* (*pderiv g*))) (**is** *?L* = *?R*)
⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

2.11 Checking if there a polynomial root on a closed segment

definition *no-proots-line*::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *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* ≠*0* ∧ *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 *proots-upper* ::*complex poly* ⇒ *nat* **where**
proots-upper p= *proots-count p* {*z*. *Im z*>*0*}

— Roots counted WITHOUT multiplicity
definition *proots-upper-card*::*complex poly* ⇒ *nat* **where**
proots-upper-card p = *card* (*proots-within p* {*x*. *Im x* >*0*})

lemma *Im-Ln-tendsto-at-top*: ((λ*x*. *Im* (*Ln* (*Complex a x*))) ⟶ *pi*/*2*) *at-top*
⟨*proof*⟩

lemma *Im-Ln-tendsto-at-bot*: ((λ*x*. *Im* (*Ln* (*Complex a x*))) ⟶ − *pi*/*2*) *at-bot*

⟨*proof*⟩

lemma *Re-winding-number-tendsto-part-circlepath*:
shows ((λ*r*. *Re* (*winding-number* (*part-circlepath z0 r 0 pi*) *a*)) ⟶ *1*/*2*) *at-top*
⟨*proof*⟩

lemma *not-image-at-top-poly-part-circlepath*:
assumes *degree p*>*0*
shows ∀$_F$ *r in at-top*. *b*∉*path-image* (*poly p o part-circlepath z0 r st tt*)
⟨*proof*⟩

lemma *not-image-poly-part-circlepath*:
assumes *degree p*>*0*
shows ∃ *r*>*0*. *b*∉*path-image* (*poly p o part-circlepath z0 r st tt*)
⟨*proof*⟩

lemma *Re-winding-number-poly-part-circlepath*:
assumes *degree p*>*0*
shows ((λ*r*. *Re* (*winding-number* (*poly p o part-circlepath z0 r 0 pi*) *0*)) ⟶ *degree p*/*2*) *at-top*
⟨*proof*⟩

lemma *Re-winding-number-poly-linepth*:
fixes *pp*::*complex poly*
defines *g* ≡ (λ*r*. *poly pp o linepath* (−*r*) (*of-real r*))
assumes *lead-coeff pp*=*1* **and** *no-real-zero*:∀ *x*∈*proots pp*. *Im x*≠*0*
shows ((λ*r*. *2*∗*Re* (*winding-number* (*g r*) *0*) + *cindex-pathE* (*g r*) *0*) ⟶ *0*) *at-top*
⟨*proof*⟩

lemma *proots-upper-cindex-eq*:

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*⟩

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-proots*:∀ *x*∈*proots r*. *Im x*=*Im a*
shows *cindexE lb ub* (λ*t*. *Im* ((*poly p* ∘ *g*) *t*) / *Re* ((*poly p* ∘ *g*) *t*)) =
cindexE lb ub (λ*t*. *Im* ((*poly q* ∘ *g*) *t*) / *Re* ((*poly q* ∘ *g*) *t*))
⟨*proof*⟩

lemma *poly-decompose-by-proots*:
fixes *p* ::′*a*::*idom poly*
assumes *p*≠*0*
shows ∃ *q r*. *p* = *q* ∗ *r* ∧ *lead-coeff q*=*1* ∧ (∀ *x*∈*proots q*. *P x*) ∧ (∀ *x*∈*proots r*. ¬*P x*) ⟨*proof*⟩

lemma *proots-upper-cindex-eq*′:
assumes *lead-coeff p*=*1*
shows *proots-upper p* = (*degree p* − *proots-count p* {*x*. *Im x*=*0*}
− *cindex-poly-ubd* (*map-poly Im p*) (*map-poly Re p*)) /*2*
⟨*proof*⟩

lemma *proots-within-upper-squarefree*:
assumes *rsquarefree p*
shows *card* (*proots-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
nat ((*degree p* − *changes-R-smods g* (*pderiv g*) − *changes-R-smods pR pI*) *div 2*)
)
⟨*proof*⟩

lemma *proots-upper-code1* [*code*]:
proots-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 ''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* ⇒ *complex* ⇒ *complex* ⇒ *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* $\circ_p$ [:*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* ⇒ *complex* ⇒ *real* ⇒ *nat* **where**
proots-ball p z0 r = *proots-count p* (*ball z0 r*)

— Roots counted WITHOUT multiplicity
definition *proots-ball-card* ::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *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 ≤ 0 then
                          0
                        else if p≠0 then
                         proots-upper (fcompose (p ∘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 ≤ 0 ∨ p=0 then
             0
          else
             proots-upper-card (fcompose (p ∘p [:z0, of-real r:]) [:i,−1:] [:i,1:])
             )
```
⟨*proof*⟩

2.16 Polynomial roots on a circle (sphere)

definition *proots-sphere*::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *nat* **where**
proots-sphere p z0 r = *proots-count p* (*sphere z0 r*)

— Roots counted WITHOUT multiplicity
definition *proots-sphere-card* ::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *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 ∨ p=0 then
             0
          else
```

(*if poly p* (*z0*−*r*) =*0 then 1 else 0*) +
proots-unbounded-line-card (*fcompose* ($p \circ_p$ [:*z0*, *of-real r*:]) [:i,−*1*:]
[:i,*1*:])
0 1
)
⟨*proof*⟩

2.17 Polynomial roots on a closed ball

definition *proots-cball*::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *nat* **where**
proots-cball p z0 r = *proots-count p* (*cball z0 r*)

