# Count the Number of Complex Roots 

Wenda Li

September 13, 2023


#### 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<br>Winding-Number-Eval.Missing-Algebraic<br>Winding-Number-Eval.Missing-Transcendental<br>Sturm-Tarski.PolyMisc<br>Budan-Fourier.BF-Misc<br>Polynomial-Interpolation.Ring-Hom-Poly<br>begin

### 1.1 Misc

lemma poly-linepath-comp':
fixes $a::^{\prime} a::\{$ real-normed-vector,comm-semiring-0,real-algebra-1 $\}$
shows poly $p$ (linepath $a b t)=\operatorname{poly}\left(p \circ_{p}[: a, b-a:]\right)(o f-r e a l t)$
by (auto simp add:poly-pcompose linepath-def scaleR-conv-of-real algebra-simps)
lemma path-poly-comp[intro]:

shows path $g \Longrightarrow$ path (poly pog)
apply (elim path-continuous-image)
by (auto intro:continuous-intros)
lemma cindex-poly-noroot:
assumes $a<b \forall x . a<x \wedge x<b \longrightarrow$ poly $p x \neq 0$
shows cindex-poly abqp=0
unfolding cindex-poly-def
apply (rule sum.neutral)
using assms by (auto intro:jump-poly-not-root)

### 1.2 More polynomial homomorphism interpretations

interpretation of-real-poly-hom:map-poly-inj-idom-hom of-real ..
interpretation Re-poly-hom:map-poly-comm-monoid-add-hom Re
by unfold-locales simp-all
interpretation Im-poly-hom:map-poly-comm-monoid-add-hom Im by unfold-locales simp-all

### 1.3 More about order

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

lemma order-gcd:
assumes $p \neq 0 \quad q \neq 0$
shows order $x(\operatorname{gcd} p q)=\min (\operatorname{order} x p)(\operatorname{order} x q)$
proof -
define $x x$ op oq where $x x=[:-x, 1:]$ and $o p=o r d e r x p$ and $o q=$ order $x q$
obtain $p p$ where $p p: p=x x \wedge o p * p p \neg x x$ dvd $p p$
using order-decomp $[O F\langle p \neq 0\rangle$,of $x$,folded $x x$-def op-def $]$ by auto
obtain $q q$ where $q q: q=x x$ ^oq* $q q \neg x x d v d q q$
using order-decomp $[O F\langle q \neq 0\rangle$, of $x$,folded $x x$-def oq-def] by auto
define $p q$ where $p q=g c d p p q q$
have $p$-unfold: $p=(p q * x x \wedge(\min o p o q)) *\left((p p\right.$ div $p q) * x x^{\wedge}(o p-\min o p$ $o q)$ )
and $[$ simp $]$ :coprime $x x$ ( $p p$ div $p q$ ) and $p p \neq 0$
proof -
have $x x^{\wedge} o p=x x^{\wedge}($ min $o p o q) * x x^{\wedge}(o p-\min o p o q)$
by (simp flip:power-add)
moreover have $p p=p q *(p p$ div $p q)$
unfolding $p q-d e f$ by simp
ultimately show $p=\left(p q * x x^{\wedge}(\min o p o q)\right) *\left((p p \operatorname{div} p q) * x x^{\wedge}(o p-\min \right.$ $o p o q)$ )
unfolding $p q$-def $p p$ by (auto simp:algebra-simps)
show coprime $x x$ ( $p p$ div $p q$ )
apply (rule prime-elem-imp-coprime $[O F$
prime-elem-linear-poly[of 1 -x,simplified],folded $x x$-def])
using $\langle p p=p q *(p p$ div $p q)\rangle p p(2)$ by auto
qed (use $p p\langle p \neq 0\rangle$ in auto)
have $q$-unfold: $q=\left(p q * x x^{\wedge}(\min o p o q)\right) *\left((q q \operatorname{div} p q) * x x^{\wedge}(o q-\min o p\right.$ $o q)$ )
and $[$ simp $]$ :coprime $x x$ ( $q q$ div $p q$ )
proof -
have $x x^{\wedge} o q=x x^{\wedge}($ min $o p o q) * x x^{\wedge}(o q-\min o p o q)$
by (simp flip:power-add)
moreover have $q q=p q *(q q$ div $p q)$
unfolding $p q$-def by simp
ultimately show $q=\left(p q * x x^{\wedge}(\min o p o q)\right) *\left((q q\right.$ div $p q) * x x^{\wedge}(o q-\min$ $o p o q)$ )
unfolding $p q$-def $q q$ by (auto simp:algebra-simps)
show coprime $x x$ ( $q q$ div $p q$ )
apply (rule prime-elem-imp-coprime $[O F$
prime-elem-linear-poly[of 1 -x,simplified],folded $x x$-def])
using $\langle q q=p q *(q q$ div $p q)\rangle q q(2)$ by auto
qed
have $g c d p q=$ normalize $\left(p q * x x^{\wedge}(\min o p o q)\right)$
proof -
have coprime $\left(p p\right.$ div $\left.p q * x x^{\wedge}(o p-\min o p o q)\right)\left(q q\right.$ div $p q * x x^{\wedge}(o q-\min$ $o p o q)$ )
proof (cases op>oq)
case True
then have $o q-\min o p o q=0$ by auto
moreover have coprime $\left(x x^{\wedge}(o p-\min o p o q)\right)(q q$ div $p q)$ by auto
moreover have coprime ( $p p$ div $p q$ ) ( $q q$ div $p q$ )
apply (rule div-gcd-coprime[of pp qq,folded $p q$-def])
using $\langle p p \neq 0$ by auto
ultimately show ?thesis by auto
next
case False
then have $o p-\min o p o q=0$ by auto
moreover have coprime ( $p p$ div $p q$ ) $\left(x x^{\wedge}(o q-\min o p o q)\right)$
by (auto simp:coprime-commute)
moreover have coprime ( $p p$ div $p q$ ) ( $q q$ div $p q$ )
apply (rule div-gcd-coprime[of pp qq,folded $p q$-def])
using $\langle p p \neq 0$ 〉 by auto
ultimately show ?thesis by auto
qed
then show?thesis unfolding $p$-unfold $q$-unfold
apply (subst gcd-mult-left)
by auto
qed
then have order $x(g c d p q)=$ order $x p q+\operatorname{order} x\left(x x{ }^{\wedge}(\min\right.$ op oq))
apply simp
apply (subst order-mult)
using assms(1) p-unfold by auto
also have $\ldots=$ order $x\left(x x^{\wedge}(\min\right.$ op oq $\left.)\right)$
using $p p(2) q q(2)$ unfolding $p q-d e f x x$-def
by (auto simp add: order-0I poly-eq-0-iff-dvd)
also have $\ldots=\min$ op oq
unfolding $x x$-def by (rule order-power- $n-n$ )
also have $\ldots=\min ($ order $x p)($ order $x q)$ unfolding op-def oq-def by simp
finally show? thesis .
qed
lemma pderiv-power: pderiv $\left(p^{\wedge} n\right)=\operatorname{smult}($ of-nat $n)\left(p^{\wedge}(n-1)\right) *$ pderiv $p$
apply (cases $n$ )
using pderiv-power-Suc by auto

```
lemma order-pderiv:
    fixes p::'a::{idom,semiring-char-0} poly
    assumes p\not=0 poly p x=0
    shows order x p = Suc (order x (pderiv p)) using assms
proof -
    define xx op where xx=[:-x,1:] and op = order x p
    have op \not=0 unfolding op-def using assms order-root by blast
    obtain pp where pp:p=xx^op * pp\neg xx dvd pp
        using order-decomp[OF <p\not=0\rangle,of x,folded xx-def op-def] by auto
    have p-der:pderiv p = smult (of-nat op) (xx^(op-1))*pp+xx`op*pderiv pp
        unfolding pp(1) by (auto simp:pderiv-mult pderiv-power xx-def algebra-simps
pderiv-pCons)
    have }x\mp@subsup{x}{}{`}(op-1) dvd (pderiv p
        unfolding p-der
            by (metis One-nat-def Suc-pred assms(1) assms(2) dvd-add dvd-mult-right
dvd-triv-left
            neq0-conv op-def order-root power-Suc smult-dvd-cancel)
    moreover have }\negxx`op dvd (pderiv p
    proof
        assume xx ^op dvd pderiv p
        then have }x\mp@subsup{x}{}{`
            unfolding p-der by (simp add:dvd-add-left-iff)
            then have }x\mp@subsup{x}{}{`}op dvd (xx`(op -1)) * p
            apply (elim dvd-monic[rotated])
            using <op\not=0\rangle by (auto simp:lead-coeff-power xx-def)
            then have }x\mp@subsup{x}{}{\wedge}(op-1)*xxdvd (xx^(op-1)
                using <\neg xx dvd pp\rangle by (simp add: <op \not=0`mult.commute power-eq-if)
            then have xx dvd 1
            using assms(1) pp(1) by auto
            then show False unfolding xx-def by (meson assms(1) dvd-trans one-dvd
order-decomp)
    qed
    ultimately have op - 1 = order x (pderiv p)
            using order-unique-lemma[of x op-1 pderiv p,folded xx-def] <op\not=0`
            by auto
    then show ?thesis using <op\not=0> unfolding op-def by auto
qed
```


### 1.4 More about rsquarefree

lemma rsquarefree- $0[$ simp $]$ : $\neg$ rsquarefree 0
unfolding rsquarefree-def by simp
lemma rsquarefree-times:
assumes rsquarefree ( $p * q$ )

```
    shows rsquarefree q using assms
proof (induct p rule:poly-root-induct-alt)
    case 0
    then show ?case by simp
next
    case (no-proots p)
    then have [simp]:p\not=0 q\not=0 \bigwedgea. order a p=0
    using order-0I by auto
    have order a }(p*q)=0\longleftrightarrow\mathrm{ order a q=0
        order }a(p*q)=1\longleftrightarrow\mathrm{ order a q=1
        for a
    subgoal by (subst order-mult) auto
    subgoal by (subst order-mult) auto
    done
    then show ?case using <rsquarefree ( }p*q)
    unfolding rsquarefree-def by simp
next
    case (root a p)
    define pq aa where pq=p*q and aa=[:-a,1:]
    have [simp]:pq\not=0 aa\not=0 order a aa=1
        subgoal using pq-def root.prems by auto
        subgoal by (simp add: aa-def)
        subgoal by (metis aa-def order-power-n-n power-one-right)
        done
    have rsquarefree (aa*pq)
    unfolding aa-def pq-def using root(2) by (simp add:algebra-simps)
    then have rsquarefree pq
    unfolding rsquarefree-def by (auto simp add:order-mult)
    from root(1)[OF this[unfolded pq-def]] show ?case .
qed
lemma rsquarefree-smult-iff:
    assumes }s\not=
    shows rsquarefree (smult s p) \longleftrightarrow rsquarefree p
    unfolding rsquarefree-def using assms by (auto simp add:order-smult)
lemma card-proots-within-rsquarefree:
    assumes rsquarefree p
    shows proots-count p s=card (proots-within p s) using assms
proof (induct rule:poly-root-induct[of - \lambdax. x\ins])
    case 0
    then have False by simp
    then show ?case by simp
next
    case (no-roots p)
    then show ?case
        by (metis all-not-in-conv card.empty proots-count-def proots-within-iff sum.empty)
next
    case (root a p)
```

```
    have proots-count \(([: a,-1:] * p) s=1+\) proots-count \(p s\)
    apply (subst proots-count-times)
    subgoal using root.prems rsquarefree-def by blast
    subgoal by (metis (no-types, opaque-lifting) add.inverse-inverse add.inverse-neutral
                                    minus-pCons proots-count-pCons-1-iff proots-count-uminus
root.hyps(1))
    done
    also have \(\ldots=1+\operatorname{card}(\) proots-within \(p s)\)
    proof -
    have rsquarefree \(p\) using 〈rsquarefree ([:a, - 1:] * p)〉
        by (elim rsquarefree-times)
    from \(\operatorname{root}(2)[O F\) this] show?thesis by simp
    qed
    also have \(\ldots=\) card \((\) proots-within \(([: a,-1:] * p) s)\) unfolding proots-within-times
    proof (subst card-Un-disjoint)
    have [simp]: \(p \neq 0\) using root.prems by auto
    show finite (proots-within [:a, - 1:] s) finite (proots-within p s)
        by auto
    show \(1+\operatorname{card}(\) proots-within \(p s)=\operatorname{card}(\) proots-within \([: a,-1:] s)\)
                        + card (proots-within p s)
        using \(\langle a \in s\rangle\)
        apply (subst proots-within-pCons-1-iff)
        by \(\operatorname{simp}\)
    have poly \(p\) a⿻二
    proof (rule ccontr)
        assume \(\neg\) poly p \(a \neq 0\)
        then have order a \(p>0\) by (simp add: order-root)
        moreover have order \(a[: a,-1:]=1\)
            by (metis (no-types, opaque-lifting) add.inverse-inverse add.inverse-neutral
minus-pCons
            order-power-n-n order-uminus power-one-right)
        ultimately have order \(a([: a,-1:] * p)>1\)
            apply (subst order-mult)
            subgoal using root.prems by auto
            subgoal by auto
            done
            then show False using 〈rsquarefree ([:a, - 1:] *p)〉
                unfolding rsquarefree-def using gr-implies-not0 less-not-reft2 by blast
    qed
    then show proots-within [:a, - 1:] \(s \cap\) proots-within \(p s=\{ \}\)
            using proots-within-pCons-1-iff(2) by auto
    qed
    finally show ?case .
qed
lemma rsquarefree-gcd-pderiv:
    fixes \(p:\) :'a::\{factorial-ring-gcd,semiring-gcd-mult-normalize,semiring-char-0\} poly
```

```
    assumes p\not=0
    shows rsquarefree (p div (gcd p (pderiv p)))
proof (cases pderiv p=0)
    case True
    have poly (unit-factor p) x\not=0 for x
        using unit-factor-is-unit[OF<p\not=0`]
        by (meson assms dvd-trans order-decomp poly-eq-0-iff-dvd unit-factor-dvd)
    then have order x (unit-factor p)=0 for x
    using order-OI by blast
    then show ?thesis using True }\langlep\not=0\rangle\mathrm{ unfolding rsquarefree-def by simp
next
    case False
    define q where q=p div (gcd p(pderiv p))
    have q\not=0 unfolding q-def by (simp add: assms dvd-div-eq-0-iff)
    have order-pq:order x p = order x q + min (order x p)(order x (pderiv p))
    for }
    proof -
    have *:p=q*gcd p(pderiv p)
        unfolding q-def by simp
    show ?thesis
        apply (subst *)
        using }\langleq\not=0\rangle\langlep\not=0\rangle\langlepderiv p\not=0\rangle by (simp add:order-mult order-gcd
    qed
    have order x q=0\vee order x q=1 for x
    proof (cases poly p x=0)
    case True
    from order-pderiv[OF \langlep\not=0> this]
    have order x p=order x (pderiv p)+1 by simp
    then show ?thesis using order-pq[of x] by auto
    next
        case False
        then have order x p = 0 by (simp add: order-0I)
        then have order x q=0 using order-pq[of x] by simp
        then show?thesis by simp
    qed
    then show ?thesis using <q\not=0\rangle\mathrm{ unfolding rsquarefree-def q-def}
    by auto
qed
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 \longleftrightarrow poly p x=0
proof (cases pderiv p=0)
    case True
    then obtain }a\mathrm{ where }p=[:a:] using pderiv-iszero by auto
    then show ?thesis by (auto simp add: unit-factor-poly-def)
next
    case False
```

then have $p \neq 0$ using pderiv- 0 by blast
define $q$ where $q=p$ div (gcd $p(p d e r i v p))$
have $q \neq 0$ unfolding $q$-def by (simp add: $\langle p \neq 0\rangle d v d$-div-eq- 0 -iff)

```
have order-pq:order x p = order x q + min (order x p) (order x (pderiv p)) for }
proof -
    have *:p=q* gcd p (pderiv p)
        unfolding q-def by simp
    show ?thesis
        apply (subst *)
        using }\langleq\not=0\rangle\langlep\not=0\rangle\langlepderiv p\not=0\rangle by (simp add:order-mult order-gcd
qed
    have order x q=0 \longleftrightarrow order x p=0
    proof (cases poly p x=0)
    case True
    from order-pderiv[OF <p\not=0> this]
    have order x p=order x (pderiv p)+1 by simp
    then show ?thesis using order-pq[of x] by auto
next
    case False
    then have order x p = 0 by (simp add: order-0I)
    then have order x q =0 using order-pq[of x] by simp
    then show ?thesis using <order x p=0> by simp
qed
then show ?thesis
    apply (fold q-def)
    unfolding order-root using }\langlep\not=0\rangle\langleq\not=0\rangle\mathrm{ by auto
qed
```


### 1.5 Composition of a polynomial and a circular path

```
lemma poly-circlepath-tan-eq:
    fixes \(z 0::\) complex and \(r::\) real and \(p::\) complex poly
    defines \(q 1 \equiv\) fcompose \(p[:(z 0+r) * \mathrm{i}, z 0-r:][: \mathrm{i}, 1:]\) and \(q 2 \equiv[: \mathrm{i}, 1:]\) へ degree \(p\)
    assumes \(0 \leq t t \leq 1 t \neq 1 / 2\)
    shows poly \(p(\) circlepath z0 \(r t)=\) poly q1 \((\tan (p i * t)) / \operatorname{poly} q 2(\tan (p i * t))\)
        (is ? \(L=? R\) )
proof -
    have \(? L=\) poly \(p(z 0+r * \exp (2 *\) of-real pi*i \(* t))\)
        unfolding circlepath by simp
    also have \(\ldots=\) ? \(R\)
    proof -
        define \(f\) where \(f=(\) poly \(p \circ(\lambda x::\) real. \(z 0+r * \exp (\mathrm{i} * x)))\)
        have \(f\)-eq:f \(t=((\lambda x:\) :real. poly \(q 1 x /\) poly q2 \(x) o(\lambda x . \tan (x / 2))) t\)
            when \(\cos (t / 2) \neq 0\) for \(t\)
        proof -
            have \(f t=\) poly \(p(z 0+r *(\cos t+\mathrm{i} * \sin t))\)
                unfolding \(f\)-def exp-Euler by (auto simp add:cos-of-real sin-of-real)
```

also have $\ldots=$ poly $p((\lambda x .((z 0-r) * x+(z 0+r) * \mathrm{i}) /(\mathrm{i}+x))(\tan (t / \mathcal{Z})))$

## proof -

define $t t$ where $t t=$ complex-of-real $(\tan (t / 2))$
define $r r$ where $r r=$ complex-of-real $r$
have $\cos t=(1-t t * t t) /(1+t t * t t)$
$\sin t=2 * t t /(1+t t * t t)$
unfolding sin-tan-half[of t/2,simplified] cos-tan-half[of t/2,OF that, simplified] tt-def
by (auto simp add:power2-eq-square)
moreover have $1+t t * t t \neq 0$ unfolding $t t$-def
apply (fold of-real-mult)
by (metis (no-types, opaque-lifting) mult-numeral-1 numeral-One of-real-add of-real-eq-0-iff
of-real-numeral sum-squares-eq-zero-iff zero-neq-one)
ultimately have $z 0+r *((\cos t)+\mathrm{i} *(\sin t))$

$$
=(z 0 *(1+t t * t t)+r r *(1-t t * t t)+\mathrm{i} * r r * 2 * t t) /(1+t t * t t)
$$

apply (fold rr-def,simp add:add-divide-distrib)
by (simp add:algebra-simps)
also have $\ldots=((z 0-r r) * t t+z 0 * \mathrm{i}+r r * \mathrm{i}) /(t t+\mathrm{i})$
proof -
have $t t+\mathrm{i} \neq 0$
using $\langle 1+t t * t t \neq 0\rangle$
by (metis $i$-squared neg-eq-iff-add-eq-0 square-eq-iff)
then show ?thesis
using $\langle 1+t t * t t \neq 0\rangle$ by (auto simp add:divide-simps algebra-simps)
qed
finally have $z 0+r *((\cos t)+\mathrm{i} *(\sin t))=((z 0-r r) * t t+z 0 * \mathrm{i}+r r * \mathrm{i}) /$ $(t t+i)$.
then show ?thesis unfolding tt-def rr-def
by (auto simp add:algebra-simps power2-eq-square)
qed
also have $\ldots=($ poly $p o((\lambda x .((z 0-r) * x+(z 0+r) * \mathrm{i}) /(\mathrm{i}+x)) o(\lambda x$. tan (x/2)) )) $t$
unfolding comp-def by (auto simp:tan-of-real)
also have $\ldots=((\lambda x::$ real. poly $q 1 x /$ poly $q 2 x) o(\lambda x \cdot \tan (x / \mathcal{Z}))) t$
unfolding q2-def q1-def
apply (subst fcompose-poly[symmetric])
subgoal for $x$
apply simp
by (metis Re-complex-of-real add-cancel-right-left complex-i-not-zero imag-inary-unit.sel(1) plus-complex.sel(1) rcis-zero-arg rcis-zero-mod)
subgoal by (auto simp:tan-of-real algebra-simps)
done
finally show ?thesis .
qed
have $\cos (p i * t) \neq 0$ unfolding cos-zero-iff-int2 proof
assume $\exists i . p i * t=$ real-of-int $i * p i+p i / 2$

```
        then obtain i where pi*t= real-of-int i*pi+pi / 2 by auto
        then have pi*t=pi*(real-of-int i+1/2) by (simp add:algebra-simps)
        then have t=real-of-int i+1/2 by auto
        then show False using <0\leqt\rangle\langlet\leq1\rangle\langlet\not=1/2\rangle by auto
    qed
    from f-eq[of 2*pi*t,simplified,OF this]
    show ?thesis
        unfolding f-def comp-def by (auto simp add:algebra-simps)
    qed
    finally show ?thesis.
qed
```


### 1.6 Combining two real polynomials into a complex one

definition cpoly-of:: real poly $\Rightarrow$ real poly $\Rightarrow$ complex poly where
cpoly-of $p R$ pI $=$ map-poly of-real $p R+$ smult $\mathrm{i}($ map-poly of-real $p I)$
lemma cpoly-of-eq-0-iff[iff]:
cpoly-of $p R p I=0 \longleftrightarrow p R=0 \wedge p I=0$
proof -
have $p R=0 \wedge p I=0$ when cpoly-of $p R p I=0$
proof -
have complex-of-real (coeff pR $n$ ) $+\mathrm{i} *$ complex-of-real $($ coeff $p I n)=0$ for $n$
using that unfolding poly-eq-iff cpoly-of-def by (auto simp:coeff-map-poly)
then have coeff $p R n=0 \wedge$ coeff $p I n=0$ for $n$
by (metis Complex-eq Im-complex-of-real Re-complex-of-real complex.sel(1)
complex.sel(2) of-real-0)
then show ?thesis unfolding poly-eq-iff by auto
qed
then show? ?thesis by (auto simp:cpoly-of-def)
qed
lemma cpoly-of-decompose:
$p=$ cpoly-of (map-poly Re p) (map-poly Im $p$ )
unfolding cpoly-of-def
apply (induct $p$ )
by (auto simp add:map-poly-pCons map-poly-map-poly complex-eq)
lemma cpoly-of-dist-right:
cpoly-of $(p R * g)(p I * g)=$ cpoly-of $p R p I *($ map-poly of-real $g)$
unfolding cpoly-of-def by (simp add: distrib-right)
lemma poly-cpoly-of-real:
poly $($ cpoly-of $p R$ pI) $($ of-real $x)=$ Complex $($ poly $p R x)($ poly $p I x)$
unfolding cpoly-of-def by (simp add: Complex-eq)
lemma poly-cpoly-of-real-iff:
shows poly $($ cpoly-of $p R p I)(o f$-real $t)=0 \longleftrightarrow$ poly $p R t=0 \wedge$ poly $p I t=0$
unfolding poly-cpoly-of-real using Complex-eq-0 by blast
lemma order-cpoly-gcd-eq:
assumes $p R \neq 0 \vee p I \neq 0$
shows order $t($ cpoly-of $p R p I)=$ order $t(g c d p R p I)$
proof -
define $g$ where $g=g c d p R p I$
have $[$ simp $]: g \neq 0$ unfolding $g$-def using assms by auto
obtain pr pi where pri: $p R=p r * g p I=p i * g$ coprime pr pi unfolding $g$-def using assms(1) gcd-coprime-exists $\langle g \neq 0\rangle g$-def by blast
then have $p r \neq 0 \vee p i \neq 0$ using assms mult-zero-left by blast
have order $t($ cpoly-of $p R p I)=$ order $t($ cpoly-of pr pi*(map-poly of-real g) $)$
unfolding pri cpoly-of-dist-right by simp
also have $\ldots=$ order $t$ (cpoly-of pr pi) + order $t g$
apply (subst order-mult)
using $\langle p r \neq 0 \vee p i \neq 0\rangle$ by (auto simp:map-poly-order-of-real)
also have $\ldots=$ order $t g$
proof -
have poly (cpoly-of pr pi) $t \neq 0$ unfolding poly-cpoly-of-real-iff using 〈coprime pr pi〉 coprime-poly- 0 by blast
then have order $t$ (cpoly-of pr pi) $=0$ by (simp add: order-0I)
then show ?thesis by auto
qed
finally show ?thesis unfolding $g$-def.
qed
lemma cpoly-of-times:
shows cpoly-of pR pI*cpoly-of $q R q I=\operatorname{cpoly}$-of $(p R * q R-p I * q I)(p I * q R+p R * q I)$
proof -
define $P R P I$ where $P R=$ map-poly complex-of-real $p R$
and $P I=$ map-poly complex-of-real pI
define $Q R \quad Q I$ where $Q R=$ map-poly complex-of-real $q R$ and $Q I=$ map-poly complex-of-real $q I$
show ?thesis
unfolding cpoly-of-def
by (simp add:algebra-simps of-real-poly-hom.hom-minus smult-add-right fip: PR-def PI-def QR-def QI-def)
qed
lemma map-poly-Re-cpoly[simp]:
map-poly Re (cpoly-of pR pI) $=p R$
unfolding cpoly-of-def smult-map-poly
apply (simp add:map-poly-map-poly Re-poly-hom.hom-add comp-def)
by (metis coeff-map-poly leading-coeff-0-iff)
lemma map-poly-Im-cpoly[simp]:
map-poly Im (cpoly-of pR pI) $=p I$
unfolding cpoly-of-def smult-map-poly
apply (simp add:map-poly-map-poly Im-poly-hom.hom-add comp-def)
by (metis coeff-map-poly leading-coeff-0-iff)
end

## 2 An alternative Sturm sequences

theory Extended-Sturm imports<br>Sturm-Tarski.Sturm-Tarski<br>Winding-Number-Eval.Cauchy-Index-Theorem<br>CC-Polynomials-Extra<br>\section*{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 $\llbracket ? a<? b ;$ poly ?p ?a $\neq 0$; poly ?p $? b \neq 0 \rrbracket \Longrightarrow$ 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 realalgebraic proof via Sturm chains" by Michael Eisermann.
hide-const Permutations.sign

### 2.1 Misc

lemma path-of-real[simp]:path (of-real :: real $\Rightarrow{ }^{\prime}$ 'a::real-normed-algebra-1)
unfolding path-def by (rule continuous-on-of-real-id)
lemma pathfinish-of-real[simp]:pathfinish of-real = 1
unfolding pathfinish-def by simp
lemma pathstart-of-real[simp]:pathstart of-real $=0$
unfolding pathstart-def by simp
lemma is-unit-pCons-ex-iff:
fixes $p::$ 'a::field poly
shows is-unit $p \longleftrightarrow(\exists a . a \neq 0 \wedge p=[: a:])$
using is-unit-poly-iff is-unit-triv
by (metis is-unit-pCons-iff)
lemma eventually-poly-nz-at-within:
fixes $x::$ ' $a::\{$ idom,euclidean-space $\}$
assumes $p \neq 0$
shows eventually $(\lambda x$. poly $p x \neq 0)($ at $x$ within $S)$
proof -
have eventually ( $\lambda x$. poly $p x \neq 0$ ) (at $x$ within $S$ )
$=\left(\forall_{F} x\right.$ in (at $x$ within $\left.S\right) . \forall y \in$ proots p. $\left.x \neq y\right)$
apply (rule eventually-subst,rule eventuallyI)
by (auto simp add:proots-def)
also have $\ldots=\left(\forall y \in\right.$ proots $p . \forall_{F} x$ in (at $x$ within $\left.\left.S\right) . x \neq y\right)$

```
    apply (subst eventually-ball-finite-distrib)
    using <p\not=0\rangle by auto
    also have ...
    unfolding eventually-at
    by (metis gt-ex not-less-iff-gr-or-eq zero-less-dist-iff)
    finally show ?thesis.
qed
lemma sgn-power:
    fixes x::'a::linordered-idom
    shows sgn (x`n)}=(\mathrm{ if }n=0\mathrm{ then 1 else if even n then |sgn x| else sgn x)
    apply (induct n)
    by (auto simp add:sgn-mult)
lemma poly-divide-filterlim-at-top:
    fixes p q::real poly
    defines ll\equiv( if degree q<degree p then
                at 0
            else if degree q=degree p then
                nhds (lead-coeff q / lead-coeff p)
            else if sgn-pos-inf q* sgn-pos-inf p>0 then
                at-top
            else
                at-bot)
    assumes p\not=0 q\not=0
    shows filterlim ( }\lambdax\mathrm{ . poly q x / poly p x) ll at-top
proof -
    define pp where pp=(\lambdax. poly px/ x`(degree p))
    define qq where qq=( }\lambda\mathrm{ x. poly qx / x`(degree q))
    define }dd\mathrm{ where }dd=(\lambdax::\mathrm{ real. if degree }p>\mathrm{ degree q then 1/x^(degree }p-\mathrm{ degree
q) else
                        x`(degree q - degree p))
    have divide-cong:}\mp@subsup{\forall}{F}{}x\mathrm{ in at-top. poly q x / poly p x = qq x / pp x * dd x
    proof (rule eventually-at-top-linorderI[of 1])
    fix }x\mathrm{ assume ( }x::\mathrm{ real ) }\geq
    then have }x\not=0\mathrm{ by auto
    then show poly qx / poly px=qqx / ppx*dd x
        unfolding qq-def pp-def dd-def using assms
        by (auto simp add:field-simps power-diff)
    qed
    have qqpp-tendsto:((\lambdax.qq x / ppx)\longrightarrow lead-coeff q / lead-coeff p) at-top
    proof -
    have (qq\longrightarrow lead-coeff q) at-top
        unfolding qq-def using poly-divide-tendsto-aux[of q]
            by (auto elim!:filterlim-mono simp:at-top-le-at-infinity)
    moreover have ( pp\longrightarrow lead-coeff p) at-top
            unfolding pp-def using poly-divide-tendsto-aux[of p]
            by (auto elim!:filterlim-mono simp:at-top-le-at-infinity)
    ultimately show ?thesis using < }p\not=0\rangle\mathrm{ by (auto intro!:tendsto-eq-intros)
```

qed
have ?thesis when degree $q<$ degree $p$
proof -
have filterlim ( $\lambda x$. poly $q x /$ poly $p x$ ) (at 0) at-top
proof (rule filterlim-atI)
show $((\lambda x$. poly $q x /$ poly $p x) \longrightarrow 0)$ at-top
using poly-divide-tendsto-0-at-infinity[OF that]
by (auto elim:filterlim-mono simp:at-top-le-at-infinity)
have $\forall_{F} x$ in at-top. poly $q x \neq 0 \forall_{F} x$ in at-top. poly p $x \neq 0$
using poly-eventually-not-zero[OF $\langle q \neq 0\rangle$ ] poly-eventually-not-zero[OF $\langle p \neq 0\rangle$ ] filter-le $D[O F$ at-top-le-at-infinity]
by auto
then show $\forall_{F} x$ in at-top. poly $q x /$ poly $p x \neq 0$
apply eventually-elim
by auto
qed
then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree $q=$ degree $p$
proof -
have $((\lambda x$. poly $q x /$ poly $p x) \longrightarrow$ lead-coeff $q /$ lead-coeff $p)$ at-top
using divide-cong qqpp-tendsto that unfolding $d d$-def
by (auto dest:tendsto-cong)
then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree $q>$ degree $p$ sgn-pos-inf $q * \operatorname{sgn}$-pos-inf $p>$ 0
proof -
have filterlim $(\lambda x .(q q x / p p x) * d d x)$ at-top at-top
proof (subst filterlim-tendsto-pos-mult-at-top-iff[OF qqpp-tendsto])
show $0<l e a d-c o e f f ~ q / l e a d-c o e f f ~ p ~ u s i n g ~ t h a t(2) ~ u n f o l d i n g ~ s g n-p o s-i n f-d e f ~$ by (simp add: zero-less-divide-iff zero-less-mult-iff)
show filterlim dd at-top at-top
unfolding $d d$-def using that(1)
by (auto intro!:filterlim-pow-at-top simp:filterlim-ident)
qed
then have LIM $x$ at-top. poly $q x /$ poly $p x:>$ at-top
using filterlim-cong[OF - divide-cong] by blast
then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree $q>$ degree $p \neg$ sgn-pos-inf $q *$ sgn-pos-inf $p>0$
proof -
have filterlim $(\lambda x .(q q x / p p x) * d d x)$ at-bot at-top
proof (subst filterlim-tendsto-neg-mult-at-bot-iff[OF qqpp-tendsto])
show lead-coeff $q /$ lead-coeff $p<0$
using that (2) $\langle p \neq 0\rangle\langle q \neq 0\rangle$ unfolding sgn-pos-inf-def
by (metis divide-eq-0-iff divide-sgn leading-coeff-0-iff

```
                    linorder-neqE-linordered-idom sgn-divide sgn-greater)
        show filterlim dd at-top at-top
            unfolding dd-def using that(1)
            by (auto intro!:filterlim-pow-at-top simp:filterlim-ident)
    qed
    then have LIM x at-top.poly q x / poly p x :> at-bot
        using filterlim-cong[OF - divide-cong] by blast
    then show ?thesis unfolding ll-def using that by auto
    qed
    ultimately show ?thesis by linarith
qed
lemma poly-divide-filterlim-at-bot:
    fixes p q::real poly
    defines ll\equiv( if degree q<degree p then
                at 0
    else if degree q=degree p then
        nhds (lead-coeff q / lead-coeff p)
        else if sgn-neg-inf q*sgn-neg-inf p>0 then
        at-top
        else
        at-bot)
    assumes p\not=0 q\not=0
    shows filterlim ( }\lambdax\mathrm{ . poly q x / poly px) ll at-bot
proof -
    define pp where pp=(\lambdax. poly px/x`(degree p))
    define qq where qq=(\lambdax. poly qx / x`(degree q))
```



```
q) else
                x(degree q - degree p))
    have divide-cong:}\mp@subsup{\forall}{F}{}x\mathrm{ in at-bot. poly q x / poly px=qqx / ppx*ddx
    proof (rule eventually-at-bot-linorderI[of -1])
    fix }x\mathrm{ assume (x::real) }\leq-
    then have }x\not=0\mathrm{ by auto
    then show poly qx / poly px=qqx / ppx*dd x
        unfolding qq-def pp-def dd-def using assms
        by (auto simp add:field-simps power-diff)
    qed
    have qqpp-tendsto:((\lambdax.qq x / pp x)\longrightarrow lead-coeff q / lead-coeff p)at-bot
    proof -
    have (qq\longrightarrow lead-coeff q) at-bot
        unfolding qq-def using poly-divide-tendsto-aux[of q]
            by (auto elim!:filterlim-mono simp:at-bot-le-at-infinity)
    moreover have ( pp\longrightarrow lead-coeff p) at-bot
            unfolding pp-def using poly-divide-tendsto-aux[of p]
            by (auto elim!:filterlim-mono simp:at-bot-le-at-infinity)
    ultimately show ?thesis using <p\not=0\rangle by (auto intro!:tendsto-eq-intros)
qed
```

```
have ?thesis when degree q<degree p
proof -
    have filterlim ( }\lambdax\mathrm{ . poly qx / poly p x) (at 0) at-bot
    proof (rule filterlim-atI)
        show (( }\lambdax\mathrm{ . poly q x / poly p x) }\longrightarrow0) at-bo
            using poly-divide-tendsto-0-at-infinity[OF that]
            by (auto elim:filterlim-mono simp:at-bot-le-at-infinity)
    have }\mp@subsup{\forall}{F}{}x\mathrm{ in at-bot. poly q }x\not=0\mp@subsup{\forall}{F}{}x\mathrm{ in at-bot. poly p }x\not=
    using poly-eventually-not-zero[OF <q\not=0〉] poly-eventually-not-zero[OF <p\not=0\rangle]
                filter-leD[OF at-bot-le-at-infinity]
            by auto
    then show }\mp@subsup{\forall}{F}{}x\mathrm{ in at-bot. poly q x / poly p x}\not=
            by eventually-elim auto
    qed
    then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree q= degree p
proof -
    have ((\lambdax. poly q x / poly p x) \longrightarrow lead-coeff q / lead-coeff p) at-bot
        using divide-cong qqpp-tendsto that unfolding dd-def
        by (auto dest:tendsto-cong)
    then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree q>degree p sgn-neg-inf q * sgn-neg-inf p>
O
proof -
    define cc where cc=lead-coeff q / lead-coeff p
    have (cc>0 ^ even (degree q- degree p))\vee(cc<0^ odd (degree q - degree
p))
    proof -
        have even (degree q - degree p) \longleftrightarrow
            (even (degree q)}\wedge\mathrm{ even (degree p)) } (\mathrm{ odd (degree q) ^ odd (degree p))
        using <degree q>degree p> by auto
        then show ?thesis
        using that \langlep\not=0\rangle\langleq\not=0\rangle unfolding sgn-neg-inf-def cc-def zero-less-mult-iff
            divide-less-0-iff zero-less-divide-iff
            apply (simp add:if-split[of (<) 0] if-split[of (>) 0])
            by argo
    qed
    moreover have filterlim ( }\lambdax.(qqx/ppx)*ddx) at-top at-bo
        when cc>0 even (degree q- degree p)
    proof (subst filterlim-tendsto-pos-mult-at-top-iff[OF qqpp-tendsto])
        show 0< lead-coeff q / lead-coeff p using {cc>0\rangle unfolding cc-def by auto
        show filterlim dd at-top at-bot
            unfolding dd-def using <degree q> degree p> that(2)
            by (auto intro!:filterlim-pow-at-bot-even simp:filterlim-ident)
    qed
    moreover have filterlim ( }\lambdax.(qqx/ppx)*ddx) at-top at-bo
        when cc<0 odd (degree q- degree p)
```

proof (subst filterlim-tendsto-neg-mult-at-top-iff[OF qqpp-tendsto])
show $0>$ lead-coeff $q /$ lead-coeff $p$ using $\langle c c<0\rangle$ unfolding $c c$-def by auto
show filterlim dd at-bot at-bot
unfolding dd-def using <degree $q>$ degree $p\rangle$ that(2)
by (auto intro!:filterlim-pow-at-bot-odd simp:filterlim-ident)
qed
ultimately have filterlim $(\lambda x .(q q x / p p x) * d d x)$ at-top at-bot by blast
then have LIM x at-bot. poly $q$ x / poly $p x$ :> at-top using filterlim-cong[OF - - divide-cong] by blast
then show ?thesis unfolding ll-def using that by auto

## qed

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

```
    qed
    ultimately show ?thesis by linarith
qed
lemma sgnx-poly-times:
    assumes F=at-bot \veeF=at-top \vee F=at-right x \vee F=at-left x
    shows sgnx (poly (p*q)) F=\operatorname{sgnx}(\mathrm{ poly p)F* sgnx (poly q) F}
    (is ?PQ =?P*?Q)
proof -
    have (poly p has-sgnx ?P) F
            (poly q has-sgnx ?Q) F
        by (rule sgnx-able-sgnx;use assms sgnx-able-poly in blast)+
    from has-sgnx-times[OF this]
    have (poly ( p*q) has-sgnx ?P*?Q) F
        by (simp flip:poly-mult)
    moreover have (poly ( p*q) has-sgnx ?PQ) F
        by (rule sgnx-able-sgnx;use assms sgnx-able-poly in blast)+
    ultimately show ?thesis
        using has-sgnx-unique assms by auto
qed
lemma sgnx-poly-plus:
    assumes poly p x=0 poly q x\not=0 and F:F=at-right x\veeF=at-left x
    shows sgnx (poly (p+q))F=\operatorname{sgnx}(poly q) F (is ?L=?R)
proof -
    have ((poly (p+q)) has-sgnx ?R) F
    proof -
        have sgnx (poly q) F= sgn (poly q x)
            using F assms(2) sgnx-poly-nz(1) sgnx-poly-nz(2) by presburger
        moreover have ((\lambdax. poly (p+q) x) has-sgnx sgn (poly q x)) F
        proof (rule tendsto-nonzero-has-sgnx)
            have ((poly p)\longrightarrow0) F
            by (metis F assms(1) poly-tendsto(2) poly-tendsto(3))
            then have (( }\lambdax.\mathrm{ poly px+ poly q x) < poly q x) F
                apply (elim tendsto-add[where a=0,simplified])
                using F poly-tendsto(2) poly-tendsto(3) by blast
            then show }((\lambdax.poly (p+q)x)\longrightarrow poly q x)
                by auto
        qed fact
        ultimately show ?thesis by metis
    qed
    from has-sgnx-imp-sgnx[OF this] F
    show ?thesis by auto
qed
```

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

```
lemma cindex-poly-combine:
    assumes \(a<b b<c\)
    shows cindex-poly abqp+jump-poly \(q p b+\) cindex-poly \(b c q p=\) cindex-poly
a \(c q p\)
proof (cases \(p \neq 0\) )
    case True
    define \(A B C D\) where \(A=\{x\). poly \(p x=0 \wedge a<x \wedge x<c\}\)
                    and \(B=\{x\). poly \(p x=0 \wedge a<x \wedge x<b\}\)
                    and \(C=(\) if poly \(p b=0\) then \(\{b\}\) else \(\{ \})\)
                    and \(D=\{x\). poly \(p x=0 \wedge b<x \wedge x<c\}\)
    let ?sum=sum ( \(\lambda x\).jump-poly q \(p x)\)
    have cindex-poly a c \(q\) p=?sum \(A\)
    unfolding cindex-poly-def \(A\)-def by simp
    also have \(\ldots=\) ?sum \((B \cup C \cup D)\)
    apply (rule arg-cong2[where \(f=\) sum])
    unfolding \(A\)-def \(B\)-def \(C\)-def \(D\)-def using less-linear assms by auto
    also have \(\ldots=\) ? sum \(B+\) ? sum \(C+\) ? sum \(D\)
    proof -
    have finite \(B\) finite \(C\) finite \(D\)
        unfolding \(B\)-def \(C\)-def \(D\)-def using True
        by (auto simp add: poly-roots-finite)
    moreover have \(B \cap C=\{ \} C \cap D=\{ \} B \cap D=\{ \}\)
        unfolding \(B\)-def \(C\)-def \(D\)-def using assms by auto
    ultimately show ?thesis
        by (subst sum.union-disjoint;auto)+
    qed
    also have \(\ldots=\) cindex-poly a b q \(p+\) jump-poly \(q\) p \(b+\) cindex-poly b c q \(p\)
    proof -
    have ?sum \(C=j u m p-p o l y ~ q ~ p b\)
        unfolding \(C\)-def using jump-poly-not-root by auto
    then show ?thesis unfolding cindex-poly-def B-def D-def
        by auto
    qed
    finally show?thesis by simp
qed auto
lemma coprime-linear-comp: - TODO: need to be generalised
    fixes \(b c:\) :real
    defines \(r 0 \equiv[: b, c:]\)
    assumes coprime p \(q c \neq 0\)
    shows coprime ( \(p \circ_{p} r 0\) ) ( \(\left.q \circ_{p} r 0\right)\)
proof -
```

define $g$ where $g=g c d\left(p \circ_{p} r 0\right)\left(q \circ_{p} r 0\right)$
define $p^{\prime}$ where $p^{\prime}=\left(\begin{array}{l}\left.p \circ_{p} r 0\right) \text { div } g\end{array}\right.$
define $q^{\prime}$ where $q^{\prime}=\left(q \circ_{p} r 0\right)$ div $g$
define $r 1$ where $r 1=[:-b / c, 1 / c:]$
have $r-i d$ :

$$
\begin{aligned}
& r 0 \\
& r
\end{aligned} o_{p} r 1=[: 0,1:] ~=[0,1:]
$$

unfolding r0-def r1-def using $\langle c \neq 0\rangle$
by (simp add: pcompose-pCons)+
have $p=\left(g \circ_{p} r 1\right) *\left(p^{\prime} \circ_{p} r 1\right)$
proof -
from $r$-id have $p=p \circ_{p}\left(r 0 \circ_{p} r 1\right)$
by (metis pcompose-idR)
also have $\ldots=\left(g * p^{\prime}\right) \circ_{p} r 1$
unfolding $g$-def $p^{\prime}$-def by (auto simp:pcompose-assoc)
also have $\ldots=\left(g \circ_{p} r 1\right) *\left(p^{\prime} \circ_{p} r 1\right)$
unfolding pcompose-mult by simp
finally show ?thesis .
qed
moreover have $q=\left(g \circ_{p} r 1\right) *\left(q^{\prime} \circ_{p} r 1\right)$
proof -
from $r$ - $i d$ have $q=q \circ_{p}\left(r 0 \circ_{p} r 1\right)$ by (metis pcompose-idR)
also have $\ldots=\left(g * q^{\prime}\right) \circ_{p} r 1$ unfolding $g$-def $q^{\prime}$-def by (auto simp:pcompose-assoc)
also have $\ldots=\left(g \circ_{p} r 1\right) *\left(q^{\prime} \circ_{p} r 1\right)$
unfolding pcompose-mult by simp
finally show ?thesis .
qed
ultimately have $\left(g \circ_{p} r 1\right) d v d \operatorname{gcd} p q$ by $\operatorname{simp}$
then have $g \circ_{p} r 1 d v d 1$
using <coprime $p q\rangle$ by auto
from pcompose-hom.hom-dvd-1[OF this]
have is-unit ( $\left.g \circ_{p}\left(r 1 \circ_{p} r 0\right)\right)$
by (auto simp:pcompose-assoc)
then have is-unit $g$
using r-id pcompose-idR by auto
then show coprime ( $p \circ_{p} r 0$ ) ( $q \circ_{p} r 0$ ) unfolding $g$-def
using is-unit-gcd by blast
qed
lemma finite-ReZ-segments-poly-rectpath:
finite-ReZ-segments (poly $p \circ$ rectpath $a b$ ) $z$
unfolding rectpath-def Let-def path-compose-join
by ((subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join

> pathfinish-compose pathstart-compose poly-pcompose)?)+
lemma valid-path-poly-linepath:
fixes a b::'a::real-normed-field
shows valid-path (poly $p$ o linepath $a b$ )
proof (rule valid-path-compose)
show valid-path (linepath a b) by simp
show $\wedge x . x \in$ path-image (linepath $a b) \Longrightarrow$ poly $p$ field-differentiable at $x$ by $\operatorname{simp}$
show continuous-on (path-image (linepath a b)) (deriv (poly p))
unfolding deriv-pderiv by (auto intro:continuous-intros)
qed
lemma valid-path-poly-rectpath: valid-path (poly porectpath ab)
unfolding rectpath-def Let-def path-compose-join
by (simp add: pathfinish-compose pathstart-compose valid-path-poly-linepath)

### 2.2 Sign difference

definition psign-diff $::$ real poly $\Rightarrow$ real poly $\Rightarrow$ real $\Rightarrow$ int where
psign-diff $p q x=($ if poly $p x=0 \wedge$ poly $q x=0$ then
1 else $\mid \operatorname{sign}($ poly $p x)-\operatorname{sign}($ poly $q x) \mid)$
lemma psign-diff-alt:
assumes coprime $p q$
shows psign-diff $p q x=\mid \operatorname{sign}($ poly $p x)-\operatorname{sign}($ poly $q x) \mid$
unfolding psign-diff-def by (meson assms coprime-poly-0)
lemma psign-diff-0[simp]:
psign-diff $0 q x=1$
psign-diff p $0 x=1$
unfolding psign-diff-def by (auto simp add:sign-def)
lemma psign-diff-poly-commute:
psign-diff $p$ q $x=$ psign-diff $q$ p $x$
unfolding psign-diff-def
by (metis abs-minus-commute gcd.commute)
lemma normalize-real-poly:
normalize $p=$ smult ( $1 /$ lead-coeff $p$ ) ( $p$ ::real poly)
unfolding normalize-poly-def
by (smt (z3) div-unit-factor normalize-eq-0-iff normalize-poly-def normalize-unit-factor smult-eq-0-iff smult-eq-iff smult-normalize-field-eq unit-factor-1)
lemma psign-diff-cancel:
assumes poly r $x \neq 0$
shows psign-diff $(r * p)(r * q) x=p s i g n$-diff $p q x$

```
proof -
    have poly (r*p)x=0 \longleftrightarrow poly p x=0
        by (simp add: assms)
    moreover have poly (r*q) x=0 \longleftrightarrow poly q x=0 by (simp add: assms)
    moreover have |sign (poly (r*p) x) - sign (poly (r*q) x)|
                = |sign (poly p x) - sign (poly q x)
    proof -
    have |sign (poly (r*p)x) - sign (poly (r*q) x)|
            = |sign (poly r x)* (sign (poly p x) - sign (poly q x) )
            by (simp add:algebra-simps sign-times)
        also have ... = |sign (poly r x) |
                        * |sign (poly p x) - sign (poly q x)|
            unfolding abs-mult by simp
    also have ... = |sign (poly p x) - sign (poly q x) |
            by (simp add: Sturm-Tarski.sign-def assms)
        finally show ?thesis.
    qed
    ultimately show ?thesis
        unfolding psign-diff-def by auto
qed
lemma psign-diff-clear: psign-diff p q x = psign-diff 1 ( }p*q)
    unfolding psign-diff-def
    apply (simp add:sign-times )
    by (simp add: sign-def)
lemma psign-diff-linear-comp:
    fixes b c::real
    defines }h\equiv(\lambdap.pcompose p [:b,c:]
    shows psign-diff (h p) (hq)x = psign-diff p q( c*x + b)
    unfolding psign-diff-def h-def poly-pcompose
    by (smt (verit, del-insts) mult.commute mult-eq-0-iff poly-0 poly-pCons)
```


### 2.3 Alternative definition of cross

```
definition cross-alt \(::\) real poly \(\Rightarrow\) real poly \(\Rightarrow\) real \(\Rightarrow\) real \(\Rightarrow\) int where cross-alt p q a b=psign-diff p \(q a-p\) sign-diff \(p q b\)
lemma cross-alt- \(0[\) simp \(]\) :
cross-alt 0 q a \(b=0\)
cross-alt p 0 a \(b=0\)
unfolding cross-alt-def by simp-all
lemma cross-alt-poly-commute:
cross-alt \(p\) q a \(b=\) cross-alt \(q p a b\)
unfolding cross-alt-def using psign-diff-poly-commute by auto
lemma cross-alt-clear:
cross-alt \(p\) q a \(b=\) cross-alt \(1(p * q) a b\)
```

unfolding cross－alt－def using psign－diff－clear by metis
lemma cross－alt－alt：
cross－alt p q a b $=\operatorname{sign}($ poly $(p * q) b)-\operatorname{sign}(\operatorname{poly}(p * q) a)$
apply（subst cross－alt－clear）
unfolding cross－alt－def psign－diff－def by（auto simp add：sign－def）
lemma cross－alt－coprime－ 0 ：
assumes coprime p $q$ p $=0 \vee q=0$
shows cross－alt p q a $b=0$
proof－
have ？thesis when $p=0$
proof－
have is－unit $q$ using that 〈coprime $p q$ 〉
by $\operatorname{simp}$
then obtain $a$ where $a \neq 0 q=[: a:]$ using is－unit－pCons－ex－iff by blast
then show ？thesis using that unfolding cross－alt－def by auto
qed
moreover have ？thesis when $q=0$
proof－
have is－unit $p$ using that 〈coprime $p q$ 〉
by $\operatorname{simp}$
then obtain $a$ where $a \neq 0 p=[: a:]$ using is－unit－pCons－ex－iff by blast
then show ？thesis using that unfolding cross－alt－def by auto
qed
ultimately show ？thesis using $\langle p=0 \vee q=0$ 〉 by auto
qed
lemma cross－alt－cancel：
assumes poly $q a \neq 0$ poly $q b \neq 0$
shows cross－alt $(q * r)(q * s) a b=$ cross－alt rsab
unfolding cross－alt－def using psign－diff－cancel assms by auto
lemma cross－alt－noroot：
assumes $a<b$ and $\forall x . a \leq x \wedge x \leq b \longrightarrow$ poly $(p * q) x \neq 0$
shows cross－alt p q ab＝0
proof－
define $p q$ where $p q=p * q$
have cross－alt p qab＝psign－diff 1 pq $a-p s i g n$－diff 1 pq $b$ apply（subst cross－alt－clear）
unfolding cross－alt－def pq－def by simp
also have $\ldots=\mid 1-\operatorname{sign}($ poly pq a）$|-| 1-\operatorname{sign}($ poly pq b） $\mid$ unfolding psign－diff－def by simp
also have $\ldots=\operatorname{sign}(p o l y p q b)-\operatorname{sign}(p o l y p q a)$
unfolding sign－def by auto
also have $\ldots=0$
proof（rule ccontr）
assume sign（poly pq $b)-\operatorname{sign}($ poly $p q a) \neq 0$
then have poly pq $a *$ poly $p q b<0$

```
by (smt (verit, best) Sturm-Tarski.sign-def assms(1) assms(2)
                    divisors-zero eq-iff-diff-eq-0 pq-def zero-less-mult-pos zero-less-mult-pos2)
    from poly-IVT[OF <a<b> this]
    have \existsx>a. x<b^ poly pq x=0.
    then show False using < }\forallx.a\leqx\wedgex\leqb\longrightarrowpoly (p*q) x\not=0\rangle\langlea<b
        apply (fold pq-def)
        by auto
    qed
    finally show ?thesis .
qed
```

```
lemma cross-alt-linear-comp:
```

lemma cross-alt-linear-comp:
fixes $b c$ ::real
fixes $b c$ ::real
defines $h \equiv(\lambda p$. pcompose $p[: b, c:])$
defines $h \equiv(\lambda p$. pcompose $p[: b, c:])$
shows cross-alt ( $h \mathrm{p})(h q) l b u b=$ cross-alt $p q(c * l b+b)(c * u b+b)$
shows cross-alt ( $h \mathrm{p})(h q) l b u b=$ cross-alt $p q(c * l b+b)(c * u b+b)$
unfolding cross-alt-def $h$-def
unfolding cross-alt-def $h$-def
by (subst (1 2) psign-diff-linear-comp;simp)

```
    by (subst (1 2) psign-diff-linear-comp;simp)
```


### 2.4 Alternative sign variation sequencse

```
fun changes-alt:: ('a ::linordered-idom) list \(\Rightarrow\) int where
    changes-alt [] \(=0 \mid\)
    changes-alt \([-]=0 \mid\)
    changes-alt \((x 1 \# x 2 \# x s)=\) abs \((\operatorname{sign} x 1-\operatorname{sign} x 2)+\) changes-alt \((x 2 \# x s)\)
```

definition changes-alt-poly-at::('a ::linordered-idom) poly list $\Rightarrow$ ' $a \Rightarrow$ int where
changes-alt-poly-at ps $a=$ changes-alt (map $(\lambda p$. poly $p a) p s)$
definition changes-alt-itv-smods:: real $\Rightarrow$ real $\Rightarrow$ real poly $\Rightarrow$ real poly $\Rightarrow$ 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
$a b q(-(p \bmod q))$
proof (cases $p=0 \vee q=0 \vee q d v d p)$
case True
moreover have $p=0 \vee q=0 \Longrightarrow$ ?thesis
using cross-alt-coprime-0
unfolding changes-alt-itv-smods-def changes-alt-poly-at-def by fastforce
moreover have $\llbracket p \neq 0 ; q \neq 0 ; p \bmod q=0 \rrbracket \Longrightarrow$ ?thesis
unfolding changes-alt-itv-smods-def changes-alt-poly-at-def cross-alt-def
psign-diff-alt[OF〈coprime p $q\rangle$ ]
by (simp add:sign-times)
ultimately show ?thesis

```
    by auto (auto elim: dvdE)
next
    case False
    hence }p\not=0q\not=0p\mathrm{ mod q#0 by auto
    then obtain ps where ps:smods p q=p#q#-(p mod q)#ps smods q (-(p mod
q))=q#-(p\operatorname{mod}q)#ps
    by auto
    define changes-diff where changes-diff \equiv\lambdax.changes-alt-poly-at ( }p#q#-(p\operatorname{mod
q)#ps)}
    - changes-alt-poly-at (q#-(p mod q)#ps) x
    have changes-diff a - changes-diff b=cross-alt p q a b
    unfolding changes-diff-def changes-alt-poly-at-def cross-alt-def
        psign-diff-alt[OF <coprime p q}>>
    by simp
    thus ?thesis unfolding changes-alt-itv-smods-def changes-diff-def changes-alt-poly-at-def
ps
    by force
qed
```


## 2.5 jumpF on polynomials

definition jumpF-polyR:: real poly $\Rightarrow$ real poly $\Rightarrow$ real $\Rightarrow$ real where jumpF-polyR q p $a=j u m p F(\lambda x$. poly $q x / \operatorname{poly} p x)($ at-right $a)$
definition jumpF-polyL:: real poly $\Rightarrow$ real poly $\Rightarrow$ real $\Rightarrow$ real where
jumpF-polyL q pa=jumpF ( $\lambda x$. poly $q x /$ poly $p x)($ at-left $a)$
definition jumpF-poly-top:: real poly $\Rightarrow$ real poly $\Rightarrow$ real where
jumpF-poly-top $q$ p $=$ jump $F(\lambda x$. poly $q x /$ poly $p x)$ at-top
definition jumpF-poly-bot:: real poly $\Rightarrow$ real poly $\Rightarrow$ real where
jumpF-poly-bot $q$ p jumpF $(\lambda x$. poly $q x /$ poly $p x)$ at-bot
lemma jump F-poly $R$ - $0[$ simp $]$ : jumpF-poly R 0 pa=0 jumpF-polyR q $0 a=0$
unfolding jumpF-polyR-def by auto
lemma jumpF-polyL-0[simp]: jumpF-polyL 0 pa=0 jumpF-polyL q $0 a=0$
unfolding jumpF-polyL-def by auto
lemma jumpF-polyR-mult-cancel:
assumes $p^{\prime} \neq 0$
shows jumpF-poly $R\left(p^{\prime} * q\right)\left(p^{\prime} * p\right) a=j u m p F-p o l y R q p a$
unfolding jumpF-polyR-def
proof (rule jumpF-cong)
obtain $u b$ where $a<u b \forall z . a<z \wedge z \leq u b \longrightarrow$ poly $p^{\prime} z \neq 0$
using next-non-root-interval $\left[O F\left\langle p^{\prime} \neq 0\right\rangle\right.$,of $\left.a\right]$ by auto
then show $\forall_{F} x$ in at-right a.poly $\left(p^{\prime} * q\right) x / \operatorname{poly}\left(p^{\prime} * p\right) x=$ poly $q x /$ poly
p x

```
    apply (unfold eventually-at-right)
    apply (intro exI[where }x=ub]\mathrm{ )
    by auto
qed simp
lemma jumpF-polyL-mult-cancel:
    assumes p'}=
    shows jumpF-polyL ( }\mp@subsup{p}{}{\prime}*q)(\mp@subsup{p}{}{\prime}*p)a=jumpF-polyL q p a
unfolding jumpF-polyL-def
proof (rule jumpF-cong)
    obtain lb where lb<a\forallz.lb\leqz\wedgez<a\longrightarrowpoly p'z\not=0
    using last-non-root-interval[OF < ' ' =0 >,of a] by auto
    then show }\mp@subsup{\forall}{F}{}x\mathrm{ in at-left a. poly ( }\mp@subsup{p}{}{\prime}*q)x/\operatorname{poly}(\mp@subsup{p}{}{\prime}*p)x=\mathrm{ poly q x / poly
p x
    apply (unfold eventually-at-left)
    apply (intro exI[where }x=lb]\mathrm{ )
    by auto
qed simp
lemma jumpF-poly-noroot:
    assumes poly p a\not=0
    shows jumpF-polyL q p a = 0 jumpF-polyR q p a=0
    subgoal unfolding jumpF-polyL-def using assms
        apply (intro jumpF-not-infinity)
        by (auto intro!:continuous-intros)
    subgoal unfolding jumpF-polyR-def using assms
        apply (intro jumpF-not-infinity)
        by (auto intro!:continuous-intros)
    done
lemma jumpF-polyR-coprime':
    assumes poly p x\not=0\vee poly q x\not=0
    shows jumpF-polyR q p x = (if p\not=0\wedgeq\not=0\wedge poly px=0 then
                            if sign-r-pos p x \longleftrightarrow poly q x>0 then 1/2 else - 1/2
else 0)
proof (cases p=0\vee q=0 \vee poly p x\not=0)
    case True
    then show ?thesis using jumpF-poly-noroot by fastforce
next
    case False
    then have asm: }p\not=0\mathrm{ q}=0\mathrm{ poly p x=0 by auto
    then have poly q x\not=0 using assms using coprime-poly-0 by blast
    have?thesis when sign-r-pos p }x\longleftrightarrow\mathrm{ poly q x>0
    proof -
        have (poly p has-sgnx sgn (poly q x)) (at-right x)
            by (smt (z3) False <poly q x = 0` has-sgnx-imp-sgnx
                    poly-has-sgnx-values(2) sgn-real-def sign-r-pos-sgnx-iff that
                    trivial-limit-at-right-real)
    then have LIM x at-right x. poly q x / poly p x :> at-top
```

```
    apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
    apply (auto simp add:<poly q x =0`)
    by (metis asm(3) poly-tendsto(3))
    then have jumpF-polyR q p x = 1/2
        unfolding jumpF-polyR-def jumpF-def by auto
    then show ?thesis using that False by auto
    qed
    moreover have ?thesis when }\neg(\mathrm{ sign-r-pos p }x\longleftrightarrow\mathrm{ poly q x>0)
    proof -
    have (poly p has-sgnx - sgn (poly q x)) (at-right x)
    proof -
        have (0::real)<1\vee\neg(1::real)<0\wedge sign-r-pos p x
            \vee ( \text { poly p has-sgnx - sgn (poly q x)) (at-right x)}
            by simp
        then show ?thesis
        by (metis (no-types) False <poly q x\not=0` add.inverse-inverse has-sgnx-imp-sgnx
                neg-less-0-iff-less poly-has-sgnx-values(2) sgn-if sgn-less sign-r-pos-sgnx-iff
                    that trivial-limit-at-right-real)
    qed
    then have LIM x at-right x. poly q x / poly p x :> at-bot
        apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
        apply (auto simp add:<poly q }x\not=0\mathrm{ ))
        by (metis asm(3) poly-tendsto(3))
        then have jumpF-polyR q px=-1/2
        unfolding jumpF-polyR-def jumpF-def by auto
    then show ?thesis using that False by auto
    qed
    ultimately show ?thesis by auto
qed
lemma jumpF-polyR-coprime:
    assumes coprime p q
    shows jumpF-polyR q p x = (if p\not=0\wedgeq\not=0^ poly px=0 then
                    if sign-r-pos p x \longleftrightarrow poly q x>0 then 1/2 else - 1/2
else 0)
    apply (rule jumpF-polyR-coprime')
    using assms coprime-poly-0 by blast
lemma jumpF-polyL-coprime':
    assumes poly p x\not=0\vee poly q x\not=0
    shows jumpF-polyL q p x = (if p\not=0 ^q\not=0 ^ poly p x=0 then
        if even (order x p) \longleftrightarrow sign-r-pos p x \longleftrightarrow poly q x>0 then 1/2 else
- 1/2 else 0)
proof (cases p=0\vee q=0 \vee poly p x\not=0)
    case True
    then show ?thesis using jumpF-poly-noroot by fastforce
next
```

```
case False
then have asm:p\not=0 q\not=0 poly px=0 by auto
then have poly q x\not=0 using assms using coprime-poly-0 by blast
have ?thesis when even (order x p) \longleftrightarrow sign-r-pos p x \longleftrightarrow poly q x>0
proof -
    consider (lt) poly q x>0 | (gt) poly q x<0 using <poly q x\not=0> by linarith
    then have sgnx (poly p)(at-left x)=sgn (poly qx)
    apply cases
    subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF <p\not=0>,of x]
        apply (subst poly-sgnx-left-right[OF<p\not=0>])
        by auto
    subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF}\langlep\not=0\rangle,of x
        apply (subst poly-sgnx-left-right[OF <p\not=0>])
        by auto
    done
    then have (poly p has-sgnx sgn (poly q x)) (at-left x)
    by (metis sgnx-able-poly(2) sgnx-able-sgnx)
    then have LIM x at-left x. poly q x / poly p x :> at-top
        apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
        apply (auto simp add:<poly q x =0`)
        by (metis asm(3) poly-tendsto(2))
    then have jumpF-polyL q p x = 1/2
        unfolding jumpF-polyL-def jumpF-def by auto
    then show ?thesis using that False by auto
qed
moreover have ?thesis when }\neg(\mathrm{ even (order x p) }\longleftrightarrow\mathrm{ sign-r-pos p x }\longleftrightarrow\mathrm{ poly
qx>0)
    proof -
    consider (lt) poly q x>0 | (gt) poly q x<0 using <poly q x\not=0> by linarith
    then have sgnx (poly p)(at-left x)= - sgn (poly q x)
        apply cases
        subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF <p\not=0>,of x]
            apply (subst poly-sgnx-left-right[OF <p\not=0>])
            by auto
        subgoal using that sign-r-pos-sgnx-iff poly-sgnx-values[OF <p\not=0\rangle,of x]
            apply (subst poly-sgnx-left-right[OF <p\not=0>])
            by auto
        done
    then have (poly p has-sgnx - sgn (poly q x)) (at-left x)
        by (metis sgnx-able-poly(2) sgnx-able-sgnx)
    then have LIM x at-left x.poly q x / poly p x :> at-bot
        apply (subst filterlim-divide-at-bot-at-top-iff[of - poly q x])
        apply (auto simp add:<poly q x\not=0`)
        by (metis asm(3) poly-tendsto(2))
    then have jumpF-polyL q p x = - 1/2
        unfolding jumpF-polyL-def jumpF-def by auto
    then show ?thesis using that False by auto
qed
ultimately show ?thesis by auto
```


## qed

lemma jumpF-polyL-coprime:
assumes coprime $p q$
shows jumpF-polyL q $p x=($ if $p \neq 0 \wedge q \neq 0 \wedge$ poly $p x=0$ then
if even (order $x p$ ) $\longleftrightarrow$ sign-r-pos $p x \longleftrightarrow$ poly $q x>0$ then 1/2 else
$-1 / 2$ else 0$)$
apply (rule jumpF-polyL-coprime')
using assms coprime-poly-0 by blast
lemma jumpF-times:
assumes tendsto: $(f \longrightarrow c) F$ and $c \neq 0 F \neq b o t$
shows jump $F(\lambda x . f x * g x) F=\operatorname{sgn} c * j u m p F g F$
proof -
have $c>0 \vee c<0$ using $\langle c \neq 0\rangle$ by auto
moreover have ?thesis when $c>0$
proof -
note filterlim-tendsto-pos-mult-at-top-iff [OF tendsto $\langle c>0\rangle, o f g]$
moreover note filterlim-tendsto-pos-mult-at-bot-iff[OF tendsto $\langle c>0\rangle, o f g]$
moreover have $\operatorname{sgn} c=1$ using $\langle c>0\rangle$ by auto
ultimately show ?thesis unfolding jumpF-def by auto
qed
moreover have ?thesis when $c<0$
proof -
define atbot where atbot $=$ filterlim $g$ at-bot $F$
define attop where attop $=$ filterlim $g$ at-top $F$
have $j u m p F(\lambda x . f x * g x) F=($ if atbot then $1 / 2$ else if attop then - $1 / 2$
else 0)
proof -
note filterlim-tendsto-neg-mult-at-top-iff $[O F$ tendsto $\langle c<0\rangle, o f g]$
moreover note filterlim-tendsto-neg-mult-at-bot-iff [OF tendsto $\langle c<0\rangle$,of $g$ ]
ultimately show ?thesis unfolding jumpF-def atbot-def attop-def by auto
qed
also have $\ldots=-($ if attop then $1 / 2$ else if atbot then $-1 / 2$ else 0$)$
proof -
have False when atbot attop
using filterlim-at-top-at-bot $[O F-\langle F \neq b o t\rangle]$ that unfolding atbot-def
attop-def by auto
then show ?thesis by fastforce
qed
also have $\ldots=\operatorname{sgn} c * j u m p F g F$
using 〈c<0〉 unfolding jumpF-def attop-def atbot-def by auto
finally show ?thesis .
qed
ultimately show ?thesis by auto
qed
lemma jumpF-polyR-inverse-add:
assumes coprime $p q$

```
    shows jumpF-polyR q p x + jumpF-polyR p q x = jumpF-polyR 1 (q*p) x
proof (cases p=0\veeq=0)
    case True
    then show ?thesis by auto
next
    case False
    have jumpF-add:
    jumpF-polyR q p x= jumpF-polyR 1 (q*p) x when poly p x=0 coprime p q for
p q
    proof (cases p=0)
        case True
        then show ?thesis by auto
    next
        case False
    have poly q x\not=0 using that coprime-poly-0 by blast
    then have q\not=0 by auto
    moreover have sign-r-pos p x = (0<poly qx)\longleftrightarrow sign-r-pos ( q*p)x
                using sign-r-pos-mult[OF <q\not=0〉\langlep\not=0\rangle] sign-r-pos-rec[OF <q\not=0〉]<poly q
x\not=0>
            by auto
            ultimately show ?thesis using <poly p x=0`
            unfolding jumpF-polyR-coprime[OF<coprime p q>,of x] jumpF-polyR-coprime[of
q*p 1 x,simplified]
            by auto
    qed
    have False when poly p x=0 poly q x=0
        using <coprime p q> that coprime-poly-0 by blast
    moreover have ?thesis when poly p x=0 poly q x\not=0
    proof -
        have jumpF-polyR p q x = 0 using jumpF-poly-noroot[OF〈poly q x\not=0〉] by
auto
    then show ?thesis using jumpF-add[OF <poly p x=0〉\langlecoprime p q>] by auto
    qed
    moreover have ?thesis when poly p x\not=0 poly q x=0
    proof -
        have jumpF-polyR q p x = 0 using jumpF-poly-noroot[OF <poly p x\not=0〉] by
auto
    then show ?thesis using jumpF-add[OF <poly q x=0 \,of p] <coprime p q〉
        by (simp add: ac-simps)
    qed
    moreover have ?thesis when poly p }x\not=0\mathrm{ poly q }x\not=
    by (simp add: jumpF-poly-noroot(2) that(1) that(2))
    ultimately show ?thesis by auto
qed
lemma jumpF-polyL-inverse-add:
    assumes coprime pq
    shows jumpF-polyL q p x + jumpF-polyL p q x = jumpF-polyL 1 (q*p) x
proof (cases p=0\veeq=0)
```

```
    case True
    then show ?thesis by auto
next
    case False
    have jumpF-add:
    jumpF-polyL q p x= jumpF-polyL 1 (q*p)x when poly p x=0 coprime p q for
pq
    proof (cases p=0)
        case True
        then show ?thesis by auto
    next
        case False
        have poly q x\not=0 using that coprime-poly-0 by blast
        then have q\not=0 by auto
    moreover have sign-r-pos p x = (0< poly q x) \longleftrightarrow sign-r-pos (q*p)x
        using sign-r-pos-mult[OF <q\not=0〉\langlep\not=0\rangle] sign-r-pos-rec[OF <q\not=0〉]<poly q
x\not=0>
            by auto
    moreover have order x p = order x (q*p)
            by (metis <poly q x = 0` add-cancel-right-left divisors-zero order-mult or-
der-root)
    ultimately show ?thesis using <poly p x=0`
    unfolding jumpF-polyL-coprime[OF<coprime p q>,of x] jumpF-polyL-coprime[of
q*p 1 x,simplified]
            by auto
    qed
    have False when poly p x=0 poly q x=0
    using <coprime p q> that coprime-poly-0 by blast
    moreover have ?thesis when poly p x=0 poly q }x\not=
    proof -
            have jumpF-polyL p q x = 0 using jumpF-poly-noroot[OF〈poly q x\not=0`] by
auto
    then show ?thesis using jumpF-add[OF <poly p x=0` \langlecoprime p q>] by auto
    qed
    moreover have ?thesis when poly p x\not=0 poly q x=0
    proof -
        have jumpF-polyL q p x = 0 using jumpF-poly-noroot[OF〈poly p x\not=0〉] by
auto
    then show ?thesis using jumpF-add[OF <poly q x=0`,of p] <coprime p q\rangle
        by (simp add: ac-simps)
    qed
    moreover have ?thesis when poly p x\not=0 poly q }x\not=
    by (simp add: jumpF-poly-noroot that(1) that(2))
    ultimately show ?thesis by auto
qed
lemma jumpF-polyL-smult-1:
    jumpF-polyL (smult c q) px=sgn c* jumpF-polyL q p x
```

```
proof (cases c=0)
    case True
    then show ?thesis by auto
next
    case False
    then show ?thesis
        unfolding jumpF-polyL-def
        apply (subst jumpF-times[of \lambda-. c,symmetric])
        by auto
qed
```

lemma jumpF-polyR-smult-1:
jumpF-polyR (smult c q) $p x=\operatorname{sgn} c * j u m p F-p o l y R q p x$
proof (cases $c=0$ )
case True
then show ?thesis by auto
next
case False
then show?thesis
unfolding jumpF-polyR-def using False
apply (subst jumpF-times[of $\lambda$-. c,symmetric])
by auto
qed

## lemma

shows jumpF-polyR-mod:jumpF-polyR q $p x=j u m p F-p o l y R(q \bmod p) p x$ and jumpF-polyL-mod:jumpF-polyL q p $x=j u m p F-p o l y L(q \bmod p) p x$
proof -
define $f$ where $f=(\lambda x$. poly ( $q$ div $p) x)$
define $g$ where $g=(\lambda x$. poly $(q \bmod p) x / \operatorname{poly} p x)$
have jumpF-eq:jumpF ( $\lambda$ x. poly $q x /$ poly $p x)($ at $y$ within $S)=j u m p F g$ (at $y$
within $S$ )
when $p \neq 0$ for $y S$
proof -
let ? $F=$ at $y$ within $S$
have $\forall_{F} x$ in at $y$ within $S$. poly p $x \neq 0$
using eventually-poly-nz-at-within $[O F\langle p \neq 0\rangle$,of $y S]$.
then have eventually $(\lambda x$. (poly $q x /$ poly $p x)=(f x+g x))$ ?F
proof (rule eventually-mono)
fix $x$
assume $P$ : poly $p x \neq 0$
have poly $q x=\operatorname{poly}(q \operatorname{div} p * p+q \bmod p) x$
by $\operatorname{simp}$
also have $\ldots=f x *$ poly $p x+\operatorname{poly}(q \bmod p) x$
by (simp only: poly-add poly-mult $f$-def $g$-def)
moreover have poly $(q \bmod p) x=g x *$ poly $p x$
using $P$ by (simp add: $g$-def)
ultimately show poly $q x /$ poly $p x=f x+g x$

```
        using P by simp
    qed
    then have jumpF ( }\lambdax.\mathrm{ poly qx / poly p x) ?F = jumpF ( }\lambdax.fx+gx)?
    by (intro jumpF-cong,auto)
    also have ... = jumpF g?F
    proof -
    have (f\longrightarrowfy)(at y within S)
        unfolding f}f\mathrm{ -def by (intro tendsto-intros)
    from filterlim-tendsto-add-at-bot-iff[OF this,of g] filterlim-tendsto-add-at-top-iff[OF
this,of g]
            show ?thesis unfolding jumpF-def by auto
    qed
    finally show ?thesis .
qed
show jumpF-polyR q p x = jumpF-polyR (q\operatorname{mod}p) px
    apply (cases p=0)
    subgoal by auto
    subgoal using jumpF-eq unfolding g-def jumpF-polyR-def by auto
    done
    show jumpF-polyL q p x = jumpF-polyL (q mod p) p x
    apply (cases p=0)
    subgoal by auto
    subgoal using jumpF-eq unfolding g-def jumpF-polyL-def by auto
    done
qed
lemma
    assumes order x p}\leq\mathrm{ order x r
    shows jumpF-polyR-order-leq: jumpF-polyR (r+q) p x = jumpF-polyR q p x
    and jumpF-polyL-order-leq: jumpF-polyL (r+q) p x jumpF-polyL q p x
proof -
    define fgh where f=( }\lambda\mathrm{ x. poly (q+r) x / poly p x)
                    and g=( }\lambdax\mathrm{ . poly q x / poly p x)
                            and }h=(\lambdax.poly r x / poly p x
have \existsc.h-x->c if }p\not=0\quadr\not=
proof -
    define xo where xo=[:- x, 1:] ^ order x p
    obtain p' where p=xo* p
            using order-decomp[OF <p\not=0\rangle,of x] unfolding xo-def by auto
    define r' where r'= r div xo
    define }\mp@subsup{h}{}{\prime}\mathrm{ where }\mp@subsup{h}{}{\prime}=(\lambdax\mathrm{ . poly r'}x/\mathrm{ poly p' }\mp@subsup{p}{}{\prime
    have }\mp@subsup{\forall}{F}{}x\mathrm{ in at }x.hx=\mp@subsup{h}{}{\prime}
    proof -
            obtain S where open S x\inS by blast
            moreover have hw= h'w if w\inS w\not=x for w
            proof -
                have r=xo * r'
```

```
    proof -
        have xo dvdr
            unfolding xo-def using <r\not=0〉 assms
            by (subst order-divides) simp
        then show ?thesis unfolding r'-def by simp
    qed
    moreover have poly xo w\not=0
        unfolding xo-def using }\langlew\not=x\rangle\mathrm{ by simp
    moreover note \langlep =xo* p'>
    ultimately show ?thesis
        unfolding h-def h'-def by auto
    qed
    ultimately show ?thesis
    unfolding eventually-at-topological by auto
qed
moreover have }\mp@subsup{h}{}{\prime}-x->\mp@subsup{h}{}{\prime}
proof -
    have poly p' }x\not=
        using < ᄀ[:- x, 1:] dvd p'> poly-eq-0-iff-dvd by blast
    then show ?thesis
        unfolding }\mp@subsup{h}{}{\prime}-de
        by (auto intro!:tendsto-eq-intros)
qed
ultimately have }h-x->\mp@subsup{h}{}{\prime}
    using tendsto-cong by auto
then show ?thesis by auto
qed
then obtain c where left:(h\longrightarrowc)(at-left x)
                    and right:(h\longrightarrowc)(at-right x)
                        if }p\not=0\quadr\not=
unfolding filterlim-at-split by auto
show jumpF-polyR (r+q) p x = jumpF-polyR q p x
proof (cases p=0\veer=0)
    case False
have jumpF-polyR (r+q) px=
            (if filterlim ( }\lambdax.hx+gx) at-top (at-right x)
            then 1 / 2
            else if filterlim ( }\lambdax.hx+gx) at-bot (at-right x)
            then - 1 / 2 else 0)
    unfolding jumpF-polyR-def jumpF-def g-def h-def
    by (simp add:poly-add add-divide-distrib)
also have ... =
            (if filterlim g at-top (at-right x) then 1 / 2
            else if filterlim g at-bot (at-right x) then - 1 / 2 else 0)
    using filterlim-tendsto-add-at-top-iff[OF right]
        filterlim-tendsto-add-at-bot-iff[OF right] False
    by simp
also have ... = jumpF-polyR q p x
```

```
        unfolding jumpF-polyR-def jumpF-def g-def by simp
        finally show jumpF-polyR (r +q) px= jumpF-polyR q px.
        qed auto
    show jumpF-polyL (r+q) p x = jumpF-polyL q p x
    proof (cases p=0\vee 
    case False
    have jumpF-polyL (r+q) px=
                (if filterlim ( }\lambdax.hx+gx) at-top(at-left x
                then 1 / 2
                else if filterlim ( }\lambdax.hx+gx) at-bot (at-left x)
                then - 1 / 2 else 0)
    unfolding jumpF-polyL-def jumpF-def g-def h-def
    by (simp add:poly-add add-divide-distrib)
    also have ... =
        (if filterlim g at-top (at-left x) then 1 / 2
            else if filterlim g at-bot (at-left x) then - 1 / 2 else 0)
    using filterlim-tendsto-add-at-top-iff[OF left]
        filterlim-tendsto-add-at-bot-iff[OF left] False
    by simp
    also have ... = jumpF-polyL q p x
    unfolding jumpF-polyL-def jumpF-def g-def by simp
    finally show jumpF-polyL (r +q) px=jumpF-polyL q p x .
    qed auto
qed
lemma
    assumes order x q< order x r q\not=0
    shows jumpF-polyR-order-le:jumpF-polyR (r+q) p x = jumpF-polyR q p x
        and jumpF-polyL-order-le:jumpF-polyL (r+q) p x =jumpF-polyL q p x
proof -
    have jumpF-polyR (r+q) px=jumpF-polyR q p x
    jumpF-polyL (r+q) p x = jumpF-polyL q p x
    if p=0 \vee r=0 \vee order x p\leqorder x r
    using jumpF-polyR-order-leq jumpF-polyL-order-leq that by auto
    moreover have
    jumpF-polyR (r+q) px=jumpF-polyR q p x
    jumpF-polyL (r+q) p x = jumpF-polyL q p x
    if p\not=0 r\not=0 order x p> order x r
proof -
    define xo where xo=[:- x, 1:] ^ order x q
    have [simp]:xo\not=0 unfolding xo-def by simp
    have xo-q:order x xo = order x q
        unfolding xo-def by (meson order-power-n-n)
    obtain q' where q:q=xo* q' and }\neg[:-x,1:] dvd q'
        using order-decomp[OF <q\not=0〉,of x] unfolding xo-def by auto
    from this(2)
    have poly q' }x\not=0\mathrm{ using poly-eq-0-iff-dvd by blast
    define }\mp@subsup{p}{}{\prime}\mp@subsup{r}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\mathrm{ div xo and }\mp@subsup{r}{}{\prime}=r\mathrm{ div xo
```

```
have \(p: p=x o * p^{\prime}\)
proof -
    have order \(x q<\) order \(x p\)
        using assms(1) less-trans that(3) by blast
    then have \(x o\) dvd \(p\)
        unfolding xo-def by (metis less-or-eq-imp-le order-divides)
    then show?thesis by (simp add: \(p^{\prime}\)-def)
qed
have \(r: r=x o * r^{\prime}\)
proof -
    have \(x o\) dvd \(r\)
        unfolding xo-def by (meson assms(1) less-or-eq-imp-le order-divides)
    then show?thesis by (simp add: r'-def)
qed
have poly \(r^{\prime} x=0\)
proof -
    have order \(x r=\) order \(x\) xo + order \(x r^{\prime}\)
        unfolding \(r\) using \(\langle r \neq 0\rangle r\) order-mult by blast
    with \(x o-q\) have order \(x r^{\prime}=\) order \(x r-\operatorname{order} x q\)
        by auto
    then have order \(x r^{\prime}>0\)
        using «order \(x r<\) order \(x\) p assms(1) by linarith
    then show poly \(r^{\prime} x=0\) using order-root by blast
qed
have poly \(p^{\prime} x=0\)
proof -
    have order \(x p=\operatorname{order} x\) xo + order \(x p^{\prime}\)
        unfolding \(p\) using \(\langle p \neq 0\rangle p\) order-mult by blast
    with \(x o-q\) have order \(x p^{\prime}=\) order \(x p-\operatorname{order} x q\)
        by auto
    then have order \(x p^{\prime}>0\)
        using <order \(x r<\) order \(x\) p assms(1) by linarith
    then show poly \(p^{\prime} x=0\) using order-root by blast
qed
have jumpF-polyL \((r+q) p x=j u m p F-p o l y L\left(x o *\left(r^{\prime}+q^{\prime}\right)\right)\left(x o * p^{\prime}\right) x\)
    unfolding \(p q r\) by (simp add:algebra-simps)
also have \(\ldots=j u m p F-p o l y L\left(r^{\prime}+q^{\prime}\right) p^{\prime} x\)
    by (rule jumpF-polyL-mult-cancel) simp
also have \(\ldots=\left(\right.\) if even (order x \(\left.p^{\prime}\right)=\left(\right.\) sign-r-pos \(p^{\prime} x\)
            \(\left.=\left(0<\operatorname{poly}\left(r^{\prime}+q^{\prime}\right) x\right)\right)\) then \(1 / 2\) else \(\left.-1 / 2\right)\)
proof -
    have poly \(\left(r^{\prime}+q^{\prime}\right) x \neq 0\)
        using \(\left\langle p o l y q^{\prime} x \neq 0\right\rangle\left\langle p o l y r^{\prime} x=0\right\rangle\) by auto
    then show ?thesis
        apply (subst jumpF-polyL-coprime')
        subgoal by \(\operatorname{simp}\)
        subgoal by \(\left(s m t(z 3)\langle p \neq 0\rangle\left\langle p o l y p^{\prime} x=0\right\rangle\right.\) mult.commute
            mult-zero-left \(p\) poly-0)
```

```
        done
    qed
    also have ... = (if even (order x p') = (sign-r-pos p' x
    =(0<poly q' x)) then 1 / 2 else - 1 / 2)
    using <poly r' }x=0\mathrm{ ` by auto
    also have ... = jumpF-polyL q' p' x
    apply (subst jumpF-polyL-coprime')
    subgoal using <poly q' }x\not=0\rangle\mathrm{ by blast
    subgoal using }\langlep\not=0\rangle\langlepoly p'x=0\rangle assms(2) p q by sim
    done
    also have ... = jumpF-polyL q p x
    unfolding pq by (subst jumpF-polyL-mult-cancel) simp-all
    finally show jumpF-polyL (r+q) px=jumpF-polyL q px.
    have jumpF-polyR (r+q) p x = jumpF-polyR (xo * (r'+q')) (xo*p') x
    unfolding p qr by (simp add:algebra-simps)
    also have ... = jumpF-polyR (r'+q') p' x
    by (rule jumpF-polyR-mult-cancel) simp
    also have ... = (if sign-r-pos p' x = (0<poly (r'+ q') x)
    then 1 / 2 else - 1 / 2)
proof -
    have poly ( }\mp@subsup{r}{}{\prime}+\mp@subsup{q}{}{\prime})x\not=
        using <poly q' }x\not=0\rangle\langlepoly \mp@subsup{r}{}{\prime}x=0`\mathrm{ by auto
    then show ?thesis
        apply (subst jumpF-polyR-coprime')
        subgoal by simp
        subgoal
            by (smt (z3)<p}\not=0\rangle\langlepoly p' x = 0\rangle mult.commute
                mult-zero-left p poly-0)
        done
    qed
    also have ... = (if sign-r-pos p' }x=(0<poly q' x)
    then 1 / 2 else - 1 / 2)
    using <poly r' }x=0\mathrm{ 〉 by auto
    also have ... = jumpF-polyR q' }\mp@subsup{p}{}{\prime}
        apply (subst jumpF-polyR-coprime')
        subgoal using <poly q' x}\not=0\mathrm{ > by blast
        subgoal using }\langlep\not=0\rangle\langlepoly p'x=0\rangle assms(2) p q by forc
        done
    also have ... = jumpF-polyR q p x
    unfolding pq by (subst jumpF-polyR-mult-cancel) simp-all
    finally show jumpF-polyR (r+q) p x = jumpF-polyR q p x.
qed
ultimately show
    jumpF-polyR (r+q) px=jumpF-polyR q p x
    jumpF-polyL (r+q) p x = jumpF-polyL q p x
    by force +
qed
```

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

## next

case False
define $P$ where $P=(\neg($ LIM $x$ at-top. poly $q x /$ poly $p x:>$ at-bot $)$

$$
\wedge \neg(\text { LIM x at-top. poly } q x / \text { poly } p x:>\text { at-top }))
$$

have $P$ when $p=0 \vee q=0$
unfolding $P$-def using that
by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds)
moreover have $P$ when $p \neq 0 \quad q \neq 0$ degree $p>$ degree $q$
proof -
have LIM $x$ at-top. poly $q x / \operatorname{poly} p x:>$ at 0
using poly-divide-filterlim-at-top $[O F$ that (1,2)] that(3) by auto
then show ?thesis unfolding $P$-def
by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds simp:filterlim-at)
qed
moreover have $P$ when $p \neq 0 \quad q \neq 0$ degree $p=$ degree $q$
proof -
have $((\lambda x$. poly $q x /$ poly $p x) \longrightarrow$ lead-coeff $q /$ lead-coeff $p)$ at-top
using poly-divide-filterlim-at-top[OF that (1,2)] using that by auto
then show ?thesis unfolding $P$-def
by (auto elim!:filterlim-at-bot-nhds filterlim-at-top-nhds)
qed
ultimately have $P$ using False by fastforce
then have jump $F(\lambda x$. poly $q x /$ poly $p x)$ at-top $=0$
unfolding jumpF-def $P$-def by auto
then show ?thesis unfolding jumpF-poly-top-def using False by presburger qed
lemma jumpF-poly-bot-code:
jumpF-poly-bot $q p=($ if $p \neq 0 \wedge q \neq 0 \wedge$ degree $q>$ degree $p$ then
if sgn-neg-inf $q *$ sgn-neg-inf $p>0$ then $1 / 2$ else $-1 / 2$ else 0$)$
proof (cases $p \neq 0 \wedge q \neq 0 \wedge$ degree $q>$ degree $p$ )
case True
have ?thesis when sgn-neg-inf $q * \operatorname{sgn}$-neg-inf $p>0$
proof -
have LIM x at-bot. poly $q x /$ poly $p x$ :> at-top
using poly-divide-filterlim-at-bot $[$ of $p q]$ True that by auto
then have $j u m p F(\lambda x$. poly $q x /$ poly $p x)$ at-bot $=1 / 2$
unfolding jumpF-def by auto
then show ?thesis unfolding jumpF-poly-bot-def using that True by auto qed
moreover have ?thesis when $\neg \operatorname{sgn}$-neg-inf $q * \operatorname{sgn}$-neg-inf $p>0$
proof -
have LIM $x$ at-bot. poly $q x /$ poly $p x:>$ at-bot
using poly-divide-filterlim-at-bot $[$ of $p q]$ True that by auto
then have $j u m p F(\lambda x$. poly $q x /$ poly $p x)$ at-bot $=-1 / 2$
unfolding jump $F$-def by auto
then show ?thesis unfolding jumpF-poly-bot-def using that True by auto qed
ultimately show ?thesis by auto

## next

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

```
    ultimately have P using False by fastforce
    then have jumpF ( }\lambdax\mathrm{ . poly q x / poly p x) at-bot = 0
    unfolding jumpF-def P-def by auto
    then show ?thesis unfolding jumpF-poly-bot-def using False by presburger
qed
lemma jump-poly-jumpF-poly:
    shows jump-poly q p x = jumpF-polyR q p x - jumpF-polyL q p x
proof (cases p=0\veeq=0)
    case True
    then show ?thesis by auto
next
    case False
    have *:jump-poly q p x = jumpF-polyR q p x - jumpF-polyL q p x
        if coprime q p for q p
    proof (cases p=0\veeq=0 \vee poly p x\not=0)
            case True
            moreover have ?thesis if p=0\veeq=0 using that by auto
            moreover have ?thesis if poly p }x\not=
            by (simp add: jumpF-poly-noroot(1) jumpF-poly-noroot(2) jump-poly-not-root
that)
            ultimately show ?thesis by blast
    next
            case False
            then have p\not=0 q\not=0 poly px=0 by auto
            have jump-poly q p x = jump ( }\lambdax\mathrm{ x. poly q x / poly p x) x
                using jump-jump-poly by simp
            also have real-of-int ... = jumpF ( }\lambdax\mathrm{ . poly q x / poly px) (at-right x) -
                    jumpF (\lambdax. poly q x / poly p x) (at-left x)
    proof (rule jump-jumpF)
            have poly q x\not=0 by (meson False coprime-poly-0 that)
            then show isCont (inverse ○ ( }\lambdax.\mathrm{ poly q x / poly p x)) x
                unfolding comp-def by simp
            define l where l = sgnx ( }\lambdax\mathrm{ . poly qx / poly p x) (at-left x)
            define r}\mathrm{ where r=sgnx ( }\lambdax\mathrm{ x. poly q x / poly p x) (at-right x)
            show ((\lambdax. poly q x / poly p x) has-sgnx l) (at-left x)
                unfolding l-def by (auto intro!:sgnx-intros sgnx-able-sgnx)
            show (( }\lambdax.\mathrm{ poly q x / poly p x) has-sgnx r) (at-right x)
                unfolding r-def by (auto intro!:sgnx-intros sgnx-able-sgnx)
            show l\not=0 unfolding l-def
                apply (subst sgnx-divide)
                using poly-sgnx-values[OF〈p\not=0\rangle, of x] poly-sgnx-values[OF <q\not=0`, of x]
                by auto
            show r\not=0 unfolding r-def
                apply (subst sgnx-divide)
                using poly-sgnx-values[OF <p\not=0\rangle, of x] poly-sgnx-values[OF <q\not=0\rangle, of x]
                by auto
```

```
    qed
    also have ... = jumpF-polyR q p x - jumpF-polyL q p x
        unfolding jumpF-polyR-def jumpF-polyL-def by simp
    finally show ?thesis .
qed
    obtain p}\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}g\mathrm{ where pq:p=g*p
    using gcd-coprime-exists[of p q]
    by (metis False coprime-commute gcd-coprime-exists gcd-eq-0-iff mult.commute)
    then have g\not=0 using False mult-zero-left by blast
    then have jump-poly q p x =jump-poly q' p' x
    unfolding pq using jump-poly-mult by auto
    also have .. = jumpF-polyR q}\mp@subsup{q}{}{\prime}\mp@subsup{p}{}{\prime}x-jumpF-polyL q' p p'
    using *[OF <coprime q' }\mp@subsup{p}{}{\prime}\rangle]
    also have ... = jumpF-polyR q p x - jumpF-polyL q p x
    unfolding pq using <g\not=0〉 jumpF-polyL-mult-cancel jumpF-polyR-mult-cancel
by auto
    finally show ?thesis.
qed
```