— Roots counted WITHOUT multiplicity
definition *proots-cball-card* ::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *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* ∨ *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*⟩

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 (⟦*open ?S*; *connected ?S*; *?f holomorphic-on ?S* − *?poles*; *?h holomorphic-on ?S*; *valid-path ?g*; *pathfinish ?g* = *pathstart ?g*; *path-image ?g* ⊆ *?S* − {*w* ∈ *?S*. *?f w* = *0* ∨ *w* ∈ *?poles*}; ∀ *z*. *z* ∉ *?S* ⟶ *winding-number ?g z* = *0*; *finite* {*w* ∈ *?S*. *?f w* = *0* ∨ *w* ∈ *?poles*}; ∀ *p*∈*?S* ∩ *?poles*. *is-pole ?f p*⟧ ⟹ *contour-integral ?g* (λ*x*. *deriv ?f x* ∗ *?h x* / *?f x*) = *complex-of-real* (*2* ∗ *pi*) ∗ i ∗ (∑ *p*∈{*w* ∈ *?S*. *?f w* = *0* ∨ *w* ∈ *?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*:] $s = 0$
  ⟨*proof*⟩

lemma *proots-count-nzero*:
  assumes $\bigwedge x.\ x \in s \Longrightarrow$ *poly p* $x \neq 0$
  shows *proots-count p s* $= 0$
  ⟨*proof*⟩

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

2.19 Counting roots in a rectangle

definition *proots-rect* ::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
  proots-rect p lb ub = *proots-count p* (*box lb ub*)

definition *proots-crect* ::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
  proots-crect p lb ub = *proots-count p* (*cbox lb ub*)

definition *proots-rect-ll* ::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
  proots-rect-ll p lb ub = *proots-count p* (*box lb ub* ∪ {*lb*}
    ∪ *open-segment lb* (*Complex* (*Re ub*) (*Im lb*))
    ∪ *open-segment lb* (*Complex* (*Re lb*) (*Im ub*)))

definition *proots-rect-border*::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
  proots-rect-border p a b = *proots-count p* (*path-image* (*rectpath a b*))

definition *not-rect-vertex*::*complex* ⇒ *complex* ⇒ *complex* ⇒ *bool* **where**
  not-rect-vertex r a b = (*r*≠*a* ∧ *r* ≠ *Complex* (*Re b*) (*Im a*) ∧ *r*≠*b* ∧ *r*≠*Complex*
(*Re a*) (*Im b*))

definition *not-rect-vanishing* :: *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *bool* **where**
  not-rect-vanishing p a b = (*poly p a*≠*0* ∧ *poly p* (*Complex* (*Re b*) (*Im a*)) ≠ *0*
    ∧ *poly p b* ≠*0* ∧ *poly p* (*Complex* (*Re a*) (*Im b*))≠ *0*)

lemma *cindexP-rectpath-edge-base*:
  assumes *Re a* < *Re b Im a* < *Im b*
    and *not-rect-vertex r a b*
    and *r*∈*path-image* (*rectpath a b*)

shows *cindexP-pathE* [:−*r*,*1*:] (*rectpath a b*) = −*1*
⟨*proof*⟩

lemma *cindexP-rectpath-vertex-base*:
assumes *Re a* < *Re b Im a* < *Im b*
and ¬ *not-rect-vertex r a b*
shows *cindexP-pathE* [:−*r*,*1*:] (*rectpath a b*) = −*1*/*2*
⟨*proof*⟩

lemma *cindexP-rectpath-interior-base*:
assumes *r*∈*box a b*
shows *cindexP-pathE* [:−*r*,*1*:] (*rectpath a b*) = −*2*
⟨*proof*⟩

lemma *cindexP-rectpath-outside-base*:
assumes *Re a* < *Re b Im a* < *Im b*
and *r*∉*cbox a b*
shows *cindexP-pathE* [:−*r*,*1*:] (*rectpath a b*) = *0*
⟨*proof*⟩

lemma *cindexP-rectpath-add-one-root*:
assumes *Re a* < *Re b Im a* < *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*∈*box a b then* −*2 else if r*∈*path-image* (*rectpath a b*) *then* − *1 else* *0*)
⟨*proof*⟩

lemma *proots-rect-cindexP-pathE*:
assumes *Re a* < *Re b Im a* < *Im b*
and *not-rect-vanishing p a b*
shows *proots-rect p a b* = −(*proots-rect-border p a b* +*cindexP-pathE p* (*rectpath a b*)) / *2*
⟨*proof*⟩

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*⟩

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*⟩

lemma *open-segment-Re-equal*:
assumes *Re x* = *Re y Im x*≠*Im y*
shows *open-segment x y* = {*z*. *Re z* = *Re x*
∧ *Im z* ∈ *open-segment* (*Im x*) (*Im y*)}
⟨*proof*⟩

lemma *Complex-eq-iff*:
x = *Complex y z* ⟷ *Re x* = *y* ∧ *Im x* = *z*
Complex y z = *x* ⟷ *Re x* = *y* ∧ *Im x* = *z*
⟨*proof*⟩

lemma *proots-rect-border-eq-lines*:
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 *proots-rect-border p lb ub* =
proots-line p lb (*Complex* (*Re ub*) (*Im lb*))
+ *proots-line p* (*Complex* (*Re ub*) (*Im lb*)) *ub*
+ *proots-line p ub* (*Complex* (*Re lb*) (*Im ub*))
+ *proots-line p* (*Complex* (*Re lb*) (*Im ub*)) *lb*
⟨*proof*⟩

lemma *proots-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 *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;
              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⟩
```

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.