### 2.6 The extended Cauchy index on polynomials

definition cindex-polyE:: real $\Rightarrow$ real $\Rightarrow$ real poly $\Rightarrow$ real poly $\Rightarrow$ real where cindex-polyE abqp=jumpF-polyR q pa+cindex-poly abqp-jumpF-polyL $q p b$
definition cindex-poly-ubd::real poly $\Rightarrow$ real poly $\Rightarrow$ int where
cindex-poly-ubd q $p=\left(\right.$ THE $l .\left(\forall_{F} r\right.$ in at-top. cindexE $(-r) r(\lambda x$. poly $q x /$ poly $p x)=o f-i n t l)$ )
lemma cindex-polyE-0[simp]: cindex-polyE abop=0 cindex-polyE a b q $0=0$ unfolding cindex-polyE-def by auto
lemma cindex-polyE-mult-cancel:
fixes $p q p^{\prime}::$ real poly
assumes $p^{\prime} \neq 0$
shows cindex-polyE ab( $\left.p^{\prime} * q\right)\left(p^{\prime} * p\right)=$ cindex-polyE ab $q$ p
unfolding cindex-polyE-def
using cindex-poly-mult[OF $\left\langle p^{\prime} \neq 0\right\rangle$ ] jump $F$-polyL-mult-cancel $\left[O F\left\langle p^{\prime} \neq 0\right\rangle\right]$ jumpF-polyR-mult-cancel $\left[O F\left\langle p^{\prime} \neq 0\right\rangle\right]$
by $\operatorname{simp}$
lemma cindexE-eq-cindex-polyE:
assumes $a<b$
shows cindexE ab( $\lambda$. poly $q x /$ poly $p x)=$ cindex-polyE a b q p
proof (cases $p=0 \vee q=0$ )
case True
then show ?thesis by (auto simp add: cindexE-constI)
next

```
case False
then have p\not=0 q\not=0 by auto
define g}\mathrm{ where g=gcd pq
define }\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\mathrm{ div }g\mathrm{ and }\mp@subsup{q}{}{\prime}=q\mathrm{ div g
define f}\mp@subsup{f}{}{\prime}\mathrm{ where }\mp@subsup{f}{}{\prime}=(\lambdax\mathrm{ . poly }\mp@subsup{q}{}{\prime}x/\mathrm{ poly }\mp@subsup{p}{}{\prime}x
have g\not=0 using False g-def by auto
have pq-f:p=g*\mp@subsup{p}{}{\prime}}q=g*\mp@subsup{q}{}{\prime}\mathrm{ and coprime }\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime
    unfolding g-def }\mp@subsup{p}{}{\prime}\mathrm{ -def }\mp@subsup{q}{}{\prime}-de
    apply simp-all
    using False div-gcd-coprime by blast
have cindexE a b ( }\lambdax\mathrm{ . poly q x/poly p x) = cindexE a b ( }\lambdax.\mathrm{ poly q' x/poly p' x)
proof -
    define }f\mathrm{ where }f=(\lambdax\mathrm{ . poly q x / poly p x)
    define }\mp@subsup{f}{}{\prime}\mathrm{ where }\mp@subsup{f}{}{\prime}=(\lambdax\mathrm{ . poly }\mp@subsup{q}{}{\prime}x/\mp@code{poly p' x)
    have jumpF f(at-right x) =jumpF f ' (at-right x) for }
    proof (rule jumpF-cong)
    obtain ub where }x<ub\forallz.x<z\wedgez\lequb\longrightarrow\mathrm{ poly gz}=
        using next-non-root-interval[OF <g\not=0\rangle,of x] by auto
    then show }\mp@subsup{\forall}{F}{}x\mathrm{ in at-right x. fx= f
        unfolding eventually-at-right f-def f'-def pq-f
        apply (intro exI[where }x=ub]\mathrm{ )
        by auto
    qed simp
    moreover have jumpFf(at-left x) = jumpF f'(at-left x) for x
    proof (rule jumpF-cong)
    obtain lb where lb<x\forallz.lb\leqz\wedgez<x\longrightarrow poly gz\not=0
        using last-non-root-interval[OF <g\not=0\rangle,of x] by auto
    then show }\mp@subsup{\forall}{F}{}x\mathrm{ in at-left x.fx= f'x
        unfolding eventually-at-left f-def f'-def pq-f
        apply (intro exI[where }x=lb]\mathrm{ )
        by auto
    qed simp
    ultimately show ?thesis unfolding cindexE-def
    apply (fold f-def f'-def)
    by auto
qed
also have ... = jumpF f'(at-right a) + real-of-int (cindex a b f') - jumpF f'
(at-left b)
    unfolding f'-def
    apply (rule cindex-eq-cindexE-divide)
    subgoal using <a<b\rangle.
    subgoal
    proof -
    have finite (proots ( }\mp@subsup{q}{}{\prime}*\mp@subsup{p}{}{\prime})\mathrm{ )
        using False poly-roots-finite pq-f(1) pq-f(2) by auto
    then show finite {x.(poly q}\mp@subsup{q}{}{\prime}x=0\vee poly p'x=0)\wedgea\leqx^x\leqb
        by (elim rev-finite-subset) auto
    qed
    subgoal using <coprime p' q'> poly-gcd-0-iff by force
```

subgoal by (auto intro:continuous-intros)
subgoal by (auto intro:continuous-intros)
done
also have $\ldots=$ cindex-polyE ab $q^{\prime} p^{\prime}$
using cindex-eq-cindex-poly unfolding cindex-polyE-def jumpF-polyR-def jumpF-polyL-def
$f^{\prime}$-def
by auto
also have $\ldots=$ cindex-polyE abqp
using cindex-polyE-mult-cancel $[O F\langle g \neq 0\rangle]$ unfolding $p q-f$ by auto
finally show ?thesis .
qed
lemma cindex-polyE-cross:
fixes $p$ ::real poly and $a b::$ real
assumes $a<b$
shows cindex-polyE ab1p=cross-alt $1 p a b / 2$
proof (induct degree $p$ arbitrary:p rule:nat-less-induct)
case induct:1
have ?case when $p=0$
using that unfolding cross-alt-def by auto
moreover have ?case when $p \neq 0$ and noroot: $\{x . a<x \wedge x<b \wedge$ poly p $x=0\}$
$=\{ \}$
proof -
have cindex-polyE ab1p=jumpF-polyR1pa-jumpF-polyL1pb
proof -
have cindex-poly a b $1 p=0$ unfolding cindex-poly-def
apply (rule sum.neutral)
using that by auto
then show ?thesis unfolding cindex-polyE-def by auto
qed
also have $\ldots=$ cross-alt 1 pab/2
proof -
define $f$ where $f=(\lambda x .1 /$ poly $p x)$
define $j a$ where $j a=j u m p F f($ at-right $a)$
define $j b$ where $j b=j u m p F f$ (at-left b)
define right where right $=(\lambda R . R$ ja ( $0::$ real $) \vee($ continuous (at-right a) $f$
$\wedge R($ poly $p a) 0))$
define left where left $=(\lambda R . R j b(0::$ real $) \vee($ continuous $($ at-left $b) f \wedge R$ (poly $p$ b) 0))
note $j a$-alt=jumpF-poly $R$-coprime[of p 1 a,unfolded jumpF-polyR-def,simplified,folded $f$-def $j a-d e f]$
note $j b$-alt=jumpF-polyL-coprime[of p 1 b,unfolded jumpF-polyL-def,simplified,folded $f$-def $j b-d e f]$
have $[$ simp $]: 0<j a \longleftrightarrow j u m p F-p o l y R 1 p a=1 / 20>j a \longleftrightarrow j u m p F-p o l y R$ $1 p a=-1 / 2$
$0<j b \longleftrightarrow j u m p F-p o l y L 1 p b=1 / 20>j b \longleftrightarrow j u m p F-p o l y L 1 p b=$ $-1 / 2$
unfolding ja－def jb－def jumpF－polyR－def jumpF－polyL－def f－def jumpF－def by auto
have［simp］：
poly $p$ a $\neq 0 \Longrightarrow$ continuous（at－right a）$f$
poly $p b \neq 0 \Longrightarrow$ continuous（at－left b）$f$
unfolding $f$－def by（auto intro！：continuous－intros ）
have not－right－left：False when（right greater $\wedge$ left less $\vee$ right less $\wedge$ left greater）
proof－
have［simp］：$f a>0 \longleftrightarrow$ poly $p a>0 f a<0 \longleftrightarrow$ poly $p a<0$
$f b>0 \longleftrightarrow$ poly $p b>0 f b<0 \longleftrightarrow$ poly $p b<0$
unfolding $f$－def by auto
have continuous－on $\{a<. .<b\} f$
unfolding $f$－def using noroot by（auto intro！：continuous－intros）
then have $\exists x>a . x<b \wedge f x=0$
apply（elim jumpF－IVT［OF $\langle a<b\rangle, o f f]$ ）
using that unfolding right－def left－def by（fold ja－def jb－def，auto）
then show False using noroot using $f$－def by auto
qed
have ？thesis when poly $p a>0 \wedge$ poly $p b>0 \vee$ poly $p a<0 \wedge$ poly $p b<0$
using that jumpF－poly－noroot
unfolding cross－alt－def psign－diff－def by auto
moreover have False when poly p $a>0 \wedge$ poly $p b<0 \vee$ poly $p a<0 \wedge$ poly p $b>0$
apply（rule not－right－left）
unfolding right－def left－def using that by auto
moreover have ？thesis when poly $p a=0$ poly $p b>0 \vee$ poly $p b<0$
proof－
have $j a>0 \vee j a<0$ using $j a$－alt $\langle p \neq 0\rangle\langle p o l y p a=0\rangle$ by argo
moreover have False when $j a>0 \wedge$ poly $p b<0 \vee j a<0 \wedge$ poly $p b>0$
apply（rule not－right－left）
unfolding right－def left－def using that by fastforce
moreover have ？thesis when $j a>0 \wedge$ poly $p b>0 \vee j a<0 \wedge$ poly $p b<0$
using that jumpF－poly－noroot 〈poly p $a=0$ 〉
unfolding cross－alt－def psign－diff－def by auto
ultimately show？？thesis using that jumpF－poly－noroot unfolding cross－alt－def by auto
qed
moreover have ？thesis when poly $p b=0$ poly $p a>0 \vee$ poly $p a<0$
proof－
have $j b>0 \vee j b<0$ using $j b$－alt $\langle p \neq 0\rangle\langle p o l y p b=0\rangle$ by argo
moreover have False when $j b>0 \wedge$ poly $p a<0 \vee j b<0 \wedge$ poly $p a>0$ apply（rule not－right－left）
unfolding right－def left－def using that by fastforce
moreover have？thesis when $j b>0 \wedge$ poly $p a>0 \vee j b<0 \wedge$ poly p $a<0$ using that jumpF－poly－noroot 〈poly $p b=0$ 〉
unfolding cross－alt－def psign－diff－def by auto
ultimately show ？thesis using that jumpF－poly－noroot unfolding cross－alt－def by auto

## qed

moreover have ？thesis when poly $p a=0$ poly $p b=0$
proof－
have $j b>0 \vee j b<0$ using $j b$－alt $\langle p \neq 0\rangle\langle$ poly $p b=0\rangle$ by argo
moreover have $j a>0 \vee j a<0$ using ja－alt $\langle p \neq 0\rangle\langle p o l y p a=0\rangle$ by argo
moreover have False when $j a>0 \wedge j b<0 \vee j a<0 \wedge j b>0$
apply（rule not－right－left）
unfolding right－def left－def using that by fastforce
moreover have ？thesis when $j a>0 \wedge j b>0 \vee j a<0 \wedge j b<0$
using that jumpF－poly－noroot 〈poly p $b=0\rangle\langle p o l y p a=0\rangle$
unfolding cross－alt－def psign－diff－def by auto
ultimately show ？thesis by blast
qed
ultimately show ？thesis by argo
qed
finally show？thesis ．
qed
moreover have ？case when $p \neq 0$ and no－empty：$\{x . a<x \wedge x<b \wedge$ poly $p x=0$ $\} \neq\{ \}$
proof－
define roots where roots $\equiv\{x . a<x \wedge x<b \wedge$ poly $p x=0\}$
have finite roots unfolding roots－def using poly－roots－finite $[O F\langle p \neq 0\rangle]$ by auto
define max－$r$ where max－$r \equiv$ Max roots
hence poly $p$ max－$r=0$ and $a<$ max－$r$ and max－$r<b$
using Max－in［OF〈finite roots〉］no－empty unfolding roots－def by auto
define max－rp where max－rp $\equiv[:-\max -r, 1:]$ order max－r $p$
then obtain $p^{\prime}$ where $p^{\prime}$－def：$p=p^{\prime} * \max -r p$ and $\neg[:-\max -r, 1:] d v d p^{\prime}$ by（metis $\langle p \neq 0\rangle$ mult．commute order－decomp）
hence $p^{\prime} \neq 0$ and $\max -r p \neq 0$ and max－$r-n z: p o l y ~ p^{\prime} \max -r \neq 0$
using $\langle p \neq 0\rangle$ by（auto simp add：dvd－iff－poly－eq－ 0 ）
define max－r－sign where max－r－sign $\equiv$ if odd（order max－r $p$ ）then－ 1 else $1::$ int
define roots＇where roots ${ }^{\prime} \equiv\left\{x . a<x \wedge x<b \wedge\right.$ poly $\left.p^{\prime} x=0\right\}$
have cindex－polyE ab1p＝jumpF－polyR $1 p a+\left(\sum x \in\right.$ roots．jump－poly $1 p$ $x)$－jumpF－polyL $1 p b$
unfolding cindex－polyE－def cindex－poly－def roots－def by（simp，meson）
also have $\ldots=$ max－r－sign $*$ cindex－poly a blat $1 p^{\prime}+$ jump－poly 1 p max－r

+ max－r－sign $*$ jumpF－polyR $1 p^{\prime} a-j u m p F-p o l y L 1 p^{\prime} b$
proof－
have $\left(\sum x \in\right.$ roots．jump－poly 1 p $\left.x\right)=$ max－r－sign $*$ cindex－poly a bllll${ }^{\prime}+$ jump－poly 1 p max－r
proof－
have $\left(\sum x \in\right.$ roots．jump－poly $\left.1 \quad p x\right)=\left(\sum x \in\right.$ roots $^{\prime}$ ．jump－poly 1 p $\left.x\right)+$ jump－poly 1 p max－r
proof－
have roots $=$ insert max－r roots ${ }^{\prime}$
unfolding roots－def roots＇－def $p^{\prime}$－def

```
            using <poly p max-r=0\rangle\langlea<max-r\rangle\langlemax- }r<b\rangle\langlep\not=0\rangle\mathrm{ order-root
            apply (subst max-rp-def)
            by auto
            moreover have finite roots'
            unfolding roots'-def using poly-roots-finite[OF \langlep'}=0\rangle]\mathrm{ by auto
            moreover have max-r & roots'
            unfolding roots'-def using max-r-nz
            by auto
                            ultimately show ?thesis using sum.insert[of roots' max-r] by auto
    qed
    moreover have (\sumx\inroots'. jump-poly 1 p x) = max-r-sign * cindex-poly
ab1 p'
    proof -
    have (\sumx\inroots'. jump-poly 1 px)=(\sumx\inroots'. max-r-sign * jump-poly
1 p' x)
    proof (rule sum.cong,rule refl)
            fix }x\mathrm{ assume }x\in\mp@subsup{\mathrm{ roots'}}{}{\prime
            hence }x\not=\mathrm{ max-r using max-r-nz unfolding roots'-def
                by auto
            hence poly max-rp x\not=0 using poly-power-n-eq unfolding max-rp-def
by auto
            hence order x max-rp=0 by (metis order-root)
            moreover have jump-poly 1 max-rp x=0
                using <poly max-rp x\not=0> by (metis jump-poly-not-root)
            moreover have x\in roots
            using <x \in roots'> unfolding roots-def roots'-def p'-def by auto
            hence x<max-r
            using Max-ge[OF< finite roots〉,of x] «x\not=max-r〉 by (fold max-r-def,auto)
            hence sign (poly max-rp x) = max-r-sign
            using <poly max-rp x\not=0〉 unfolding max-r-sign-def max-rp-def sign-def
            by (subst poly-power,simp add:linorder-class.not-less zero-less-power-eq)
            ultimately show jump-poly 1 px=max-r-sign * jump-poly 1 p'x
                using jump-poly-1-mult[of p' x max-rp] unfolding p'-def
            by (simp add: <poly max-rp x = 0`)
    qed
    also have ... = max-r-sign * (\sumx\inroots'. jump-poly 1 p'x)
            by (simp add: sum-distrib-left)
    also have ... = max-r-sign * cindex-poly a b 1 p'
            unfolding cindex-poly-def roots'-def by meson
    finally show ?thesis.
qed
    ultimately show ?thesis by simp
    qed
    moreover have jumpF-polyR 1 pa= max-r-sign * jumpF-polyR 1 p'a
    proof -
    define f}\mathrm{ where f}=(\lambdax.1/ poly max-rp x
    define g}\mathrm{ where }g=(\lambdax.1/ poly p p'x
    have jumpF-polyR 1 pa=jumpF ( }\lambdax.fx*gx)(at-right a
        unfolding jumpF-polyR-def f-def g-def p'-def
```

```
    by (auto simp add:field-simps)
    also have ... = sgn (f a)* jumpFg(at-right a)
    proof (rule jumpF-times)
    have [simp]: poly max-rp a\not=0
        unfolding max-rp-def using <max-r>a> by auto
    show (f\longrightarrowf a) (at-right a)fa\not=0
        unfolding f-def by (auto intro:tendsto-intros)
    qed auto
    also have ... = max-r-sign * jumpF-polyR 1 p'a
    proof -
    have sgn (fa) = max-r-sign
        unfolding max-r-sign-def f-def max-rp-def using <a<max-r>
        by (auto simp add:sgn-power)
    then show ?thesis unfolding jumpF-polyR-def g-def by auto
    qed
    finally show ?thesis.
qed
moreover have jumpF-polyL 1pb=jumpF-polyL 1 p'b
proof -
    define f}\mathrm{ where f}=(\lambdax.1 / poly max-rp x
    define g}\mathrm{ where }g=(\lambdax.1/\mathrm{ poly }\mp@subsup{p}{}{\prime}x
    have jumpF-polyL 1 pb=jumpF( }\lambdax.fx*gx)(at-left b
    unfolding jumpF-polyL-def f-def g-def p'-def
    by (auto simp add:field-simps)
    also have ... = sgn (f b)* jumpF g (at-left b)
    proof (rule jumpF-times)
    have [simp]: poly max-rp b}\not=
        unfolding max-rp-def using <max-r<b> by auto
    show (f\longrightarrowfb) (at-left b) fb\not=0
        unfolding f}f\mathrm{ -def by (auto intro:tendsto-intros)
    qed auto
    also have ... = jumpF-polyL 1 p'b
    proof -
        have sgn (fb)=1
            unfolding max-r-sign-def f-def max-rp-def using \b>max-r>
            by (auto simp add:sgn-power)
        then show ?thesis unfolding jumpF-polyL-def g-def by auto
    qed
    finally show ?thesis.
qed
    ultimately show ?thesis by auto
qed
also have ... = max-r-sign * cindex-polyE a b 1 p' + jump-poly 1p max-r
    +(max-r-sign - 1)* jumpF-polyL 1 p'b
    unfolding cindex-polyE-def roots'-def by (auto simp add:algebra-simps)
also have ... = max-r-sign * cross-alt 1 1 p'ab/2 + jump-poly 1p max-r
    +(max-r-sign - 1)* jumpF-polyL 1 p'b
proof -
    have degree max-rp>0 unfolding max-rp-def degree-linear-power
```

```
            using <poly p max-r=0〉 order-root \langlep\not=0> by blast
    then have degree p'<degree p unfolding p}\mp@subsup{p}{}{\prime}\mathrm{ -def
            using degree-mult-eq[OF \langle\mp@subsup{p}{}{\prime}\not=0\rangle\langle<max-rp\not=0\rangle] by auto
        from induct[rule-format, OF this]
        have cindex-polyE a b 1 p'= real-of-int (cross-alt 1 p'ab) / 2 by auto
        then show ?thesis by auto
    qed
    also have ... = real-of-int (cross-alt 1 pab) / 2
    proof -
    have sjump-p:jump-poly 1 p max-r = (if odd (order max-r p) then sign (poly
p' max-r) else 0)
    proof -
            note max-r-nz
            moreover then have poly max-rp max-r=0
            using <poly p max-r = 0` p'-def by auto
            ultimately have jump-poly 1 p max-r = sign (poly p' max-r) * jump-poly
1 \text { max-rp max-r}
            unfolding }\mp@subsup{p}{}{\prime}\mathrm{ -def using jump-poly-1-mult[of p' max-r max-rp]
            by auto
            also have ... = (if odd (order max-r max-rp) then sign (poly p' max-r) else
0)
            proof -
            have sign-r-pos max-rp max-r
                unfolding max-rp-def using sign-r-pos-power by auto
            then show ?thesis using <max-rp\not=0` unfolding jump-poly-def by auto
    qed
    also have ... = (if odd (order max-r p) then sign (poly p' max-r) else 0)
    proof -
            have order max-r p'=0 by (simp add: <poly p' max-r \not=0` order-0I)
            then have order max-r max-rp = order max-r p
                unfolding }\mp@subsup{p}{}{\prime}\mathrm{ -def using < p
                    apply (subst order-mult)
                by auto
            then show ?thesis by auto
        qed
            finally show ?thesis .
    qed
    have ?thesis when even (order max-r p)
    proof -
            have sign (poly px)=(\operatorname{sign}(\mathrm{ poly p' x)::int) when }x\not=\mathrm{ max- r for }x
            proof -
            have sign (poly max-rp x)=(1::int)
                unfolding max-rp-def using <even (order max-r p)> that
                    apply (simp add:sign-power )
                    by (simp add: Sturm-Tarski.sign-def)
                            then show ?thesis unfolding p'-def by (simp add:sign-times)
qed
from this[of a] this[of b] <a<max-r\rangle\langlemax-r<b>
have cross-alt 1 p' a b cross-alt 1 p ab
```

unfolding cross－alt－def psign－diff－def by auto
then show ？thesis using that unfolding max－r－sign－def sjump－p by auto qed
moreover have？thesis when odd（order max－r p）
proof－
let ？thesis2 $=\operatorname{sign}\left(\right.$ poly $p^{\prime}$ max－r $) * 2-$ cross－alt $1 p^{\prime} a b-4 * j u m p F-p o l y L$ $1 p^{\prime} b$

$$
=\text { cross-alt } 1 \text { pab }
$$

have ？thesis2 when poly $p^{\prime} b=0$
proof－
have jumpF－polyL $1 p^{\prime} b=1 / 2 \vee j u m p F-p o l y L 1 p^{\prime} b=-1 / 2$
using jumpF－polyL－coprime［of $p^{\prime} 1 b$ ，simplified $]\left\langle p^{\prime} \neq 0\right\rangle\left\langle\right.$ poly $\left.p^{\prime} b=0\right\rangle$ by
auto
moreover have poly $p^{\prime}$ max－$r>0 \vee$ poly $p^{\prime}$ max－$r<0$
using max－r－nz by auto
moreover have False when poly $p^{\prime} \max -r>0 \wedge j u m p F-p o l y L 1 p^{\prime} b=-1 / 2$
$\vee$ poly $p^{\prime}$ max－$r<0 \wedge$ jumpF－polyL $1 p^{\prime} b=1 / 2$
proof－
define $f$ where $f=\left(\lambda x\right.$ ． $1 /$ poly $\left.p^{\prime} x\right)$
have noroots：poly $p^{\prime} x \neq 0$ when $x \in\{$ max $-r<. .<b\}$ for $x$
proof（rule ccontr）
assume $\neg$ poly $p^{\prime} x \neq 0$
then have poly $p x=0$ unfolding $p^{\prime}$－def by auto
then have $x \in$ roots unfolding roots－def using that $\langle a<$ max－$r\rangle$ by auto
then have $x \leq$ max－$r$ using Max－ge $[O F$ 〈finite roots〉］unfolding
max－r－def by auto
moreover have $x>$ max－$r$ using that by auto
ultimately show False by auto
qed
have continuous－on $\{$ max－$r<. .<b\} f$
unfolding $f$－def using noroots by（auto intro！：continuous－intros）
moreover have continuous（at－right max－r）$f$
unfolding $f$－def using max－$r$－$n z$
by（auto intro！：continuous－intros）
moreover have $f$ max－$r>0 \wedge j u m p F f($ at－left b）$<0$
$\vee f$ max－$r<0 \wedge$ jump $F f($ at－left b）$>0$
using that unfolding $f$－def jumpF－polyL－def by auto
ultimately have $\exists x>$ max－$r . x<b \wedge f x=0$
apply（intro jumpF－IVT［OF 〈max－r＜b〉］）
by auto
then show False using noroots unfolding $f$－def by auto
qed
moreover have ？thesis when poly $p^{\prime}$ max－$r>0 \wedge j u m p F-p o l y L 1 p^{\prime} b=1 / 2$
$\checkmark$ poly $p^{\prime}$ max－$r<0 \wedge$ jumpF－polyL $1 p^{\prime} b=-1 / 2$
proof－
have poly max－rp a 00 poly max－rp $b>0$
unfolding max－rp－def using $\langle$ odd（order max－r $p$ ）〉 $\langle a<\max -r\rangle\langle\max -r<b\rangle$ by（simp－all add：zero－less－power－eq）

```
            then have cross-alt 1 pab= - cross-alt 1 p' a b
                    unfolding cross-alt-def p'-def using <poly p}\mp@subsup{p}{}{\prime}b=0\mathrm{ 〉
            apply (simp add:sign-times)
            by (auto simp add: Sturm-Tarski.sign-def psign-diff-def zero-less-mult-iff)
            with that show ?thesis by auto
            qed
            ultimately show ?thesis by blast
    qed
    moreover have ?thesis2 when poly p' b\not=0
    proof -
    have [simp]:jumpF-polyL 1 p' b=0
        using jumpF-polyL-coprime[of p' 1 b,simplified] <poly p' b\not=0` by auto
    have [simp]:poly max-rp a<0 poly max-rp b>0
    unfolding max-rp-def using <odd (order max-r p)\rangle\langlea<max-r\rangle\langlemax-r<b\rangle
        by (simp-all add:zero-less-power-eq)
    have poly p' b>0\vee poly p' b<0
        using <poly p' b\not=0> by auto
    moreover have poly p' max-r>0 \vee poly p' max-r<0
        using max-r-nz by auto
    moreover have ?thesis when poly p' b>0 ^ poly p' max-r>0
        using that unfolding cross-alt-def p}\mp@subsup{p}{}{\prime}\mathrm{ -def psign-diff-def
        apply (simp add:sign-times)
        by (simp add: Sturm-Tarski.sign-def)
    moreover have ?thesis when poly p' b<0 ^ poly p' max- }r<
        using that unfolding cross-alt-def p'-def psign-diff-def
        apply (simp add:sign-times)
        by (simp add: Sturm-Tarski.sign-def)
        moreover have False when poly p'b>0 ^ poly p' max-r<0 \vee poly p'
b<0^ poly p' max-r>0
    proof -
    have \existsx>max-r. x<b^ poly p'x=0
                apply (rule poly-IVT[OF<max-r<b>,of p])
                    using that mult-less-0-iff by blast
            then obtain x where max-r<x x<b poly p x=0 unfolding p'-def by
auto
            then have x\inroots using < }a<\mathrm{ max-r> unfolding roots-def by auto
                    then have x\leqmax-r unfolding max-r-def using Max-ge[OF<finite
roots>] by auto
            then show False using <max-r<x\rangle by auto
            qed
            ultimately show ?thesis by blast
            qed
            ultimately have ?thesis2 by auto
            then show ?thesis unfolding max-r-sign-def sjump-p using that by simp
        qed
        ultimately show ?thesis by auto
    qed
    finally show ?thesis.
qed
```

```
    ultimately show ?case by fast
qed
lemma cindex-polyE-inverse-add:
    fixes p q::real poly
    assumes cp:coprime p q
    shows cindex-polyE a b q p + cindex-polyE a b p q=cindex-polyE a b 1 (q*p)
    unfolding cindex-polyE-def
    using cindex-poly-inverse-add[OF cp,symmetric] jumpF-polyR-inverse-add[OF
cp,symmetric]
    jumpF-polyL-inverse-add[OF cp,symmetric]
    by auto
lemma cindex-polyE-inverse-add-cross:
    fixes p q::real poly
    assumes a<b coprime p q
    shows cindex-polyE a b q p + cindex-polyE a b p q= cross-alt p qab / 2
    apply (subst cindex-polyE-inverse-add[OF <coprime p q>])
    apply (subst cindex-polyE-cross[OF <a<b>])
    apply (subst mult.commute)
    apply (subst (2) cross-alt-clear)
    by simp
lemma cindex-polyE-inverse-add-cross':
    fixes p q::real poly
    assumes }a<b\mathrm{ poly p a⿻=0` poly q aキ0 poly p b}=0\vee\mp@code{poly q b}=
    shows cindex-polyE ab q p + cindex-polyE ab pq=cross-alt p qab/2
proof -
    define g1 where g1 = gcd pq
    obtain p' q}\mp@subsup{q}{}{\prime}\mathrm{ where pq:p=g1* 'p}q=g1*\mp@subsup{q}{}{\prime}\mathrm{ and coprime p' }\mp@subsup{q}{}{\prime
    unfolding g1-def
    by (metis assms(2) coprime-commute div-gcd-coprime dvd-mult-div-cancel gcd-dvd1
        gcd-dvd2 order-root)
    have [simp]:g1\not=0
    unfolding g1-def using assms(2) by force
    have cindex-polyE a b q' p' + cindex-polyE a b p' q}\mp@subsup{q}{}{\prime}=(\mathrm{ cross-alt p' q}\mp@subsup{q}{}{\prime}ab)/
    using cindex-polyE-inverse-add-cross[OF \langlea<b\rangle\langlecoprime p' q}\mp@subsup{q}{}{\prime}\rangle]
moreover have cindex-polyE ab p}\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}=\mathrm{ cindex-polyE ab p q
    unfolding pq
    apply (subst cindex-polyE-mult-cancel)
    by simp-all
moreover have cindex-polyE a b q' p'= cindex-polyE a b q p
    unfolding pq
    apply (subst cindex-polyE-mult-cancel)
    by simp-all
moreover have cross-alt p' q' a b cross-alt p q a b
    unfolding pq
```

```
    apply (subst cross-alt-cancel)
    subgoal using assms(2) g1-def poly-gcd-0-iff by blast
    subgoal using assms(3) g1-def poly-gcd-0-iff by blast
    by simp
    ultimately show ?thesis by auto
qed
lemma cindex-polyE-smult-1:
    fixes p q::real poly and c::real
    shows cindex-polyE a b (smult c q) p=(sgn c)* cindex-polyE a b q p
proof -
    have real-of-int (sign c) = sgn c
        by (simp add: sgn-if)
    then show ?thesis
            unfolding cindex-polyE-def jumpF-polyL-smult-1 jumpF-polyR-smult-1 cin-
dex-poly-smult-1
    by (auto simp add: algebra-simps)
qed
lemma cindex-polyE-smult-2:
    fixes p q::real poly and c::real
    shows cindex-polyE a b q (smult c p)=(sgn c) * cindex-polyE a b q p
proof (cases c=0)
    case True
    then show ?thesis by simp
next
    case False
    then have cindex-polyE a b q (smult c p)
                    = cindex-polyE a b ([:1/c:]*q) ([:1/c:]*(smult c p))
    apply (subst cindex-polyE-mult-cancel)
    by simp-all
    also have ... = cindex-polyE a b (smult (1/c)q) p
    by simp
    also have ... = (sgn (1/c))* cindex-polyE a b q p
        using cindex-polyE-smult-1 by simp
    also have ... = (sgn c)* cindex-polyE a b q p
    by simp
    finally show ?thesis.
qed
lemma cindex-polyE-mod:
    fixes p q::real poly
    shows cindex-polyE a b q p = cindex-polyE ab(q\operatorname{mod}p)p
    unfolding cindex-polyE-def
    apply (subst cindex-poly-mod)
    apply (subst jumpF-polyR-mod)
    apply (subst jumpF-polyL-mod)
    by simp
```

```
lemma cindex-polyE-rec:
    fixes p q::real poly
    assumes a<b coprime pq
    shows cindex-polyE a b q p = cross-alt q p ab/2 + cindex-polyE ab (- (p
mod q)) q
proof -
    note cindex-polyE-inverse-add-cross[OF assms]
    moreover have cindex-polyE a b (- (p\operatorname{mod}q)) q= - cindex-polyE abpq
        using cindex-polyE-mod cindex-polyE-smult-1[of a b -1]
        by auto
    ultimately show ?thesis by (auto simp add:field-simps cross-alt-poly-commute)
qed
lemma cindex-polyE-changes-alt-itv-mods:
    assumes a<b coprime p q
    shows cindex-polyE a b q p = changes-alt-itv-smods a b pq/2 using<coprime
p q>
proof (induct smods p q arbitrary:p q)
    case Nil
    then have p=0 by (metis smods-nil-eq)
    then show ?case by (simp add:changes-alt-itv-smods-def changes-alt-poly-at-def)
next
    case (Cons x xs)
    then have p\not=0 by auto
    have ?case when q=0
        using that by (simp add:changes-alt-itv-smods-def changes-alt-poly-at-def)
    moreover have ?case when q}\not=
    proof -
        define r where r\equiv- ( }p\operatorname{mod}q
        obtain ps where ps:smods p q=p#q#ps smods q r=q#ps and xs=q#ps
            unfolding r-def using <q\not=0\rangle\langlep\not=0\rangle\langlex # xs= smods p q\rangle
            by (metis list.inject smods.simps)
    from Cons.prems }\langleq\not=0\rangle\mathrm{ have coprime q r
            by (simp add: r-def ac-simps)
        then have cindex-polyE a b r q = real-of-int (changes-alt-itv-smods a b q r) /
2
            apply (rule-tac Cons.hyps(1))
            using ps <xs=q#ps` by simp-all
    moreover have changes-alt-itv-smods a b p q=cross-alt p qab+changes-alt-itv-smods
abqr
            using changes-alt-itv-smods-rec[OF <a<b\rangle\langlecoprime p q>,folded r-def].
        moreover have cindex-polyE a b q p = real-of-int (cross-alt q p a b)/2 +
cindex-polyE a b r q
            using cindex-polyE-rec[OF «a<b\rangle<coprime p q>,folded r-def].
            ultimately show ?case
            by (auto simp add:field-simps cross-alt-poly-commute)
    qed
    ultimately show ?case by blast
```

qed

```
lemma cindex-poly-ubd-eventually:
    shows \(\forall_{F} r\) in at-top. cindexE \((-r) r(\lambda x\). poly \(q x /\) poly \(p x)=o f\)-int (cindex-poly-ubd
\(q\) p)
proof -
    define \(f\) where \(f=(\lambda x\). poly \(q x /\) poly \(p x)\)
    obtain \(R\) where \(R\)-def: \(R>0\) proots \(p \subseteq\{-R<. .<R\}\)
        if \(p \neq 0\)
    proof (cases \(p=0\) )
    case True
    then show ?thesis using that[of 1] by auto
next
    case False
    then have finite (proots \(p\) ) by auto
    from finite-ball-include[OF this,of 0]
    obtain \(r\) where \(r>0\) and \(r\)-ball:proots \(p \subseteq\) ball \(0 r\)
        by auto
    have proots \(p \subseteq\{-r<. .<r\}\)
    proof
        fix \(x\) assume \(x \in\) proots \(p\)
        then have \(x \in\) ball \(0 r\) using \(r\)-ball by auto
        then have abs \(x<r\) using mem-ball-0 by auto
        then show \(x \in\{-r<. .<r\}\) using \(\langle r\rangle 0\rangle\) by auto
    qed
    then show ?thesis using that [of \(r\) ] False \(\langle r>0\rangle\) by auto
qed
define \(l\) where \(l=(\) if \(p=0\) then 0 else cindex-poly \((-R) R q p)\)
define \(P\) where \(P=\left(\lambda l .\left(\forall_{F} r\right.\right.\) in at-top. cindex \(E(-r) r f=o f\)-int \(\left.\left.l\right)\right)\)
have \(P l\)
proof (cases \(p=0\) )
    case True
    then show?thesis
        unfolding \(P\)-def f-def l-def using True
        by (auto intro!: eventuallyI cindexE-constI)
    next
    case False
    have \(P l\) unfolding \(P\)-def
    proof (rule eventually-at-top-linorderI[of R])
        fix \(r\) assume \(R \leq r\)
        then have cindexE \((-r) r f=\) cindex-poly \(E(-r) r q p\)
        unfolding \(f\)-def using \(R\)-def[OF \(\langle p \neq 0\rangle]\) by (auto intro: cindexE-eq-cindex-polyE)
        also have \(\ldots=o f-i n t(\) cindex-poly \((-r) r q p)\)
        proof -
                have jumpF-polyR q \(p(-r)=0\)
            apply (rule jumpF-poly-noroot)
            using \(\langle R \leq r\rangle R-\operatorname{def}[O F\langle p \neq 0\rangle]\) by auto
            moreover have jumpF-polyL q p \(r=0\)
                apply (rule jumpF-poly-noroot)
```

```
            using <R\leqr> R-def[OF<p\not=0>] by auto
            ultimately show ?thesis unfolding cindex-polyE-def by auto
    qed
    also have ... = of-int (cindex-poly (-R) R q p)
    proof -
```



```
        define Rs where Rs={x. poly px=0^-R<x\wedgex<R}
        have }rs=R
            using R-def[OF <p\not=0\rangle]< R\leqr\rangle unfolding rs-def Rs-def by force
            then show ?thesis
            unfolding cindex-poly-def by (fold rs-def Rs-def,auto)
        qed
        also have ... = of-int l unfolding l-def using False by auto
        finally show cindexE (-r)rf = real-of-int l.
        qed
        then show ?thesis unfolding P-def by auto
    qed
    moreover have }x=l\mathrm{ when }Px\mathrm{ for }
    proof -
    have }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. cindexE (-r)rf= real-of-int x
            \forallF}r\mathrm{ in at-top. cindexE (-r)rf = real-of-int l
        using }\langlePx\rangle\langlePl\rangle\mathrm{ unfolding P-def by auto
    from eventually-conj[OF this]
    have }\mp@subsup{\forall}{F}{}r:\mathrm{ :real in at-top. real-of-int }x=\mathrm{ real-of-int }
        by (elim eventually-mono,auto)
    then have real-of-int x = real-of-int l by auto
    then show?thesis by simp
qed
ultimately have P(THE x. P x) using theI[of P l] by blast
then show ?thesis unfolding P-def f-def cindex-poly-ubd-def by auto
qed
lemma cindex-poly-ubd-0:
    assumes p=0\veeq=0
    shows cindex-poly-ubd q p = 0
proof -
    have }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. cindexE (-r)r ( }\lambda\mathrm{ x. poly q x/poly p x)=0
        apply (rule eventuallyI)
        using assms by (auto intro:cindexE-constI)
    from eventually-conj[OF this cindex-poly-ubd-eventually[of q p]]
    have }\mp@subsup{\forall}{F}{}r\mathrm{ r::real in at-top. (cindex-poly-ubd q p)=(0::int)
        apply (elim eventually-mono)
        by auto
    then show ?thesis by auto
qed
lemma cindex-poly-ubd-code:
    shows cindex-poly-ubd q p = changes-R-smods p q
proof (cases p=0)
```

```
    case True
    then show ?thesis using cindex-poly-ubd-0 by auto
next
    case False
    define ps where ps\equivsmods p q
    have p\inset ps using ps-def \langlep\not=0\rangle by auto
    obtain lb where lb:\forallp\inset ps. }\forallx\mathrm{ . poly p x=0 }\longrightarrowx>l
        and lb-sgn:\forallx\leqlb.}\forallp\in\mathrm{ set ps.sgn (poly p x) = sgn-neg-inf p
        and lb<0
    using root-list-lb[OF no-0-in-smods,of p q,folded ps-def]
    by auto
    obtain ub where ub:\forallp\inset ps. }\forall\mathrm{ x. poly p x=0 }\longrightarrowx<u
        and ub-sgn:\forallx\gequb.\forallp\inset ps.sgn (poly p x)=sgn-pos-inf p
        and ub>0
    using root-list-ub[OF no-O-in-smods,of p q,folded ps-def]
    by auto
    define f}\mathrm{ where f=( }\lambdat\mathrm{ . poly q t/ poly p t)
    define P where P=(\lambdal. (\forallF r in at-top. cindexE (-r)rf =of-int l))
    have P (changes-R-smods p q) unfolding P-def
    proof (rule eventually-at-top-linorderI[of max |lb| |ub| + 1])
    fix r assume r-asm:r\geqmax |lb| |ub| +1
    have cindexE (-r)rf= cindex-polyE (-r) rqp
        unfolding f}f\mathrm{ -def using r-asm by (auto intro: cindexE-eq-cindex-polyE)
    also have ... =of-int (cindex-poly (-r) r q p)
    proof -
            have jumpF-polyR q p (-r)=0
            apply (rule jumpF-poly-noroot)
            using r-asm lb[rule-format,OF<p\inset ps`,of -r] by linarith
            moreover have jumpF-polyL q pr=0
                apply (rule jumpF-poly-noroot)
            using r-asm ub[rule-format,OF <p\inset ps\rangle,of r] by linarith
        ultimately show ?thesis unfolding cindex-polyE-def by auto
    qed
    also have ... = of-int (changes-itv-smods (-r) r p q)
        apply (rule cindex-poly-changes-itv-mods[THEN arg-cong])
            using r-asm lb[rule-format,OF <p\inset ps>,of -r] ub[rule-format,OF}\langlep\inse
ps>,of r]
            by linarith+
    also have ... =of-int (changes-R-smods p q)
    proof -
            have map (sgn ○ ( }\lambda\mathrm{ p. poly p (-r))) ps = map sgn-neg-inf ps
                and map (sgn ○ (\lambdap. poly pr)) ps= map sgn-pos-inf ps
            using lb-sgn[THEN spec,of -r,simplified] ub-sgn[THEN spec,of r,simplified]
r-asm
            by auto
            hence changes-poly-at ps (-r)=changes-poly-neg-inf ps
                ^changes-poly-at ps r=changes-poly-pos-inf ps
            unfolding changes-poly-neg-inf-def changes-poly-at-def changes-poly-pos-inf-def
                by (subst (1 3) changes-map-sgn-eq,metis map-map)
```

```
        thus ?thesis unfolding changes-R-smods-def changes-itv-smods-def ps-def
        by metis
    qed
    finally show cindexE (-r)rf=of-int (changes-R-smods p q).
    qed
    moreover have x= changes-R-smods p q when P f for x
    proof -
    have }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. cindexE (-r)rf=real-of-int (changes-R-smods p q)
            \mp@subsup{\forall}{F}{}r in at-top. cindexE (-r) rf = real-of-int x
        using <P (changes-R-smods p q)\rangle\langleP x\rangle unfolding P-def by auto
    from eventually-conj[OF this]
    have }\mp@subsup{\forall}{F}{}(r::real) in at-top. of-int x =of-int (changes-R-smods p q)
        by (elim eventually-mono,auto)
    then have of-int x =of-int (changes-R-smods p q)
        using eventually-const-iff by auto
    then show ?thesis using of-int-eq-iff by blast
    qed
    ultimately have (THE x. P x) = changes-R-smods p q
    using the-equality[of P changes-R-smods p q] by blast
    then show ?thesis unfolding cindex-poly-ubd-def P-def f-def by auto
qed
```

lemma cindexE-ubd-poly: cindexE-ubd ( $\lambda$ x. poly $q x /$ poly $p x)=$ cindex-poly-ubd $q$
p
proof (cases $p=0$ )
case True
then show ?thesis using cindex-poly-ubd-0 unfolding cindexE-ubd-def
by auto
next
case False
define $m x m n$ where $m x=\operatorname{Max}\{x$. poly $p x=0\}$ and $m n=\operatorname{Min}\{x$. poly $p$
$x=0\}$
define $r r$ where $r r=1+(\max |m x||m n|)$
have $r r:-r r<x \wedge x<r r$ when poly $p x=0$ for $x$
proof -
have finite $\{x$. poly $p x=0\}$ using $\langle p \neq 0\rangle$ poly-roots-finite by blast
then have $m n \leq x x \leq m x$
using Max-ge Min-le that unfolding mn-def mx-def by simp-all
then show ?thesis unfolding rr-def by auto
qed
define $f$ where $f=(\lambda x$. poly $q x /$ poly $p x)$
have $\forall_{F} r$ in at-top. cindexE $(-r) r f=\operatorname{cindexE} E u b d f$
proof (rule eventually-at-top-linorderI $[$ of rr])
fix $r$ assume $r \geq r r$
define $R 1$ R2 where $R 1=\{x$.jumpF $f($ at-right $x) \neq 0 \wedge-r \leq x \wedge x<r\}$
and $R 2=\{x$.jumpF $f($ at-right $x) \neq 0\}$
define L1 L2 where L1 $=\{x$. jumpF $f($ at-left $x) \neq 0 \wedge-r<x \wedge x \leq r\}$
and $L 2=\{x$.jumpF $f($ at-left $x) \neq 0\}$

```
    have \(R 1=R 2\)
    proof -
    have jumpF \(f(\) at-right \(x)=0\) when \(\neg(-r \leq x \wedge x<r)\) for \(x\)
    proof -
            have jumpF \(f(\) at-right \(x)=\) jumpF-poly R q \(p x\)
            unfolding \(f\)-def jumpF-poly \(R\)-def by simp
            also have \(\ldots=0\)
                apply (rule jumpF-poly-noroot)
                using that \(\langle r \geq r r\rangle\) by (auto dest:rr)
            finally show?thesis.
    qed
    then show ?thesis unfolding R1-def R2-def by blast
qed
moreover have \(L 1=L\) 2
proof -
    have jumpF \(f(\) at-left \(x)=0\) when \(\neg(-r<x \wedge x \leq r)\) for \(x\)
    proof -
            have jumpF \(f(\) at-left \(x)=j u m p F-p o l y L q p x\)
            unfolding \(f\)-def jumpF-polyL-def by simp
        also have ... \(=0\)
            apply (rule jumpF-poly-noroot)
            using that \(\langle r \geq r r\rangle\) by (auto dest:rr)
            finally show ?thesis.
    qed
    then show ?thesis unfolding L1-def L2-def by blast
    qed
    ultimately show cindexE \((-r) r f=\operatorname{cindexE-ubd} f\)
    unfolding cindexE-def cindexE-ubd-def
    apply (fold R1-def R2-def L1-def L2-def)
    by auto
qed
moreover have \(\forall_{F} r\) in at-top. cindexE \((-r) r f=\) cindex-poly-ubd q \(p\)
    using cindex-poly-ubd-eventually unfolding \(f\)-def by auto
ultimately have \(\forall_{F} r\) in at-top. cindexE \((-r) r f=\operatorname{cindexE}\)-ubd \(f\)
                        \(\wedge\) cindexE \((-r) r f=\) cindex-poly-ubd \(q p\)
    using eventually-conj by auto
then have \(\forall_{F}(r::\) real \()\) in at-top. cindexE-ubd \(f=\) cindex-poly-ubd q \(p\)
    by (elim eventually-mono) auto
    then show ?thesis unfolding \(f\)-def by auto
qed
lemma cindex-polyE-noroot:
    assumes \(a<b \forall x . a \leq x \wedge x \leq b \longrightarrow\) poly \(p x \neq 0\)
    shows cindex-polyE a b q p=0
proof -
    have jumpF-polyR q p \(a=0\)
        apply (rule jumpF-poly-noroot)
        using assms by auto
    moreover have jumpF-polyL q pb=0
```

```
    apply (rule jumpF-poly-noroot)
    using assms by auto
    moreover have cindex-poly a b q p =0
    apply (rule cindex-poly-noroot)
    using assms by auto
    ultimately show ?thesis unfolding cindex-polyE-def by auto
qed
lemma cindex-polyE-combine:
    assumes a<b b<c
    shows cindex-polyE a b q p+cindex-polyE b cq p = cindex-polyE a c q p
proof -
    define }AB\mathrm{ where }A=\mathrm{ cindex-poly a b q p-jumpF-polyL q p b
            and B=jumpF-polyR q pb+cindex-poly b cqp
    have cindex-polyE a b q p + cindex-polyE b cqp=
                            jumpF-polyR q p a + (A+B) - jumpF-polyL q p c
    unfolding cindex-polyE-def A-def B-def by auto
    also have ... = jumpF-polyR q p a + cindex-poly a c q p - jumpF-polyL q p c
    proof -
    have }A+B=\mathrm{ cindex-poly a b q p +(jumpF-polyR q pb-jumpF-polyL q pb)
                + cindex-poly b c q p
            unfolding A-def B-def by auto
    also have ... = cindex-poly a b q p + real-of-int (jump-poly q p b) + cindex-poly
b c q p
            using jump-poly-jumpF-poly by auto
    also have ... = cindex-poly a c q p
            using assms
            apply (subst (3) cindex-poly-combine[symmetric,of - b])
            by auto
            finally show ?thesis by auto
    qed
    also have ... = cindex-polyE a c q p
    unfolding cindex-polyE-def by simp
    finally show ?thesis.
qed
lemma cindex-polyE-linear-comp:
    fixes b c::real
    defines }h\equiv(\lambdap. pcompose p [:b,c:]
    assumes lb<ub c\not=0
    shows cindex-polyE lb ub (h q) (h p)=
                (if 0<c then cindex-polyE (c*lb +b) (c*ub+b)qp
                else - cindex-polyE (c*ub + b) (c*lb + b)q p)
proof -
    have cindex-polyE lb ub (h q) (h p) = cindexE lb ub (\lambdax. poly (h q) x / poly (h
p) x)
    apply (subst cindexE-eq-cindex-polyE[symmetric,OF <lb<ub>])
    by simp
    also have ... = cindexE lb ub ((\lambdax. poly q x / poly p x) ○ (\lambdax.c* x + b))
```

unfolding comp－def h－def poly－pcompose by（simp add：algebra－simps）
also have $\ldots=($ if $0<c$ then cindex $E(c * l b+b)(c * u b+b)(\lambda x$ ．poly $q x /$ poly $p x$ ）
else $-\operatorname{cindexE}(c * u b+b)(c * l b+b)(\lambda x$. poly $q x / \operatorname{poly} p x))$
apply（subst cindexE－linear－comp［OF〈c申0〉］）
by $\operatorname{simp}$
also have $\ldots=($ if $0<c$ then cindex－polyE $(c * l b+b)(c * u b+b) q p$ else－cindex－polyE $(c * u b+b)(c * l b+b) q p)$
proof－
have cindex $E(c * l b+b)(c * u b+b)(\lambda x$ ．poly $q x / \operatorname{poly} p x)$

$$
=\text { cindex-polyE }(c * l b+b)(c * u b+b) q p \text { if } c>0
$$

apply（subst cindexE－eq－cindex－polyE）
using that $\langle l b<u b\rangle$ by auto
moreover have cindex $E(c * u b+b)(c * l b+b)(\lambda x$ ．poly $q x /$ poly $p x)$
$=$ cindex－poly $E(c * u b+b)(c * l b+b) q p$ if $\neg c>0$
apply（subst cindexE－eq－cindex－polyE）
using that assms by auto
ultimately show ？thesis by auto
qed
finally show ？thesis．
qed
lemma cindex－polyE－product＇：
fixes $p$ r $q$ s：：real poly and $a b$ ：：real
assumes $a<b$ coprime $q p$ coprime s $r$
shows cindex－polyE ab（ $p * r-q * s)(p * s+q * r)$
$=$ cindex－polyE abpq＋cindex－polyE abrs
－cross－alt $(p * s+q * r)(q * s) a b / 2(i s ? L=? R)$
proof（cases $q=0 \vee s=0 \vee p=0 \vee r=0 \vee p * s+q * r=0$ ）
case True
moreover have ？thesis if $q=0$
proof－
have $p \neq 0$
using assms（2）coprime－poly－0 poly－0 that by blast
then show ？thesis using that cindex－polyE－mult－cancel by simp
qed
moreover have ？thesis if $s=0$
proof－
have $r \neq 0$ using assms（3）coprime－poly－0 poly－0 that by blast
then have ？$L=$ cindex－polyE ab $(r * p)(r * q)$
using that by（simp add：algebra－simps）
also have ．．．$=$ ？$R$
using that cindex－polyE－mult－cancel $\langle r \neq 0\rangle$ by simp
finally show ？thesis．
qed
moreover have ？thesis if $p * s+q * r=0 s \neq 0 \quad q \neq 0$
proof－
have cindex－polyE a b p q＝cindex－polyE ab（s＊p）（ $s * q$ ）
using cindex－polyE－mult－cancel $[O F\langle s \neq 0\rangle]$ by simp

```
    also have \(\ldots=\) cindex-polyE ab( \(-(q * r))(q * s)\)
    using that(1)
    by (metis add.inverse-inverse add.inverse-unique mult.commute)
    also have \(\ldots=-\) cindex-polyE ab \((q * r)(q * s)\)
    using cindex-polyE-smult-1[where \(c=-1\), simplified] by simp
    also have \(\ldots=-\) cindex-polyE a brs
    using cindex-polyE-mult-cancel \([O F\langle q \neq 0\rangle]\) by simp
    finally have cindex-poly E abpq=-cindex-polyE abrs.
    then show ?thesis using that(1) by simp
qed
moreover have ?thesis if \(p=0\)
proof -
    have poly \(q\) af0
    using assms(2) coprime-poly-0 order-root that(1) by blast
    have poly \(q b \neq 0\)
    by (metis assms(2) coprime-poly-0 mpoly-base-conv(1) that)
    then have \(q \neq 0\) using poly- 0 by blast
    have ? \(L=-\) cindex-polyE absr
    using that cindex-polyE-smult-1[where \(c=-1\),simplified]
        cindex-polyE-mult-cancel[OF \(\langle q \neq 0\rangle\) ]
    by \(\operatorname{simp}\)
    also have \(\ldots=\) cindex-polyE \(a b r s-(\) cross-alt rsab)/2
    apply (subst cindex-polyE-inverse-add-cross[symmetric])
    using \(\langle a\langle b\rangle\langle c o p r i m e s r\rangle\) by (auto simp:coprime-commute)
    also have \(\ldots=\) ? \(R\)
        using \(\langle p=0\rangle\langle p o l y q a \neq 0\rangle\langle p o l y q b \neq 0\rangle\) cross-alt-cancel
    by simp
    finally show ?thesis .
qed
moreover have ?thesis if \(r=0\)
proof -
    have poly \(s a \neq 0\)
        using assms(3) coprime-poly-0 order-root that by blast
    have poly \(s b \neq 0\)
        using assms(3) coprime-poly-0 order-root that by blast
    then have \(s \neq 0\) using poly- 0 by blast
    have cindex-polyE ab(-(q*s))(p*s)
        \(=-\) cindex-polyE ab(q*s)(p*s)
        using cindex-polyE-smult-1[where \(c=-1\), simplified] by auto
    also have \(\ldots=-\) cindex-polyE ab( \(s * q)(s * p)\)
        by (simp add:algebra-simps)
    also have \(\ldots=-\) cindex-polyE a b q p
        using cindex-polyE-mult-cancel \([O F\langle s \neq 0\rangle]\) by simp
    finally have cindex-polyE ab(-(q*s))(p*s)
        \(=-\) cindex-polyE abqp.
    moreover have cross-alt \((p * s)(q * s) a b / 2\)
        \(=\) cindex-polyE abqp+cindex-polyE abpq
```

```
    proof -
    have cross-alt (p*s) (q*s) ab
                cross-alt (s*p) (s*q) ab
            by (simp add:algebra-simps)
    also have ... = cross-alt p qa b
            using cross-alt-cancel by (simp add: <poly s a }=0\rangle\langlepoly s b\not=0〉
    also have ... / 2 = cindex-polyE abqp+cindex-polyE abpq
            apply (subst cindex-polyE-inverse-add-cross[symmetric])
            using \langlea<b\rangle\langlecoprime q p> coprime-commute by auto
            finally show ?thesis .
qed
    ultimately show ?thesis using that by simp
qed
ultimately show ?thesis by argo
next
case False
define P where P=(p*s+q*r)
define Q where Q =q*s*P
from False have }q\not=0\quads\not=0\quadp\not=0 r\not=0 P\not=0 Q\not=
    unfolding }P\mathrm{ -def Q-def by auto
then have finite:finite (proots-within Q {x.a\leqx ^x\leqb})
    unfolding P-def Q-def
    by (auto intro: finite-proots)
have sign-pos-eq:
        sign-r-pos Q a =(poly Q b>0)
        poly Q a =0 \Longrightarrow poly Q a>0 = (poly Q b>0)
    if }a<b\mathrm{ and noroot: }\forallx.a<x\wedgex\leqb\longrightarrow\mathrm{ poly Q x}=0\mathrm{ for a b Q
proof -
    have sign-r-pos Q a = (sgnx (poly Q) (at-right a)>0)
        unfolding sign-r-pos-sgnx-iff by simp
    also have ... = (sgnx (poly Q) (at-left b)>0)
    proof (rule ccontr)
        assume (0< sgnx (poly Q) (at-right a))
                        \not=(0<\operatorname{sgnx}(\mathrm{ poly Q) (at-left b))}
        then have }\existsx>a.x<b\wedge poly Qx=
            using sgnx-at-left-at-right-IVT[OF - <a<b>] by auto
        then show False using that(2) by auto
    qed
    also have ... = (poly Q b>0)
        apply (subst sgnx-poly-nz)
        using that by auto
    finally show sign-r-pos Q a = (poly Q b>0) .
    show (poly Q a>0) = (poly Q b>0) if poly Q a\not=0
    proof (rule ccontr)
    assume (0<poly Q a)}\not=(0<poly Q b
    then have poly Qa* poly Q b<0
        by (metis «sign-r-pos Q a = (0 < poly Q b)` poly-0 sign-r-pos-rec that)
```

```
    from poly-IVT[OF <a<b> this]
    have }\existsx>a. x<b\wedge poly Q x=0
    then show False using noroot by auto
    qed
qed
define Case where Case=(\lambdaab.cindex-polyE ab (p*r-q*s)P
                        = cindex-polyE a b pq+ cindex-polyE a brs
                        - (cross-alt P (q*s)ab) / 2)
have basic-case:Case a b
    if noroot0:proots-within Q {x. a<x^x<b} ={}
        and noroot-disj:poly Q a\not=0 \vee poly Q b\not=0
        and a<b
    for ab
proof -
    let ?thesis' = \lambdaprqs a. cindex-polyE a b (p*r-q*s) (p*s+q*r)=
                        cindex-polyE a b pq+cindex-polyE a brs-
                        (cross-alt (p*s+q*r)(q*s)ab) / 2
    have base-case:?thesis' p r q s a
        if proots-within (q*s*(p*s+q*r)){x.a<x\wedgex\leqb}={}
            and coprime q p coprime s r
                q\not=0 s\not=0 p\not=0 r\not=0p*s+q*r\not=0
                a<b
        for prqsa
    proof -
    define P where P=(p*s+q*r)
    have noroot1:proots-within (q*s*P) {x.a<x\wedgex\leqb}={}
        using that(1) unfolding P-def .
    have}P\not=0\mathrm{ using }\langlep*s+q*r\not=0\rangle\mathrm{ unfolding P-def by simp
    have cind1:cindex-polyE a b (p*r-q*s)P
                =(if poly Pa=0 then jumpF-polyR ( }p*r-q*s)Pa else 0) 
    proof -
            have cindex-poly a b (p*r-q*s)P=0
                apply (rule cindex-poly-noroot[OF <a<b〉])
                using noroot1 by fastforce
            moreover have jumpF-polyL ( p*r-q*s)Pb=0
                apply (rule jumpF-poly-noroot)
                using noroot1 <a<b> by auto
            ultimately show ?thesis
                unfolding cindex-polyE-def by (simp add: jumpF-poly-noroot(2))
    qed
    have cind2:cindex-polyE a b p q
                =(if poly q a = 0 then jumpF-polyR p q a else 0)
    proof -
            have cindex-poly a b p q=0
                apply (rule cindex-poly-noroot)
                using noroot1 <a<b\rangle by auto fastforce
```

```
    moreover have jumpF-polyL p q b=0
    apply (rule jumpF-poly-noroot)
    using noroot1 \(\langle a<b\rangle\) by auto
    ultimately show ?thesis
    unfolding cindex-polyE-def
    by (simp add: jumpF-poly-noroot(2))
qed
have cind3:cindex-polyE abrs
    \(=(\) if poly s \(a=0\) then jumpF-polyR r s a else 0\()\)
proof -
    have cindex-poly a br s=0
    apply (rule cindex-poly-noroot)
    using noroot1 \(\langle a<b\rangle\) by auto fastforce
    moreover have jumpF-polyL rs \(b=0\)
    apply (rule jumpF-poly-noroot)
    using noroot1 \(\langle a<b\rangle\) by auto
    ultimately show ?thesis
        unfolding cindex-polyE-def
    by (simp add: jumpF-poly-noroot(2))
qed
have ?thesis if poly \((q * s * P) a \neq 0\)
proof -
    have noroot2:proots-within \((q * s * P)\{x . a \leq x \wedge x \leq b\}=\{ \}\)
    using that noroot1 by force
    have cindex-polyE a \(b(p * r-q * s) P=0\)
    apply (rule cindex-polyE-noroot)
    using noroot2 \(\langle a<b\rangle\) by auto
    moreover have cindex-polyE abpq=0
    apply (rule cindex-polyE-noroot)
    using noroot2 \(\langle a<b\rangle\) by auto
    moreover have cindex-polyE abrs=0
    apply (rule cindex-polyE-noroot)
    using noroot2 \(\langle a<b\rangle\) by auto
    moreover have cross-alt \(P(q * s) a b=0\)
    apply (rule cross-alt-noroot \([\) OF \(\langle a<b\rangle]\) )
    using noroot2 by auto
    ultimately show ?thesis unfolding \(P\)-def by auto
qed
moreover have ?thesis if poly \((q * s * P) a=0\)
proof -
    have ?thesis if poly \(q a=0\) poly s \(a \neq 0\)
    proof -
    have poly \(P a \neq 0\)
            using that coprime-poly- \(0[O F 〈\) coprime \(q\) p〉] unfolding \(P\)-def
            by \(\operatorname{simp}\)
            then have cindex-polyE ab(p*r-q*s) P=0
            using cind1 by auto
            moreover have cindex-polyE abpq=(cross-alt \(P(q * s) a b) / 2\)
```

```
proof -
    have cindex-polyE a b p q=jumpF-polyR p qa
        using cind2 that(1) by auto
    also have ... =(cross-alt 1 (q*s*P)ab)/2
    proof -
        have sign-eq:(sign-r-pos q a \longleftrightarrow poly p a>0)
                =(poly (q*s*P)b>0)
        proof -
            have (sign-r-pos q a \longleftrightarrow poly p a>0)
                = (sgnx (poly (q*p)) (at-right a)>0)
            proof -
            have (poly p a>0) =(sgnx (poly p)(at-right a)>0)
                apply (subst sgnx-poly-nz)
                using <coprime q p coprime-poly-0 that(1) by auto
            then show ?thesis
                unfolding sign-r-pos-sgnx-iff
                apply (subst sgnx-poly-times[of - a])
                subgoal by simp
                using poly-sgnx-values }\langlep\not=0\rangle\langleq\not=0
                by (metis (no-types, opaque-lifting) add.inverse-inverse
                    mult.right-neutral mult-minus-right zero-less-one)
            qed
            also have ... = (sgnx (poly ((q*p)*s`2)) (at-right a)>0)
            proof (subst (2) sgnx-poly-times)
            have sgnx (poly (s}\mp@subsup{s}{}{2}))(\mathrm{ at-right a)>0
                using sgn-zero-iff sgnx-poly-nz(2) that(2) by auto
            then show (0< sgnx (poly (q* p)) (at-right a)) =
                    (0< sgnx (poly (q* p)) (at-right a)
                    * sgnx (poly (s}\mp@subsup{s}{}{2}))(\mathrm{ at-right a))
                    by (simp add: zero-less-mult-iff)
            qed auto
            also have ... = (sgnx (poly (q*s)) (at-right a)
                * sgnx (poly (p*s)) (at-right a)>0)
            unfolding power2-eq-square
            apply (subst sgnx-poly-times[where x=a],simp)+
            by (simp add:algebra-simps)
            also have ... = (sgnx (poly (q*s))(at-right a)
                * sgnx (poly P) (at-right a)> 0)
            proof -
            have sgnx (poly P) (at-right a) =
                    sgnx (poly (q*r+p*s)) (at-right a)
                unfolding P-def by (simp add:algebra-simps)
            also have ... = sgnx (poly (p*s))(at-right a)
                apply (rule sgnx-poly-plus[where x=a])
                subgoal using <poly q a=0` by simp
                    subgoal using <coprime q p` coprime-poly-0 poly-mult-zero-iff
                    that(1) that(2) by blast
                by simp
            finally show ?thesis by auto
```

```
    qed
    also have \(\ldots=(0<\operatorname{sgnx}(\) poly \((q * s * P))(\) at-right a \())\)
    apply (subst sgnx-poly-times \([\) where \(x=a]\), simp \()+\)
    by (simp add:algebra-simps)
    also have \(\ldots=(0<\operatorname{sgnx}(\operatorname{poly}(q * s * P))(\) at-left b) \()\)
    proof -
    have sgnx \((\) poly \((q * s * P))(\) at-right a)
            \(=\operatorname{sgnx}(\) poly \((q * s * P))(\) at-left \(b)\)
    proof (rule ccontr)
        assume sgnx (poly \((q * s * P))(\) at-right a)
                \(\neq \operatorname{sgnx}(\) poly \((q * s * P))(\) at-left b)
            from sgnx-at-left-at-right-IVT[OF this \(\langle a<b\rangle]\)
            have \(\exists x>a . x<b \wedge \operatorname{poly}(q * s * P) x=0\).
            then show False using noroot1 by fastforce
    qed
    then show ?thesis by auto
    qed
    also have \(\ldots=(\) poly \((q * s * P) b>0)\)
    apply (subst sgnx-poly-nz)
    using noroot \(1\langle a<b\rangle\) by auto
    finally show ?thesis.
qed
have psign-a:psign-diff \(1(q * s * P) a=1\)
    unfolding psign-diff-def using <poly \((q * s * P) a=0\) 〉
    by \(\operatorname{simp}\)
have poly \((q * s * P) b \neq 0\)
    using noroot1 \(\langle a<b\rangle\) by blast
moreover have ?thesis if poly \((q * s * P) b>0\)
proof -
    have psign-diff \(1(q * s * P) b=0\)
        using that unfolding psign-diff-def by auto
    moreover have jumpF-polyR p qaa=1/2
        unfolding jumpF-polyR-coprime[OF <coprime q \(p\) 〉]
        using \(\langle p \neq 0\rangle\langle\) poly \(q a=0\rangle\langle q \neq 0\rangle\) sign-eq that by presburger
    ultimately show ?thesis
        unfolding cross-alt-def using psign- \(a\) by auto
qed
moreover have ?thesis if poly \((q * s * P) b<0\)
proof -
    have psign-diff \(1(q * s * P) b=2\)
        using that unfolding psign-diff-def by auto
    moreover have jumpF-polyR p q a = 1/2
        unfolding jumpF-polyR-coprime[OF 〈coprime q \(p\) 〉]
        using \(\langle p \neq 0\rangle\langle p o l y q a=0\rangle\langle q \neq 0\rangle\) sign-eq that by auto
    ultimately show ?thesis
        unfolding cross-alt-def using psign- \(a\) by auto
qed
ultimately show ?thesis by argo
```

```
    qed
    also have ... =(cross-alt P(q*s)ab) / 2
    apply (subst cross-alt-clear[symmetric])
    using <poly P a\not= 0〉 noroot1 <a<b〉 cross-alt-poly-commute
    by auto
    finally show ?thesis.
qed
moreover have cindex-polyE a brs=0
    using cind3 that by auto
    ultimately show ?thesis using that
    apply (fold P-def)
    by auto
qed
moreover have ?thesis if poly q a\not=0 poly s a=0
proof -
    have poly P a\not=0
        using that coprime-poly- O[OF<coprime s r>] unfolding P-def
    by simp
then have cindex-polyE ab(p*r-q*s)P=0
    using cind1 by auto
    moreover have cindex-polyE a brs=(cross-alt P (q*s)ab) / 2
    proof
    have cindex-polyE a b r s = jumpF-polyR r s a
        using cind3 that by auto
    also have ... =(cross-alt 1 (s*q*P)ab)/2
    proof -
        have sign-eq:(sign-r-pos s a \longleftrightarrow poly r a>0)
                        =(poly (s*q*P)b>0)
        proof -
        have (sign-r-pos s a u poly r a>0)
                        = (sgnx (poly (s*r)) (at-right a)>0)
            proof -
            have (poly r a>0) =(sgnx (poly r) (at-right a)>0)
                apply (subst sgnx-poly-nz)
                using <coprime s r` coprime-poly-0 that(2) by auto
            then show ?thesis
                unfolding sign-r-pos-sgnx-iff
                apply (subst sgnx-poly-times[of - a])
                subgoal by simp
                subgoal using <r\not=0\rangle\langles\not=0\rangle
                    by (metis (no-types, opaque-lifting) add.inverse-inverse
                        mult.right-neutral mult-minus-right poly-sgnx-values(2)
                            zero-less-one)
                done
            qed
            also have ... =(sgnx (poly ((s*r)*q^2)) (at-right a)>0)
            proof (subst (2) sgnx-poly-times)
                have sgnx (poly ( }\mp@subsup{q}{}{2}))(\mathrm{ at-right a) >0
        by (metis }<q\not=0`\mathrm{ power2-eq-square sign-r-pos-mult sign-r-pos-sgnx-iff)
```

```
    then show \((0<\operatorname{sgnx}(\) poly \((s * r))(\) at-right \(a))=\)
        \((0<\operatorname{sgnx}(\) poly \((s * r))(\) at-right \(a)\)
        * sgnx \(\left(\right.\) poly \(\left.\left(q^{2}\right)\right)(\) at-right a) \()\)
        by (simp add: zero-less-mult-iff)
    qed auto
    also have \(\ldots=(\operatorname{sgnx}(\) poly \((s * q))(\) at-right a)
        * \(\operatorname{sgnx}(\) poly \((r * q))(\) at-right a) \(>0)\)
    unfolding power2-eq-square
    apply (subst sgnx-poly-times \([\) where \(x=a]\),simp \()+\)
    by (simp add:algebra-simps)
also have \(\ldots=(\operatorname{sgnx}(\) poly \((s * q))(\) at-right \(a)\)
        * sgnx \((\) poly \(P)(\) at-right \(a)>0)\)
    proof -
    have sgnx \((\) poly \(P)(\) at-right a \()=\)
        \(\operatorname{sgnx}(p o l y(p * s+q * r))(\) at-right \(a)\)
        unfolding \(P\)-def by (simp add:algebra-simps)
    also have \(\ldots=\operatorname{sgnx}(\) poly \((q * r))(\) at-right a)
        apply (rule sgnx-poly-plus[where \(x=a]\) )
        subgoal using <poly s \(a=0\) 〉 by simp
        subgoal
            using 〈coprime s r〉 coprime-poly-0 poly-mult-zero-iff that(1)
                that(2) by blast
        by \(\operatorname{simp}\)
    finally show ?thesis by (auto simp:algebra-simps)
qed
also have \(\ldots=(0<\operatorname{sgnx}(\) poly \((s * q * P))(\) at-right a \()\) )
    apply (subst sgnx-poly-times \([\) where \(x=a]\),simp) +
    by (simp add:algebra-simps)
also have \(\ldots=(0<\operatorname{sgnx}(\) poly \((s * q * P))(\) at-left b) \()\)
proof -
    have sgnx (poly \((s * q * P))\) (at-right a)
        \(=\operatorname{sgnx}(\) poly \((s * q * P))(\) at-left b)
    proof (rule ccontr)
        assume sgnx (poly \((s * q * P))(\) at-right a)
                \(\neq \operatorname{sgnx}(\) poly \((s * q * P))(\) at-left \(b)\)
            from sgnx-at-left-at-right-IVT[OF this \(\langle a<b\rangle]\)
            have \(\exists x>a . x<b \wedge \operatorname{poly}(s * q * P) x=0\).
            then show False using noroot1 by fastforce
    qed
    then show ?thesis by auto
qed
also have \(\ldots=(\) poly \((s * q * P) b>0)\)
    apply (subst sgnx-poly-nz)
    using noroot1 \(\langle a<b\rangle\) by auto
    finally show ?thesis .
qed
have psign-a:psign-diff \(1(s * q * P) a=1\)
    unfolding psign-diff-def using 〈poly \((q * s * P) a=0\) 〉
    by (simp add:algebra-simps)
```

```
    have poly (s*q*P)b\not=0
    using noroot1 }\langlea<b\rangle\mathrm{ by (auto simp:algebra-simps)
    moreover have ?thesis if poly }(s*q*P)b>
    proof -
    have psign-diff 1(s*q*P)b=0
        using that unfolding psign-diff-def by auto
    moreover have jumpF-polyR rs a=1/2
        unfolding jumpF-polyR-coprime[OF <coprime s r>]
        using〈poly s a = 0\rangle\langler\not=0\rangle\langles\not=0\rangle sign-eq that by presburger
    ultimately show ?thesis
        unfolding cross-alt-def using psign-a by auto
    qed
    moreover have ?thesis if poly (s*q*P)b<0
    proof -
        have psign-diff 1 (s*q*P)b=2
        using that unfolding psign-diff-def by auto
        moreover have jumpF-polyR r s a = - 1/2
        unfolding jumpF-polyR-coprime[OF <coprime s r>]
        using <poly s a = 0\rangle\langler\not=0\rangle sign-eq that by auto
        ultimately show ?thesis
        unfolding cross-alt-def using psign-a by auto
    qed
    ultimately show ?thesis by argo
    qed
    also have ... =(cross-alt P (q*s)ab) / 2
        apply (subst cross-alt-clear[symmetric])
        using <poly P a\not= 0〉 noroot1 <a<b〉 cross-alt-poly-commute
        by (auto simp:algebra-simps)
    finally show ?thesis.
qed
moreover have cindex-polyE a b p q=0
    using cind2 that by auto
    ultimately show ?thesis using that
    apply (fold P-def)
    by auto
qed
moreover have ?thesis if poly P a=0 poly q a\not=0 poly s a\not=0
proof -
    have cindex-polyE ab(p*r-q*s)P
        = jumpF-polyR(p*r-q*s)Pa
    using cind1 that by auto
    also have ... = (if sign-r-pos Pa=(0<poly (p*r-q*s)a)
        then 1 / 2 else - 1 / 2) (is - = ?R)
    proof (subst jumpF-polyR-coprime')
        let ?C=(P\not=0^p*r-q*s\not=0^ poly Pa=0)
        have ?C
        by (smt (z3) P-def \langleP\not=0\rangle add.left-neutral diff-add-cancel
                        poly-add poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec that(1)
```

```
that(2) that(3))
            then show (if ?C then ?R else 0)}=
    show poly Pa\not=0\vee poly (p*r-q*s) a\not=0
        by (smt (z3) P-def mult-less-0-iff poly-add poly-diff poly-mult
                poly-mult-zero-iff that(2) that(3))
    qed
    also have ... = - cross-alt P (q*s)ab / 2
    proof -
    have (sign-r-pos P a = (0<poly (p*r-q*s)a))
                =(\neg(poly (q*s*P)b>0))
    proof -
        have (poly (q*s*P)b>0)
                =(sgnx (poly (q*s*P))(at-left b)>0)
            apply (subst sgnx-poly-nz)
            using noroot1 < }a<b\rangle\mathrm{ by auto
    also have ... =(sgnx (poly (q*s*P))(at-right a)>0)
    proof (rule ccontr)
            define F where F=(q*s*P)
            assume (0< sgnx (poly F) (at-left b))
                    \not=(0< sgnx (poly F) (at-right a))
            then have sgnx (poly F) (at-right a) = sgnx (poly F) (at-left b)
            by auto
            then have }\existsx>a.x<b\wedge poly Fx=
                using sgnx-at-left-at-right-IVT[OF - <a<b\rangle] by auto
            then show False using noroot1[folded F-def]\a<b\rangle by fastforce
    qed
    also have ... = sign-r-pos (q*s*P)a
            using sign-r-pos-sgnx-iff by simp
    also have ... = (sign-r-pos Pa=sign-r-pos (q*s)a)
            apply (subst sign-r-pos-mult[symmetric])
            using \langleP\not=0\rangle\langleq\not=0\rangle\langles\not=0\rangle by (auto simp add:algebra-simps)
    also have ... = (sign-r-pos Pa=(0\geqpoly (p*r-q*s)a))
    proof -
            have sign-r-pos (q*s) a=(poly (q*s)a>0)
            by (metis poly-0 poly-mult-zero-iff sign-r-pos-rec
                that(2) that(3))
            also have ... = (0\geq poly ( p*r-q*s)a)
                using <poly P a =0` unfolding P-def
                by (smt (verit, ccfv-threshold) \langlep\not=0\rangle\langleq\not=0\rangle\langler\not=0\rangle\langles\not=0\rangle
divisors-zero
                poly-add poly-diff poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec
that(2)
                that(3))
            finally show ?thesis by simp
    qed
    finally have (0<poly (q*s*P)b)
        =(sign-r-pos Pa=(poly (p*r-q*s)a\leq0)).
    then show ?thesis by argo
    qed
```

```
    moreover have cross-alt P(q*s) ab=
        (if poly (q*s*P)b>0 then 1 else -1)
    proof -
    have psign-diff P(q*s)a=1
            by (smt (verit, ccfv-threshold) Sturm-Tarski.sign-def
                dvd-div-mult-self gcd-dvd1 gcd-dvd2 poly-mult-zero-iff
                psign-diff-def that(1) that(2) that(3))
    moreover have psign-diff P(q*s)b
                =(if poly (q*s*P)b>0 then 0 else 2)
    proof -
            define F where F=q*s*P
            have psign-diff P(q*s)b=psign-diff 1Fb
                apply (subst psign-diff-clear)
            using noroot1 «a<b\rangle unfolding F-def
            by (auto simp:algebra-simps)
            also have ... = (if 0<poly F b then 0 else 2)
            proof -
                have poly F b\not=0
                unfolding F-def using «a<b> noroot1 by auto
                then show ?thesis
                unfolding psign-diff-def by auto
            qed
            finally show ?thesis unfolding F-def.
        qed
        ultimately show ?thesis unfolding cross-alt-def by auto
    qed
    ultimately show ?thesis by auto
qed
finally have cindex-polyE a b (p*r-q*s)P
                    = - cross-alt P (q*s)ab/2 .
    moreover have cindex-polyE a b p q=0
        using cind2 that by auto
    moreover have cindex-polyE a brs=0
        using cind3 that by auto
    ultimately show?thesis
        by (fold P-def) auto
qed
moreover have ?thesis if poly q a=0 polys a=0
proof -
    have poly p a\not=0
        using «coprime q p` coprime-poly-0 that(1) by blast
    have poly r a\not=0
        using \coprime s r` coprime-poly-0 that(2) by blast
    have poly P a=0
        unfolding P-def using that by simp
    define ff where ff=(\lambdax. if x then 1/(2::real) else -1/2)
    define C1 C2 C3 C4 C5 where C1 =(sign-r-pos Pa)
        and C2 =(0< poly p a)
```

```
    and C3=(0 < poly r a)
    and C4}=(\mathrm{ sign-r-pos q a)
    and C5 =(sign-r-pos s a)
note CC-def = C1-def C2-def C3-def C4-def C5-def
have cindex-polyE a b (p*r-q*s) P=ff((C1 = C2) = C3)
proof -
    have cindex-polyE a b (p*r-q*s)P
                = jumpF-polyR (p*r-q*s)Pa
    using cind1 <poly P a=0〉 by auto
    also have ... = (ff (sign-r-pos P a
        =(0<poly (p*r-q*s)a)))
    unfolding ff-def
    apply (subst jumpF-polyR-coprime')
    subgoal
        by (simp add: <poly p a \not=0`<poly r a \not=0`that(1))
    subgoal
        by (smt (z3) <P\not=0\rangle\langlepoly Pa=0\rangle
            poly P a f=0\vee poly (p*r-q*s) a\not=0`poly-0)
    done
    also have ... =(ff (sign-r-pos P a = (0<poly (p*r)a)))
    proof -
    have (0<poly (p*r-q*s)a)=(0<poly (p*r)a)
            by (simp add: that(1))
        then show?thesis by simp
    qed
    also have ... = ff ((C1 = C2) = C3)
        unfolding CC-def
            by (smt (z3)<p\not=0\rangle\langlepoly p a \not=0\rangle\langlepoly r a\not=0\rangle\langler\not=0\rangle
            poly-mult-zero-iff sign-r-pos-mult sign-r-pos-rec)
    finally show ?thesis .
qed
moreover have cindex-polyE a b p q
    =ff (C4 = C2)
proof -
    have cindex-polyE a b p q= jumpF-polyR p q a
        using cind2 <poly q a=0〉 by auto
    also have ... = ff (sign-r-pos q a = (0<poly p a))
        apply (subst jumpF-polyR-coprime')
        subgoal using <poly pa\not=0` by auto
        subgoal using }<p\not=0\rangle\langleq\not=0\rangleff-def that(1) by presburger
        done
    also have ... = ff (C4 = C2)
        using <a<b> noroot1 unfolding CC-def by auto
    finally show ?thesis.
qed
moreover have cindex-polyE a b r s = ff (C5 = C3)
proof -
```

no-zero-divisors

```
have cindex-polyE a brs=jumpF-polyR rsa
using cind3 〈poly s \(a=0\) 〉 by auto
also have \(\ldots=f f(\) sign-r-pos s \(a=(0<p o l y r a))\)
    apply (subst jumpF-polyR-coprime')
    subgoal using <poly \(r a \neq 0\) 〉 by auto
    subgoal using \(\langle r \neq 0\rangle\langle s \neq 0\rangle\) ff-def that(2) by presburger
    done
also have \(\ldots=f f(C 5=C 3)\)
    using \(\langle a<b\rangle\) noroot1 unfolding \(C C\)-def by auto
    finally show ?thesis.
qed
moreover have cross-alt \(P(q * s) a b=2 * \int f((C 1=C 4)=C 5)\)
proof -
    have cross-alt \(P(q * s) a b\)
        \(=\operatorname{sign}(\) poly \(P b *(\) poly \(q b *\) poly s \(b))\)
    apply (subst cross-alt-clear)
    apply (subst cross-alt-alt)
    using that by auto
    also have \(\ldots=2 * f f((C 1=C 4)=C 5)\)
    proof -
        have sign-r-pos \(P a=(\) poly \(P b>0)\)
            apply (rule sign-pos-eq)
            using \(\langle a<b\rangle\) noroot1 by auto
    moreover have sign-r-pos \(q a=(\) poly \(q b>0)\)
            apply (rule sign-pos-eq)
            using \(\langle a<b\rangle\) noroot1 by auto
    moreover have sign-r-pos s \(a=(\) poly s \(b>0)\)
            apply (rule sign-pos-eq)
            using \(\langle a<b\rangle\) noroot1 by auto
    ultimately show ?thesis
            unfolding \(C C\)-def ff-def
            apply (simp add:sign-times)
            using noroot1 \(\langle a<b\rangle\) by (auto simp:sign-def)
qed
    finally show? thesis .
qed
ultimately have ?thesis \(=(f f((C 1=C 2)=C 3)=f f(C 4=C 2)+\)
                        ff \((C 5=C 3)-f f((C 1=C 4)=C 5))\)
    by (fold P-def) auto
moreover have \(f f((C 1=C 2)=C 3)=f f(C 4=C 2)+\)
                    \(f f(C 5=C 3)-f f((C 1=C 4)=C 5)\)
proof -
    have \(p p:(0<p o l y p a)=\) sign-r-pos \(p a\)
        apply (subst sign-r-pos-rec)
        using 〈poly \(p a \neq 0\) 〉 by auto
    have \(r r:(0<p o l y r a)=\) sign-r-pos \(r\) a
            apply (subst sign-r-pos-rec)
        using <poly \(r a \neq 0\) 〉 by auto
```

```
    have \(C 1\) if \(C 2=C 5 C 3=C 4\)
    proof -
    have sign-r-pos \((p * s) a\)
        apply (subst sign-r-pos-mult)
        using \(p p\langle C 2=C 5\rangle\langle p \neq 0\rangle\langle s \neq 0\rangle\) unfolding \(C C\)-def by auto
    moreover have sign-r-pos \((q * r) a\)
        apply (subst sign-r-pos-mult)
        using \(r r\langle C 3=C 4\rangle\langle q \neq 0\rangle\langle r \neq 0\rangle\) unfolding \(C C\)-def by auto
        ultimately show ?thesis unfolding \(C C\)-def \(P\)-def
            using sign-r-pos-plus-imp by auto
    qed
    moreover have foo \(2: \neg C 1\) if \(C 2 \neq C 5 C 3 \neq C 4\)
    proof -
        have \((0<\) poly \(p a)=\operatorname{sign-r-pos}(-s) a\)
        apply (subst sign-r-pos-minus)
        using \(\langle s \neq 0\rangle\langle C 2 \neq C 5\rangle\) unfolding \(C C\)-def by auto
        then have sign-r-pos \((p *(-s)) a\)
            apply (subst sign-r-pos-mult)
            unfolding \(p p\) using \(\langle p \neq 0\rangle\langle s \neq 0\rangle\) by auto
    moreover have \((0<p o l y r a)=\operatorname{sign-r-pos}(-q) a\)
        apply (subst sign-r-pos-minus)
        using \(\langle q \neq 0\rangle\langle C 3 \neq C 4\rangle\) unfolding \(C C\)-def by auto
    then have sign-r-pos \((r *(-q)) a\)
        apply (subst sign-r-pos-mult)
        unfolding \(r r\) using \(\langle r \neq 0\rangle\langle q \neq 0\rangle\) by auto
    ultimately have sign-r-pos \((p *(-s)+r *(-q)) a\)
        using sign-r-pos-plus-imp by blast
    then have sign-r-pos \((-(p * s+q * r)) a\)
        by (simp add:algebra-simps)
    then have \(\neg\) sign-r-pos \(P a\)
        apply (subst sign-r-pos-minus)
        using \(\langle P \neq 0\rangle\) unfolding \(P\)-def by auto
        then show? ?thesis unfolding \(C C\)-def .
    qed
    ultimately show ?thesis unfolding ff-def by auto
qed
ultimately show ?thesis by simp
qed
ultimately show ?thesis using that by auto
qed
    ultimately show ?thesis by auto
qed
have ?thesis' prqsa if poly \(Q b \neq 0\)
    apply (rule base-case[OF - <coprime \(q\) p \(\rangle\langle\) coprime s \(r\rangle\) )
    subgoal using noroot0 that unfolding \(Q\)-def \(P\)-def by fastforce
    using False \(\langle a<b\rangle\) by auto
moreover have ?thesis' prqsa if poly \(Q b=0\)
proof -
```


## have poly $Q$ a $=0$ using noroot－disj that by auto

define $h$ where $h=\left(\lambda p . p \circ_{p}[: a+b,-1:]\right)$
have $h$－rw：
$h p-h q=h(p-q)$
$h p * h q=h(p * q)$
$h p+h q=h(p+q)$
cindex－polyE ab（hq）（hp）＝－cindex－polyE abqp
cross－alt $(h p)(h q) a b=$ cross－alt $p q b a$
for $p q$
unfolding $h$－def pcompose－diff pcompose－mult pcompose－add
cindex－polyE－linear－comp $[O F\langle a<b\rangle$ ，of $-1-a+b$ ，simplified $]$
cross－alt－linear－comp［of pa＋b－1 q a b，simplified］
by simp－all
have ？thesis＇$(h p)(h r)(h q)(h s) a$
proof（rule base－case）
have proots－within $(h q * h s *(h p * h s+h q * h r))\{x . a<x \wedge x \leq b\}$ $=$ proots－within $(h Q)\{x . a<x \wedge x \leq b\}$
unfolding $Q$－def $P$－def $h$－def
by（simp add：pcompose－diff pcompose－mult pcompose－add）
also have ．．．$=\{ \}$
unfolding proots－within－def $h$－def poly－pcompose
using $\langle a<b\rangle$ that $[$ folded $Q$－def］noroot0［unfolded P－def，folded $Q$－def］＜poly $Q a \neq 0$ 〉
by（auto simp：order．order－iff－strict proots－within－def）
finally show proots－within $(h q * h s *(h p * h s+h q * h r))$

$$
\{x . a<x \wedge x \leq b\}=\{ \}
$$

show coprime（ $h$ q）（ $h$ p）unfolding $h$－def
apply（rule coprime－linear－comp）
using 〈coprime $q p$ 〉 by auto
show coprime $\left(\begin{array}{l}h\end{array}\right)(h r)$ unfolding $h$－def
apply（rule coprime－linear－comp）
using 〈coprime s $r$ 〉 by auto
show $h q \neq 0 h s \neq 0 \quad h p \neq 0 h r \neq 0$
using False unfolding $h$－def
by（subst pcompose－eq－0；auto）＋
have $h(p * s+q * r) \neq 0$
using False unfolding $h$－def
by（subst pcompose－eq－ $0 ;$ auto）
then show $h p * h s+h q * h r \neq 0$
unfolding $h$－def pcompose－mult pcompose－add by simp
show $a<b$ by fact
qed
moreover have cross－alt $(p * s+q * r)(q * s) b a$ $=-$ cross－alt $(p * s+q * r)(q * s) a b$
unfolding cross－alt－def by auto
ultimately show ？thesis unfolding $h-r w$ by auto qed
ultimately show ?thesis unfolding Case-def P-def by blast qed
show ?thesis using $\langle a<b\rangle$
proof (induct card (proots-within $(q * s * P)\{x . a<x \wedge x \leq b\})$ arbitrary:a)
case 0
have Case a b
proof (rule basic-case)
have $*:$ proots-within $Q\{x . a<x \wedge x \leq b\}=\{ \}$ using $0\langle Q \neq 0\rangle$ unfolding $Q$-def by auto
then show proots-within $Q\{x . a<x \wedge x<b\}=\{ \}$ by force
show poly $Q a \neq 0 \vee$ poly $Q b \neq 0$
using $*\langle a<b\rangle$ by blast
show $a<b$ by fact
qed
then show ?case unfolding Case-def $P$-def by simp
next
case (Suc n)
define $S$ where $S=(\lambda a$. proots-within $Q\{x . a<x \wedge x \leq b\})$
have $S a-S u c: S u c ~ n=\operatorname{card}\left(\begin{array}{ll}S & a\end{array}\right)$
using $S u c(2)$ unfolding $S$-def $Q$-def by auto
define mroot where mroot $=\operatorname{Min}(S a)$
have fin-S:finite ( $S$ a) for $a$
using $\operatorname{Suc}(2)$ unfolding $S$-def $Q$-def
by ( simp add: $\langle P \neq 0\rangle\langle q \neq 0\rangle\langle s \neq 0\rangle$ )
have mroot-in:mroot $\in S$ a and mroot-min $: \forall x \in S$ a. mroot $\leq x$
proof -
have $S a \neq\{ \}$
unfolding $S$-def $Q$-def using Suc.hyps(2) by force
then show mroot $\in S$ a unfolding mroot-def
using Min-in fin-S by auto
show $\forall x \in S$ a. mroot $\leq x$
using 〈finite $\left(\begin{array}{ll}S & a)\rangle \text { Min-le unfolding mroot-def by auto }\end{array}\right.$
qed
have mroot-nzero:poly $Q x \neq 0$ if $a<x x<m r o o t$ for $x$
using mroot-in mroot-min that unfolding $S$-def
by (metis (no-types, lifting) dual-order.strict-trans leD
le-less-linear mem-Collect-eq proots-within-iff )
define $C 1$ where $C 1=(\lambda a b$. cindex-polyE $a b(p * r-q * s) P)$
define C2 where $C 2=(\lambda a b$. cindex-polyE abr $\quad$ )
define $C 3$ where $C 3=\left(\begin{array}{ll}\lambda a b & b \text {. cindex-polyE a }\end{array}\right.$ brrs)
define $C 4$ where $C 4=(\lambda a b$. cross-alt $P(q * s) a b)$
note $C C$-def $=$ C1-def C2-def C3-def C4-def
have hyps:C1 mroot $b=C 2$ mroot $b+C 3$ mroot $b-C 4$ mroot $b / 2$

```
    if mroot < b
    unfolding C1-def C2-def C3-def C4-def P-def
    proof (rule Suc.hyps(1)[OF - that])
    have Suc n = card (S a) using Sa-Suc by auto
    also have .. = card (insert mroot (S mroot))
    proof -
        have Sa= proots-within Q {x.a<x^x\leqb}
        unfolding S-def Q-def by simp
    also have ... = proots-within Q ({x.a<x\wedge x\leqmroot }}\cup{x.mroot <
\wedge x < b})
        apply (rule arg-cong2[where f=proots-within])
        using mroot-in unfolding S-def by auto
    also have ... = proots-within Q {x. a<x\wedge x\leqmroot }}\cupS\mathrm{ mroot
        unfolding S-def Q-def
        apply (subst proots-within-union)
        by auto
    also have ... ={mroot }}\cupS\mathrm{ mroot
    proof -
        have proots-within Q {x.a<x\wedgex\leqmroot }}={\mathrm{ mroot }
            using mroot-in mroot-min unfolding S-def
            by auto force
        then show ?thesis by auto
    qed
    finally have Sa= insert mroot (S mroot) by auto
    then show ?thesis by auto
    qed
    also have ... = Suc (card (S mroot))
    apply (rule card-insert-disjoint)
    using fin-S unfolding S-def by auto
    finally have Suc n=Suc (card (S mroot)).
    then have n= card (S mroot) by simp
    then show n = card (proots-within (q*s*P){x.mroot <x\wedgex\leqb})
        unfolding S-def Q-def by simp
qed
have ?case if mroot = b
proof -
    have nzero:poly Q x\not=0 if a<x x<b for x
        using mroot-nzero <mroot = b that by auto
    define m where m=(a+b)/2
    have [simp]: a<m m<b using <a<b> unfolding m-def by auto
    have Case a m
    proof (rule basic-case)
    show proots-within }Q{x.a<x\wedgex<m}={
        using nzero <a<b> unfolding m-def by auto
    have poly Q m\not=0 using nzero <a<m\rangle\langlem<b\rangle by auto
    then show poly Q a}\not=0\vee\mathrm{ poly Q m}=0\mathrm{ by auto
```

qed $\operatorname{simp}$
moreover have Case $m b$
proof (rule basic-case)
show proots-within $Q\{x . m<x \wedge x<b\}=\{ \}$
using nzero $\langle a<b\rangle$ unfolding $m$-def by auto
have poly $Q m \neq 0$ using nzero $\langle a<m\rangle\langle m<b\rangle$ by auto
then show poly $Q m \neq 0 \vee$ poly $Q b \neq 0$ by auto
qed $\operatorname{simp}$
ultimately have C1 $a m+C 1 m b=(C 2 a m+C 2 m b)$

$$
+(C 3 a m+C 3 m b)-(C 4 a m+C 4 m b) / 2
$$

unfolding Case-def C1-def
apply simp
unfolding C2-def C3-def C4-def by (auto simp:algebra-simps)
moreover have
$C 1 a m+C 1 m b=C 1 a b$
C2 $a m+C 2 m b=C 2 a b$
$C 3 a m+C 3 m b=C 3 a b$
unfolding $C C$-def
by (rule cindex-polyE-combine;auto)+
moreover have C4 $a m+C 4 m b=C 4 a b$
unfolding C4-def cross-alt-def by simp
ultimately have C1 $a b=C 2 a b+C 3 a b-C 4 a b / 2$
by auto
then show ?thesis unfolding $C C$-def $P$-def by auto
qed
moreover have ? case if mroot $\neq b$
proof -
have $[$ simp $]: a<$ mroot mroot $<b$
using mroot-in that unfolding $S$-def by auto
define $m$ where $m=(a+m r o o t) / 2$
have $[$ simp $]: a<m$ momroot using mroot-in unfolding $m$-def $S$-def by auto
have poly $Q m \neq 0$
by (rule mroot-nzero) auto
have C1 mroot $b=C 2$ mroot $b+C 3$ mroot $b-C 4$ mroot $b / 2$ using hyps $\langle m r o o t<b\rangle$ by simp
moreover have Case a m
apply (rule basic-case)
subgoal
by (smt (verit) Collect-empty-eq $\langle m<m r o o t\rangle$ mem-Collect-eq mroot-nzero proots-within-def)
subgoal using $\langle p o l y ~ Q m \neq 0$ 〉 by auto
by fact
then have C1 a $m=C 2 a m+C 3 a m-C 4 a m / 2$
unfolding Case-def CC-def by auto
moreover have Case m mroot
apply (rule basic-case)

## subgoal

by（smt（verit）Collect－empty－eq $\langle a<m\rangle$ mem－Collect－eq mroot－nzero proots－within－def）
subgoal using $\langle p o l y ~ Q m \neq 0$ 〉 by auto
by fact
then have C1 m mroot $=$ C2 $m$ mroot $+C 3 m$ mroot $-C 4 m$ mroot $/ 2$
unfolding Case－def CC－def by auto
ultimately have $C 1 a m+C 1 m$ mroot $+C 1$ mroot $b=$

$$
(C 2 a m+C 2 m \text { mroot }+C 2 \text { mroot } b)
$$

$$
+(\text { C3 a m }+ \text { C3 m mroot }+ \text { C3 mroot } b)
$$

$$
-\left(C_{4} a m+C_{4} m \text { mroot }+C_{4} m r o o t b\right) / 2
$$

by $\operatorname{simp}$（simp add：algebra－simps）
moreover have
$C 1 a m+C 1 m$ mroot $+C 1$ mroot $b=C 1 a b$
C2 $a m+$ C2 $m$ mroot + C2 mroot $b=$ C2 $a b$
$C 3 a m+C 3 m$ mroot $+C 3$ mroot $b=C 3 a b$
unfolding $C C$－def
by（subst cindex－polyE－combine；simp？）＋
moreover have $C_{4} a m+C 4 m$ mroot $+C 4$ mroot $b=C 4 a b$
unfolding C4－def cross－alt－def by simp
ultimately have C1 $a b=C 2 a b+C 3 a b-C 4 a b / 2$
by auto
then show ？thesis unfolding $C C$－def $P$－def by auto
qed
ultimately show ？case by auto
qed
qed
lemma cindex－polyE－product：
fixes $p$ r $q$ s：：real poly and $a b$ ：：real
assumes $a<b$
and poly $p$ aキ0 $\vee$ poly $q$ a⿻丷 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－polyE ab（p＊r－q＊s）（p＊s＋q＊r）
$=$ cindex－polyE abpq＋cindex－polyE abrs
－cross－alt $(p * s+q * r)(q * s) a b / 2$
proof－
define $g 1$ where $g 1=g c d p q$
obtain $p^{\prime} q^{\prime}$ where $p q: p=g 1 * p^{\prime} q=g 1 * q^{\prime}$ and coprime $q^{\prime} p^{\prime}$ unfolding g1－def
by（metis assms（2）coprime－commute div－gcd－coprime dvd－mult－div－cancel gcd－dvd1
gcd-dvd2 order-root)
define $g 2$ where $g 2=g c d r s$
obtain $r^{\prime} s^{\prime}$ where $r s: r=g 2 * r^{\prime} s=g 2 * s^{\prime}$ coprime $s^{\prime} r^{\prime}$
unfolding g2－def using assms（4）
by（metis coprime－commute div－gcd－coprime dvd－mult－div－cancel gcd－dvd1 gcd－dvd2

```
order-root)
    define g}\mathrm{ where }g=g1*g
    have [simp]:g\not=0 g1\not=0 g2\not=0
        unfolding g-def g1-def g2-def
        using assms by auto
    have [simp]:poly g a\not=0 poly g b}\not=
        unfolding g-def g1-def g2-def
        subgoal by (metis assms(2) assms(4) poly-gcd-0-iff poly-mult-zero-iff)
        subgoal by (metis assms(3) assms(5) poly-gcd-0-iff poly-mult-zero-iff)
        done
    have cindex-polyE a b ( p'* *r' - q'* *') ( }\mp@subsup{p}{}{\prime}*\mp@subsup{s}{}{\prime}+\mp@subsup{q}{}{\prime}*\mp@subsup{r}{}{\prime})
        cindex-polyE a b p' q}\mp@subsup{\mp@code{'}}{+}{+}\mathrm{ cindex-polyE a b r r}\mp@subsup{s}{}{\prime
            (cross-alt ( }\mp@subsup{p}{}{\prime}*\mp@subsup{s}{}{\prime}+\mp@subsup{q}{}{\prime}*\mp@subsup{r}{}{\prime})(\mp@subsup{q}{}{\prime}*\mp@subsup{s}{}{\prime})ab)/
        using cindex-polyE-product'[OF <a<b\rangle<coprime q}\mp@subsup{q}{}{\prime}\mp@subsup{p}{}{\prime}\rangle\langlecoprime s' r'>]. 
    moreover have cindex-polyE a b (p*r-q*s)(p*s+q*r)
    = cindex-polyE a b (g*(\mp@subsup{p}{}{\prime}*\mp@subsup{r}{}{\prime}-\mp@subsup{q}{}{\prime}*\mp@subsup{s}{}{\prime}))(g*(\mp@subsup{p}{}{\prime}*\mp@subsup{s}{}{\prime}+\mp@subsup{q}{}{\prime}*\mp@subsup{r}{}{\prime}))
    unfolding pq rs g-def by (auto simp:algebra-simps)
    then have cindex-polyE ab(p*r-q*s)(p*s+q*r)
                        = cindex-polyE a b (p'* r' - q'* * ')( (p'* *' 
    apply (subst (asm) cindex-polyE-mult-cancel)
    by simp
    moreover have cindex-polyE a b p q= cindex-polyE ab p' q'
    unfolding pq using cindex-polyE-mult-cancel by simp
    moreover have cindex-polyE a brs = cindex-polyE a b r' s
    unfolding rs using cindex-polyE-mult-cancel by simp
    moreover have cross-alt (p*s+q*r)(q*s) ab
                        cross-alt (g*(\mp@subsup{p}{}{\prime}*\mp@subsup{s}{}{\prime}+\mp@subsup{q}{}{\prime}*\mp@subsup{r}{}{\prime}))(g*(\mp@subsup{q}{}{\prime}*\mp@subsup{s}{}{\prime})) ab
    unfolding pq rs g-def by (auto simp:algebra-simps)
    then have cross-alt (p*s+q*r)(q*s)ab
                        cross-alt ( }\mp@subsup{p}{}{\prime}*\mp@subsup{s}{}{\prime}+\mp@subsup{q}{}{\prime}*\mp@subsup{r}{}{\prime})(\mp@subsup{q}{}{\prime}*\mp@subsup{s}{}{\prime})a
    apply (subst (asm) cross-alt-cancel)
    by simp-all
    ultimately show ?thesis by auto
qed
lemma cindex-pathE-linepath-on:
    assumes z f closed-segment a b
    shows cindex-pathE (linepath a b) z=0
proof -
    obtain u where }0\lequu\leq
        and z-eq:z = complex-of-real (1 -u)*a+complex-of-real u*b
    using assms unfolding in-segment scaleR-conv-of-real
    by auto
    define U where U = [:-u,1:]
    have U\not=0 unfolding U-def by auto
```

```
    have cindex-pathE (linepath a b) z
        = cindexE 01 (\lambdat. (Ima+t*\operatorname{Im}b-(\operatorname{Im}z+t*\operatorname{Im}a))
        /(Rea+t*Reb-(Rez+t*Rea)))
    unfolding cindex-pathE-def
    by (simp add:linepath-def algebra-simps)
also have ... = cindexE 0 1
    (\lambdat. ((Im b - Im a)* (t-u))
        / ((Re b - Rea)* (t-u)))
    unfolding z-eq
    by (simp add:algebra-simps)
also have ... = cindex-polyE 0 1 (U*[:Im b - Im a:]) (U*[:Re b - Re a:])
proof (subst cindexE-eq-cindex-polyE[symmetric])
    have}(\operatorname{Im}b-\operatorname{Im}a)*(t-u)/((Reb-Rea)*(t-u)
                = poly (U* [:Im b - Im a:]) t / poly (U* [:Re b - Re a:]) t for t
    unfolding U-def by (simp add:algebra-simps)
    then show cindexE 0 1 (\lambdat. (Im b - Im a)* (t-u)/((Reb-Rea)* (t-
u))) =
                            cindexE 0 1 (\lambdax. poly (U* [:Im b - Im a:]) x / poly (U * [:Re b -
Re a:]) x)
    by auto
    qed simp
    also have ... = cindex-polyE 0 1 [:Im b - Im a:] [:Re b - Re a:]
        apply (rule cindex-polyE-mult-cancel)
        by fact
    also have ... = cindexE 01 (\lambdax. (Im b - Im a) / (Re b - Re a))
        apply (subst cindexE-eq-cindex-polyE[symmetric])
        by auto
    also have ... = 0
        apply (rule cindexE-constI)
        by auto
    finally show ?thesis.
qed
```


### 2.7 More Cauchy indices on polynomials

```
definition cindexP-pathE::complex poly }=>\mathrm{ (real }=>\mathrm{ complex })=>\mathrm{ real where
```

    cindexP-pathE p \(g=\) cindex-pathE (poly pog) 0
    definition cindexP-lineE :: complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ real where
cindexP-lineE pab=cindexP-pathE $p$ (linepath $a b$ )
lemma cindexP-pathE-const:cindexP-pathE [:c:] $g=0$
unfolding cindexP-pathE-def by (auto intro:cindex-pathE-constI)
lemma cindex-poly-pathE-joinpaths:
assumes finite-ReZ-segments (poly pog1) 0
and finite-ReZ-segments (poly pog2) 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 -
    have path (poly p o g1) path (poly p o g2)
        using <path g1> <path g2` by auto
    moreover have pathfinish (poly p o g1) = pathstart (poly p o g2)
        using <pathfinish g1 = pathstart g2`
        by (simp add: pathfinish-compose pathstart-def)
    ultimately have
        cindex-pathE ((poly p\circg1) +++(poly p\circg2)) 0=
            cindex-pathE (poly p\circg1) 0 + cindex-pathE (poly p\circg2) 0
    using cindex-pathE-joinpaths[OF assms(1,2)] by auto
    then show ?thesis
    unfolding cindexP-pathE-def
    by (simp add:path-compose-join)
qed
lemma cindexP-lineE-polyE:
    fixes p::complex poly and a b::complex
    defines pp \equiv pcompose p[:a,b-a:]
    defines pR\equiv map-poly Re pp
        and pI \equiv map-poly Im pp
    shows cindexP-lineE pab= cindex-polyE 0 1 pI pR
proof -
    have cindexP-lineE pab= cindexE 0 1
                    (\lambdat.Im (poly ( }p\mp@subsup{\circ}{p}{}[:a,b-a:])(complex-of-real t)) /
                    Re (poly ( }p\mp@subsup{\circ}{p}{[:a,b - a:]) (complex-of-real t)))
        unfolding cindexP-lineE-def cindexP-pathE-def cindex-pathE-def
        by (simp add:poly-linepath-comp')
    also have ... = cindexE 0 1 ( }\lambdat\mathrm{ . poly pIt/poly pR t)
        unfolding pI-def pR-def pp-def
        by (simp add:Im-poly-of-real Re-poly-of-real)
    also have ... = cindex-polyE 0 1 pI pR
        apply (subst cindexE-eq-cindex-polyE)
        by simp-all
    finally show ?thesis .
qed
definition psign-aux :: complex poly }=>\mathrm{ complex poly }=>\mathrm{ complex }=>\mathrm{ int where
    psign-aux p q b =
        sign (Im (poly p b* poly q b) *(Im (poly p b)*\operatorname{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 }=>\mathrm{ complex poly }=>\mathrm{ complex }=>\mathrm{ complex }=>\mathrm{ c 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\not=0 poly p b\not=0 poly q a\not=0 poly q b\not=0
    shows cindexP-lineE ( }p*q) ab=cindexP-lineE pab+cindexP-lineE qa
b+cdiff-aux p q a b/2
proof -
    define pR pI where pR= map-poly Re ( }p\mp@subsup{\circ}{p}{}[:a,b-a:]
                    and pI = map-poly Im ( }p\mp@subsup{\circ}{p}{}[:a,b-a:]
define qR qI where qR = map-poly Re (q\mp@subsup{\circ}{p}{}[:a,b-a:])
        and qI = map-poly Im (q\circop [:a,b - a:])
define P1 P2 where P1 = pR*qI + pI*qR and P2 =pR*qR-pI*qI
have p-poly:
        poly pR 0 = Re (poly pa)
        poly pI 0 = Im (poly pa)
        poly pR 1 = Re (poly pb)
        poly pI 1 = Im (poly p b)
    unfolding pR-def pI-def
    by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose)+
have q-poly:
    poly qR 0 = Re (poly q a)
    poly qI 0 = Im (poly qa)
    poly qR 1 = Re (poly q b)
    poly qI 1 = Im (poly q b)
    unfolding qR-def qI-def
    by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose)+
have P2-poly:
        poly P2 0 = Re (poly (p*q) a)
        poly P2 1 = Re (poly (p*q) b)
    unfolding P2-def pR-def qI-def pI-def qR-def
    by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose)+
have P1-poly:
        poly P1 O = Im (poly (p*q)a)
        poly P1 1 = Im (poly (p*q) b)
    unfolding P1-def pR-def qI-def pI-def qR-def
    by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose)+
have p-nzero:poly pR 0 = 0 \vee poly pI 0 f 0 poly pR 1 f=0 \vee poly pI 1 f=0
    unfolding p-poly
    using assms(1,2) complex-eqI by force+
have q-nzero:poly qR 0}=0\vee\mathrm{ poly qI 0}=0\mathrm{ poly qR 1 
    unfolding q-poly using assms(3,4) complex-eqI by force+
have P12-nzero:poly P2 0}\not=0\vee\mathrm{ poly P1 0}\not=0\mathrm{ poly P2 1 f=0 v poly P1 1 f=0
    unfolding P1-poly P2-poly using assms
    by (metis Im-poly-hom.base.hom-zero Re-poly-hom.base.hom-zero
        complex-eqI poly-mult-zero-iff)+
```

define C1 C2 where $C 1=\left(\begin{array}{ll}\lambda p q . ~ c i n d e x-p o l y E ~ & 0 \\ 1 & p\end{array}\right)$ and $\left.C 2=\left(\begin{array}{ll}\lambda p & q \text {. real-of-int }(\text { cross-alt } p q 01\end{array}\right) / \mathcal{Z}\right)$
define $C R$ where $C R=C 2 P 1(p I * q I)+C 2 P 2 P 1-C 2 p R p I-C 2 q R q I$
have cindexP-lineE $(p * q)$ a $b=$
cindex-polyE 01 (map-poly Im (cpoly-of pR pI*cpoly-of qR qI))
(map-poly Re (cpoly-of pR pI* cpoly-of $q R q I)$ )
proof -
have $p \circ_{p}[: a, b-a:]=$ cpoly-of $p R p I$
using cpoly-of-decompose pI-def pR-def by blast
moreover have $q \circ_{p}[: a, b-a:]=c p o l y$-of $q R q I$
using cpoly-of-decompose qI-def qR-def by blast
ultimately show ?thesis
apply (subst cindexP-lineE-polyE)
unfolding pcompose-mult by simp
qed
also have $\ldots=$ cindex-polyE $01(p R * q I+p I * q R)(p R * q R-p I * q I)$
unfolding cpoly-of-times by (simp add:algebra-simps)
also have $\ldots=$ cindex-polyE 01 P1 P2
unfolding P1-def P2-def by simp
also have $\ldots=$ cindex-polyE $01 p I p R+$ cindex-polyE $01 q I q R+C R$
proof -
have C1 P2 P1 $=C 1 p R p I+C 1 q R q I-C 2 P 1(p I * q I)$
unfolding P1-def P2-def C1-def C2-def
apply (rule cindex-polyE-product) thm cindex-polyE-product
by simp fact+
moreover have C1 P2 P1 = C2 P2 P1 - C1 P1 P2
unfolding C1-def C2-def
apply (subst cindex-polyE-inverse-add-cross' ${ }^{\prime}$ symmetric $]$ )
using P12-nzero by simp-all
moreover have $C 1 p R p I=C 2 p R p I-C 1 p I p R$
unfolding C1-def C2-def
apply (subst cindex-polyE-inverse-add-cross'[symmetric])
using $p$-nzero by simp-all
moreover have C1 $q R q I=C 2 q R q I-C 1 q I q R$
unfolding C1-def C2-def
apply (subst cindex-polyE-inverse-add-cross' ${ }^{\text {[symmetric] }]}$ )
using $q$-nzero by simp-all
ultimately have C2 P2 P1 - C1 P1 P2 $=(C 2 p R p I-C 1 p I p R)$

$$
+(C 2 q R q I-C 1 q I q R)-C 2 P 1(p I * q I)
$$

by auto
then have C1 P1 P2 $=C 1 p I p R+C 1 q I q R+C R$
unfolding CR-def by (auto simp:algebra-simps)
then show ?thesis unfolding C1-def .
qed
also have $\ldots=$ cindexP-lineE pab+cindexP-lineE qab+CR
unfolding C1-def $p I$-def $p R$-def $q I-$ def $q R$-def
apply (subst (1 2) cindexP-lineE-polyE)
by $\operatorname{simp}$
also have $\ldots=$ cindexP-lineE pab+cindexP-lineE q ab+cdiff-aux p q ab/2 proof -
have $C R=$ cdiff-aux p q a b/2
unfolding CR-def C2-def cross-alt-alt cdiff-aux-def psign-aux-def
by (simp add:P1-poly P2-poly p-poly $q$-poly del:times-complex.sel)
then show ?thesis by simp
qed
finally show ?thesis.
qed
lemma cindexP-lineE-changes:
fixes $p$ ::complex poly and $a b$ ::complex
assumes $p \neq 0 \quad a \neq b$
shows cindexP-lineE pab=
(let $p 1=$ pcompose $p[: a, b-a:]$;
$p R 1=$ map-poly Re p1;
pI1 $=$ map-poly Im p1;
$g c 1=g c d p R 1 p I 1$
in
real-of-int (changes-alt-itv-smods 01
(pR1 div gc1) (pI1 div gc1)) / 2)
proof -
define $p 1$ pR1 pI1 gc1 where $p 1=$ pcompose $p[: a, b-a:]$
and $p R 1=$ map-poly Re $p 1$ and $p I 1=$ map-poly Im $p 1$
and $g c 1=g c d p R 1 p I 1$
have $g c 1 \neq 0$
proof (rule ccontr)
assume $\neg g c 1 \neq 0$
then have $p I 1=0 p R 1=0$ unfolding gc1-def by auto
then have $p 1=0$ unfolding $p I 1$-def $p R 1$-def
by (metis cpoly-of-decompose map-poly-0)
then have $p=0$ unfolding $p 1$-def
apply (subst (asm) pcompose-eq-0)
using $\langle a \neq b$ by auto
then show False using $\langle p \neq 0\rangle$ by auto
qed
have cindexP-lineE pab=
cindexE $01(\lambda t . \operatorname{Im}($ poly $p$ (linepath a $b t))$
/ Re (poly p (linepath abt))
unfolding cindexP-lineE-def cindex-pathE-def cindexP-pathE-def by simp
also have $\ldots=$ cindexE $01(\lambda t$. poly pI1 $t /$ poly pR1 $t)$
unfolding pI1-def pR1-def p1-def poly-linepath-comp'
by (simp add:Im-poly-of-real Re-poly-of-real)
also have $\ldots=$ cindex-polyE 01 pI1 pR1
by (simp add: cindexE-eq-cindex-polyE)
also have $\ldots=$ cindex-polyE 01 (pI1 div gc1) (pR1 div gc1)
using $\langle g c 1 \neq 0\rangle$

```
    apply (subst (2) cindex-polyE-mult-cancel[of gc1,symmetric])
    by (simp-all add: gc1-def)
    also have ... = real-of-int (changes-alt-itv-smods 0 1
                (pR1 div gc1) (pI1 div gc1)) / 2
    apply (rule cindex-polyE-changes-alt-itv-mods)
    apply simp
    by (metis 〈gc1 = 0` div-gcd-coprime gc1-def gcd-eq-0-iff)
    finally show ?thesis
    by (metis gc1-def p1-def pI1-def pR1-def)
qed
lemma cindexP-lineE-code[code]:
    cindexP-lineE p a b=( if p\not=0 ^a\not=b then
    (let p1 = pcompose p [:a,b-a:];
        pR1 = map-poly Re p1;
        pI1 = map-poly Im p1;
        gc1 = gcd pR1 pI1
    in
        real-of-int (changes-alt-itv-smods 0 1
                            (pR1 div gc1) (pI1 div gc1)) / 2)
    else
    Code.abort (STR "cindexP-lineE fails for now')
            ( }\lambda\mathrm{ -. cindexP-lineE p a b))
using cindexP-lineE-changes by auto
```


## end

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


## begin

### 2.8 Misc

```
lemma closed-segment-imp-Re-Im:
fixes \(x\) ::complex
assumes \(x \in\) closed-segment \(l b u b\)
shows Re \(l b \leq R e u b \Longrightarrow R e l b \leq R e x \wedge R e x \leq R e u b\)
\(\operatorname{Im} l b \leq \operatorname{Im} u b \Longrightarrow \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im} u b\)
proof -
obtain \(u\) where \(x\) - \(u: x=(1-u) *_{R} l b+u *_{R} u b\) and \(0 \leq u u \leq 1\)
using assms unfolding closed-segment-def by auto
have Re \(l b \leq R e x\) when \(R e l b \leq R e u b\)
proof -
have \(\operatorname{Re} x=\operatorname{Re}\left((1-u) *_{R} l b+u *_{R} u b\right)\)
```

using $x-u$ by blast
also have $\ldots=\operatorname{Re}\left(l b+u *_{R}(u b-l b)\right)$ by (auto simp add:algebra-simps)
also have $\ldots=\operatorname{Re} l b+u *(\operatorname{Re} u b-R e l b)$ by auto
also have $\ldots \geq$ Re $l b$ using $\langle u \geq 0\rangle\langle R e l b \leq R e u b\rangle$ by auto
finally show ?thesis.
qed
moreover have $\operatorname{Im} l b \leq \operatorname{Im} x$ when $\operatorname{Im} l b \leq \operatorname{Im} u b$ proof -
have $\operatorname{Im} x=\operatorname{Im}\left((1-u) *_{R} l b+u *_{R} u b\right)$ using $x-u$ by blast
also have $\ldots=\operatorname{Im}\left(l b+u *_{R}(u b-l b)\right)$ by (auto simp add:algebra-simps)
also have $\ldots=\operatorname{Im} l b+u *(\operatorname{Im} u b-\operatorname{Im} l b)$ by auto
also have $\ldots \geq \operatorname{Im} l b$ using $\langle u \geq 0\rangle\langle\operatorname{Im} l b \leq \operatorname{Im} u b\rangle$ by auto
finally show ?thesis .
qed
moreover have $R e x \leq R e u b$ when $R e l b \leq R e u b$
proof -
have $\operatorname{Re} x=\operatorname{Re}\left((1-u) *_{R} l b+u *_{R} u b\right)$
using $x-u$ by blast
also have $\ldots=(1-u) * \operatorname{Re} l b+u * \operatorname{Re} u b$ by auto
also have $\ldots \leq(1-u) * R e u b+u * R e u b$ using $\langle u \leq 1\rangle\langle R e l b \leq R e ~ u b\rangle$ by (auto simp add: mult-left-mono)
also have $\ldots=R e u b$ by (auto simp add:algebra-simps)
finally show? thesis.
qed
moreover have $\operatorname{Im} x \leq \operatorname{Im} u b$ when $\operatorname{Im} l b \leq \operatorname{Im} u b$
proof -
have $\operatorname{Im} x=\operatorname{Im}\left((1-u) *_{R} l b+u *_{R} u b\right)$
using $x-u$ by blast
also have $\ldots=(1-u) * \operatorname{Im} l b+u * \operatorname{Im} u b$ by auto
also have $\ldots \leq(1-u) * \operatorname{Im} u b+u * \operatorname{Im} u b$
using $\langle u \leq 1\rangle\langle\operatorname{Im} l b \leq I m u b\rangle$ by (auto simp add: mult-left-mono)
also have $\ldots=\operatorname{Im} u b$ by (auto simp add:algebra-simps)
finally show ?thesis.
qed
ultimately show
$R e l b \leq R e u b \Longrightarrow \operatorname{Re} l b \leq \operatorname{Re} x \wedge \operatorname{Re} x \leq R e u b$
$\operatorname{Im} l b \leq \operatorname{Im} u b \Longrightarrow \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im} u b$
by auto
qed
lemma closed-segment-degen-complex:
$\llbracket R e l b=R e u b ; \operatorname{Im} l b \leq \operatorname{Im} u b \rrbracket$
$\Longrightarrow x \in$ closed-segment $l b u b \longleftrightarrow \operatorname{Re} x=\operatorname{Re} l b \wedge \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im}$ ub
$\llbracket I m l b=I m u b ; R e l b \leq R e u b \rrbracket$
$\Longrightarrow x \in$ closed-segment $l b u b \longleftrightarrow \operatorname{Im} x=\operatorname{Im} l b \wedge \operatorname{Re} l b \leq \operatorname{Re} x \wedge \operatorname{Re} x \leq R e$
$u b$
proof -
show $x \in$ closed-segment $l b u b \longleftrightarrow \operatorname{Re} x=\operatorname{Re} l b \wedge \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im}$ ub
when Re $l b=$ Re ub Im $l b \leq \operatorname{Im} u b$

## proof

show Re $x=\operatorname{Re} l b \wedge \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im} u b$ when $x \in$ closed-segment $l b u b$
using closed-segment-imp-Re-Im[OF that] $\langle R e l b=R e u b\rangle\langle\operatorname{Im} l b \leq \operatorname{Im} u b\rangle$ by fastforce
next
assume asm:Re $x=\operatorname{Re} l b \wedge \operatorname{Im} l b \leq \operatorname{Im} x \wedge \operatorname{Im} x \leq \operatorname{Im} u b$
define $u$ where $u=(\operatorname{Im} x-\operatorname{Im} l b) /(\operatorname{Im} u b-\operatorname{Im} l b)$
have $x=(1-u) *_{R} l b+u *_{R} u b$
unfolding $u$-def using asm $\langle R e l b=R e u b\rangle\langle I m l b \leq I m u b\rangle$
apply (intro complex-eqI)
apply (auto simp add:field-simps)
apply (cases Im $u b-\operatorname{Im} l b=0$ )
apply (auto simp add:field-simps)
done
moreover have $0 \leq u u \leq 1$ unfolding $u$-def
using $\langle I m l b \leq I m u b\rangle$ asm
by (cases Im ub - Im $l b=0$,auto simp add:field-simps) +
ultimately show $x \in$ closed-segment $l b u b$ unfolding closed-segment-def by auto

## qed

show $x \in$ closed-segment $l b u b \longleftrightarrow \operatorname{Im} x=\operatorname{Im} l b \wedge R e l b \leq R e x \wedge R e x \leq R e$ ub
when $\operatorname{Im} l b=\operatorname{Im} u b \operatorname{Re} l b \leq R e u b$
proof
show $\operatorname{Im} x=\operatorname{Im} l b \wedge \operatorname{Re} l b \leq \operatorname{Re} x \wedge \operatorname{Re} x \leq \operatorname{Re} u b$ when $x \in$ closed-segment $l b u b$
using closed-segment-imp-Re-Im[OF that $]\langle\operatorname{Im} l b=\operatorname{Im} u b\rangle\langle R e l b \leq R e u b\rangle$ by fastforce
next
assume asm:Im $x=\operatorname{Im} l b \wedge \operatorname{Re} l b \leq \operatorname{Re} x \wedge \operatorname{Re} x \leq \operatorname{Re} u b$
define $u$ where $u=($ Re $x-R e l b) /($ Re $u b-R e l b)$
have $x=(1-u) *_{R} l b+u *_{R} u b$
unfolding $u$-def using $a s m \prec I m l b=I m u b\rangle\langle R e l b \leq R e u b\rangle$
apply (intro complex-eqI)
apply (auto simp add:field-simps)
apply (cases Re ub - Re $l b=0$ )
apply (auto simp add:field-simps)
done
moreover have $0 \leq u u \leq 1$ unfolding $u$-def
using $\langle R e l b \leq R e u b\rangle$ asm
by (cases Re ub - Re lb=0,auto simp add:field-simps) +
ultimately show $x \in$ closed-segment $l b u b$ unfolding closed-segment-def by auto
qed
qed

```
corollary path-image-part-circlepath-subset:
    assumes r\geq0
    shows path-image(part-circlepath z r st tt)\subseteq sphere z r
proof (cases st\leqtt)
    case True
    then show ?thesis
        by (auto simp: assms path-image-part-circlepath sphere-def dist-norm alge-
bra-simps norm-mult)
next
    case False
    then have path-image(part-circlepath z r tt st)\subseteq sphere zr
            by (auto simp: assms path-image-part-circlepath sphere-def dist-norm alge-
bra-simps norm-mult)
    moreover have path-image(part-circlepath zr tt st) = path-image(part-circlepath
z r st tt)
    using path-image-reversepath by fastforce
    ultimately show ?thesis by auto
qed
```

proposition in-path-image-part-circlepath:
assumes $w \in$ path-image(part-circlepath zr st tt) $0 \leq r$
shows $\operatorname{norm}(w-z)=r$
proof -
have $w \in\{c$. dist $z c=r\}$
by (metis (no-types) path-image-part-circlepath-subset sphere-def subset-eq assms)
thus ?thesis
by (simp add: dist-norm norm-minus-commute)
qed
lemma infinite-ball:
fixes $a$ :: 'a::euclidean-space
assumes $r>0$
shows infinite (ball a r)
using uncountable-ball[OF assms, THEN uncountable-infinite] .
lemma infinite-cball:
fixes $a$ :: 'a::euclidean-space
assumes $r>0$
shows infinite (cball a r)
using uncountable-cball[OF assms, THEN uncountable-infinite,of a].
lemma infinite-sphere:
fixes $a$ :: complex
assumes $r>0$
shows infinite (sphere a r)

```
proof -
    have uncountable (path-image (circlepath a r))
        apply (rule simple-path-image-uncountable)
        using simple-path-circlepath assms by simp
    then have uncountable (sphere a r)
        using assms by simp
    from uncountable-infinite[OF this] show ?thesis.
qed
lemma infinite-halfspace-Im-gt: infinite {x.Im x>b}
    apply (rule connected-uncountable[THEN uncountable-infinite,of - (b+1)* i (b+2)*i])
    by (auto intro!:convex-connected simp add: convex-halfspace-Im-gt)
lemma (in ring-1) Ints-minus2: - }a\in\mathbb{Z}\Longrightarrowa\in\mathbb{Z
    using Ints-minus[of -a] by auto
lemma dvd-divide-Ints-iff:
```



```
proof
    assume asm:b dvd a \vee b=0
    let ?thesis = of-int a / of-int b 
    have ?thesis when b dvd a
    proof -
        obtain c where }a=b*c\mathrm{ using <b dvd a> unfolding dvd-def by auto
        then show ?thesis by (auto simp add:field-simps)
    qed
    moreover have ?thesis when b=0
        using that by auto
    ultimately show ?thesis using asm by auto
next
    assume of-int a / of-int b\in(\mathbb{Z :: 'a :: {field,ring-char-0} set)}}=\mp@code{*}
    from Ints-cases[OF this] obtain c where *:(of-int::- = ' 'a)c=of-int a / of-int
b
    by metis
    have b dvd a when }b\not=
    proof -
        have (of-int::- # ' }a\mathrm{ ) a =of-int b*of-int c using that * by auto
        then have }a=b*c\mathrm{ using of-int-eq-iff by fastforce
        then show ?thesis unfolding dvd-def by auto
    qed
    then show b dvd a\veeb=0 by auto
qed
lemma of-int-div-field:
assumes \(d\) dvd \(n\)
shows (of-int::- \(\boldsymbol{A}^{\prime} a::\) field-char-0) \((n\) div d) \(=\) of-int \(n /\) of-int \(d\)
apply (subst (2) dvd-mult-div-cancel[OF assms,symmetric])
by (auto simp add:field-simps)
```

lemma powr-eq-1-iff:
assumes $a>0$
shows ( $a::$ real) powr $b=1 \longleftrightarrow a=1 \vee b=0$
proof
assume $a$ powr $b=1$
have $b * \ln a=0$
using $\langle a$ powr $b=1$ 〉 ln-powr $[$ of $a b]$ assms by auto
then have $b=0 \vee \ln a=0$ by auto
then show $a=1 \vee b=0$ using assms by auto
qed (insert assms, auto)
lemma tan-inj-pi:
$-(p i / 2)<x \Longrightarrow x<p i / 2 \Longrightarrow-(p i / 2)<y \Longrightarrow y<p i / 2 \Longrightarrow \tan x=\tan y$
$\Longrightarrow x=y$
by (metis arctan-tan)
lemma finite-ReZ-segments-poly-circlepath:
finite-ReZ-segments (poly $p \circ$ circlepath z0 r) 0
proof (cases $\forall t \in(\{0 . .1\}-\{1 / 2\})$. Re $(($ poly $p \circ$ circlepath $z 0 r) t)=0)$
case True
have isCont (Re $\circ$ poly $p \circ$ circlepath z0 r) (1/2)
by (auto intro!:continuous-intros simp:circlepath)
moreover have (Re o poly $p \circ$ circlepath $z 0 r)-1 / 2 \rightarrow 0$
proof -
have $\forall_{F} x$ in at (1/2). (Re $\circ$ poly $p \circ$ circlepath $\left.z 0 r\right) x=0$
unfolding eventually-at-le
apply (rule exI[where $x=1 / 2]$ )
unfolding dist-real-def abs-diff-le-iff
by (auto intro!: True[rule-format, unfolded comp-def])
then show?thesis by (rule tendsto-eventually)
qed
ultimately have $\operatorname{Re}(($ poly $p \circ$ circlepath z0 r) $(1 / 2))=0$
unfolding comp-def by (simp add: LIM-unique continuous-within)
then have $\forall t \in\{0 . .1\}$. Re $(($ poly $p \circ$ circlepath $z 0 r) t)=0$
using True by blast
then show?thesis
apply (rule-tac finite-ReZ-segments-constI[THEN finite-ReZ-segments-congE]) by auto
next
case False
define $q 1 q 2$ where $q 1=$ fcompose $p[:(z 0+r) * \mathrm{i}, z 0-r:][: \mathrm{i}, 1:]$ and $q 2=([: i, 1:]$ ^ degree $p)$
define $q 1 R q 1 I$ where $q 1 R=$ map-poly $R e q 1$ and $q 1 I=$ map-poly Im $q 1$ define $q 2 R q 2 I$ where $q 2 R=$ map-poly $R e q 2$ and $q 2 I=$ map-poly $\operatorname{Im} q 2$ define $q q$ where $q q=q 1 R * q 2 R+q 1 I * q 2 I$
have poly-eq:Re $(($ poly $p \circ$ circlepath $z 0 r) t)=0 \longleftrightarrow$ poly $q q(\tan (p i * t))=0$ when $0 \leq t t \leq 1 t \neq 1 / 2$ for $t$

```
proof -
    define \(t t\) where \(t=\tan (p i * t)\)
    have Re \(((\) poly \(p \circ\) circlepath \(z 0 r) t)=0 \longleftrightarrow R e(\) poly q1 tt \(/\) poly q2 \(t t)=0\)
        unfolding comp-def
        apply (subst poly-circlepath-tan-eq[of t p z0 r,folded q1-def q2-def tt-def])
        using that by simp-all
    also have \(\ldots \longleftrightarrow\) poly \(q 1 R\) tt \(*\) poly \(q 2 R\) tt + poly \(q 1 I t t *\) poly \(q 2 I t t=0\)
        unfolding \(q 1 I\)-def \(q 1 R\)-def \(q 2 R\)-def \(q 2 I\)-def
        by (simp add:Re-complex-div-eq-0 Re-poly-of-real Im-poly-of-real)
    also have \(\ldots \longleftrightarrow\) poly \(q q t t=0\)
        unfolding \(q q\)-def by simp
    finally show ?thesis unfolding \(t t-d e f\).
qed
have finite \(\{t\). Re \(((\) poly \(p \circ\) circlepath \(z 0 r) t)=0 \wedge 0 \leq t \wedge t \leq 1\}\)
proof -
    define \(P\) where \(P=(\lambda t\). Re \(((\) poly \(p \circ\) circlepath \(z 0 r) t)=0)\)
    define \(A\) where \(A=(\{0 . .1\}::\) real set \()\)
    define \(S\) where \(S=\{t \in A-\{1,1 / 2\} . P t\}\)
    have finite \(\{t\). poly \(q q(\tan (p i * t))=0 \wedge 0 \leq t \wedge t<1 \wedge t \neq 1 / 2\}\)
    proof -
    define \(A\) where \(A=\{t::\) real. \(0 \leq t \wedge t<1 \wedge t \neq 1 / 2\}\)
    have finite \(((\lambda t\). tan \((p i * t))-‘\{x\). poly \(q q x=0\} \cap A)\)
    proof (rule finite-vimage-IntI)
            have \(x=y\) when \(\tan (p i * x)=\tan (p i * y) x \in A y \in A\) for \(x y\)
            proof -
                define \(x^{\prime}\) where \(x^{\prime}=(\) if \(x<1 / 2\) then \(x\) else \(x-1)\)
            define \(y^{\prime}\) where \(y^{\prime}=(\) if \(y<1 / 2\) then \(y\) else \(y-1)\)
            have \(x^{\prime} * p i=y^{\prime} * p i\)
            proof (rule tan-inj-pi)
                    have \(*:-1 / 2<x^{\prime} x^{\prime}<1 / 2-1 / 2<y^{\prime} y^{\prime}<1 / 2\)
                    using that(2,3) unfolding \(x^{\prime}\)-def \(y^{\prime}\)-def \(A\)-def by simp-all
                    show \(-(p i / 2)<x^{\prime} * p i x^{\prime} * p i<p i / 2-(p i / 2)<y^{\prime} * p i\)
                    \(y^{\prime} * p i<p i / 2\)
                    using mult-strict-right-mono[OF *(1),of pi]
                        mult-strict-right-mono \([O F *\) (2), of pi]
                                    mult-strict-right-mono[OF *(3),of pi]
                                    mult-strict-right-mono[OF *(4),of pi]
                    by auto
            next
                have \(\tan \left(x^{\prime} * p i\right)=\tan (x * p i)\)
                    unfolding \(x^{\prime}\)-def using tan-periodic-int[of - - 1 ,simplified \(]\)
                    by (auto simp add:algebra-simps)
                    also have \(\ldots=\tan (y * p i)\)
                    using \(\langle\tan (p i * x)=\tan (p i * y)\rangle\) by (auto simp:algebra-simps)
                    also have \(\ldots=\tan \left(y^{\prime} * p i\right)\)
                    unfolding \(y^{\prime}\)-def using tan-periodic-int[of - 1 ,simplified]
                    by (auto simp add:algebra-simps)
                    finally show \(\tan \left(x^{\prime} * p i\right)=\tan \left(y^{\prime} * p i\right)\).
```

```
        qed
        then have }\mp@subsup{x}{}{\prime}=\mp@subsup{y}{}{\prime}\mathrm{ by auto
        then show ?thesis
            using that(2,3) unfolding x'-def y'-def A-def by (auto split:if-splits)
    qed
    then show inj-on ( }\lambdat\mathrm{ t. tan (pi*t)) A
        unfolding inj-on-def by blast
    next
    have qq\not=0
    proof (rule ccontr)
        assume }\negqq\not=
        then have Re ((poly p\circ circlepath z0 r)t)=0 when t\in{0..1} - {1/2}
for }
            apply (subst poly-eq)
            using that by auto
            then show False using False by blast
        qed
        then show finite {x. poly qq x=0} by (simp add: poly-roots-finite)
        qed
        then show ?thesis by (elim rev-finite-subset) (auto simp:A-def)
    qed
    moreover have {t.poly qq (tan (pi*t))=0\wedge0\leqt\wedget<1\wedget\not=1/2}=S
        unfolding S-def P-def A-def using poly-eq by force
    ultimately have finite S by blast
    then have finite (S\cup(if P1 then {1} else {})\cup(if P(1/2) then {1/2} else
{}))
            by auto
    moreover have }(S\cup(\mathrm{ if P 1 then {1} else {}) }\cup(\mathrm{ if }P(1/2) then {1/2} els
{}))
                        ={t.Pt\wedge0\leqt\wedget\leq1}
    proof -
        have }1\inA1/2\inA\mathrm{ unfolding }A\mathrm{ -def by auto
        then have }(S\cup(\mathrm{ if }P1\mathrm{ then {1} else {}) }\cup(\mathrm{ if }P(1/\mathcal{Z})\mathrm{ then {1/2} else {}))
                        ={t\inA.Pt}
            unfolding S-def
            apply auto
            by (metis eq-divide-eq-numeral1(1) zero-neq-numeral)+
            also have ... ={t. Pt\wedge0\leqt\wedget\leq1}
            unfolding }A\mathrm{ -def by auto
            finally show ?thesis .
    qed
    ultimately have finite {t.Pt\wedge0\leqt\wedget\leq1} by auto
    then show ?thesis unfolding P-def by simp
qed
then show ?thesis
    apply (rule-tac finite-imp-finite-ReZ-segments)
    by auto
qed
```

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

### 2.9 Some useful conformal/bij-betw properties

lemma bij-betw-plane-ball:bij-betw $(\lambda x .(\mathrm{i}-x) /(\mathrm{i}+x))\{x$. Im $x>0\}($ ball 01 ) proof (rule bij-betw-imageI)
have neq:i $+x \neq 0$ when $\operatorname{Im} x>0$ for $x$
using that
by (metis add-less-same-cancel2 add-uminus-conv-diff diff-0 diff-add-cancel imaginary-unit.simps(2) not-one-less-zero uminus-complex.sel(2))
then show inj-on $(\lambda x$. $(\mathrm{i}-x) /(\mathrm{i}+x))\{x .0<\operatorname{Im} x\}$
unfolding inj-on-def by (auto simp add:divide-simps algebra-simps)
have $\operatorname{cmod}((\mathrm{i}-x) /(\mathrm{i}+x))<1$ when $0<\operatorname{Im} x$ for $x$
proof -
have $\operatorname{cmod}(\mathrm{i}-x)<\operatorname{cmod}(\mathrm{i}+x)$
unfolding norm-lt inner-complex-def using that
by (auto simp add:algebra-simps)
then show?thesis
unfolding norm-divide using neq[OF that] by auto
qed
moreover have $x \in(\lambda x$. $(\mathrm{i}-x) /(\mathrm{i}+x))$ ' $\{x .0<\operatorname{Im} x\}$ when $\operatorname{cmod} x<1$ for $x$
proof (rule rev-image-eqI[of $\mathrm{i} *(1-x) /(1+x)])$
have $1+x \neq 0 \mathrm{i} * 2+\mathrm{i} *(x * 2) \neq 0$
subgoal using that by (metis complex-mod-triangle-sub norm-one norm-zero not-le pth-7(1)) subgoal using that by (metis $\langle 1+x \neq 0\rangle$ complex-i-not-zero div-mult-self 4 mult-2
mult-zero-right nonzero-mult-div-cancel-left nonzero-mult-div-cancel-right one-add-one zero-neq-numeral)

## done

then show $x=(\mathrm{i}-\mathrm{i} *(1-x) /(1+x)) /(\mathrm{i}+\mathrm{i} *(1-x) /(1+x))$ by (auto simp add:field-simps)
show $\mathrm{i} *(1-x) /(1+x) \in\{x .0<\operatorname{Im} x\}$
apply (auto simp:Im-complex-div-gt-0 algebra-simps)
using that unfolding cmod-def by (auto simp:power2-eq-square)
qed
ultimately show $(\lambda x .(\mathrm{i}-x) /(\mathrm{i}+x)) '\{x .0<\operatorname{Im} x\}=$ ball 01
by auto
qed
lemma bij-betw-axis-sphere:bij-betw $(\lambda x .(\mathrm{i}-x) /(\mathrm{i}+x))\{x$. Im $x=0\}$ (sphere $01-$ $\{-1\}$ )
proof (rule bij-betw-imageI)
have neq: $\mathrm{i}+x \neq 0$ when $\operatorname{Im} x=0$ for $x$
using that
by (metis add-diff-cancel-left' imaginary-unit.simps(2) minus-complex.simps(2)
right-minus-eq zero-complex.simps(2) zero-neq-one)
then show inj-on $(\lambda x .(\mathrm{i}-x) /(\mathrm{i}+x))\{x . \operatorname{Im} x=0\}$
unfolding inj-on-def by (auto simp add:divide-simps algebra-simps)
have $\operatorname{cmod}((\mathrm{i}-x) /(\mathrm{i}+x))=1(\mathrm{i}-x) /(\mathrm{i}+x) \neq-1$ when $\operatorname{Im} x=0$ for $x$
proof -
have $\operatorname{cmod}(\mathrm{i}+x)=\operatorname{cmod}(\mathrm{i}-x)$
using that unfolding cmod-def by auto
then show $\operatorname{cmod}((\mathrm{i}-x) /(\mathrm{i}+x))=1$
unfolding norm-divide using neq[OF that] by auto
show $(\mathrm{i}-x) /(\mathrm{i}+x) \neq-1$ using neq $[$ OF that $]$ by (auto simp add:divide-simps)
qed
moreover have $x \in(\lambda x$. $(\mathrm{i}-x) /(\mathrm{i}+x))$ ' $\{x . \operatorname{Im} x=0\}$
when $\operatorname{cmod} x=1 x \neq-1$ for $x$
proof (rule rev-image-eqI[of $\mathrm{i} *(1-x) /(1+x)])$
have $1+x \neq 0 \mathrm{i} * 2+\mathrm{i} *(x * 2) \neq 0$
subgoal using that(2) by algebra
subgoal using that by (metis $\langle 1+x \neq 0\rangle$ complex- $i$-not-zero div-mult-self 4
mult-2
mult-zero-right nonzero-mult-div-cancel-left nonzero-mult-div-cancel-right one-add-one zero-neq-numeral)
done
then show $x=(\mathrm{i}-\mathrm{i} *(1-x) /(1+x)) /(\mathrm{i}+\mathrm{i} *(1-x) /(1+x))$ by (auto simp add:field-simps)
show $\mathrm{i} *(1-x) /(1+x) \in\{x . \operatorname{Im} x=0\}$
apply (auto simp:algebra-simps Im-complex-div-eq-0)
using that (1) unfolding cmod-def by (auto simp:power2-eq-square)
qed
ultimately show $(\lambda x .(\mathrm{i}-x) /(\mathrm{i}+x)) '\{x . \operatorname{Im} x=0\}=$ sphere $01-\{-1\}$
by force
qed
lemma bij-betw-ball-uball:
assumes $r>0$
shows bij-betw $(\lambda x$. complex-of-real $r * x+z 0)($ ball 01$)($ ball z0 r)
proof (rule bij-betw-imageI)
show inj-on $(\lambda x$. complex-of-real $r * x+z 0)($ ball 01$)$
unfolding inj-on-def using assms by simp
have dist $z 0$ (complex-of-real $r * x+z 0$ ) $<r$ when cmod $x<1$ for $x$
using that assms by (auto simp:dist-norm norm-mult abs-of-pos)
moreover have $x \in(\lambda x$. complex-of-real $r * x+z 0)$ 'ball 01 when dist $z 0 x$ $<r$ for $x$
apply (rule rev-image-eqI[of $(x-z 0) / r])$
using that assms by (auto simp add: dist-norm norm-divide norm-minus-commute)
ultimately show ( $\lambda x$. complex-of-real $r * x+z 0$ )'ball $01=$ ball z0 $r$
by auto
qed

```
lemma bij-betw-sphere-usphere:
    assumes r>0
    shows bij-betw ( }\lambda\mathrm{ x. complex-of-real r*x + z0) (sphere 0 1) (sphere z0 r)
proof (rule bij-betw-imageI)
    show inj-on ( }\lambdax\mathrm{ . complex-of-real r *x+z0) (sphere 0 1)
        unfolding inj-on-def using assms by simp
    have dist z0 (complex-of-real r*x+z0)=r when cmod x=1 for x
    using that assms by (auto simp:dist-norm norm-mult abs-of-pos)
    moreover have }x\in(\lambdax\mathrm{ . complex-of-real r*x+z0)' sphere 0 1 when dist z0
x=r for }
    apply (rule rev-image-eqI[of (x-z0)/r])
    using that assms by (auto simp add: dist-norm norm-divide norm-minus-commute)
    ultimately show ( }\lambdax\mathrm{ . complex-of-real r *x+z0)'sphere 0 1 = sphere z0 r
    by auto
qed
lemma proots-ball-plane-eq:
    defines q1 \equiv[:i,-1:] and q2\equiv[:i,1:]
    assumes p\not=0
    shows proots-count p (ball 0 1) = proots-count (fcompose p q1 q2) {x.0 < Im
x}
    unfolding q1-def q2-def
proof (rule proots-fcompose-bij-eq[OF - <p\not=0\rangle])
    show }\forallx\in{x.0<\operatorname{Im}x}.poly [:i, 1:] x\not=
        apply simp
        by (metis add-less-same-cancel2 imaginary-unit.simps(2) not-one-less-zero
                plus-complex.simps(2) zero-complex.simps(2))
    show infinite (UNIV ::complex set) by (simp add: infinite-UNIV-char-0)
qed (use bij-betw-plane-ball in auto)
lemma proots-sphere-axis-eq:
    defines q1\equiv[:i,-1:] and q2\equiv[:i,1:]
    assumes p\not=0
    shows proots-count p (sphere 0 1 - {-1}) = proots-count (fcompose p q1 q2)
{x.0 = Im x }
unfolding q1-def q2-def
proof (rule proots-fcompose-bij-eq[OF - <p\not=0`])
    show }\forallx\in{x.0=Im x}. poly [:i, 1:] x = 0 by (simp add: Complex-eq-0
plus-complex.code)
    show infinite (UNIV ::complex set) by (simp add: infinite-UNIV-char-0)
qed (use bij-betw-axis-sphere in auto)
lemma proots-card-ball-plane-eq
    defines q1\equiv[:i,-1:] and q2\equiv[:i,1:]
    assumes p\not=0
    shows card (proots-within p (ball 0 1)) = card (proots-within (fcompose p q1 q2)
{x.0<Im x})
unfolding q1-def q2-def
```

```
proof (rule proots-card-fcompose-bij-eq[OF - \(\langle p \neq 0\rangle]\) )
    show \(\forall x \in\{x .0<\operatorname{Im} x\}\). poly [:i, 1:] \(x \neq 0\)
        apply simp
    by (metis add-less-same-cancel2 imaginary-unit.simps(2) not-one-less-zero
                plus-complex.simps(2) zero-complex.simps(2))
qed (use bij-betw-plane-ball infinite-UNIV-char-0 in auto)
lemma proots-card-sphere-axis-eq:
    defines \(q 1 \equiv[: i,-1:]\) and \(q 2 \equiv[: i, 1:]\)
    assumes \(p \neq 0\)
    shows card (proots-within \(p\) (sphere \(01-\{-1\})\) )
            \(=\operatorname{card}(\) proots-within (fcompose p q1 q2) \(\{x .0=\operatorname{Im} x\})\)
unfolding \(q 1\)-def \(q 2\)-def
proof (rule proots-card-fcompose-bij-eq[OF - \(\langle p \neq 0\rangle]\) )
    show \(\forall x \in\{x .0=\operatorname{Im} x\}\). poly [:i, 1:] \(x \neq 0\) by (simp add: Complex-eq-0
plus-complex.code)
qed (use bij-betw-axis-sphere infinite-UNIV-char-0 in auto)
lemma proots-uball-eq:
    fixes \(z 0::\) complex and \(r::\) real
    defines \(q \equiv[: z 0\), of-real \(r:]\)
    assumes \(p \neq 0\) and \(r>0\)
    shows proots-count p(ball z0 r) \(=\) proots-count \(\left(p \circ_{p} q\right)(\) ball 01\()\)
proof -
    show ?thesis
        apply (rule proots-pcompose-bij-eq[OF - \(\langle p \neq 0\rangle]\) )
        subgoal unfolding \(q\)-def using bij-betw-ball-uball \([O F\langle r>0\rangle, o f z 0]\) by (auto
simp:algebra-simps)
    subgoal unfolding \(q\)-def using \(\langle r\rangle 0\rangle\) by auto
        done
qed
lemma proots-card-uball-eq:
    fixes \(z 0::\) complex and \(r::\) real
    defines \(q \equiv[: z 0\), of-real \(r:]\)
    assumes \(r>0\)
    shows card (proots-within p(ball z0 r)) \(=\) card \(\left(\right.\) proots-within \(\left(p \circ_{p} q\right)(b a l l ~ 0\)
1))
proof -
    have ?thesis
        when \(p=0\)
    proof -
        have card \((b a l l ~ z 0 r)=0\) card \((\) ball \((0::\) complex \() 1)=0\)
            using infinite-ball[OF〈r>0〉,of z0] infinite-ball[ of \(10::\) complex] by auto
            then show ?thesis using that by auto
    qed
    moreover have ?thesis
        when \(p \neq 0\)
        apply (rule proots-card-pcompose-bij-eq[OF-〈p申0〉])
```

subgoal unfolding $q$-def using bij-betw-ball-uball $[O F\langle r>0\rangle, o f z 0]$ by (auto simp:algebra-simps)
subgoal unfolding $q$-def using $\langle r>0\rangle$ by auto done
ultimately show ?thesis
by blast
qed
lemma proots-card-usphere-eq:
fixes $z 0::$ complex and $r::$ real
defines $q \equiv[: z 0$, of-real $r:]$
assumes $r>0$
shows card (proots-within $p($ sphere $z 0 r))=\operatorname{card}\left(\right.$ proots-within $\left(p \circ_{p} q\right)($ sphere 0 1))
proof -
have ?thesis
when $p=0$
proof -
have card (sphere z0r)=0 card (sphere $(0::$ complex $) 1)=0$
using infinite-sphere[OF $\langle r>0\rangle$,of $z 0]$ infinite-sphere[of $10::$ complex] by auto
then show ?thesis using that by auto
qed
moreover have ?thesis
when $p \neq 0$
apply (rule proots-card-pcompose-bij-eq[OF-<p$\neq 0\rangle]$ )
subgoal unfolding $q$-def using bij-betw-sphere-usphere[OF $\langle r>0\rangle$, of $z 0]$
by (auto simp:algebra-simps)
subgoal unfolding $q$-def using $\langle r\rangle 0\rangle$ by auto
done
ultimately show $\operatorname{card}($ proots-within $p(s p h e r e z 0 r))=\operatorname{card}(p r o o t s-w i t h i n ~(p$ $\circ_{p} q$ ) (sphere 0 1) )
by blast
qed

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

definition unbounded-line::'a::real-vector $\Rightarrow^{\prime} a \Rightarrow$ 'a set where
unbounded-line a $b=\left(\left\{x . \exists u::\right.\right.$ real. $\left.\left.x=(1-u) *_{R} a+u *_{R} b\right\}\right)$
definition proots-line-card:: complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots-line-card $p$ st $t t=\operatorname{card}($ proots-within $p($ open-segment st tt) $)$
definition proots-unbounded-line-card:: complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where
proots-unbounded-line-card p st tt $=$ card (proots-within $p$ (unbounded-line st tt))
definition proots-unbounded-line :: complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where

```
    proots-unbounded-line p st tt = proots-count p(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 $t t)$ ) $=$
(let pc $=$ pcompose $p[: s t, t t-s t:]$;
$p R=$ map-poly Re $p c ;$
$p I=$ map-poly Im pc;
$g=g c d p R p I$
in changes-itv-smods $01 \mathrm{~g}($ pderiv $g))($ is ? $L=? R)$
proof -
define $p c p R p I g$ where
$p c=$ pcompose $p[: s t, t t-s t:]$ and
$p R=$ map-poly Re pc and
$p I=$ map-poly $\operatorname{Im} p c$ and
$g=g c d p R p I$
have poly-iff:poly $g t=0 \longleftrightarrow$ poly pc $t=0$ for $t$
proof -
have poly $g t=0 \longleftrightarrow$ poly $p R t=0 \wedge$ poly $p I t=0$
unfolding $g$-def using poly-gcd-0-iff by auto
also have $\ldots \longleftrightarrow$ poly pc $t=0$
proof -
have cpoly-of $p R p I=p c$
unfolding $p c$-def $p R$-def $p I$-def using cpoly-of-decompose by auto
then show ?thesis using poly-cpoly-of-real-iff by blast
qed
finally show ?thesis by auto
qed
have ? $R=$ changes-itv-smods 01 g (pderiv $g)$
unfolding $p c$-def $g$-def $p I$-def $p R$-def by (auto simp add:Let-def)
also have $\ldots=\operatorname{card}\{t$. poly $g t=0 \wedge 0<t \wedge t<1\}$
proof -
have poly g $0 \neq 0$
using poly-iff[of 0] assms unfolding pc-def by (auto simp add:poly-pcompose)
moreover have poly g $1 \neq 0$
using poly-iff[of 1] assms unfolding pc-def by (auto simp add:poly-pcompose)
ultimately show ?thesis using sturm-interval[of 01 g$]$ by auto
qed
also have $\ldots=$ card $\{t::$ real. poly $p c($ of-real $t)=0 \wedge 0<t \wedge t<1\}$
unfolding poly-iff by simp
also have ... $=$ ? $L$
proof (cases st=tt)
case True
then show ?thesis unfolding pc-def poly-pcompose using 〈poly ptt $\neq 0$ 〉
by auto
next
case False
define $f f$ where $f f=(\lambda t::$ real. st $+t *(t t-s t))$

```
    define \(l l\) where \(l l=\{t\). poly pc (complex-of-real \(t)=0 \wedge 0<t \wedge t<1\}\)
    have \(f f\) ' \(l l=\) proots-within \(p\) (open-segment st \(t t\) )
    proof (rule equalityI)
    show ff' \(l l \subseteq\) proots-within \(p\) (open-segment st tt)
        unfolding ll-def ff-def pc-def poly-pcompose
        by (auto simp add:in-segment False scaleR-conv-of-real algebra-simps)
    next
    show proots-within \(p\) (open-segment st \(t t) \subseteq f f\) ' \(l l\)
    proof clarify
        fix \(x\) assume asm: \(x \in\) proots-within \(p\) (open-segment st tt)
        then obtain \(u\) where \(0<u\) and \(u<1\) and \(u: x=(1-u) *_{R} s t+u *_{R} t t\)
            by (auto simp add:in-segment)
        then have poly \(p\left((1-u) *_{R} s t+u *_{R} t t\right)=0\) using asm by simp
        then have \(u \in l l\)
            unfolding \(l l\)-def pc-def poly-pcompose
            by (simp add:scaleR-conv-of-real algebra-simps \(\langle 0<u\rangle\langle u<1\rangle)\)
        moreover have \(x=\int f u\)
        unfolding ff-def using \(u\) by (auto simp add:algebra-simps scaleR-conv-of-real)
            ultimately show \(x \in f f\) ' \(l l\) by (rule rev-image-eqI[of \(u\) ])
        qed
    qed
    moreover have inj-on ff \(l l\)
    unfolding ff-def using False inj-on-def by fastforce
    ultimately show ?thesis unfolding ll-def
    using card-image \([\) of ff \(]\) by fastforce
    qed
    finally show ?thesis by simp
qed
lemma unbounded-line-closed-segment: closed-segment \(a b \subseteq\) unbounded-line \(a b\)
    unfolding unbounded-line-def closed-segment-def by auto
lemma card-proots-unbounded-line:
    assumes \(s t \neq t t\)
    shows card (proots-within \(p\) (unbounded-line st tt)) \(=\)
            (let \(p c=\) pcompose \(p[: s t, t t-s t:]\);
                        \(p R=\) map-poly Re \(p c ;\)
                        \(p I=\) map-poly \(\operatorname{Im} p c\);
                        \(g=g c d p R p I\)
            in nat (changes- \(R\)-smods \(g(\) pderiv \(g)))(\) is \(? L=? R)\)
proof -
    define \(p c p R p I g\) where
        \(p c=p\) compose \(p[: s t, t t-s t:]\) and
        \(p R=\) map-poly Re pc and
        \(p I=\) map-poly \(\operatorname{Im} p c\) and
        \(g=g c d p R p I\)
    have poly-iff:poly \(g t=0 \longleftrightarrow\) poly pc \(t=0\) for \(t\)
    proof -
        have poly \(g t=0 \longleftrightarrow\) poly \(p R \quad t=0 \wedge\) poly \(p I t=0\)
```

unfolding $g$-def using poly-gcd-0-iff by auto also have $\ldots \longleftrightarrow$ poly pc $t=0$
proof -
have cpoly-of $p R p I=p c$
unfolding $p c$-def $p R$-def $p I$-def using cpoly-of-decompose by auto
then show ?thesis using poly-cpoly-of-real-iff by blast
qed
finally show ?thesis by auto
qed
have $? R=$ nat (changes- $R$-smods $g(p d e r i v g))$
unfolding $p c$-def $g$-def $p I$-def $p R$-def by (auto simp add:Let-def)
also have $\ldots=$ card $\{t$. poly $g t=0\}$
using sturm- $R[$ of $g]$ by simp
also have $\ldots=$ card $\{t:$ :real. poly pc $t=0\}$
unfolding poly-iff by simp
also have $\ldots=$ ? $L$
proof (cases st=tt)
case True
then show ?thesis unfolding pc-def poly-pcompose unbounded-line-def using
assms
by (auto simp add:proots-within-def)
next
case False
define $f f$ where $f f=(\lambda t:$ :real. st $+t *(t t-s t))$
define $l l$ where $l l=\{t$. poly pc (complex-of-real $t)=0\}$
have $f f$ ' $l l=$ proots-within $p$ (unbounded-line st $t t$ )
proof (rule equalityI)
show ff' $l l \subseteq$ proots-within $p$ (unbounded-line st tt)
unfolding $l l$-def ff-def pc-def poly-pcompose
by (auto simp add:unbounded-line-def False scaleR-conv-of-real algebra-simps)
next
show proots-within $p$ (unbounded-line st $t t) \subseteq f f$ ' $l l$
proof clarify
fix $x$ assume asm: $x \in$ proots-within $p$ (unbounded-line st tt)
then obtain $u$ where $u: x=(1-u) *_{R}$ st $+u *_{R} t t$
by (auto simp add:unbounded-line-def)
then have poly $p\left((1-u) *_{R} s t+u *_{R} t t\right)=0$ using asm by simp
then have $u \in l l$
unfolding $l l$-def $p c$-def poly-pcompose
by (simp add:scaleR-conv-of-real algebra-simps unbounded-line-def)
moreover have $x=f f u$
unfolding ff-def using $u$ by (auto simp add:algebra-simps scaleR-conv-of-real)
ultimately show $x \in f f$ ' $l l$ by (rule rev-image-eqI[of u])
qed
qed
moreover have inj-on ff $l l$
unfolding ff-def using False inj-on-def by fastforce
ultimately show ?thesis unfolding $l l$-def

```
        using card-image[of ff] by metis
    qed
    finally show ?thesis by simp
qed
lemma proots-count-gcd-eq:
    fixes p::complex poly and st tt::complex
        and g::real poly
    defines pc \equiv pcompose p [:st,tt - st:]
    defines pR\equiv map-poly Re pc and pI\equiv map-poly Im pc
    defines g \equivgcd pR pI
    assumes st\not=tt p\not=0
        and s1-def:s1 = (\lambdax. poly [:st, tt - st:] (of-real x))'s2
    shows proots-count p s1 = proots-count g s2
proof -
    have [simp]: g\not=0 pc\not=0
    proof -
        show pc\not=0 using assms pc-def pcompose-eq-0
            by (metis cancel-comm-monoid-add-class.diff-cancel degree-pCons-eq-if
                        diff-eq-diff-eq less-nat-zero-code pCons-eq-0-iff zero-less-Suc)
    then have pR\not=0\vee pI\not=0 unfolding pR-def pI-def by (metis cpoly-of-decompose
map-poly-0)
    then show g\not=0 unfolding g-def by simp
    qed
    have order-eq:order t g = order t pc for t
    apply (subst order-cpoly-gcd-eq[of pR pI,folded g-def,symmetric])
    subgoal using < g\not=0〉 unfolding g-def by simp
    subgoal unfolding pR-def pI-def by (simp add:cpoly-of-decompose[symmetric])
    done
    have proots-count g s2 = proots-count (map-poly complex-of-real g)
                                    (of-real's2)
    apply (subst proots-count-of-real)
    by auto
    also have ... = proots-count pc (of-real'sQ)
    apply (rule proots-count-cong)
    by (auto simp add: map-poly-order-of-real order-eq)
    also have ... = proots-count p s1
    unfolding pc-def s1-def
    apply (subst proots-pcompose)
    using <st\not=tt\rangle\langlep\not=0\rangle by (simp-all add:image-image)
    finally show ?thesis by simp
qed
lemma proots-unbounded-line:
    assumes st\not=tt p\not=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 -
    define pc pR pIg where
        pc = pcompose p [:st,tt-st:] and
        pR = map-poly Re pc and
        pI= map-poly Im pc and
        g = gcd pR pI
    have [simp]: g\not=0 pc\not=0
    proof -
        show pc\not=0 using assms(1) assms(2) pc-def pcompose-eq-0
            by (metis cancel-comm-monoid-add-class.diff-cancel degree-pCons-eq-if
                diff-eq-diff-eq less-nat-zero-code pCons-eq-O-iff zero-less-Suc)
    then have pR\not=0\vee pI\not=0 unfolding pR-def pI-def by (metis cpoly-of-decompose
map-poly-0)
    then show }g\not=0\mathrm{ unfolding g-def by simp
    qed
    have order-eq:order t g=order tpc for t
    apply (subst order-cpoly-gcd-eq[of pR pI,folded g-def,symmetric])
    subgoal using < g\not=0\rangle unfolding g-def by simp
    subgoal unfolding pR-def pI-def by (simp add:cpoly-of-decompose[symmetric])
    done
    have ?R = nat (changes-R-smods-ext g (pderiv g))
    unfolding pc-def g-def pI-def pR-def by (auto simp add:Let-def)
    also have ... = proots-count g UNIV
    using sturm-ext-R[OF<g\not=0`] by auto
    also have ... = proots-count (map-poly complex-of-real g)(of-real'UNIV)
    apply (subst proots-count-of-real)
    by auto
    also have ... = proots-count (map-poly complex-of-real g) {x. Im x = 0 }
    apply (rule arg-cong2[where f=proots-count])
    using Reals-def complex-is-Real-iff by auto
    also have ... = proots-count pc {x. Im x = 0}
    apply (rule proots-count-cong)
    apply (metis (mono-tags) Im-complex-of-real Re-complex-of-real }\langleg\not=0\rangle\mathrm{ com-
plex-surj
                    map-poly-order-of-real mem-Collect-eq order-eq)
    by auto
    also have ... = proots-count p (unbounded-line st tt)
proof -
    have poly [:st, tt - st:]' {x. Im x=0} = unbounded-line st tt
        unfolding unbounded-line-def
        apply safe
        subgoal for - }
            apply (rule-tac x=Re x in exI)
            apply (simp add:algebra-simps)
            by (simp add: mult.commute scaleR-complex.code times-complex.code)
```

```
        subgoal for - u
            apply (rule rev-image-eqI[of of-real u])
            by (auto simp:scaleR-conv-of-real algebra-simps)
        done
    then show ?thesis
        unfolding pc-def
        apply (subst proots-pcompose)
        using <p\not=0\rangle\langlest\not=tt\rangle by auto
    qed
    finally show ?thesis by simp
qed
lemma proots-unbounded-line-card-code[code]:
    proots-unbounded-line-card p st tt =
        (if st\not=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.")
                            (\lambda-. proots-unbounded-line-card p st tt))
    unfolding proots-unbounded-line-card-def using card-proots-unbounded-line[of st
tt p] by auto
lemma proots-unbounded-line-code[code]:
    proots-unbounded-line p st tt =
        ( if st\not=tt then
        if p\not=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')
                    (\lambda-. proots-unbounded-line p st tt)
        else
Code.abort (STR "proots-unbounded-line fails due to invalid
hyperplanes.")
( \(\lambda\)-. proots-unbounded-line \(p\) st \(t t\) ) )
unfolding proots-unbounded-line-def using proots-unbounded-line by auto
```


### 2.11 Checking if there a polynomial root on a closed segment

definition no-proots-line::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ bool where no-proots-line p st tt $=($ proots-within $p($ closed-segment st tt $)=\{ \})$
lemma no-proots-line-code[code]: no-proots-line $p$ st $t=$ (if poly $p$ st $\neq 0 \wedge$ poly $p$ $t t \neq 0$ then

$$
\begin{aligned}
& \text { (let } p c=\text { pcompose } p[: s t, \text { tt }-s t:] \\
& \quad p R=\text { map-poly Re } p c \\
& \quad p I=\text { map-poly } \operatorname{Im} p c \\
& \quad g=\text { gcd } p R \text { pI } \\
& \text { in if changes-itv-smods } 01 \mathrm{~g}(\text { pderiv } g)=0 \text { then True else False })
\end{aligned}
$$

else False)
(is $? L=? R)$
proof (cases poly p st $\neq 0 \wedge$ poly ptt $\neq 0$ )
case False
thus ?thesis unfolding no-proots-line-def by auto
next
case True
then have poly $p$ st $\neq 0$ poly $p$ tt $\neq 0$ by auto
define $p c p R p I g$ where
$p c=$ pcompose $p[: s t, t t-s t:]$ and
$p R=$ map-poly Re pc and
$p I=$ map-poly $\operatorname{Im} p c$ and
$g=g c d p R p I$
have poly-iff:poly $g t=0 \longleftrightarrow$ poly pc $t=0$ for $t$
proof -
have poly $g t=0 \longleftrightarrow$ poly $p R \quad t=0 \wedge$ poly $p I t=0$
unfolding $g$-def using poly-gcd-0-iff by auto
also have $\ldots \longleftrightarrow$ poly pc $t=0$
proof -
have cpoly-of $p R p I=p c$
unfolding $p c$-def $p R$-def $p I$-def using cpoly-of-decompose by auto
then show ?thesis using poly-cpoly-of-real-iff by blast
qed
finally show? ?thesis by auto
qed
have $? R=($ changes-itv-smods $01 g(p d e r i v g)=0)$
using True unfolding $p c$-def $g$-def $p I$-def $p R$-def
by (auto simp add:Let-def)
also have $\ldots=(\operatorname{card}\{x$. poly $g x=0 \wedge 0<x \wedge x<1\}=0)$
proof -
have poly g $0 \neq 0$
using poly-iff[of 0] True unfolding pc-def by (auto simp add:poly-pcompose)
moreover have poly g $1 \neq 0$
using poly-iff[of 1] True unfolding pc-def by (auto simp add:poly-pcompose)
ultimately show ?thesis using sturm-interval[ of 01 g ] by auto
qed
also have $\ldots=(\{x$. poly $g($ of-real $x)=0 \wedge 0<x \wedge x<1\}=\{ \})$
proof -
have $g \neq 0$
proof (rule ccontr)

```
        assume \(\neg g \neq 0\)
        then have poly pc \(0=0\)
        using poly-iff \([\) of 0\(]\) by auto
    then show False using True unfolding pc-def by (auto simp add:poly-pcompose)
    qed
    from poly-roots-finite[OF this] have finite \(\{x\). poly \(g x=0 \wedge 0<x \wedge x<1\}\)
        by auto
    then show ?thesis using card-eq-0-iff by auto
    qed
    also have \(\ldots=\) ? \(L\)
    proof -
        have \((\exists t\). poly \(g(\) of-real \(t)=0 \wedge 0<t \wedge t<1) \longleftrightarrow\)
            \((\exists t::\) real. poly pc \((\) of-real \(t)=0 \wedge 0<t \wedge t<1)\)
        using poly-iff by auto
    also have \(\ldots \longleftrightarrow(\exists x . x \in\) closed-segment st tt \(\wedge\) poly \(p x=0)\)
    proof
        assume \(\exists t\). poly pc (complex-of-real \(t)=0 \wedge 0<t \wedge t<1\)
        then obtain \(t\) where \(*:\) poly pc (of-real \(t)=0\) and \(0<t t<1\) by auto
        define \(x\) where \(x=p o l y[: s t, t t-s t:] t\)
    have \(x \in\) closed-segment st \(t t\) using \(\langle 0<t\rangle\langle t<1\rangle\) unfolding \(x\)-def in-segment
        by (intro exI[where \(x=t]\), auto simp add: algebra-simps scaleR-conv-of-real)
        moreover have poly p \(x=0\) using \(*\) unfolding pc-def \(x\)-def
            by (auto simp add:poly-pcompose)
            ultimately show \(\exists x . x \in\) closed-segment st \(t t \wedge\) poly \(p x=0\) by auto
    next
    assume \(\exists x . x \in\) closed-segment st \(t t \wedge\) poly \(p x=0\)
    then obtain \(x\) where \(x \in\) closed-segment st tt poly \(p x=0\) by auto
    then obtain \(t::\) real where \(*: x=(1-t) *_{R}\) st \(+t *_{R} t t\) and \(0 \leq t t \leq 1\)
            unfolding in-segment by auto
    then have \(x=\) poly \([: s t, t t-s t:] t\) by (auto simp add: algebra-simps scaleR-conv-of-real)
        then have poly pc (complex-of-real \(t)=0\)
            using 〈poly \(p x=0\rangle\) unfolding \(p c\)-def by (auto simp add:poly-pcompose)
            moreover have \(t \neq 0 \quad t \neq 1\) using True * 〈poly \(p x=0\) 〉 by auto
            then have \(0<t t<1\) using \(\langle 0 \leq t\rangle\langle t \leq 1\rangle\) by auto
            ultimately show \(\exists t\). poly \(p c\) (complex-of-real \(t\) ) \(=0 \wedge 0<t \wedge t<1\) by
auto
    qed
    finally show? ?thesis
        unfolding no-proots-line-def proots-within-def
        by blast
    qed
    finally show ?thesis by simp
qed
```


## 2．12 Number of roots on a bounded open segment

definition proots－line：：complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots－line $p$ st $t t=$ proots－count $p$（open－segment st $t t)$

```
lemma proots-line-commute:
    proots-line p st tt = proots-line p tt st
    unfolding proots-line-def by (simp add: open-segment-commute)
lemma proots-line-smods:
    assumes poly p st \not=0 poly p tt \not=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 -
    have p\not=0 using assms(2) poly-0 by blast
    define pc pR pI g}\mathrm{ where
        pc= pcompose p[:st,tt-st:] and
        pR = map-poly Re pc and
        pI = map-poly Im pc and
        g = gcd pR pI
    have [simp]: g\not=0 pc\not=0
    proof -
        show pc\not=0
            by (metis assms(1) coeff-pCons-0 pCons-0-0 pc-def pcompose-coeff-0)
            then have }pR\not=0\veepI\not=0\mathrm{ unfolding }pR\mathrm{ -def pI-def
            by (metis cpoly-of-decompose map-poly-0)
    then show }g\not=0\mathrm{ unfolding }g\mathrm{ -def by simp
qed
have order-eq:order t g=order tpc for t
    apply (subst order-cpoly-gcd-eq[of pR pI,folded g-def,symmetric])
    subgoal using < g\not=0> unfolding g-def by simp
    subgoal unfolding pR-def pI-def by (simp add:cpoly-of-decompose[symmetric])
    done
have poly-iff:poly g t=0 \longleftrightarrow poly pc t=0 for t
    using order-eq by (simp add: order-root)
have poly g 0 = 0 poly g 1 \not=0
    unfolding poly-iff pc-def
    using assms by (simp-all add:poly-pcompose)
have ?R = changes-itv-smods-ext 01g(pderiv g)
    unfolding Let-def
    apply (fold pc-def g-def pI-def pR-def)
    using assms changes-itv-smods-ext-geq-0[OF - <poly g 0\not=0〉\langlepoly g 1\not=0〉]
    by auto
also have ... = int (proots-count g{x. 0<x\wedge x<1})
    apply (rule sturm-ext-interval[symmetric])
    by simp fact+
```

```
    also have ... = int (proots-count p (open-segment st tt))
    proof -
    define f}\mathrm{ where f=( }\lambdax\mathrm{ . poly [:st, tt - st:] (complex-of-real x))
    have }x\inf\mathrm{ ' {x. 0 < x^ x<1} if xGopen-segment st tt for }
    proof -
    obtain u where u:u>0 u<1x=(1-u)*R}st+u\mp@subsup{*}{R}{}t
        using <x\inopen-segment st tt> unfolding in-segment by auto
    show ?thesis
        apply (rule rev-image-eqI[where }x=u]\mathrm{ )
        using u unfolding f-def
        by (auto simp:algebra-simps scaleR-conv-of-real)
    qed
    moreover have x\inopen-segment st tt if x\inf'{x.0<x\wedge x<1} for x
        using that <st }\not=tt\rangle\mathrm{ unfolding in-segment f-def
        by (auto simp:scaleR-conv-of-real algebra-simps)
    ultimately have open-segment st tt =f'{x.0<x\wedgex<1}
        by auto
    then have proots-count p (open-segment st tt)
                = proots-count g{x. 0<x\wedgex<1}
        using proots-count-gcd-eq[OF <st\not=tt\rangle\langlep\not=0\rangle,
                folded pc-def pR-def pI-def g-def] unfolding f-def
    by auto
    then show ?thesis by auto
qed
also have ... =proots-line p st tt
    unfolding proots-line-def by simp
    finally show ?thesis by simp
qed
lemma proots-line-code[code]:
    proots-line p st tt =
        (if poly p st }\not=0\wedge\mathrm{ poly p tt }\not=0\mathrm{ then
            (if st\not=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")
                                    (\lambda-. proots-line p st tt)) (is ?L = ?R)
proof (cases poly p st \not=0 ^ poly p tt \not=0^ st\not=tt)
    case False
    moreover have ?thesis if st=tt p\not=0
        using that unfolding proots-line-def by auto
    ultimately show ?thesis by fastforce
next
```

case True
then show ?thesis using proots-line-smods by auto
qed
end
theory Count-Half-Plane imports
Count-Line
begin

### 2.13 Polynomial roots on the upper half-plane

```
definition proots-upper ::complex poly }=>\mathrm{ nat where
    proots-upper p= proots-count p {z.Im z>0}
- Roots counted WITHOUT multiplicity
definition proots-upper-card::complex poly }=>\mathrm{ nat where
    proots-upper-card p = card (proots-within p {x. Im x>0})
lemma Im-Ln-tendsto-at-top: ((\lambdax.Im (Ln (Complex a x))) \longrightarrow pi/2 ) at-top
proof (cases a=0)
    case False
    define f}\mathrm{ where f=( }\lambda\mathrm{ x. if a>0 then arctan (x/a) else arctan (x/a)+pi)
    define g}\mathrm{ where g=( }\lambdax\mathrm{ . Im (Ln (Complex a x)))
    have (f\longrightarrowpi / 2) at-top
    proof (cases a>0)
    case True
    then have }(f\longrightarrowpi/2) at-top \longleftrightarrow ((\lambdax.\operatorname{arctan }(x*\mathrm{ inverse a ) ) }\longrightarrowp
/ 2) at-top
            unfolding f
        also have ... \longleftrightarrow(arctan \longrightarrowpi/2) at-top
        apply (subst filterlim-at-top-linear-iff[of inverse a arctan 0 nhds (pi/2),simplified])
            using True by auto
    also have ... using tendsto-arctan-at-top .
    finally show ?thesis.
    next
    case False
    then have }(f\longrightarrowpi/2) at-top \longleftrightarrow((\lambdax. arctan (x* inverse a) + pi)
pi / 2) at-top
            unfolding }f\mathrm{ -def field-class.field-divide-inverse by auto
    also have ... \longleftrightarrow((\lambdax. arctan (x* inverse a))\longrightarrow - pi/ 2) at-top
        apply (subst tendsto-add-const-iff[of -pi,symmetric])
        by auto
    also have }\ldots\longleftrightarrow(\mathrm{ arctan }\longrightarrow-pi/2) at-bo
        apply (subst filterlim-at-top-linear-iff[of inverse a arctan 0,simplified])
        using False <a\not=0> by auto
    also have ... using tendsto-arctan-at-bot by simp
    finally show ?thesis.
qed
```

```
moreover have \(\forall_{F} x\) in at-top. \(f x=g x\)
    unfolding \(f\)-def \(g\)-def using \(\langle a \neq 0\rangle\)
    apply (subst Im-Ln-eq)
    subgoal for \(x\) using Complex-eq-0 by blast
    subgoal unfolding eventually-at-top-linorder by auto
    done
ultimately show ?thesis
    using tendsto-cong[offgat-top] unfolding \(g\)-def by auto
next
    case True
    show ?thesis
    apply (rule tendsto-eventually)
    apply (rule eventually-at-top-linorderI[of 1])
    using True by (subst Im-Ln-eq,auto simp add:Complex-eq-0)
qed
lemma Im-Ln-tendsto-at-bot: \(((\lambda x\). Im \((\operatorname{Ln}(C o m p l e x ~ a ~ x))) \longrightarrow-p i / 2)\) at-bot
proof (cases \(a=0\) )
    case False
    define \(f\) where \(f=(\lambda x\). if \(a>0\) then \(\arctan (x / a)\) else arctan \((x / a)-p i)\)
    define \(g\) where \(g=(\lambda x\). Im \((\operatorname{Ln}(\) Complex a \(x)))\)
    have \((f \longrightarrow-p i / 2)\) at-bot
    proof (cases \(a>0\) )
    case True
    then have \((f \longrightarrow-p i / 2)\) at-bot \(\longleftrightarrow((\lambda x . \arctan (x *\) inverse \(a)) \longrightarrow\)
- pi / 2) at-bot
            unfolding \(f\)-def field-class.field-divide-inverse by auto
    also have \(\ldots \longleftrightarrow\) (arctan \(\longrightarrow-p i / 2)\) at-bot
            apply (subst filterlim-at-bot-linear-iff [of inverse a arctan 0 ,simplified \(]\) )
            using True by auto
    also have ... using tendsto-arctan-at-bot by simp
    finally show ?thesis .
    next
    case False
    then have \((f \longrightarrow-p i / 2)\) at-bot \(\longleftrightarrow((\lambda x . \arctan (x *\) inverse \(a)-p i)\)
        \(\rightarrow-p i / 2) a t-b o t\)
        unfolding \(f\)-def field-class.field-divide-inverse by auto
    also have \(\ldots \longleftrightarrow((\lambda x . \arctan (x *\) inverse \(a)) \longrightarrow p i / 2)\) at-bot
        apply (subst tendsto-add-const-iff[of pi,symmetric])
        by auto
    also have \(\ldots \longleftrightarrow\) (arctan \(\longrightarrow\) pi / 2) at-top
        apply (subst filterlim-at-bot-linear-iff[of inverse a arctan 0,simplified])
        using False \(\langle a \neq 0\) 〉 by auto
    also have ... using tendsto-arctan-at-top by simp
    finally show ?thesis .
qed
moreover have \(\forall_{F} x\) in at-bot. \(f x=g x\)
    unfolding \(f\)-def \(g\)-def using \(\langle a \neq 0\rangle\)
```

```
    apply (subst Im-Ln-eq)
    subgoal for x using Complex-eq-0 by blast
    subgoal unfolding eventually-at-bot-linorder by (auto intro:exI[where x=-1])
    done
    ultimately show ?thesis
    using tendsto-cong[of fg at-bot] unfolding g-def by auto
next
    case True
    show ?thesis
    apply (rule tendsto-eventually)
    apply (rule eventually-at-bot-linorderI[of - 1])
    using True by (subst Im-Ln-eq,auto simp add:Complex-eq-0)
qed
lemma Re-winding-number-tendsto-part-circlepath:
    shows ((\lambdar.Re (winding-number (part-circlepath z0 r 0 pi ) a)) \longrightarrow 1/2 )
at-top
proof (cases Im z0\leqIm a)
    case True
    define g1 where g1=(\lambdar. part-circlepath z0 r 0 pi)
    define g2 where g2=(\lambdar. part-circlepath z0 r pi (2*pi))
    define f1 where f1=(\lambdar. Re (winding-number (g1 r ) a))
    define f2 where f2=(\lambdar.Re (winding-number (g2 r) a))
    have (f2 \longrightarrow1/2 ) at-top
    proof -
    define h1 where h1 = (\lambdar. Im (Ln (Complex (Im a-Im z0) (Rez0 - Re a
+r))))
    define h2 where h2 = ( \lambdar. Im (Ln (Complex ( Im a - Im z0) (Rez0 - Re a
-r))))
    have }\mp@subsup{\forall}{F}{}x\mathrm{ in at-top. f% }x=(h1x-h2x)/(2*pi
    proof (rule eventually-at-top-linorderI[of cmod (a-z0) + 1])
        fix r assume asm:r \geq cmod (a-z0)+1
        have Im p\leqIm a when p\inpath-image (g2 r) for p
        proof -
            obtain t where p-def:p=z0 + of-real r * exp (i * of-real t) and pi\leqtt\leq2*pi
            using <p\inpath-image (g2 r)>
            unfolding g2-def path-image-part-circlepath[of pi 2*pi,simplified]
            by auto
            then have Im p=Im z0 + sin t*r by (auto simp add:Im-exp)
            also have ... \leqIm z0
            proof -
                    have sin t\leq0 using <pi\leqt><t\leq2*pi\rangle sin-le-zero by fastforce
                    moreover have r\geq0
            using asm by (metis add.inverse-inverse add.left-neutral add-uminus-conv-diff
                        diff-ge-0-iff-ge norm-ge-zero order-trans zero-le-one)
                    ultimately have sin t*r\leq0 using mult-le-0-iff by blast
                    then show ?thesis by auto
            qed
            also have ... \leqIm a using True .
```

```
    finally show ?thesis.
    qed
    moreover have valid-path ( \(g 2 r\) ) unfolding \(g 2\)-def by auto
    moreover have \(a \notin\) path-image ( \(g 2 r\) )
    unfolding 92 -def
    apply (rule not-on-circlepathI)
    using asm by auto
    moreover have [symmetric]:Im \((\operatorname{Ln}(\mathrm{i} *\) pathfinish \((g 2 r)-\mathrm{i} * a))=h 1 r\)
    unfolding \(h 1\)-def \(g 2\)-def
    apply (simp only:pathfinish-pathstart-partcirclepath-simps)
    apply (subst (4 10) complex-eq)
    by (auto simp add:algebra-simps Complex-eq)
    moreover have [symmetric]:Im \((\operatorname{Ln}(\mathrm{i} *\) pathstart \((g 2 r)-\mathrm{i} * a))=h 2 r\)
    unfolding h2-def g2-def
    apply (simp only:pathfinish-pathstart-partcirclepath-simps)
    apply (subst (4 10) complex-eq)
    by (auto simp add:algebra-simps Complex-eq)
    ultimately show \(f 2 r=(h 1 r-h 2 r) /(2 * p i)\)
    unfolding \(f 2\)-def
    apply (subst Re-winding-number-half-lower)
    by (auto simp add:exp-Euler algebra-simps)
qed
moreover have \(((\lambda x .(h 1 x-h 2 x) /(2 * p i)) \longrightarrow 1 / 2)\) at-top
proof -
    have ( \(h 1 \longrightarrow p i / 2\) ) at-top
        unfolding h1-def
    apply (subst filterlim-at-top-linear-iff[of 1 - Re a - Re zo ,simplified,symmetric])
        using Im-Ln-tendsto-at-top by (simp del:Complex-eq)
    moreover have ( \(h 2 \longrightarrow-p i / 2\) ) at-top
        unfolding \(h 2\)-def
    apply (subst filterlim-at-bot-linear-iff[of - 1 - - Re a + Re zo ,simplified,symmetric])
        using Im-Ln-tendsto-at-bot by (simp del:Complex-eq)
    ultimately have \(((\lambda x . h 1 x-h 2 x) \longrightarrow p i)\) at-top
        by (auto intro: tendsto-eq-intros)
    then show ?thesis
    by (auto intro: tendsto-eq-intros)
qed
ultimately show ?thesis by (auto dest:tendsto-cong)
qed
moreover have \(\forall_{F} r\) in at-top. f2 \(r=1-f 1 r\)
proof (rule eventually-at-top-linorderI[of \(\operatorname{cmod}(a-z 0)+1])\)
    fix \(r\) assume asm: \(r \geq \operatorname{cmod}(a-z 0)+1\)
    have \(f 1 r+f 2 r=\operatorname{Re}(\) winding-number \((g 1 r+++g 2 r) a)\)
        unfolding f1-def f2-def g1-def g2-def
        apply (subst winding-number-join)
        using asm by (auto intro!:not-on-circlepathI)
also have \(\ldots=\operatorname{Re}(\) winding-number \((\) circlepath z0 r) a)
```

```
    proof -
    have g1r+++ g2 r = circlepath z0 r
            unfolding circlepath-def g1-def g2-def joinpaths-def part-circlepath-def
linepath-def
            by (auto simp add:field-simps)
            then show ?thesis by auto
    qed
    also have ... = 1
    proof -
        have winding-number (circlepath z0 r) a = 1
            apply (rule winding-number-circlepath)
            using asm by auto
        then show ?thesis by auto
    qed
    finally have f1 r+f2 r=1.
    then show f2 r=1 - f1 r by auto
    qed
    ultimately have ((\lambdar.1 - f1r)\longrightarrow1/2 ) at-top
    using tendsto-cong[of f2 \lambdar.1 - f1r at-top] by auto
    then have (f1 \longrightarrow 1/2) at-top
        apply (rule-tac tendsto-minus-cancel)
        apply (subst tendsto-add-const-iff[of 1,symmetric])
        by auto
    then show ?thesis unfolding f1-def g1-def by auto
next
    case False
    define g}\mathrm{ where g=( }\lambdar\mathrm{ . part-circlepath z0 r 0 pi)
    define f}\mathrm{ where f=( }\lambdar\mathrm{ . Re (winding-number (gr)a))
    have (f\longrightarrow1/2) at-top
    proof -
        define h1 where h1 = (\lambdar. Im (Ln (Complex (Im z0-Ima) (Re a - Rez0
+r))))
    define h2 where h2 = (\lambdar. Im (Ln (Complex ( Im z0 -Im a ) (Re a - Re
z0 - r))))
    have }\mp@subsup{\forall}{F}{}x\mathrm{ in at-top. fx=(h1 x - h2 x) / (2 * pi)
    proof (rule eventually-at-top-linorderI[of cmod (a-z0) + 1])
        fix r assume asm:r\geqcmod (a-z0)+1
        have Im p\geqIm a when p\inpath-image (gr) for p
        proof -
            obtain t where p-def:p=z0 + of-real r * exp (i * of-real t) and 0\leqt t\leqpi
                    using < p\inpath-image ( g r )}
                    unfolding g-def path-image-part-circlepath[of 0 pi,simplified]
                    by auto
            then have Im p=Im z0 + sin t*r by (auto simp add:Im-exp)
            moreover have sin t*r\geq0
            proof -
            have sin t\geq0 using <0\leqt\rangle\langlet\leqpi\rangle sin-ge-zero by fastforce
            moreover have r\geq0
            using asm by (metis add.inverse-inverse add.left-neutral add-uminus-conv-diff
```

    qed
    ultimately show ?thesis using False by auto
    qed
    moreover have valid-path ( \(g r\) ) unfolding \(g\)-def by auto
    moreover have \(a \notin\) path-image ( \(g r\) )
        unfolding \(g\)-def
        apply (rule not-on-circlepathI)
        using asm by auto
    moreover have [symmetric]:Im \((\operatorname{Ln}(\mathrm{i} * a-\mathrm{i} * \operatorname{pathfinish}(g r)))=h 1 r\)
        unfolding \(h 1\)-def \(g\)-def
        apply (simp only:pathfinish-pathstart-partcirclepath-simps)
        apply (subst (49) complex-eq)
        by (auto simp add:algebra-simps Complex-eq)
    moreover have [symmetric]:Im (Ln (i *a-i * pathstart \((g r)))=h 2 r\)
        unfolding h2-def \(g\)-def
        apply (simp only:pathfinish-pathstart-partcirclepath-simps)
        apply (subst (49) complex-eq)
        by (auto simp add:algebra-simps Complex-eq)
    ultimately show \(f r=(h 1 r-h 2 r) /(2 * p i)\)
        unfolding \(f\)-def
        apply (subst Re-winding-number-half-upper)
        by (auto simp add:exp-Euler algebra-simps)
    qed
    moreover have \(((\lambda x .(h 1 x-h 2 x) /(2 * p i)) \longrightarrow 1 / 2)\) at-top
    proof -
    have (h1 \(\longrightarrow p i / 2)\) at-top
        unfolding \(h 1\)-def
    apply (subst filterlim-at-top-linear-iff[of \(1--R e a+R e z 0\),simplified,symmetric])
        using Im-Ln-tendsto-at-top by (simp del:Complex-eq)
    moreover have \((h 2 \longrightarrow-p i / 2)\) at-top
        unfolding \(h 2\)-def
    apply (subst filterlim-at-bot-linear-iff [of - 1 - Re a-Re z0 ,simplified,symmetric])
        using Im-Ln-tendsto-at-bot by (simp del:Complex-eq)
    ultimately have \(((\lambda x . h 1 x-h 2 x) \longrightarrow p i)\) at-top
        by (auto intro: tendsto-eq-intros)
    then show?thesis
        by (auto intro: tendsto-eq-intros)
    qed
    ultimately show ?thesis by (auto dest:tendsto-cong)
    qed
then show ?thesis unfolding $f$-def $g$-def by auto
qed
lemma not-image-at-top-poly-part-circlepath:

```
    assumes degree p>0
    shows }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. b&path-image (poly p o part-circlepath z0 r st tt)
proof -
    have finite (proots (p-[:b:]))
        apply (rule finite-proots)
        using assms by auto
    from finite-ball-include[OF this]
    obtain R::real where }R>0\mathrm{ and }R\mathrm{ -ball:proots ( }p-[:b:])\subseteq\mathrm{ ball z0 R by auto
    show ?thesis
    proof (rule eventually-at-top-linorderI[of R])
        fix r assume r\geqR
        show b\not\inpath-image (poly p o part-circlepath z0 r st tt)
            unfolding path-image-compose
    proof clarify
        fix x assume asm:b = poly p x x path-image (part-circlepath z0 r st tt)
        then have x\inproots ( }p-[:b:])\mathrm{ unfolding proots-def by auto
        then have x\inball z0 r using R-ball \langler\geqR\rangle by auto
        then have cmod (x-z0)<r
            by (simp add: dist-commute dist-norm)
        moreover have cmod (x-z0)=r
            using asm(2) in-path-image-part-circlepath }\langleR>0\rangle\langler\geqR\rangle by aut
            ultimately show False by auto
        qed
    qed
qed
lemma not-image-poly-part-circlepath:
    assumes degree p>0
    shows \existsr>0. b\not\inpath-image (poly p o part-circlepath z0 r st tt)
proof -
    have finite (proots (p-[:b:]))
        apply (rule finite-proots)
        using assms by auto
    from finite-ball-include[OF this]
    obtain r::real where r>0 and r-ball:proots (p-[:b:])\subseteq ball z0 r by auto
    have b\not\inpath-image (poly p o part-circlepath z0 r st tt)
    unfolding path-image-compose
    proof clarify
        fix x assume asm:b = poly p x x f path-image (part-circlepath z0 r st tt)
        then have x\inproots (p-[:b:]) unfolding proots-def by auto
        then have x\inball z0 r using r-ball by auto
        then have cmod (x-z0)<r
            by (simp add: dist-commute dist-norm)
            moreover have cmod (x-z0)=r
            using asm(2) in-path-image-part-circlepath }\langler>0\rangle\mathrm{ by auto
            ultimately show False by auto
    qed
    then show ?thesis using <r>0\rangle by blast
qed
```

```
lemma Re-winding-number-poly-part-circlepath:
    assumes degree p>0
    shows ((\lambdar.Re (winding-number (poly p o part-circlepath z0 r 0 pi) 0)) \longrightarrow
degree p/2 ) at-top
using assms
proof (induct rule:poly-root-induct-alt)
    case 0
    then show ?case by auto
next
    case (no-proots p)
    then have False
    using Fundamental-Theorem-Algebra.fundamental-theorem-of-algebra constant-degree
neq0-conv
    by blast
    then show ?case by auto
next
    case (root a p)
    define g}\mathrm{ where g=( ( rr. part-circlepath z0 r 0 pi)
    define q}\mathrm{ where q=[:-a,1:]*p
    define w}\mathrm{ where w = ( }\lambdar\mathrm{ . winding-number (poly q}\circggr)0
    have ?case when degree p=0
    proof -
    obtain pc where pc-def:p=[:pc:] using <degree p=0\rangle degree-eq-zeroE by blast
    then have pc\not=0 using root(2) by auto
    have }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. Re (wr)=Re (winding-number (g r) a)
    proof (rule eventually-at-top-linorderI[of cmod (( pc*a)/pc-z0) + 1])
            fix r::real assume asm:cmod ((pc*a)/pc-z0)+1\leqr
            have wr= winding-number ((\lambdax. pc*x - pc*a)\circ(gr)) 0
                    unfolding w-def pc-def g-def q-def
            apply auto
        by (metis (no-types, opaque-lifting) add.right-neutral mult.commute mult-zero-right
                    poly-0 poly-pCons uminus-add-conv-diff)
            also have ... = winding-number ( gr)a
                    apply (subst winding-number-comp-linear[where b=-pc*a,simplified])
                    subgoal using < }c\not=0\mathrm{ \ .
                    subgoal unfolding g-def by auto
                    subgoal unfolding g-def
                    apply (rule not-on-circlepathI)
                    using asm by auto
                    subgoal using <pc\not=0\rangle by (auto simp add:field-simps)
                    done
            finally have wr= winding-number (g r) a .
            then show Re (wr)=Re (winding-number (gr) a) by simp
    qed
    moreover have ((\lambdar.Re (winding-number (gr)a))\longrightarrow 1/2) at-top
            using Re-winding-number-tendsto-part-circlepath unfolding g-def by auto
    ultimately have ((\lambdar.Re (wr))\longrightarrow1/2) at-top
```

```
        by (auto dest!:tendsto-cong)
    moreover have degree ([:-a,1:]*p)=1 unfolding pc-def using <pc\not=0>
by auto
    ultimately show ?thesis unfolding w-def g-def comp-def q-def by simp
    qed
    moreover have ?case when degree p>0
    proof -
    have }\mp@subsup{\forall}{F}{}r\mathrm{ in at-top. 0 & path-image (poly q ○gr)
        unfolding g-def
        apply (rule not-image-at-top-poly-part-circlepath)
        unfolding q-def using root.prems by blast
    then have }\mp@subsup{\forall}{F}{}r\mathrm{ rin at-top. Re (wr)=Re (winding-number (gr)a)
                +Re (winding-number (poly p\circgr)0)
    proof (rule eventually-mono)
        fix r assume asm:0 & path-image (poly q\circgr)
        define cc where cc=1 / (of-real (2*pi)* i)
        define pf where pf=(\lambdaw. deriv (poly p) w/ poly p w)
        define af where af=(\lambdaw. 1/(w-a))
        have wr=cc* contour-integral (gr) (\lambdaw.deriv (poly q) w / poly q w)
            unfolding w-def
            apply (subst winding-number-comp[of UNIV,simplified])
            using asm unfolding g-def cc-def by auto
        also have ... = cc * contour-integral (gr) (\lambdaw. deriv (poly p)w / poly p w
+1/(w-a))
    proof -
            have contour-integral (g r) (\lambdaw. deriv (poly q) w / poly q w)
            = contour-integral (gr) (\lambdaw. deriv (poly p) w/ poly pw+1/(w-a))
            proof (rule contour-integral-eq)
                fix x assume }x\in\mathrm{ path-image (gr)
                have deriv (poly q) x = deriv (poly p) x* (x-a) + poly p x
                    proof -
                        have poly q = ( }\lambdax.(x-a)* poly p x
                            apply (rule ext)
                    unfolding q-def by (auto simp add:algebra-simps)
                    then show ?thesis
                        apply simp
                        apply (subst deriv-mult[of \lambdax. x-a - poly p])
                        by (auto intro:derivative-intros)
                    qed
                    moreover have poly p x\not=0\wedgex-a\not=0
                    proof (rule ccontr)
                        assume }\neg(\mathrm{ poly p x}\not=0\wedgex-a\not=0
                    then have poly qx=0 unfolding q-def by auto
                    then have 0\inpoly q' path-image ( g r)
                        using <x \in path-image ( g r ) > by auto
                    then show False using < 0 # path-image (poly q\circg r)>
                        unfolding path-image-compose by auto
                    qed
                    ultimately show deriv (poly q) x / poly q x = deriv (poly p) x / poly p x
```

```
+1/(x-a)
            unfolding q-def by (auto simp add:field-simps)
        qed
        then show ?thesis by auto
    qed
    also have ... = cc * contour-integral (g r) (\lambdaw. deriv (poly p) w / poly p w)
        +cc* contour-integral (gr) (\lambdaw.1/(w-a))
    proof (subst contour-integral-add)
        have continuous-on (path-image (g r)) (\lambdaw. deriv (poly p) w)
            unfolding deriv-pderiv by (intro continuous-intros)
        moreover have }\forallw\inpath-image (gr). poly p w\not=
            using asm unfolding q-def path-image-compose by auto
        ultimately show (\lambdaw. deriv (poly p) w / poly p w) contour-integrable-on g
r
        unfolding g-def
                        by (auto intro!: contour-integrable-continuous-part-circlepath continu-
ous-intros)
        show (\lambdaw. 1 / (w-a)) contour-integrable-on gr
            apply (rule contour-integrable-inversediff)
            subgoal unfolding g-def by auto
            subgoal using asm unfolding q-def path-image-compose by auto
            done
    qed (auto simp add:algebra-simps)
    also have ... = winding-number (gr)a+ winding-number (poly pogr) 0
    proof -
        have winding-number (poly pog r) 0
            =cc* contour-integral (gr) (\lambdaw. deriv (poly p)w / poly p w)
            apply (subst winding-number-comp[of UNIV,simplified])
        using <0 # path-image (poly q\circgr)\rangle unfolding path-image-compose q-def
g-def cc-def
            by auto
            moreover have winding-number (gr) a = cc * contour-integral (gr) ( }\lambdaw
1/(w-a))
            apply (subst winding-number-valid-path)
        using <0 # path-image (poly q\circgr)\rangle unfolding path-image-compose q-def
g-def cc-def
                by auto
            ultimately show ?thesis by auto
            qed
            finally show Re (wr)=Re(winding-number (gr)a) + Re(winding-number
(poly p\circgr)0)
            by auto
    qed
    moreover have ((\lambdar. Re (winding-number (gr) a)
                    + Re (winding-number (poly pogr) 0)) \longrightarrow degree q/ 2) at-top
    proof -
        have ((\lambdar.Re (winding-number (gr)a)) \longrightarrow1 / 2) at-top
            unfolding g-def by (rule Re-winding-number-tendsto-part-circlepath)
            moreover have ((\lambdar.Re (winding-number (poly p\circgr)0)) \longrightarrow degree p
```

```
(2) at-top
unfolding g-def by (rule root(1)[OF that])
moreover have degree q}=\mathrm{ degree }p+
unfolding q-def
apply (subst degree-mult-eq)
    using that by auto
ultimately show ?thesis
    by (simp add: tendsto-add add-divide-distrib)
    qed
    ultimately have ((\lambdar. Re (wr)) \longrightarrow degree q/2) at-top
    by (auto dest!:tendsto-cong)
    then show ?thesis unfolding w-def q-def g-def by blast
    qed
    ultimately show ?case by blast
qed
lemma Re-winding-number-poly-linepth:
    fixes pp::complex poly
    defines g}\equiv(\lambdar.poly pp o linepath ( -r) (of-real r)
    assumes lead-coeff pp=1 and no-real-zero:\forall }\forall\in\mathrm{ proots pp. Im }x\not=
    shows ((\lambdar.2*Re (winding-number (gr) 0) + cindex-pathE (gr) 0) \longrightarrow0
) at-top
proof -
    define p where p=map-poly Re pp
    define q}\mathrm{ where q=map-poly Im pp
    define f}\mathrm{ where f=( }\lambdat\mathrm{ . poly q t / poly p t)
    have sgnx-top:sgnx (poly p) at-top = 1
    unfolding sgnx-poly-at-top sgn-pos-inf-def p-def using <lead-coeff pp=1`
    by (subst lead-coeff-map-poly-nz,auto)
    have not-g-image:0 # path-image ( }gr\mathrm{ ) for r
    proof (rule ccontr)
    assume ᄀ0 & path-image (g r)
    then obtain x where poly pp x=0 x\inclosed-segment (- of-real r) (of-real r)
            unfolding g-def path-image-compose of-real-linepath by auto
    then have Im x=0 x\inproots pp
            using closed-segment-imp-Re-Im(2) unfolding proots-def by fastforce+
    then show False using <\forallx\inproots pp. Im x\not=0〉 by auto
    qed
    have arctan-f-tendsto:((\lambdar. (arctan (fr) - arctan (f (-r))) / pi)\longrightarrow0)
at-top
    proof (cases degree p>0)
    case True
    have degree p>degree q
    proof -
        have degree p=degree pp
            unfolding p-def using <lead-coeff pp=1>
            by (auto intro:map-poly-degree-eq)
        moreover then have degree q<degree pp
                unfolding q-def using <lead-coeff pp=1> True
```

```
            by (auto intro!:map-poly-degree-less)
            ultimately show ?thesis by auto
    qed
    then have ( f\longrightarrow 0) at-infinity
        unfolding f-def using poly-divide-tendsto-0-at-infinity by auto
    then have (f\longrightarrow0) at-bot (f\longrightarrow0) at-top
    by (auto elim!:filterlim-mono simp add:at-top-le-at-infinity at-bot-le-at-infinity)
    then have }((\lambdar.\operatorname{arctan}(fr))\longrightarrow0) at-top ((\lambdar.arctan (f (-r)))\longrightarrow0
at-top
    apply -
    subgoal by (auto intro:tendsto-eq-intros)
    subgoal
            apply (subst tendsto-compose-filtermap[of - uminus,unfolded comp-def])
            by (auto intro:tendsto-eq-intros simp add:at-bot-mirror[symmetric])
    done
    then show ?thesis
        by (auto intro:tendsto-eq-intros)
next
    case False
    obtain c where f=( \lambdar.c)
    proof -
        have degree p=0 using False by auto
        moreover have degree q\leqdegree p
        proof -
            have degree }p=\mathrm{ degree }p
            unfolding p-def using <lead-coeff pp=1`
            by (auto intro:map-poly-degree-eq)
            moreover have degree q\leqdegree pp
                    unfolding q-def by simp
            ultimately show ?thesis by auto
        qed
        ultimately have degree q=0 by simp
        then obtain pa qa where p=[:pa:] q=[:qa:]
            using <degree p=0` by (meson degree-eq-zeroE)
        then show ?thesis using that unfolding f-def by auto
    qed
    then show ?thesis by auto
qed
have [simp]:valid-path (gr) path (gr) finite-ReZ-segments (g r) 0 for r
proof -
    show valid-path ( g r ) unfolding g-def
        apply (rule valid-path-compose-holomorphic[where S=UNIV])
        by (auto simp add:of-real-linepath)
    then show path (g r) using valid-path-imp-path by auto
    show finite-ReZ-segments (gr) 0
            unfolding g-def of-real-linepath using finite-ReZ-segments-poly-linepath by
simp
qed
have g-f-eq:Im (grt)/Re (grt)=(fo(\lambdax.2*r*x-r))t for rt
```

```
proof -
    have Im (grt)/Re (grt) = Im ((poly pp o of-real o (\lambdax.2*r*x-r))t)
            / Re ((poly pp o of-real o (\lambdax. 2*r*x - r)) t)
        unfolding g-def linepath-def comp-def
        by (auto simp add:algebra-simps)
    also have ... = (fo(\lambdax.2*r*x-r))t
        unfolding comp-def
        by (simp only:Im-poly-of-real diff-0-right Re-poly-of-real f-def q-def p-def)
    finally show ?thesis.
qed
have ?thesis when proots p={}
proof -
    have \forall}\mp@subsup{}{F}{}r\mathrm{ in at-top. 2 * Re (winding-number (gr) 0) + cindex-pathE (gr) 0
        =(arctan (fr) - arctan (f(-r)))/ pi
    proof (rule eventually-at-top-linorderI[of 1])
        fix r::real assume r\geq1
        have image-pos:\forall p\inpath-image (gr). 0<Re p
        proof (rule ccontr)
            assume }\neg(\forallp\inpath-image (gr).0<Re p
            then obtain t where poly pt\leq0
                unfolding g-def path-image-compose of-real-linepath p-def
                    using Re-poly-of-real
                    apply (simp add:closed-segment-def)
                by (metis not-less of-real-def real-vector.scale-scale scaleR-left-diff-distrib)
            moreover have False when poly p t<0
            proof -
                    have sgnx (poly p) (at-right t)= -1
                    using sgnx-poly-nz that by auto
                    then obtain }x\mathrm{ where }x>t\mathrm{ poly p x=0
                    using sgnx-at-top-IVT[of p t] sgnx-top by auto
                    then have x\inproots p unfolding proots-def by auto
                    then show False using <proots p={}` by auto
            qed
            moreover have False when poly p t=0
                    using <proots p={}> that unfolding proots-def by auto
            ultimately show False by linarith
        qed
        have Re (winding-number (gr) 0) =(Im (Ln (pathfinish (gr))) - Im (Ln
(pathstart (gr))))
                    / (2*pi)
            apply (rule Re-winding-number-half-right[of g r 0,simplified])
            subgoal using image-pos by auto
            subgoal by (auto simp add:not-g-image)
            done
        also have ... =(arctan (fr) - arctan (f(-r)))/(2*pi)
        proof -
            have Im (Ln (pathfinish (g r))) = arctan (fr)
```

```
    proof -
            have Re (pathfinish \((g r))>0\)
            by (auto intro: image-pos[rule-format \(]\) )
            then have \(\operatorname{Im}(\operatorname{Ln}(\) pathfinish \((g r)))\)
                \(=\arctan (\operatorname{Im}(\) pathfinish \((g r)) / \operatorname{Re}(\) pathfinish \((g r)))\)
                by (subst Im-Ln-eq,auto)
            also have \(\ldots=\arctan (f r)\)
            unfolding path-defs by (subst g-f-eq,auto)
            finally show? thesis .
    qed
    moreover have \(\operatorname{Im}(\operatorname{Ln}(\) pathstart \((g r)))=\arctan (f(-r))\)
    proof -
        have \(\operatorname{Re}\) (pathstart \((g r))>0\)
            by (auto intro: image-pos[rule-format \(]\) )
            then have \(\operatorname{Im}(\operatorname{Ln}(\) pathstart \((g r)))\)
            \(=\arctan (\operatorname{Im}(\) pathstart \((g r)) / \operatorname{Re}(\) pathstart \((g r)))\)
            by (subst Im-Ln-eq,auto)
            also have \(\ldots=\arctan (f(-r))\)
            unfolding path-defs by (subst g-f-eq,auto)
            finally show ?thesis .
    qed
    ultimately show ?thesis by auto
qed
                            finally have \(\operatorname{Re}(\) winding-number \((g r) 0)=(\arctan (f r)-\arctan (f\)
\((-r))) /(2 * p i)\).
    moreover have cindex-pathE (g r) \(0=0\)
    proof -
    have cindex-pathE (g r) \(0=\) cindex-pathE (poly pp o of-real o \((\lambda x\). \(2 * r * x\)
\(-r)) 0\)
            unfolding \(g\)-def linepath-def comp-def
            by (auto simp add:algebra-simps)
            also have \(\ldots=\operatorname{cindexE} 01(f o(\lambda x .2 * r * x-r))\)
            unfolding cindex-pathE-def comp-def
            by (simp only:Im-poly-of-real diff-0-right Re-poly-of-real f-def \(q\)-def p-def)
            also have \(\ldots=\) cindexE \((-r) r f\)
            apply (subst cindexE-linear-comp[of 2*r 0 1--r,simplified \(]\) )
            using \(\langle r \geq 1\rangle\) by auto
    also have \(\ldots=0\)
    proof -
            have jumpF \(f(\) at-left \(x)=0\) jumpF \(f(\) at-right \(x)=0\) when \(x \in\{-r . . r\}\)
for \(x\)
            proof -
            have poly \(p x \neq 0\) using 〈proots \(p=\{ \}\) 〉unfolding proots-def by auto
            then show jumpF \(f(\) at-left \(x)=0\) jumpF \(f(\) at-right \(x)=0\)
            unfolding \(f\)-def by (auto intro!: jumpF-not-infinity continuous-intros)
    qed
    then show ?thesis unfolding cindexE-def by auto
    qed
    finally show ?thesis.
```

```
    qed
    ultimately show 2 * Re (winding-number (gr) 0) + cindex-pathE (gr) 0
        =(arctan (fr) - arctan (f(-r))) / pi
        unfolding path-defs by (auto simp add:field-simps)
    qed
    with arctan-f-tendsto show ?thesis by (auto dest:tendsto-cong)
qed
moreover have ?thesis when proots p\not={}
proof -
    define max-r where max-r=Max (proots p)
    define min-r where min-r=Min (proots p)
    have max-r \inproots p min-r \inproots p min-r\leqmax-r and
        min-max-bound:\forall p\inproots p. p\in{min-r..max-r}
    proof -
        have p\not=0
        proof -
            have (0::real)}\not=
                by simp
            then show ?thesis
            by (metis (full-types) <p \equiv map-poly Re pp> assms(2) coeff-0 coeff-map-poly
one-complex.simps(1) zero-complex.sel(1))
        qed
        then have finite (proots p) by auto
        then show max-r \inproots p min-r \inproots p
            using Min-in Max-in that unfolding max-r-def min-r-def by fast+
        then show }\forallp\in\mathrm{ proots p. pe{min-r..max-r}
            using Min-le Max-ge<finite (proots p)〉 unfolding max-r-def min-r-def by
auto
        then show min-r\leqmax-r using <max-r\inproots p` by auto
    qed
```



```
                =(arctan (fr) - arctan (f(-r))) / pi
    proof (rule eventually-at-top-linorderI[of max (norm max-r) (norm min-r) +
1])
    fix r assume r-asm:max (norm max-r) (norm min-r) + 1\leqr
    then have r\not=0 min-r>-r max-r<r by auto
    define }u\mathrm{ where }u=(\mathrm{ min-r +r)/(2*r)
    define v}\mathrm{ where v=(max-r +r)/(2*r)
    have uv:u\in{0..1} v\in{0..1} u\leqv
            unfolding u-def v-def using r-asm 〈min-r\leqmax-r\rangle
            by (auto simp add:field-simps)
    define g1 where g1=subpath 0 u (gr)
    define g2 where g2=subpath uv(gr)
    define g3 where g3=subpath v 1 ( g r)
    have path g1 path g2 path g3 valid-path g1 valid-path g2 valid-path g3
            unfolding g1-def g2-def g3-def using uv
            by (auto intro!:path-subpath valid-path-subpath)
            define wc-add where wc-add = (\lambdag. 2*Re(winding-number g 0) +cin-
dex-pathE g 0)
```

have $w c$-add $(g r)=w c$ - $a d d g 1+w c$-add $g 2+w c$ - $a d d g 3$
proof -
have winding-number ( $g r$ ) $0=$ winding-number $g 10+$ winding-number g2 $0+$ winding-number g3 0
unfolding g1-def g2-def g3-def using $\langle u \in\{0 . .1\}\rangle\langle v \in\{0 . .1\}\rangle$ not-g-image
by (subst winding-number-subpath-combine,simp-all)+
moreover have cindex-pathE ( $g$ r) $0=$ cindex-pathE $g 10+$ cindex-pathE g2 $0+$ cindex-pathE g3 0
unfolding g1-def g2-def g3-def using $\langle u \in\{0 . .1\}\rangle\langle v \in\{0 . .1\}\rangle\langle u \leq v\rangle$ not-g-image
by (subst cindex-pathE-subpath-combine,simp-all)+
ultimately show ?thesis unfolding wc-add-def by auto
qed
moreover have wc-add $g 2=0$
proof -
have 2 * Re (winding-number g2 0) $=-$ cindex-pathE g2 0
unfolding $g 2$-def
apply (rule winding-number-cindex-pathE-aux)
subgoal using $u v$ by (auto intro:finite-ReZ-segments-subpath)
subgoal using $u v$ by (auto intro:valid-path-subpath)
subgoal using Path-Connected.path-image-subpath-subset «\r. path (g $r$ ) not- $g$-image uv
by blast
subgoal unfolding subpath-def $v$-def $g$-def linepath-def using $r$-asm «max-r $\in$ proots $p$ >
by (auto simp add:field-simps Re-poly-of-real p-def)
subgoal unfolding subpath-def $u$-def $g$-def linepath-def using $r$-asm «min-r $\in$ proots $p>$
by (auto simp add:field-simps Re-poly-of-real p-def)
done
then show ?thesis unfolding wc-add-def by auto
qed
moreover have wc-add g1=- arctan $(f(-r)) / p i$
proof -
have $g 1-p q$ :
Re (g1 t) $=$ poly $p(\min -r * t+r * t-r)$
$\operatorname{Im}(g 1 t)=$ poly $q(\min -r * t+r * t-r)$
$\operatorname{Im}(g 1 t) / \operatorname{Re}(g 1 t)=(f o(\lambda x .(\min -r+r) * x-r)) t$
for $t$
proof -
have $g 1 t=$ poly pp (of-real (min-r*t+r*t-r))
using $\langle r \neq 0\rangle$ unfolding $g 1$-def $g$-def linepath-def subpath-def $u$-def $p$-def
by (auto simp add:field-simps)
then show
$\operatorname{Re}(g 1 t)=$ poly $p(\min -r * t+r * t-r)$
$\operatorname{Im}(g 1 t)=$ poly $q($ min- $r * t+r * t-r)$
unfolding $p$-def $q$-def
by (simp only:Re-poly-of-real Im-poly-of-real) +

```
    then show \(\operatorname{Im}(g 1 t) / R e(g 1 t)=(f o(\lambda x .(\min -r+r) * x-r)) t\)
    unfolding \(f\)-def by (auto simp add:algebra-simps)
    qed
have \(\operatorname{Re}\left(\begin{array}{ll}g 1 & 1\end{array}\right)=0\)
    using \(\langle r \neq 0\rangle\) Re-poly-of-real \(\langle m i n-r \in\) proots \(p\rangle\)
    unfolding g1-def subpath-def \(u\)-def \(g\)-def linepath-def
    by (auto simp add:field-simps p-def)
have \(0 \notin\) path-image g1
    by (metis (full-types) path-image-subpath-subset 〈へr. path (g r)〉
    atLeastAtMost-iff g1-def le-less not-g-image subsetCE uv(1) zero-le-one)
```

    have \(w c\)-add-pos:wc-add \(h=-\arctan (p o l y q(-r) / p o l y p(-r)) / p i\)
    when
Re-pos: $\forall x \in\{0 . .<1\} .0<(\operatorname{Re} \circ h) x$
and $h p: \forall t$. Re $(h t)=c *$ poly $p($ min $-r * t+r * t-r)$
and $h q: \forall t$. Im $(h t)=c * \operatorname{poly} q(\min -r * t+r * t-r)$
and $[$ simp $]: c \neq 0$
and $\operatorname{Re}\left(\begin{array}{ll}h & 1\end{array}\right)=0$
and valid-path $h$
and $h$-img:0 $\notin$ path-image $h$
for $h c$
proof -
define $f$ where $f=(\lambda t . c *$ poly $q t /(c *$ poly $p t))$
define farg where farg $=\left(\right.$ if $0<\operatorname{Im}\left(\begin{array}{ll}h & 1)\end{array}\right)$ then pi / 2 else - pi / 2)
have Re (winding-number h0) $=(\operatorname{Im}(\operatorname{Ln}($ pathfinish $h))$
- Im $($ Ln $($ pathstart h) $)) /(2 * p i)$
apply (rule Re-winding-number-half-right[of h 0,simplified])
subgoal using that $\left\langle R e\left(\begin{array}{ll}h & 1\end{array}\right)=0\right\rangle$ unfolding path-image-def
by (auto simp add:le-less)
subgoal using <valid-path $h$ 〉.
subgoal using $h$-img .
done
also have $\ldots=(f \arg -\arctan (f(-r))) /(2 * p i)$
proof -
have $\operatorname{Im}(\operatorname{Ln}($ pathfinish $h))=$ farg
using $\langle\operatorname{Re}(h 1)=0\rangle$ unfolding farg-def path-defs
apply (subst Im-Ln-eq)
subgoal using $h$-img unfolding path-defs by fastforce
subgoal by simp
done
moreover have $\operatorname{Im}($ Ln $($ pathstart $h))=\arctan (f(-r))$
proof -
have pathstart $h \neq 0$
using $h$-img
by (metis pathstart-in-path-image)
then have $\operatorname{Im}(\operatorname{Ln}($ pathstart $h))=\arctan (\operatorname{Im}($ pathstart $h) / R e$
(pathstart h))
using Re-pos[rule-format,of 0]

```
                    by (simp add: Im-Ln-eq path-defs)
                    also have ... = arctan (f(-r))
                    unfolding f-def path-defs hp[rule-format] hq[rule-format]
                    by simp
                            finally show ?thesis.
    qed
    ultimately show ?thesis by auto
        qed
        finally have Re (winding-number h 0) = (farg - arctan (f (-r))) / (2*
pi).
    moreover have cindex-pathE h 0 = (-farg/pi)
    proof -
        have cindex-pathE h 0 = cindexE 0 1 (f\circ (\lambdax. (min-r +r)*x-r))
        unfolding cindex-pathE-def using <c\not=0\rangle
        by (auto simp add:hp hq f-def comp-def algebra-simps)
    also have ... = cindexE (-r) min-r f
        apply (subst cindexE-linear-comp[where b=-r,simplified])
        using r-asm by auto
    also have ... = - jumpFf(at-left min-r)
    proof -
        define right where right ={x.jumpF f(at-right x)\not=0\wedge-r\leqx
\wedge x< min-r}
        define left where left ={x.jumpF f(at-left x)\not=0\wedge -r<x\wedge x
smin-r}
                            have *:jumpF f (at-right x) =0 jumpF f(at-left x) =0 when
x\in{-r..<min-r} for x
        proof -
            have False when poly p x=0
            proof -
            have }x\geq\mathrm{ min-r
                using min-max-bound[rule-format,of x] that by auto
            then show False using <x\in{-r..<min-r}> by auto
        qed
        then show jumpF f(at-right x)=0 jumpF f(at-left x) =0
        unfolding f-def by (auto intro!:jumpF-not-infinity continuous-intros)
        qed
        then have right = {}
            unfolding right-def by force
            moreover have left = (if jumpF f(at-left min-r) = 0 then {} else
{min-r})
            unfolding left-def le-less using * r-asm by force
        ultimately show ?thesis
            unfolding cindexE-def by (fold left-def right-def,auto)
    qed
    also have ... = (-farg/pi)
    proof -
        have p-pos:c*poly px>0 when }x\in{-r<..<\mathrm{ min-r }}\mathrm{ for }
        proof -
```

```
    define hh where hh=(\lambdat. min-r*t+r*t-r)
    have }(x+r)/(min-r+r)\in{0..<1
    using that r-asm by (auto simp add:field-simps)
    then have 0<c*poly p (hh ((x+r)/(min-r+r)))
    apply (drule-tac Re-pos[rule-format])
    unfolding comp-def hp[rule-format] hq[rule-format] hh-def .
    moreover have hh ((x+r)/(min-r+r)) =x
    unfolding hh-def using <min-r>-r`
    apply (auto simp add:divide-simps)
    by (auto simp add:algebra-simps)
    ultimately show ?thesis by simp
    qed
    have c*poly q min-r }\not=
    using no-real-zero 〈c\not=0>
by (metis Im-complex-of-real UNIV-I «min-r \in proots p` cpoly-of-decompose
mult-eq-0-iff p-def poly-cpoly-of-real-iff proots-within-iff q-def)
    moreover have ?thesis when c*poly q min-r>0
    proof -
    have 0 < Im (h 1) unfolding hq[rule-format] hp[rule-format] using
    moreover have jumpF f(at-left min-r) = 1/2
    proof -
        have (( }\lambdat.c*\mathrm{ poly p t) has-sgnx 1) (at-left min-r)
        unfolding has-sgnx-def
        apply (rule eventually-at-leftI[of -r])
        using p-pos \langlemin-r>-r\rangle by auto
            then have filterlim f at-top (at-left min-r)
                        unfolding f-def
                                apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q min-r])
                                using that <min-r\inproots p> by (auto intro!:tendsto-eq-intros)
            then show ?thesis unfolding jumpF-def by auto
        qed
        ultimately show ?thesis unfolding farg-def by auto
    qed
    moreover have ?thesis when c*poly q min-r < 0
    proof -
    have 0 > Im (h 1) unfolding hq[rule-format] hp[rule-format] using
            moreover have jumpFf(at-left min-r) = - 1/2
            proof -
            have ((\lambdat. c*poly p t) has-sgnx 1) (at-left min-r)
                unfolding has-sgnx-def
                apply (rule eventually-at-leftI[of -r])
                        using p-pos \langlemin-r>-r\rangle by auto
                    then have filterlim f at-bot (at-left min-r)
                unfolding f-def
```

that by auto
that by auto

```
                    apply (subst filterlim-divide-at-bot-at-top-iff[of-c*poly q min-r])
                    using that «min-r\inproots p> by (auto intro!:tendsto-eq-intros)
                    then show ?thesis unfolding jumpF-def by auto
                qed
                ultimately show ?thesis unfolding farg-def by auto
            qed
            ultimately show ?thesis by linarith
        qed
        finally show ?thesis.
    qed
        ultimately show ?thesis unfolding wc-add-def f-def by (auto simp
add:field-simps)
    qed
    have }\forallx\in{0..<1}.(Re\circg1) x\not=
    proof (rule ccontr)
        assume }\neg(\forallx\in{0..<1}.(Re\circg1) x\not=0
        then obtain t where t-def:Re (g1 t) =0 t\in{0..<1}
        unfolding path-image-def by fastforce
    define m}\mathrm{ where m=min-r*t+r*t-r
    have poly p m=0
    proof -
        have Re (g1 t)=Re (poly pp (of-real m))
            unfolding m-def g1-def g-def linepath-def subpath-def u-def using
<r\not=0\rangle
            by (auto simp add:field-simps)
        then show ?thesis using t-def unfolding Re-poly-of-real p-def by auto
        qed
        moreover have m<min-r
        proof -
            have min-r+r>0 using r-asm by simp
            then have (min-r +r)*(t-1)<0 using <t\in{0..<1}>
            by (simp add: mult-pos-neg)
        then show ?thesis unfolding m-def by (auto simp add:algebra-simps)
        qed
        ultimately show False using min-max-bound unfolding proots-def by
auto
    qed
    then have }(\forallx\in{0..<1}.0<(Re\circg1) x)\vee(\forallx\in{0..<1}.(Re\circg1)
<0)
            apply (elim continuous-on-neq-split)
            using <path g1` unfolding path-def
            by (auto intro!:continuous-intros elim:continuous-on-subset)
    moreover have ?thesis when }\forallx\in{0..<1}.(Re\circg1) x<
    proof -
    have wc-add (uminus o g1) = - arctan (f (-r)) / pi
        unfolding f-def
        apply (rule wc-add-pos[of - -1])
        using g1-pq that <min-r \inproots p><valid-path g1>\langle0 # path-image g1>
```

by（auto simp add：path－image－compose）
moreover have $w c$－add（uminus $\circ g 1$ ）$=w c$－add $g 1$
unfolding wc－add－def cindex－pathE－def
apply（subst winding－number－uminus－comp）
using 〈valid－path g1〉〈0 $\notin$ path－image g1〉 by auto
ultimately show ？thesis by auto
qed
moreover have ？thesis when $\forall x \in\{0 . .<1\}$ ．（Re $\circ g 1) x>0$
unfolding $f$－def
apply（rule wc－add－pos［of－1］）
using g1－pq that 〈min－r $\in$ proots $p\rangle\langle v a l i d-p a t h ~ g 1\rangle\langle 0 \notin$ path－image g1〉
by（auto simp add：path－image－compose）
ultimately show ？thesis by blast
qed
moreover have $w c$－add $g 3=\arctan (f r) / p i$
proof－
have $g 3-p q$ ：
Re $(g 3 t)=$ poly $p((r-$ max $-r) * t+$ max－$r)$
$\operatorname{Im}(g 3 t)=$ poly $q((r-m a x-r) * t+$ max－$r)$
$\operatorname{Im}(g 3 t) / \operatorname{Re}(g 3 t)=(f o(\lambda x .(r-\max -r) * x+\max -r)) t$
for $t$
proof－
have $g 3 t=$ poly pp $(o f$－real $((r-$ max－$r) * t+$ max－$r))$
using $\langle r \neq 0\rangle\langle$ max－$r<r\rangle$ unfolding $g 3$－def $g$－def linepath－def subpath－def
$v$－def $p$－def
by（auto simp add：algebra－simps）
then show

$$
\operatorname{Re}(g 3 t)=\text { poly } p((r-\max -r) * t+\max -r)
$$

$$
\operatorname{Im}(g 3 t)=\text { poly } q((r-\text { max }-r) * t+\text { max }-r)
$$

unfolding $p$－def $q$－def
by（simp only：Re－poly－of－real Im－poly－of－real）＋
then show $\operatorname{Im}(g 3 t) / \operatorname{Re}(g 3 t)=(f o(\lambda x .(r-\max -r) * x+\max -r)) t$
unfolding $f$－def by（auto simp add：algebra－simps）
qed
have $\operatorname{Re}\left(\begin{array}{ll}g 3 & 0\end{array}\right)=0$
using $\langle r \neq 0\rangle$ Re－poly－of－real $\langle$ max－$r \in$ proots $p\rangle$
unfolding $g 3$－def subpath－def $v$－def $g$－def linepath－def
by（auto simp add：field－simps p－def）
have $0 \notin$ path－image g3
proof－
have $(1::$ real $) \in\{0 . .1\}$
by auto
then show ？thesis
using Path－Connected．path－image－subpath－subset 〈＾r．path（g r）〉g3－def
not－g－image uv（2）by blast
qed
have wc－add－pos：wc－add $h=\arctan ($ poly $q r / p o l y p r) / p i$ when Re－pos：$\forall x \in\{0<. .1\} .0<(R e \circ h) x$

```
    and hp:\forallt.Re (ht)=c*poly p ((r-max-r)*t + max-r)
    and hq:\forallt.Im (ht)=c*poly q ((r-max-r)*t + max-r)
    and [simp]:c\not=0
    and Re(h0) = 0
    and valid-path h
    and h-img:0 # path-image h
    for hc
proof -
    define f}\mathrm{ where f=( }\lambdat.c*\mathrm{ poly q t / (c*poly p t))
    define farg where farg=( if 0<Im(ll 0) then pi / 2 else - pi / 2)
    have Re (winding-number h 0) =(Im (Ln (pathfinish h))
        - Im (Ln (pathstart h))) / (2 * pi)
        apply (rule Re-winding-number-half-right[of h 0,simplified])
        subgoal using that <Re (h 0) = 0> unfolding path-image-def
            by (auto simp add:le-less)
    subgoal using <valid-path h>.
    subgoal using h-img .
    done
    also have ... =(arctan (fr) - farg) / (2* pi)
    proof -
    have Im}(Ln(\mathrm{ pathstart h)) = farg
        using <Re(h 0)=0〉 unfolding farg-def path-defs
        apply (subst Im-Ln-eq)
        subgoal using h-img unfolding path-defs by fastforce
        subgoal by simp
        done
    moreover have Im (Ln (pathfinish h)) = arctan (fr)
    proof -
        have pathfinish h\not=0
            using h-img
                by (metis pathfinish-in-path-image)
            then have Im (Ln (pathfinish h)) = arctan (Im (pathfinish h) / Re
        (pathfinish h))
            using Re-pos[rule-format,of 1]
            by (simp add: Im-Ln-eq path-defs)
            also have ... = arctan (fr)
                unfolding f-def path-defs hp[rule-format] hq[rule-format]
                by simp
            finally show ?thesis.
    qed
    ultimately show ?thesis by auto
    qed
    finally have Re (winding-number h 0) = (arctan (fr) - farg) / (2*pi).
    moreover have cindex-pathE h 0 = farg/pi
    proof -
    have cindex-pathE h 0 = cindexE 0 1 (f\circ (\lambdax. (r-max-r)*x + max-r ))
        unfolding cindex-pathE-def using < c\not=0`
        by (auto simp add:hp hq f-def comp-def algebra-simps)
```

```
    also have ... = cindexE max-r r f
    apply (subst cindexE-linear-comp)
    using r-asm by auto
    also have ... = jumpF f(at-right max-r)
    proof -
    define right where right ={x.jumpF f(at-right x) = 0 ^ max-r \leqx
\wedge x<r}
    define left where left ={x.jumpFf(at-left x)}\not=0\wedge max-r < x ^ x
sr}
            have *:jumpF f (at-right x) =0 jumpF f (at-left x) =0 when
x\in{max-r<..r} for }
    proof -
        have False when poly p x =0
        proof -
        have }x\leqmax-
            using min-max-bound[rule-format,of x] that by auto
            then show False using <x\in{max-r<..r}> by auto
        qed
        then show jumpFf(at-right x ) =0 jumpF f(at-left x ) =0
        unfolding f-def by (auto intro!:jumpF-not-infinity continuous-intros)
    qed
    then have left = {}
        unfolding left-def by force
    moreover have right = (if jumpFf(at-right max-r) =0 then {} else
{max-r})
            unfolding right-def le-less using * r-asm by force
    ultimately show ?thesis
    unfolding cindexE-def by (fold left-def right-def,auto)
    qed
    also have ... = farg/pi
    proof -
    have p-pos:c*poly p x>0 when }x\in{\mathrm{ max- }r<..<r}\mathrm{ for }
    proof -
        define }hh\mathrm{ where }hh=(\lambdat.(r-max-r)*t+max-r
        have (x-max-r)/(r-max-r) \in{0<..1}
            using that r-asm by (auto simp add:field-simps)
        then have 0<c*poly p(hh ((x-max-r)/(r-max-r)))
            apply (drule-tac Re-pos[rule-format])
            unfolding comp-def hp[rule-format] hq[rule-format] hh-def .
            moreover have hh ((x-max-r)/(r-max-r)) =x
            unfolding hh-def using <max-r<r>
            by (auto simp add:divide-simps)
        ultimately show ?thesis by simp
    qed
    have c*poly q max-r }\not=
        using no-real-zero 〈c\not=0>
    by (metis Im-complex-of-real UNIV-I <max-r \in proots p\rangle cpoly-of-decompose
```

$$
\text { mult-eq-O-iff p-def poly-cpoly-of-real-iff proots-within-iff } q \text {-def) }
$$

moreover have ?thesis when $c *$ poly $q$ max- $r>0$
proof -
have $0<\operatorname{Im}\left(\begin{array}{ll}h & 0\end{array}\right)$ unfolding $h q[$ rule-format $] h p[$ rule-format $]$ using
that by auto
moreover have jumpF $f$ (at-right max-r) $=1 / 2$
proof -
have ( $(\lambda t . c *$ poly $p t)$ has-sgnx 1) (at-right max-r) unfolding has-sgnx-def apply (rule eventually-at-rightI $[o f-r]$ ) using $p$-pos $\langle$ max- $r<r\rangle$ by auto
then have filterlim $f$ at-top (at-right max-r) unfolding $f$-def apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q max-r]) using that 〈max-r $\in$ proots $p\rangle$ by (auto intro!:tendsto-eq-intros)
then show ?thesis unfolding jumpF-def by auto
qed
ultimately show ?thesis unfolding farg-def by auto
qed
moreover have ?thesis when $c *$ poly $q$ max- $r<0$
proof -
have $0>\operatorname{Im}\left(\begin{array}{ll}h & 0\end{array}\right)$ unfolding $h q[$ rule-format $] h p[r u l e-f o r m a t]$ using that by auto
moreover have jumpF $f($ at-right max-r $)=-1 / 2$
proof -
have (( $\lambda t . c *$ poly $p t)$ has-sgnx 1) (at-right max-r) unfolding has-sgnx-def apply (rule eventually-at-rightI[of $-r]$ ) using $p$-pos $\langle$ max- $r<r\rangle$ by auto
then have filterlim $f$ at-bot (at-right max-r) unfolding $f$-def apply (subst filterlim-divide-at-bot-at-top-iff [of - c*poly q max-r]) using that $\langle$ max-r $r$ proots $p\rangle$ by (auto intro!:tendsto-eq-intros)
then show ?thesis unfolding jumpF-def by auto
qed
ultimately show ?thesis unfolding farg-def by auto
qed
ultimately show ?thesis by linarith
qed
finally show ?thesis.
qed
ultimately show ?thesis unfolding wc-add-def $f$-def by (auto simp
add:field-simps)
qed
have $\forall x \in\{0<. .1\} .(R e \circ g 3) x \neq 0$
proof (rule ccontr)

```
assume \(\neg(\forall x \in\{0<. .1\} .(R e \circ g 3) x \neq 0)\)
then obtain \(t\) where \(t\)-def:Re \((g 3 t)=0 \quad t \in\{0<. .1\}\)
    unfolding path-image-def by fastforce
define \(m\) where \(m=(r-\) max- \(r) * t+\) max- \(r\)
have poly \(p m=0\)
proof -
    have \(\operatorname{Re}(g 3 t)=R e(\) poly pp \((o f\)-real \(m))\)
    unfolding \(m\)-def \(g 3\)-def \(g\)-def linepath-def subpath-def \(v\)-def using \(\langle r \neq 0\rangle\)
        by (auto simp add:algebra-simps)
    then show ?thesis using \(t\)-def unfolding Re-poly-of-real p-def by auto
qed
moreover have \(m>\) max- \(r\)
proof -
    have \(r\)-max- \(r>0\) using \(r\)-asm by simp
    then have \((r-\max -r) * t>0\) using \(\langle t \in\{0<. .1\}\rangle\)
        by (simp add: mult-pos-neg)
    then show ?thesis unfolding m-def by (auto simp add:algebra-simps)
qed
ultimately show False using min-max-bound unfolding proots-def by
auto
qed
    then have \((\forall x \in\{0<. .1\} .0<(R e \circ g 3) x) \vee(\forall x \in\{0<. .1\} .(R e \circ g 3) x\)
\(<0\) )
    apply (elim continuous-on-neq-split)
    using «path g3〉 unfolding path-def
    by (auto intro!:continuous-intros elim:continuous-on-subset)
    moreover have ?thesis when \(\forall x \in\{0<. .1\}\). (Re \(\circ\) g3) \(x<0\)
    proof -
    have \(w c\)-add \((\) uminus o \(g 3)=\arctan (f r) / p i\)
        unfolding \(f\)-def
        apply (rule wc-add-pos[of - -1])
        using g3-pq that 〈max-r \(\in\) proots \(p\rangle\langle v a l i d-p a t h ~ g 3\rangle\langle 0 \notin\) path-image g3〉
        by (auto simp add:path-image-compose)
    moreover have wc-add (uminus \(\circ\) g3) \(=w c\)-add \(g 3\)
        unfolding wc-add-def cindex-pathE-def
        apply (subst winding-number-uminus-comp)
        using 〈valid-path g3〉〈0 \(\notin\) path-image g3〉 by auto
    ultimately show ?thesis by auto
    qed
    moreover have ?thesis when \(\forall x \in\{0<. .1\}\). (Re \(\circ\) g3) \(x>0\)
        unfolding \(f\)-def
    apply (rule wc-add-pos[of - 1])
    using g3-pq that 〈max-r \(\in\) proots \(p\rangle\langle v a l i d-p a t h ~ g 3\rangle\langle 0 \notin\) path-image g3〉
    by (auto simp add:path-image-compose)
    ultimately show ?thesis by blast
qed
ultimately have \(w c\) - \(a d d(g r)=(\arctan (f r)-\arctan (f(-r))) / p i\)
    by (auto simp add:field-simps)
then show 2 * Re (winding-number ( \(\begin{aligned} & \text { r }) 0 \text { ) }+ \text { cindex-pathE }(g r) 0 \\ & 0\end{aligned}\)
```

```
            =(arctan (fr) - arctan (f(-r))) / pi
            unfolding wc-add-def .
    qed
    with arctan-f-tendsto show ?thesis by (auto dest:tendsto-cong)
    qed
    ultimately show ?thesis by auto
qed
lemma proots-upper-cindex-eq:
    assumes lead-coeff p=1 and no-real-roots: }\forallx\in\mathrm{ proots p. Im x}=
    shows proots-upper p =
                            (degree p - cindex-poly-ubd (map-poly Im p) (map-poly Re p))/2
proof (cases degree p=0)
    case True
    then obtain c where p=[:c:] using degree-eq-zeroE by blast
    then have p-def:p=[:1:] using <lead-coeff p=1> by simp
    have proots-count p {x. Im x>0} = 0 unfolding p-def proots-count-def by auto
    moreover have cindex-poly-ubd (map-poly Im p) (map-poly Re p)=0
        apply (subst cindex-poly-ubd-code)
        unfolding p-def
    by (auto simp add:map-poly-pCons changes-R-smods-def changes-poly-neg-inf-def
        changes-poly-pos-inf-def)
    ultimately show ?thesis using True unfolding proots-upper-def by auto
next
    case False
    then have degree p>0 p\not=0 by auto
    define w1 where w1=(\lambdar. Re (winding-number (poly p\circ
```



```
    define w2 where w2=(\lambdar. Re (winding-number (poly p o part-circlepath 0 r 0
pi) 0))
    define cp where cp=( }\lambdar\mathrm{ . cindex-pathE (poly p}\circ(\lambdax
        of-real (linepath (-r) (of-real r) x))) 0)
    define ci where ci=(\lambdar.cindexE (-r)r ( }\lambda\mathrm{ x. poly (map-poly Im p) x/poly
(map-poly Re p) x))
    define cubd where cubd = cindex-poly-ubd (map-poly Im p) (map-poly Re p)
    obtain R where proots p\subseteqball 0 R and R>0
    using \langlep\not=0\rangle finite-ball-include[of proots p 0] by auto
    have ((\lambdar.w1r +w2 r + cpr / 2 -cir/2)
                real (degree p) / 2 - of-int cubd / 2) at-top
    proof -
    have t1:((\lambdar. 2*w1r + cpr)\longrightarrow0) at-top
            using Re-winding-number-poly-linepth[OF assms] unfolding w1-def cp-def
by auto
    have t2:(w2 \longrightarrow real (degree p) / 2) at-top
        using Re-winding-number-poly-part-circlepath[OF <degree p>0〉,of 0] unfold-
ing w2-def by auto
    have t3:(ci\longrightarrowof-int cubd) at-top
```

apply (rule tendsto-eventually)
using cindex-poly-ubd-eventually[of map-poly Im p map-poly Re p]
unfolding ci-def cubd-def by auto
from tendsto-add[OF tendsto-add[OF tendsto-mult-left[OF t3,of -1/2,simplified]

```
tendsto-mult-left[OF t1,of 1/2,simplified]]
```

$t 2]$
show ?thesis by (simp add:algebra-simps)
qed
moreover have $\forall_{F} r$ in at-top. w1 $r+w 2 r+c p r / 2-c i r / 2=$ proots-count
$p\{x$. Im $x>0\}$
proof (rule eventually-at-top-linorderI[of $R]$ )
fix $r$ assume $r \geq R$
then have $r$-ball:proots $p \subseteq$ ball $0 r$ and $r>0$
using $\langle R>0\rangle\langle$ proots $p \subseteq$ ball $0 R\rangle$ by auto
define $l l$ where $l l=$ linepath ( - complex-of-real $r$ ) $r$
define $r r$ where $r r=$ part-circlepath 0 r 0 pi
define $l r$ where $l r=l l+++r r$
have img-ll:path-image $l l \subseteq-$ proots $p$ and img-rr: path-image $r r \subseteq-$ proots
subgoal unfolding ll-def using $\langle 0<r\rangle$ closed-segment-degen-complex(2)
no-real-roots by auto
subgoal unfolding rr-def using in-path-image-part-circlepath $\langle 0<r\rangle r$-ball
by fastforce
done
have [simp]:valid-path (poly poll) valid-path (poly porr) valid-path ll valid-path rr pathfinish rr=pathstart ll pathfinish $l l=$ pathstart $r r$
proof -
show valid-path (poly poll) valid-path (poly porr)
unfolding ll-def rr-def by (auto intro:valid-path-compose-holomorphic)
then show valid-path ll valid-path rr unfolding ll-def rr-def by auto
show pathfinish $r r=$ pathstart $l l$ pathfinish $l l=$ pathstart $r r$
unfolding ll-def rr-def by auto
qed
have proots-count $p\{x$. Im $x>0\}=\left(\sum x \in\right.$ proots $p$. winding-number lr $x *$ of-nat (order x p))
unfolding proots-count-def of-nat-sum
proof (rule sum.mono-neutral-cong-left)
show finite (proots p) proots-within p $\{x .0<\operatorname{Im} x\} \subseteq$ proots $p$ using $\langle p \neq 0\rangle$ by auto
next
have winding-number $l r x=0$ when $x \in$ proots $p$ - proots-within $p\{x .0<\operatorname{Im}$
$x\}$ for $x$
unfolding $l r$-def $l l$-def $r r$-def
proof (eval-winding,simp-all)
show $*: x \notin$ closed-segment ( - complex-of-real r) (complex-of-real r)
using img-ll that unfolding $l l$-def by auto
show $x \notin$ path-image (part-circlepath 0 r 0 pi)
using img-rr that unfolding $r$ r-def by auto
have $x r$-facts: $0>\operatorname{Im} x-r<\operatorname{Re} x \operatorname{Re} x<r \operatorname{cmod} x<r$
proof -
have Im $x \leq 0$ using that by auto
moreover have Im $x \neq 0$ using no-real-roots that by blast
ultimately show $0>\operatorname{Im} x$ by auto
next
show cmod $x<r$ using that $r$-ball by auto
then have $\mid$ Re $x \mid<r$
using abs-Re-le-cmod [of $x$ ] by argo
then show $-r<\operatorname{Re} x$ Re $x<r$ by linarith +
qed
then have cindex-pathE $l l x=1$
using $\langle r>0\rangle$ unfolding cindex-pathE-linepath[OF *] ll-def
by (auto simp add: mult-pos-neg)
moreover have cindex-pathE rr $x=-1$
unfolding $r$ r-def using $r$-ball that
by (auto intro!: cindex-pathE-circlepath-upper)
ultimately show - cindex-pathE (linepath $(-$ of-real $r)(o f-r e a l r)) x=$
cindex-pathE (part-circlepath 0 r 0 pi) $x$
unfolding ll-def rr-def by auto
qed
then show $\forall i \in$ proots $p$ - proots-within $p\{x .0<\operatorname{Im} x\}$.
winding-number lr $i *$ of-nat (order i $p$ ) $=0$
by auto
next
fix $x$ assume $x$-asm: $x \in$ proots-within $p\{x .0<\operatorname{Im} x\}$
have winding-number lr $x=1$ unfolding $l r$-def $l l-d e f r r$-def
proof (eval-winding,simp-all)
show $*: x \notin$ closed-segment ( - complex-of-real r) (complex-of-real r)
using img-ll $x$-asm unfolding $l l$-def by auto
show $x \notin$ path-image (part-circlepath 0 r 0 pi)
using img-rr $x$-asm unfolding $r r$-def by auto
have $x r$-facts: $0<\operatorname{Im} x-r<\operatorname{Re} x \operatorname{Re} x<r \operatorname{cmod} x<r$
proof -
show $0<\operatorname{Im} x$ using $x$-asm by auto
next
show cmod $x<r$ using $x$-asm $r$-ball by auto
then have $\mid$ Re $x \mid<r$
using abs-Re-le-cmod $[$ of $x]$ by argo
then show $-r<R e x$ Re $x<r$ by linarith +
qed
then have cindex-pathE $l l x=-1$
using $\langle r>0\rangle$ unfolding cindex-pathE-linepath $[O F *]$ ll-def
by (auto simp add: mult-less-0-iff)
moreover have cindex-pathE rr $x=-1$
unfolding $r$-def using $r$-ball $x$-asm
by (auto intro!: cindex-pathE-circlepath-upper)
ultimately show - of-real (cindex-pathE (linepath (- of-real r) (of-real

```
r)) x) -
            of-real (cindex-pathE (part-circlepath 0 r 0 pi) x)=2
            unfolding ll-def rr-def by auto
        qed
        then show of-nat (order x p) = winding-number lr x* of-nat (order x p) by
auto
    qed
    also have ... = 1/(2*pi*i)* contour-integral lr ( }\lambda\mathrm{ x. deriv (poly p) x / poly p
x)
    apply (subst argument-principle-poly[of p lr])
    using <p\not=0\rangle img-ll img-rr unfolding lr-def ll-def rr-def
    by (auto simp add:path-image-join)
    also have ... = winding-number (poly p\circlr) 0
        apply (subst winding-number-comp[of UNIV poly p lr 0])
        using〈p\not=0\rangle img-ll img-rr unfolding lr-def ll-def rr-def
        by (auto simp add:path-image-join path-image-compose)
    also have ... = Re (winding-number (poly p\circlr)0)
    proof -
        have winding-number (poly p\circlr) 0\in Ints
            apply (rule integer-winding-number)
            using < p\not=0` img-ll img-rr unfolding lr-def
            by (auto simp add:path-image-join path-image-compose path-compose-join
                    pathstart-compose pathfinish-compose valid-path-imp-path)
        then show ?thesis by (simp add: complex-eqI complex-is-Int-iff)
    qed
    also have ... = Re (winding-number (poly p\circll)0) + Re (winding-number
(poly p\circrr) 0)
        unfolding lr-def path-compose-join using img-ll img-rr
        apply (subst winding-number-join)
        by (auto simp add:valid-path-imp-path path-image-compose pathstart-compose
        pathfinish-compose)
    also have ... = w1r +w2 r
            unfolding w1-def w2-def ll-def rr-def of-real-linepath by auto
    finally have of-nat (proots-count p {x.0<Im x}) = complex-of-real (w1r +
w2 r).
    then have proots-count p {x.0<Im x} =w1r + w2 r
        using of-real-eq-iff by fastforce
    moreover have cpr=cir
    proof -
        define f}\mathrm{ where f=( }\lambdax\mathrm{ . Im (poly p (of-real x)) / Re (poly p x))
        have cpr = cindex-pathE (poly p}\circ(\lambdax.2*r*x - r)) 0 
            unfolding cp-def linepath-def by (auto simp add:algebra-simps)
        also have ... = cindexE 0 1 (fo ( }\lambdax.2*r*x-r)
            unfolding cp-def ci-def cindex-pathE-def f-def comp-def by auto
    also have ... = cindexE (-r) rf
            apply (subst cindexE-linear-comp[of 2*r 0 1f -r,simplified])
            using \langler>0\rangle by auto
        also have ... = cir
            unfolding ci-def f-def Im-poly-of-real Re-poly-of-real by simp
```

```
        finally show ?thesis .
    qed
    ultimately show w1r + w2 r + cpr / 2 - cir / 2 = real (proots-count p
{x.0<Im x})
            by auto
    qed
    ultimately have ((\lambdar::real. real (proots-count p {x.0<Im x}))
        \longrightarroweal (degree p) / 2 - of-int cubd / 2) at-top
    by (auto dest: tendsto-cong)
    then show ?thesis
    apply (subst (asm) tendsto-const-iff)
    unfolding cubd-def proots-upper-def by auto
qed
lemma cindexE-roots-on-horizontal-border:
    fixes a::complex and s::real
    defines g\equivlinepath a (a+of-real s)
    assumes pqr:p=q*r and r-monic:lead-coeff r=1 and r-proots: }\forallx\in\mathrm{ proots r.
Im x=Im a
    shows cindexE lb ub (\lambdat. Im ((poly p\circg)t) / Re ((poly p\circg)t))=
        cindexE lb ub (\lambdat. Im ((poly q\circg) t) / Re ((poly q\circg)t))
    using assms
proof (induct r arbitrary:p rule:poly-root-induct-alt)
    case 0
    then have False
            by (metis Im-complex-of-real UNIV-I imaginary-unit.simps(2) proots-within-0
zero-neq-one)
    then show ?case by simp
next
    case (no-proots r)
    then obtain b}\mathrm{ where b}=0r=[:b:
            using fundamental-theorem-of-algebra-alt by blast
    then have r=1 using <lead-coeff r=1> by simp
    with }\langlep=q*r\rangle\mathrm{ show ?case by simp
next
    case (root b r)
    then have ?case when s=0
        using that(1) unfolding cindex-pathE-def by (simp add:cindexE-constI)
    moreover have ?case when }s\not=
    proof -
        define qrg where qrg = poly (q*r) \circg
        have cindexE lb ub (\lambdat. Im ((poly p\circg) t) / Re ((poly p\circg)t))
                = cindexE lb ub (\lambdat. Im (qrg t* (gt-b)) / Re (qrgt* (gt-b)))
            unfolding qrg-def \langlep=q*([:- b, 1:]*r)\rangle comp-def
            by (simp add:algebra-simps)
    also have ... = cindexE lb ub
                (\lambdat. ((Rea+t*s-Reb)* Im (qrg t)) /
                    ((Rea+t*s - Re b )* Re (qrg t)))
    proof -
```

```
    have \(\operatorname{Im} b=\operatorname{Im} a\)
        using \(\langle\forall x \in\) proots \(([:-b, 1:] * r) . \operatorname{Im} x=\operatorname{Im} a\rangle\) by auto
    then show ?thesis
        unfolding cindex-pathE-def g-def linepath-def
        by (simp add:algebra-simps)
    qed
    also have \(\ldots=\operatorname{cindexE} l b u b(\lambda t\). \(\operatorname{Im}(q r g t) / \operatorname{Re}(q r g t))\)
    proof (rule cindexE-cong[of \{t. Re \(a+t * s-\operatorname{Re} b=0\}])\)
    show finite \(\{t\). Re \(a+t * s-R e b=0\}\)
    proof (cases Re \(a=\) Re b)
        case True
        then have \(\{t\). Re \(a+t * s-\operatorname{Re} b=0\}=\{0\}\)
            using \(\langle s \neq 0\rangle\) by auto
        then show ?thesis by auto
    next
        case False
        then have \(\{t\). Re \(a+t * s-\operatorname{Re} b=0\}=\{(\operatorname{Re} b-\operatorname{Re} a) / s\}\)
            using \(\langle s \neq 0\rangle\) by (auto simp add:field-simps)
        then show ?thesis by auto
    qed
    next
    fix \(x\) assume asm: \(x \notin\{t\). Re \(a+t * s-\operatorname{Re} b=0\}\)
    define \(t t\) where \(t t=R e a+x * s-R e b\)
    have \(t t \neq 0\) using asm unfolding \(t t\)-def by auto
    then show \(t t * \operatorname{Im}(q r g x) /(t t * \operatorname{Re}(q r g x))=\operatorname{Im}(q r g x) / \operatorname{Re}(\operatorname{qrg} x)\)
        by auto
    qed
    also have \(\ldots=\) cindexE \(l b u b(\lambda t\). \(\operatorname{Im}((\) poly \(q \circ g) t) / \operatorname{Re}((p o l y q \circ g) t))\)
    unfolding \(q r g\)-def
    proof (rule root(1))
    show lead-coeff \(r=1\)
    by (metis lead-coeff-mult lead-coeff-pCons(1) mult-cancel-left2 one-poly-eq-simps(2)
        root.prems(2) zero-neq-one)
    qed (use root in simp-all)
    finally show? ?thesis .
qed
ultimately show ?case by auto
qed
lemma poly-decompose-by-proots:
fixes \(p::{ }^{\prime} a:: i d o m\) poly
assumes \(p \neq 0\)
shows \(\exists q r . p=q * r \wedge\) lead-coeff \(q=1 \wedge(\forall x \in\) proots \(q\). \(P x) \wedge(\forall x \in\) proots \(r\).
\(\neg P x)\) using assms
proof (induct \(p\) rule:poly-root-induct-alt)
case 0
```

```
    then show ?case by simp
next
    case (no-proots p)
    then show ?case
        apply (rule-tac x=1 in exI)
        apply (rule-tac x=p in exI)
    by (simp add:proots-def)
next
    case (root a p)
    then obtain q r where pqr:p=q*r and leadq:lead-coeff q=1
                            and qball:}\foralla\inproots q. P a and rball: \forallx\inproots r.\negP x
        using mult-zero-right by metis
    have ?case when P a
        apply (rule-tac x=[:- a, 1:]*q in exI)
        apply (rule-tac x=r in exI)
        using pqr qball rball that leadq unfolding lead-coeff-mult
        by (auto simp add:algebra-simps)
    moreover have ?case when \negPa
        apply (rule-tac x=q in exI)
        apply (rule-tac x=[:-a, 1:] *r in exI)
        using pqr qball rball that leadq unfolding lead-coeff-mult
        by (auto simp add:algebra-simps)
    ultimately show ?case by blast
qed
lemma proots-upper-cindex-eq':
    assumes lead-coeff p=1
    shows 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 -
    have p\not=0 using assms by auto
    from poly-decompose-by-proots[OF this,of \lambdax. Im x\not=0]
    obtain qr where pqr:p=q*r and leadq:lead-coeff q=1
                and qball: }\forallx\in\mathrm{ proots q. Im x}\not=0\mathrm{ and rball: }\forallx\in\mathrm{ proots r. Im x =0
    by auto
    have real-of-int (proots-upper p) = proots-upper q + proots-upper r
    using 〈p\not=0\rangle unfolding proots-upper-def pqr by (auto simp add:proots-count-times)
    also have ... = proots-upper q
    proof -
    have proots-within r {z.0<Im z}={}
            using rball by auto
    then have proots-upper r =0
            unfolding proots-upper-def proots-count-def by simp
    then show ?thesis by auto
    qed
    also have ... = (degree q-cindex-poly-ubd (map-poly Im q) (map-poly Re q))
/ 2
    by (rule proots-upper-cindex-eq[OF leadq qball])
    also have ... = (degree p - proots-count p {x. Im x=0}
```

- cindex-poly-ubd (map-poly Im p) (map-poly Re p))/2
proof -
have degree $q=$ degree $p$ - proots-count $p\{x$. Im $x=0\}$
proof -
have degree $p=$ degree $q+$ degree $r$
unfolding $p q r$
apply (rule degree-mult-eq)
using $\langle p \neq 0$ 〉 pqr by auto
moreover have degree $r=$ proots-count $p\{x$. Im $x=0\}$
unfolding degree-proots-count proots-count-def
proof (rule sum.cong)
fix $x$ assume $x \in$ proots-within $p\{x$. Im $x=0\}$
then have $\operatorname{Im} x=0$ by auto
then have order $x q=0$
using qball order-0I by blast
then show order $x r=$ order $x p$
using $\langle p \neq 0\rangle$ unfolding $p q r$ by (simp add: order-mult)
next
show proots $r=$ proots-within $p\{x$. Im $x=0\}$
unfolding pqr proots-within-times using qball rball by auto
qed
ultimately show ?thesis by auto
qed
moreover have cindex-poly-ubd (map-poly Im q) (map-poly Re q)
$=$ cindex-poly-ubd (map-poly Im p) (map-poly Re p)
proof -
define $i q$ rq $i p r p$ where $i q=$ map-poly $\operatorname{Im} q$ and rq=map-poly Re $q$
and $i p=m a p-p o l y$ Im $p$ and $r p=$ map-poly Re $p$
have cindexE $(-x) x(\lambda x$. poly iq $x /$ poly rq $x)$ $=$ cindexE $(-x) x(\lambda x$. poly ip $x /$ poly rp $x)$ for $x$
proof -
have lead-coeff $r=1$
using pqr leadq 〈lead-coeff $p=1\rangle$ by (simp add: coeff-degree-mult)
then have cindexE $(-x) x\left(\lambda t . \operatorname{Im}\left(\right.\right.$ poly $\left.p\left(t *_{R} 1\right)\right) / \operatorname{Re}\left(\operatorname{poly} p\left(t *_{R}\right.\right.$
1))) $=$

$$
\operatorname{cindex} E(-x) x\left(\lambda t . \operatorname{Im}\left(\text { poly } q\left(t *_{R} 1\right)\right) / \operatorname{Re}\left(\operatorname{poly} q\left(t *_{R} 1\right)\right)\right)
$$

using cindexE-roots-on-horizontal-border[OF pqr,of $0-x \times 1$
,unfolded linepath-def comp-def,simplified] rball by simp
then show ?thesis
unfolding scaleR-conv-of-real iq-def ip-def rq-def rp-def by (simp add:Im-poly-of-real Re-poly-of-real)
qed
then have $\forall_{F}$ r::real in at-top.
real-of-int (cindex-poly-ubd iq rq) $=$ cindex-poly-ubd ip rp
using eventually-conj[OF cindex-poly-ubd-eventually[of iq rq]
cindex-poly-ubd-eventually[of ip rp]]
by (elim eventually-mono,auto)
then show ?thesis
apply (fold iq-def rq-def ip-def rp-def)

```
            by simp
        qed
        ultimately show ?thesis by auto
    qed
    finally show ?thesis by simp
qed
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 -
    define pp where pp=smult (inverse (lead-coeff p)) p
    define pI where pI= map-poly Im pp
    define pR where pR=map-poly Re pp
    define g}\mathrm{ where g=gcd pR pI
    have card (proots-within p{x. Im x>0})= proots-upper p
    unfolding proots-upper-def using card-proots-within-rsquarefree[OF assms] by
auto
    also have ... = proots-upper pp
        unfolding proots-upper-def pp-def
        apply (subst proots-count-smult)
        using assms by auto
    also have ... = (degree pp - proots-count pp {x. Im x = 0} - cindex-poly-ubd
pI pR) div 2
    proof -
            define rr where rr = proots-count pp {x. Im x = 0}
            define cpp where cpp = cindex-poly-ubd pI pR
            have *:proots-upper pp=(degree pp-rr - cpp)/2
            apply (rule proots-upper-cindex-eq'[of pp,folded rr-def cpp-def pR-def pI-def])
            unfolding pp-def using assms by auto
    also have ... = (degree pp-rr - cpp) div 2
    proof (subst real-of-int-div)
            define tt where tt=int (degree pp - rr) - cpp
            have real-of-int tt=2*proots-upper pp
                by (simp add:*[folded tt-def])
            then show even tt by (metis dvd-triv-left even-of-nat of-int-eq-iff of-int-of-nat-eq)
            qed simp
            finally show ?thesis unfolding rr-def cpp-def by simp
    qed
    also have ... = (degree pp - changes-R-smods g (pderiv g)
```

```
proof -
    have rsquarefree \(p p\)
        using assms rsquarefree-smult-iff unfolding \(p p\)-def
        by (metis inverse-eq-imp-eq inverse-zero leading-coeff-neq-0 rsquarefree-0)
    from card-proots-within-rsquarefree[OF this]
    have proots-count pp \(\{x . \operatorname{Im} x=0\}=\) card (proots-within pp \(\{x \operatorname{Im} x=0\}\) )
        by \(\operatorname{simp}\)
    also have \(\ldots=\operatorname{card}(\) proots-within pp (unbounded-line 01\()\) )
    proof -
        have \(\{x\). Im \(x=0\}=\) unbounded-line 01
            unfolding unbounded-line-def
            apply auto
            subgoal for \(x\)
                apply (rule-tac \(x=R e x\) in exI)
                by (metis complex-is-Real-iff of-real-Re of-real-def)
            done
        then show? ?thesis by simp
    qed
    also have \(\ldots=\) changes- \(R\)-smods \(g\) (pderiv \(g)\)
    unfolding card-proots-unbounded-line[of 01 pp,simplified,folded pI-def pR-def]
\(g\)-def
    by (auto simp add:Let-def sturm- \(R[\) symmetric \(]\) )
    finally have proots-count \(p p\{x\). Im \(x=0\}=\) changes- \(R\)-smods \(g(p d e r i v g)\).
    moreover have degree \(p p \geq\) proots-count \(p p\{x . \operatorname{Im} x=0\}\)
        by (metis 〈rsquarefree pp〉 proots-count-leq-degree rsquarefree-0)
    ultimately show ?thesis
        by auto
qed
also have \(\ldots=(\) degree \(p-\) changes- \(R\)-smods \(g(p d e r i v g)\)
                        - changes- \(R\)-smods \(p R p I\) ) div 2
        using cindex-poly-ubd-code unfolding pp-def by simp
finally have card (proots-within \(p\{x .0<\operatorname{Im} x\})=(\) degree \(p-\) changes- \(R\)-smods
\(g(\) pderiv \(g)\) -
                        changes-R-smods \(p R\) pI) div 2.
    then show ?thesis unfolding Let-def
    apply (fold \(p p\)-def \(p R\)-def \(p I\)-def \(g\)-def)
    by (simp add: pp-def)
qed
lemma proots-upper-code1 [code]:
    proots-upper \(p=\)
    (if \(p \neq 0\) then
        (let \(p p=\) smult (inverse (lead-coeff \(p)\) ) \(p\);
                        \(p I=m a p-p o l y\) Im \(p p ;\)
                        \(p R=\) map-poly Re \(p p\);
                        \(g=g c d p I p R\)
            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.'') (\lambda-. proots-upper p))
proof -
    define pp where pp = smult (inverse (lead-coeff p)) p
    define }pI\mathrm{ where }pI=\mathrm{ map-poly Im pp
    define }pR\mathrm{ where }pR=\mathrm{ map-poly Re pp
    define g}\mathrm{ where g=gcd pI pR
    have ?thesis when p=0
        using that by auto
    moreover have ?thesis when p\not=0
    proof -
        have pp\not=0 unfolding pp-def using that by auto
        define rr where rr=int (degree pp - proots-count pp {x. Im x = 0}) -
cindex-poly-ubd pI pR
    have lead-coeff p\not=0 using <p\not=0` by simp
    then have proots-upper pp =rr / 2 unfolding rr-def
            apply (rule-tac proots-upper-cindex-eq'[of pp, folded pI-def pR-def])
            unfolding pp-def lead-coeff-smult by auto
    then have proots-upper pp = nat (rr div 2) by linarith
    moreover have
            rr = degree p - nat (changes-R-smods-ext g(pderiv g)) - changes-R-smods
pR pI
    proof -
        have degree pp = degree p unfolding pp-def by auto
        moreover have proots-count pp {x. Im x = 0} = nat (changes-R-smods-ext
g(pderiv g))
        proof -
            have {x. Im x=0}=unbounded-line 0 1
                    unfolding unbounded-line-def by (simp add: complex-eq-iff)
            then show ?thesis
                using proots-unbounded-line[of 0 1 pp,simplified, folded pI-def pR-def]
<pp\not=0>
            by (auto simp:Let-def g-def gcd.commute)
        qed
        moreover have cindex-poly-ubd pI pR = changes-R-smods pR pI
            using cindex-poly-ubd-code by auto
            ultimately show ?thesis unfolding rr-def by auto
    qed
    moreover have proots-upper pp = proots-upper p
        unfolding pp-def proots-upper-def
        apply (subst proots-count-smult)
        using that by auto
    ultimately show ?thesis
            unfolding Let-def using that
            apply (fold pp-def pI-def pR-def g-def)
        by argo
    qed
```

```
    ultimately show ?thesis by blast
qed
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 (cases p=0)
    case True
    then show ?thesis unfolding proots-upper-card-def using infinite-halfspace-Im-gt
by auto
next
    case False
    define pf pp pI pR g where
        pf = p div (gcd p (pderiv p))
    and pp= smult (inverse (lead-coeff pf)) pf
    and pI= map-poly Im pp
    and pR=map-poly Re pp
    and}g=gcd pR p
    have proots-upper-card p = proots-upper-card pf
    proof -
    have proots-within p {x.0<Im x} = proots-within pf {x.0<Im x}
            unfolding proots-within-def using poly-gcd-pderiv-iff[of p,folded pf-def]
            by auto
    then show ?thesis unfolding proots-upper-card-def by auto
    qed
    also have ... = nat ((degree pf - changes-R-smods g (pderiv g) - changes-R-smods
pR pI) div 2)
    using proots-within-upper-squarefree[OF rsquarefree-gcd-pderiv[OF <p\not=0>]
            ,unfolded Let-def,folded pf-def,folded pp-def pI-def pR-def g-def]
    unfolding proots-upper-card-def by blast
    finally show ?thesis unfolding Let-def
    apply (fold pf-def,fold pp-def pI-def pR-def g-def)
    using False by auto
qed
```


### 2.14 Polynomial roots on a general half-plane

the number of roots of polynomial $p$, counted with multiplicity, on the left half plane of the vector $b-a$.
definition proots-half ::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where

```
    proots-half p a b = proots-count p {w. Im ((w-a)/ (b-a))>0}
lemma proots-half-empty:
    assumes }a=
    shows proots-half p a b = 0
unfolding proots-half-def using assms by auto
lemma proots-half-proots-upper:
    assumes a\not=b p\not=0
    shows proots-half p a b= proots-upper (pcompose p [:a,(b-a):])
proof -
    define q where q=[:a,(b-a):]
    define }f\mathrm{ where f}=(\lambdax.(b-a)*x+a
    have (\sumr\inproots-within p{w. Im ((w-a)/(b-a))>0}.order r p)
        =(\sumr\inproots-within (p\mp@subsup{\circ}{p}{}q){z.0<Im z}. order r ( p opq))
    proof (rule sum.reindex-cong[of f])
    have injf
        using assms unfolding f}\mathrm{ -def inj-on-def by fastforce
    then show inj-on f (proots-within ( }p\mp@subsup{\circ}{p}{}q){z.0<Imz}
        by (elim inj-on-subset,auto)
    next
        show proots-within p {w. Im ((w-a)/(b-a))>0} =f'proots-within (p\circp
q) {z.0<Im z}
    proof safe
        fix x assume x-asm:x \in proots-within p {w. Im ((w-a) / (b-a))>0}
        define }xx\mathrm{ where }xx=(x-a)/(b-a
        have poly ( }p\mp@subsup{\circ}{p}{}q\mathrm{ ) xx=0
            unfolding q-def xx-def poly-pcompose using assms x-asm by auto
        moreover have Im xx>0
                unfolding xx-def using x-asm by auto
        ultimately have }xx\in\mathrm{ proots-within ( }p\mp@subsup{\circ}{p}{}q){z.0<Imz} by aut
        then show }x\inf`\mathrm{ ' proots-within ( }p\mp@subsup{\circ}{p}{}q){z.0<Im z
                apply (intro rev-image-eqI[of xx])
                unfolding f-def xx-def using assms by auto
    next
                fix x assume }x\in\mathrm{ proots-within ( }p\mp@subsup{\circ}{p}{}q){z.0<Imz
                then show fx\in proots-within p {w.0<Im ((w-a)/(b-a))}
                unfolding f-def q-def using assms
                apply (auto simp add:poly-pcompose)
                by (auto simp add:algebra-simps)
    qed
    next
        fix x assume x f proots-within ( }p\mp@subsup{\circ}{p}{}q){z.0<Imz
        show order ( fx) p = order x ( }p\mp@subsup{\circ}{p}{}q)\mathrm{ using < p}=0
        proof (induct p rule:poly-root-induct-alt)
            case 0
            then show ?case by simp
        next
```

```
case (no-proots p)
have order (fx) p=0
    apply (rule order-OI)
    using no-proots by auto
moreover have order x ( }p\mp@subsup{\circ}{p}{}q)=
    apply (rule order-0I)
    unfolding poly-pcompose q-def using no-proots by auto
    ultimately show ?case by auto
next
    case (root c p)
    have order (fx) ([:- c, 1:]* p)= order (fx) [:-c,1:] + order (fx)p
    apply (subst order-mult)
    using root by auto
also have ... = order x ([:- c, 1:] [opq)+\operatorname{order x ( }p\mp@subsup{\circ}{p}{}q)
proof -
    have order (fx)[:- c, 1:] = order x ([:- c, 1:] 别q)
    proof (cases f x=c)
        case True
        have [:- c, 1:] 㘯 q = smult (b-a) [:-x,1:]
            using True unfolding q-def f-def pcompose-pCons by auto
        then have order x ([:-c,1:] op q) = order x (smult (b-a) [:-x,1:])
            by auto
    then have order x ([:-c,1:] 的q)=1
                apply (subst (asm) order-smult)
            using assms order-power-n-n[of-1,simplified] by auto
            moreover have order (fx)[:- c, 1:] = 1
                using True order-power-n-n[of-1,simplified] by auto
            ultimately show ?thesis by auto
    next
        case False
        have order (fx)[:-c,1:]=0
            apply (rule order-0I)
            using False unfolding f-def by auto
        moreover have order x ([:-c, 1:] 趹q)=0
            apply (rule order-0I)
            using False unfolding f}f\mathrm{ -def q-def poly-pcompose by auto
        ultimately show ?thesis by auto
    qed
    moreover have order (fx) p= order x (p op q)
        apply (rule root)
        using root by auto
    ultimately show ?thesis by auto
qed
also have ... = order x (([:-c, 1:]*p) }\mp@subsup{\circ}{p}{}q
    unfolding pcompose-mult
    apply (subst order-mult)
    subgoal
        unfolding q-def using assms(1) pcompose-eq-0 root.prems
        by (metis One-nat-def degree-pCons-eq-if mult-eq-0-iff
```

```
                    one-neq-zero pCons-eq-0-iff right-minus-eq)
```

            by \(\operatorname{simp}\)
        finally show ?case .
        qed
    qed
    then show ?thesis unfolding proots-half-def proots-upper-def proots-count-def
    $q$-def
by auto
qed
lemma proots-half-code1 [code]:
proots-half $p$ a $b=($ if $a \neq b$ then
if $p \neq 0$ then proots-upper $\left(p \circ_{p}[: a, b-a:]\right)$
else Code.abort (STR "proots-half fails when $p=0 .{ }^{\prime \prime}$ )
( $\lambda$-. proots-half p ab)
else 0)
proof -
have ?thesis when $a=b$
using proots-half-empty that by auto
moreover have ?thesis when $a \neq b p=0$
using that by auto
moreover have ?thesis when $a \neq b p \neq 0$
using proots-half-proots-upper $[$ OF that $]$ that by auto
ultimately show ?thesis by auto
qed
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 pz0r card (proots-within p (ball z0 r))
lemma proots-ball-code1 [code]:
proots-ball pzor $=($ if $r \leq 0$ then
0
else if $p \neq 0$ then
proots-upper (fcompose ( $p \circ_{p}[: z 0$, of-real $r:]$ ) [:i,-1:] [:i, $\left.1:\right]$ )
else
Code.abort (STR "proots-ball fails when $\left.p=0 .{ }^{\prime \prime}\right)$
( $\lambda$-. proots-ball p z0 r)

```
proof (cases p=0\vee )
    case False
    have proots-ball p z0 r = proots-count (p op [:z0,of-real r:])(ball 0 1)
        unfolding proots-ball-def
        apply (rule proots-uball-eq[THEN arg-cong])
        using False by auto
    also have ... = proots-upper (fcompose ( }p\mp@subsup{\circ}{p}{}[:z0\mathrm{ , of-real r:]) [:i,-1:] [:i,1:])
        unfolding proots-upper-def
        apply (rule proots-ball-plane-eq[THEN arg-cong])
        using False pcompose-eq-0[of p [:z0, of-real r:]]
        by (simp add: pcompose-eq-0)
    finally show ?thesis using False by auto
qed (auto simp:proots-ball-def ball-empty)
lemma proots-ball-card-code1 [code]:
    proots-ball-card p z0 r=
        ( if r\leq0\vee p=0 then
            0
            else
                proots-upper-card (fcompose (p op [:z0, of-real r:]) [:i,-1:] [:i,1:])
                    )
proof (cases p=0\veer\leq0)
    case True
    moreover have ?thesis when r\leq0
    proof -
        have proots-within p (ball z0 r)={}
            by (simp add: ball-empty that)
            then show ?thesis unfolding proots-ball-card-def using that by auto
    qed
    moreover have ?thesis when r>0 p=0
        unfolding proots-ball-card-def using that infinite-ball[of r z0]
        by auto
    ultimately show ?thesis by argo
next
    case False
    then have p\not=0 r>0 by auto
    have proots-ball-card pz0 r = card (proots-within ( }p\mp@subsup{\circ}{p}{}[:z0\mathrm{ , of-real r:]) (ball 0
1))
    unfolding proots-ball-card-def
    by (rule proots-card-uball-eq[OF <r>0\rangle, THEN arg-cong])
    also have ... = proots-upper-card (fcompose ( }p\mp@subsup{\circ}{p}{}[:z0\mathrm{ , of-real r:]) [:i,-1:] [:i,1:])
        unfolding proots-upper-card-def
        apply (rule proots-card-ball-plane-eq[THEN arg-cong])
        using False pcompose-eq-0[of p [:z0, of-real r:]] by (simp add: pcompose-eq-0)
    finally show ?thesis using False by auto
qed
```


### 2.16 Polynomial roots on a circle (sphere)

```
definition proots-sphere::complex poly \(\Rightarrow\) complex \(\Rightarrow\) real \(\Rightarrow\) nat where
    proots-sphere \(p\) z \(0 r=\) 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 pz0r=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 \(p z 0=0\) then 1 else 0 )
        else if \(r<0 \vee p=0\) then
            0
        else
            (if poly \(p(z 0-r)=0\) then 1 else 0\()+\)
        proots-unbounded-line-card (fcompose ( \(p \circ_{p}[: z 0\), of-real \(r:]\) ) [:i,-1:]
[:i, \(1:])\)
                    01
            )
proof -
    have ?thesis when \(r=0\)
    proof -
        have proots-within \(p\{z 0\}=(\) if poly \(p z 0=0\) then \(\{z 0\}\) else \(\{ \}\) )
            by auto
        then show ?thesis unfolding proots-sphere-card-def using that by simp
    qed
    moreover have ?thesis when \(r \neq 0 r<0 \vee p=0\)
    proof -
        have ?thesis when \(r<0\)
        proof -
            have proots-within \(p\) (sphere z0 r) \(=\{ \}\)
                by (auto simp add: ball-empty that)
            then show ?thesis unfolding proots-sphere-card-def using that by auto
    qed
    moreover have ?thesis when \(r>0 p=0\)
            unfolding proots-sphere-card-def using that infinite-sphere[of r z0]
            by auto
    ultimately show ?thesis using that by argo
    qed
    moreover have ?thesis when \(r>0 \quad p \neq 0\)
    proof -
        define \(p p\) where \(p p=p \circ_{p}[: z 0\), of-real \(r:]\)
        define \(p p p\) where \(p p p=\) fcompose \(p p[: i,-1:][: i, 1:]\)
    have \(p p \neq 0\) unfolding \(p p\)-def using that pcompose-eq-0
            by force
    have proots-sphere-card pz0r=card (proots-within pp (sphere 0 1))
```

unfolding proots-sphere-card-def pp-def
by (rule proots-card-usphere-eq[OF 〈r>0〉, THEN arg-cong])
also have $\ldots=$ card (proots-within pp $\{-1\} \cup$ proots-within pp (sphere $01-$ $\{-1\})$ )
by (simp add: insert-absorb proots-within-union)
also have $\ldots=$ card (proots-within pp $\{-1\}$ ) + card (proots-within pp (sphere $01-\{-1\})$ )
apply (rule card-Un-disjoint)
using $\langle p p \neq 0\rangle$ by auto
also have $\ldots=\operatorname{card}($ proots-within pp $\{-1\})+\operatorname{card}$ (proots-within ppp $\{x .0$ $=\operatorname{Im} x\}$ )
using proots-card-sphere-axis-eq[OF $\langle p p \neq 0\rangle$,folded ppp-def] by simp
also have $\ldots=($ if poly $p(z 0-r)=0$ then 1 else 0$)+$ proots-unbounded-line-card ppp 01
proof -
have proots-within $p p\{-1\}=($ if poly $p(z 0-r)=0$ then $\{-1\}$ else $\{ \}$ )
unfolding $p p$-def by (auto simp:poly-pcompose)
then have card (proots-within pp $\{-1\})=$ (if poly $p(z 0-r)=0$ then 1 else 0)
by auto
moreover have $\{x . \operatorname{Im} x=0\}=$ unbounded-line 01
unfolding unbounded-line-def
apply auto
by (metis complex-is-Real-iff of-real-Re of-real-def)
then have card (proots-within ppp $\{x .0=\operatorname{Im} x\}$ )
$=$ proots-unbounded-line-card ppp 01
unfolding proots-unbounded-line-card-def by simp
ultimately show ?thesis by auto
qed
finally show?thesis
apply (fold pp-def,fold ppp-def)
using that by auto
qed
ultimately show ?thesis by auto
qed

### 2.17 Polynomial roots on a closed ball

definition proots-cball::complex poly $\Rightarrow$ complex $\Rightarrow$ real $\Rightarrow$ nat where proots-cball p z0 r $=$ proots-count $p(c b a l l ~ z 0 ~ r) ~$

- Roots counted WITHOUT multiplicity
definition proots-cball-card ::complex poly $\Rightarrow$ complex $\Rightarrow$ real $\Rightarrow$ nat where proots-cball-card pz0r=card (proots-within $p($ cball z0 r))
lemma proots-cball-card-code1[code]:
proots-cball-card p z0 r =
( if $r=0$ then

```
                    (if poly p z0=0 then 1 else 0)
        else if r<0\vee p=0 then
            O
        else
            ( let pp=fcompose ( }p\mp@subsup{\circ}{p}{}[:z0\mathrm{ , of-real r:]) [:i,-1:] [:i,1:]
        in
                        (if poly p (z0-r)=0 then 1 else 0)
            + proots-unbounded-line-card pp 0 1
            + proots-upper-card pp
        )
        )
proof -
    have ?thesis when r=0
    proof -
        have proots-within p {z0}=(if poly pz0=0 then {z0} else {})
            by auto
        then show ?thesis unfolding proots-cball-card-def using that by simp
    qed
    moreover have ?thesis when r\not=0 r<0\veep=0
    proof -
        have ?thesis when r<0
        proof -
            have proots-within p (cball z0 r)={}
                    by (auto simp add: ball-empty that)
            then show ?thesis unfolding proots-cball-card-def using that by auto
        qed
        moreover have ?thesis when r>0 p=0
            unfolding proots-cball-card-def using that infinite-cball[of r z0]
            by auto
        ultimately show ?thesis using that by argo
    qed
    moreover have ?thesis when p\not=0 r>0
    proof -
        define pp where pp=fcompose ( }p\mp@subsup{\circ}{p}{}[:z0\mathrm{ ,of-real r:]) [:i,-1:] [:i,1:]
    have proots-cball-card pz0r= card (proots-within p (sphere z0 r)
                        \cup ~ p r o o t s - w i t h i n ~ p ~ ( b a l l ~ z 0 ~ r ) ) ~
            unfolding proots-cball-card-def
            apply (simp add:proots-within-union)
            by (metis Diff-partition cball-diff-sphere sphere-cball)
            also have ... = card (proots-within p (sphere z0 r)) + card (proots-within p
(ball z0 r))
            apply (rule card-Un-disjoint)
            using < p}\not=0\mathrm{ \ by auto
    also have ... = (if poly p (z0-r)=0 then 1 else 0) + proots-unbounded-line-card
pp 0 1
                        + proots-upper-card pp
        using proots-sphere-card-code1[of p z0 r,folded pp-def,unfolded proots-sphere-card-def]
```

```
            proots-ball-card-code1[of p z0 r,folded pp-def,unfolded proots-ball-card-def]
            that
        by simp
    finally show ?thesis
        apply (fold pp-def)
        using that by auto
    qed
    ultimately show ?thesis by auto
qed
end
```


## theory Count-Rectangle imports Count-Line begin

Counting roots in a rectangular area can be in a purely algebraic approach without introducing (analytic) winding number (winding-number) nor the argument principle (【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 $p \rrbracket \Longrightarrow$ contour-integral ? $g(\lambda x$. deriv ?f $x *$ ?h $x /$ ?f $x)=$ complex-of-real $(2 * p i) * \mathrm{i} *\left(\sum p \in\{w \in\right.$ ?s. ?f $w=0 \vee w$ $\in$ ?poles $\}$. winding-number ? g $p *$ ?h $p *$ complex-of-int (zorder ?f p))). This has been illustrated by Michael Eisermann [1]. We lightly make use of winding-number here only to shorten the proof of one of the technical lemmas.

### 2.18 Misc

lemma proots-count-const:
assumes $c \neq 0$
shows proots-count [:c:] $s=0$
unfolding proots-count-def using assms by auto
lemma proots-count-nzero:
assumes $\bigwedge x . x \in s \Longrightarrow$ poly $p x \neq 0$
shows proots-count $p s=0$
unfolding proots-count-def
by (rule sum.neutral) (use assms in auto)
lemma complex-box-ne-empty:
fixes a b::complex
shows
cbox $a b \neq\{ \} \longleftrightarrow(\operatorname{Re} a \leq \operatorname{Re} b \wedge \operatorname{Im} a \leq \operatorname{Im} b)$
box $a b \neq\{ \} \longleftrightarrow(\operatorname{Re} a<\operatorname{Re} b \wedge \operatorname{Im} a<\operatorname{Im} b)$
by (auto simp add:box-ne-empty Basis-complex-def)

### 2.19 Counting roots in a rectangle

definition proots-rect ::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots-rect plbub= proots-count $p$ (box lb ub)
definition proots-crect ::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots-crect $p l b u b=$ proots-count $p($ cbox lb $u b)$
definition proots-rect-ll ::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots-rect-ll plb $u b=$ proots-count $p$ (box $l b u b \cup\{l b\}$
$\cup$ open-segment lb $($ Complex $($ Re ub $)(\operatorname{Im} l b))$
$\cup$ open-segment lb $($ Complex $($ Re lb) $(\operatorname{Im} u b)))$
definition proots-rect-border::complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ nat where proots-rect-border pab=proots-count $p$ (path-image (rectpath ab))
definition not-rect-vertex::complex $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ bool where not-rect-vertex r a $b=(r \neq a \wedge r \neq$ Complex $($ Re $b)(\operatorname{Im} a) \wedge r \neq b \wedge r \neq$ Complex (Rea) (Im b))
definition not-rect-vanishing :: complex poly $\Rightarrow$ complex $\Rightarrow$ complex $\Rightarrow$ bool where not-rect-vanishing pab=(poly paf0 $\wedge$ poly $p(\operatorname{Complex}(\operatorname{Re} b)(\operatorname{Im} a)) \neq 0$ $\wedge$ poly $p b \neq 0 \wedge$ poly $p($ Complex $(\operatorname{Re} a)(\operatorname{Im} b)) \neq 0)$
lemma cindexP-rectpath-edge-base:
assumes Re $a<\operatorname{Re} b \operatorname{Im} a<\operatorname{Im} b$
and not-rect-vertex rab
and $r \in$ path-image (rectpath a b)
shows cindexP-pathE [:-r, 1:] (rectpath a $b$ ) $=-1$
proof -
have $r$-nzero: $r \neq a \quad r \neq C o m p l e x ~(R e b)(\operatorname{Im} a) r \neq b r \neq C o m p l e x ~(R e a)(\operatorname{Im} b)$ using «not-rect-vertex $r$ a $b$ 〉 unfolding not-rect-vertex-def by auto
define $r r$ where $r r=[:-r, 1:]$
have rr-linepath:cindexP-pathE rr (linepath ab)
$=$ cindex-path $E($ linepath $(a-r)(b-r)) 0$ for $a b$
unfolding $r$ r-def
unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-
bra-simps)
have cindexP-pathE-eq:cindexP-pathE rr (rectpath ab) =

$$
\begin{aligned}
& \text { cindexP-pathE rr (linepath a (Complex }(\operatorname{Re} b)(\operatorname{Im} a))) \\
& \text { + cindexP-pathE rr (linepath (Complex }(\operatorname{Re} b)(\operatorname{Im} a)) \text { b) } \\
& \text { + cindexP-pathE rr (linepath b (Complex }(\operatorname{Re} a)(\operatorname{Im} b))) \\
& \text { + cindexP-pathE rr (linepath (Complex }(\operatorname{Re} a)(\operatorname{Im} b)) \text { a) }
\end{aligned}
$$

unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths
$\mid$ subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
pathfinish-compose pathstart-compose poly-pcompose)?)+

```
have (Im r = Im a ^Re a < Rer ^ Rer<Reb)
    \vee (Rer = Re b ^Im a<Im r ^ Im r<Im b)
    \vee ( I m r = I m ~ b \wedge ~ R e ~ a < R e r \wedge ~ R e r < R e ~ b ) ~
    \vee ( \operatorname { R e } r = \operatorname { R e } a \wedge \operatorname { I m } a < \operatorname { I m } r \wedge \operatorname { I m } r < \operatorname { I m } b )
proof -
    have r c closed-segment a (Complex (Re b) (Im a))
            \vee r c closed-segment (Complex (Re b) (Im a)) b
            \vee r closed-segment b (Complex (Re a) (Im b))
            \vee \mp@code { v }
    using <repath-image (rectpath a b)>
    unfolding rectpath-def Let-def
    by (subst (asm) path-image-join;simp)+
    then show ?thesis
            by (smt (verit, del-insts) assms(1) assms(2) r-nzero
            closed-segment-commute closed-segment-imp-Re-Im(1) closed-segment-imp-Re-Im(2)
                complex.sel(1) complex.sel(2) complex-eq-iff)
qed
moreover have cindexP-pathE rr (rectpath a b) = -1
    if Im r=Im a Re a < Re r Re r<Reb
proof -
    have cindexP-pathE rr (linepath a (Complex (Re b) (Im a))) =0
        unfolding rr-linepath
        apply (rule cindex-pathE-linepath-on)
        using closed-segment-degen-complex(2) that(1) that(2) that(3) by auto
    moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) =0
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using closed-segment-imp-Re-Im(1) that(3) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath b (Complex (Re a) (Im b))) = -1
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)=0
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using closed-segment-imp-Re-Im(1) that(2) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
```

```
moreover have cindexP-pathE \(\operatorname{rr}(\) rectpath \(a b)=-1\)
    if \(R e r=\operatorname{Re} b \operatorname{Im} a<\operatorname{Im} r \operatorname{Im} r<\operatorname{Im} b\)
proof -
    have cindexP-pathE rr (linepath a (Complex \((\operatorname{Re} b)(\operatorname{Im} a)))=-1 / 2\)
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using closed-segment-imp-Re-Im(2) that(2) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) \(=0\)
        unfolding rr-linepath
        apply (rule cindex-pathE-linepath-on)
        using closed-segment-degen-complex(1) that(1) that(2) that(3) by auto
        moreover have cindexP-pathE rr (linepath b(Complex \((\operatorname{Re} a)(\operatorname{Im} b)))=\)
\(-1 / 2\)
    unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using closed-segment-imp-Re-Im(2) that(3) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a) =0
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using assms(1) closed-segment-imp-Re-Im(1) that(1) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
moreover have cindexP-pathE rr (rectpath a b) \(=-1\)
    if Im \(r=\operatorname{Im} b\) Re \(a<\operatorname{Re} r \operatorname{Re} r<\operatorname{Re} b\)
proof -
    have cindexP-pathE rr (linepath a (Complex \((\operatorname{Re} b)(\operatorname{Im} a)))=-1\)
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) \(=0\)
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using closed-segment-imp-Re-Im(1) that(3) by force
        subgoal using that assms unfolding Let-def by auto
        done
    moreover have cindexP-pathE rr (linepath b(Complex \((\operatorname{Re} a)(\operatorname{Im} b)))=0\)
        unfolding rr-linepath
        apply (rule cindex-pathE-linepath-on)
    by (smt (verit, del-insts) Im-poly-hom.base.hom-zero Re-poly-hom.base.hom-zero
```

closed-segment-commute closed-segment-degen-complex(2) complex.sel(1) complex.sel(2) minus-complex.simps(1) minus-complex.simps(2) that(1) that(2) that(3))
moreover have cindexP-pathE rr (linepath $(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)) a)=0$ unfolding rr-linepath apply (subst cindex-pathE-linepath) subgoal using closed-segment-imp-Re-Im(1) that(2) by fastforce subgoal using that assms unfolding Let-def by auto done
ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
moreover have cindexP-pathE $\operatorname{rr}($ rectpath $a b)=-1$
if Re $r=\operatorname{Re} a \operatorname{Im} a<\operatorname{Im} r \operatorname{Im} r<\operatorname{Im} b$
proof -
have cindexP-pathE rr (linepath a (Complex $(\operatorname{Re} b)(\operatorname{Im} a)))=-1 / 2$ unfolding rr-linepath apply (subst cindex-pathE-linepath) subgoal using closed-segment-imp-Re-Im(2) that(2) by fastforce subgoal using that assms unfolding Let-def by auto done
moreover have cindexP-pathE rr (linepath (Complex (Re b) (Ima)) b) =0 unfolding rr-linepath apply (subst cindex-pathE-linepath)
subgoal using assms(1) closed-segment-imp-Re-Im(1) that(1) by fastforce subgoal using that assms unfolding Let-def by auto done
moreover have cindexP-pathE rr (linepath b(Complex $(\operatorname{Re} a)(\operatorname{Im} b)))=$ $-1 / 2$
unfolding rr-linepath
apply (subst cindex-pathE-linepath)
subgoal using closed-segment-imp-Re-Im(2) that(3) by fastforce
subgoal using that assms unfolding Let-def by auto
done
moreover have cindexP-pathE $r$ r (linepath $(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)) a)=0$
unfolding rr-linepath
apply (rule cindex-pathE-linepath-on)
by (smt (verit) Im-poly-hom.base.hom-zero Re-poly-hom.base.hom-zero closed-segment-commute closed-segment-degen-complex(1) complex.sel(1) complex.sel(2) minus-complex.simps(1) minus-complex.simps(2) that(1)
that(2) that(3))
ultimately show ?thesis unfolding cindexP-pathE-eq by auto
qed
ultimately show ?thesis unfolding rr-def by auto
qed
lemma cindexP-rectpath-vertex-base:
assumes Re $a<\operatorname{Re} b \operatorname{Im} a<\operatorname{Im} b$
and $\neg$ not-rect-vertex $r$ a $b$
shows cindexP-path $E[:-r, 1:]($ rectpath $a b)=-1 / 2$

```
proof -
    have r-cases: }r=a\veer=Complex (Re b) (Im a)\veer=b\veer=Complex (Re a) (Im
b)
    using «\neg not-rect-vertex r a b> unfolding not-rect-vertex-def by auto
    define rr where rr = [:-r,1:]
    have rr-linepath:cindexP-pathE rr (linepath a b)
                = cindex-pathE (linepath (a-r) (b-r)) 0 for ab
        unfolding rr-def
        unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
            by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-
bra-simps)
    have cindexP-pathE-eq:cindexP-pathE rr (rectpath a b)=
                        cindexP-pathE rr (linepath a (Complex (Re b) (Im a)))
                        + cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b)
                        + cindexP-pathE rr (linepath b (Complex (Re a) (Im b)))
                        + cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)
    unfolding rectpath-def Let-def
    by ((subst cindex-poly-pathE-joinpaths
                |subst finite-ReZ-segments-joinpaths
                |intro path-poly-comp conjI);
        (simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
            pathfinish-compose pathstart-compose poly-pcompose)?)+
    have cindexP-pathE rr (rectpath a b) = - 1/2
    if }r=
    proof -
    have cindexP-pathE rr (linepath a (Complex (Re b) (Im a))) =0
            unfolding rr-linepath
            apply (rule cindex-pathE-linepath-on)
            by (simp add: that)
    moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) =0
            unfolding rr-linepath
            apply (subst cindex-pathE-linepath)
            subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce
            subgoal using that assms unfolding Let-def by auto
            done
            moreover have cindexP-pathE rr (linepath b (Complex (Re a) (Im b))) =
-1/2
            unfolding rr-linepath
            apply (subst cindex-pathE-linepath)
            subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
            subgoal using that assms unfolding Let-def by auto
            done
            moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)=0
            unfolding rr-linepath
            apply (rule cindex-pathE-linepath-on)
            by (simp add: that)
```

ultimately show ?thesis unfolding cindexP-pathE-eq by auto

## qed

moreover have cindexP-pathE $r r($ rectpath $a b)=-1 / 2$
if $r=$ Complex $(R e b)(\operatorname{Im} a)$
proof -
have cindexP-pathE rr (linepath a (Complex $(\operatorname{Re} b)(\operatorname{Im} a)))=0$ unfolding rr-linepath
apply (rule cindex-pathE-linepath-on)
by (simp add: that)
moreover have cindexP-pathE rr (linepath (Complex $(\operatorname{Re} b)(\operatorname{Im} a)) b)=0$ unfolding rr-linepath
apply (rule cindex-pathE-linepath-on)
by ( simp add: that)
moreover have cindexP-pathE $\operatorname{rr}($ linepath $b(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)))=$ $-1 / 2$
unfolding rr-linepath
apply (subst cindex-pathE-linepath)
subgoal using assms(2) closed-segment-imp-Re-Im(2) that(1) by fastforce
subgoal using that assms unfolding Let-def by auto
done
moreover have cindexP-pathE $\operatorname{rr}(\operatorname{linepath}(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)) a)=0$ unfolding rr-linepath apply (subst cindex-pathE-linepath)
subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce subgoal by (smt (z3) complex.sel(1) minus-complex.simps(1)) done
ultimately show ?thesis unfolding cindexP-pathE-eq by auto

## qed

moreover have cindexP-pathE $r r($ rectpath $a b)=-1 / 2$
if $r=b$
proof -
have cindexP-pathE rr (linepath a (Complex $(\operatorname{Re} b)(\operatorname{Im} a)))=-1 / 2$ unfolding rr-linepath apply (subst cindex-pathE-linepath)
subgoal using assms(2) closed-segment-imp-Re-Im(2) that by fastforce subgoal using assms(1) assms(2) that by auto done
moreover have cindexP-pathE rr (linepath (Complex (Reb) (Im a)) b) $=0$
unfolding rr-linepath
apply (rule cindex-pathE-linepath-on)
by ( simp add: that)
moreover have cindexP-pathE rr (linepath b(Complex $(\operatorname{Re} a)(\operatorname{Im} b)))=0$ unfolding rr-linepath
apply (rule cindex-pathE-linepath-on)
by (simp add: that)
moreover have cindexP-pathE $\operatorname{rr}($ linepath $(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)) a)=0$ unfolding rr-linepath apply (subst cindex-pathE-linepath)
subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce

```
        subgoal by (smt (z3) complex.sel(1) minus-complex.simps(1))
        done
    ultimately show ?thesis unfolding cindexP-pathE-eq by auto
    qed
    moreover have cindexP-pathE rr (rectpath a b) = -1/2
    if r=Complex (Re a) (Im b)
    proof -
        have cindexP-pathE rr (linepath a (Complex (Re b) (Im a))) = - 1/2
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using assms(2) closed-segment-imp-Re-Im(2) that by fastforce
        subgoal using assms(1) assms(2) that by auto
        done
    moreover have cindexP-pathE rr (linepath (Complex (Re b) (Im a)) b) =0
        unfolding rr-linepath
        apply (subst cindex-pathE-linepath)
        subgoal using assms(1) closed-segment-imp-Re-Im(1) that by fastforce
        subgoal by (smt (z3) complex.sel(1) minus-complex.simps(1))
        done
    moreover have cindexP-pathE rr (linepath b (Complex (Re a) (Im b))) =0
        unfolding rr-linepath
        apply (rule cindex-pathE-linepath-on)
        by (simp add: that)
    moreover have cindexP-pathE rr (linepath (Complex (Re a) (Im b)) a)=0
        unfolding rr-linepath
        apply (rule cindex-pathE-linepath-on)
        by (simp add: that)
    ultimately show ?thesis unfolding cindexP-pathE-eq by auto
    qed
    ultimately show ?thesis using r-cases unfolding rr-def by auto
qed
lemma cindexP-rectpath-interior-base:
    assumes r\inbox a b
    shows cindexP-pathE [:-r,1:] (rectpath a b)= -2
proof -
    have inbox:Re r \in{Re a<..<Re b} ^ Im r f { Im a<..<Im b}
        using 〈r\inbox a b〉 unfolding in-box-complex-iff by auto
    then have r-nzero:r\not=a r\not=Complex (Re b) (Im a) r\not=b r\not=Complex (Re a) (Im
b)
    by auto
    have Re a<Re b Im a<Im b
    using «r\inbox a b〉 complex-box-ne-empty by blast+
    define rr where rr = [:-r,1:]
    have rr-linepath:cindexP-pathE rr (linepath a b)
                        = cindex-pathE (linepath (a-r) (b-r)) 0 for ab
        unfolding rr-def
        unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
```

by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-bra-simps)
have cindexP-pathE rr (rectpath a b) =

$$
\begin{aligned}
& \text { cindexP-pathE rr (linepath a (Complex (Re b) }(\operatorname{Im} a)) \text { ) } \\
& \text { + cindexP-pathE rr (linepath (Complex }(\operatorname{Re} b)(\operatorname{Im} a)) \text { b) } \\
& \text { + cindexP-pathE rr (linepath b (Complex }(\operatorname{Re} a)(\operatorname{Im} b))) \\
& \text { + cindexP-pathE rr (linepath (Complex }(\operatorname{Re} a)(\operatorname{Im} b)) \text { a) }
\end{aligned}
$$

unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths |subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
( simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?)+
also have $\ldots=-2$
proof -
have cindexP-pathE rr (linepath a (Complex $(\operatorname{Re} b)(\operatorname{Im} a)))=-1$
unfolding rr-linepath
apply (subst cindex-pathE-linepath)
subgoal using closed-segment-imp-Re-Im(2) inbox by fastforce
using inbox by auto
moreover have cindexP-pathE rr (linepath (Complex $(\operatorname{Re} b)(\operatorname{Im} a)) b)=0$ unfolding r-linepath apply (subst cindex-pathE-linepath)
subgoal using closed-segment-imp-Re-Im(1) inbox by fastforce using inbox by auto
moreover have cindexP-pathE rr (linepath $b(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)))=-1$ unfolding $r$-linepath apply (subst cindex-pathE-linepath)
subgoal using closed-segment-imp-Re-Im(2) inbox by fastforce using inbox by auto
moreover have cindexP-pathE rr $(\operatorname{linepath}(\operatorname{Complex}(\operatorname{Re} a)(\operatorname{Im} b)) a)=0$
unfolding rr-linepath
apply (subst cindex-pathE-linepath)
subgoal using closed-segment-imp-Re-Im(1) inbox by fastforce using inbox by auto
ultimately show ?thesis by auto
qed
finally show ?thesis unfolding rr-def .
qed
lemma cindexP-rectpath-outside-base:
assumes Re $a<\operatorname{Re} b \operatorname{Im} a<\operatorname{Im} b$
and $r \notin c b o x$ a $b$
shows cindexP-path $E[:-r, 1:]($ rectpath $a b)=0$
proof -
have not-cbox: $\neg(R e r \in\{R e a . . R e b\} \wedge \operatorname{Im} r \in\{\operatorname{Im} a . . \operatorname{Im} b\})$
using $\langle r \notin c b o x$ a $b\rangle$ unfolding in-cbox-complex-iff by auto
then have $r$-nzero: $r \neq a r \neq$ Complex (Re b) (Im a) $r \neq b r \neq$ Complex (Re a) (Im b)
using assms by auto
define $r r$ where $r r=[:-r, 1:]$
have rr-linepath:cindexP-pathE rr (linepath ab) $=$ cindex-pathE $($ linepath $(a-r)(b-r)) 0$ for $a b$
unfolding $r$ r-def
unfolding cindexP-lineE-def cindexP-pathE-def poly-linepath-comp
by (simp add:poly-pcompose comp-def linepath-def scaleR-conv-of-real alge-bra-simps)
have cindexP-pathE $r r($ rectpath a $b)=$

$$
\begin{aligned}
& \text { cindexP-pathE rr (linepath a (Complex (Re b) (Im a ) )) } \\
& \text { + cindexP-pathE rr (linepath (Complex (Re b) } \operatorname{Im} a)) \text { b) } \\
& \text { + cindexP-pathE rr (linepath b (Complex }(\operatorname{Re} a)(\operatorname{Im} b))) \\
& \text { + cindexP-pathE rr (linepath (Complex }(\operatorname{Re} a)(\operatorname{Im} b)) \text { a) }
\end{aligned}
$$

unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths
|subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?)+
have cindexP-pathE rr (rectpath $a b)=$ cindex-pathE (poly rr $\circ$ rectpath ab) 0 unfolding cindexP-pathE-def by simp
also have $\ldots=-2 *$ winding-number $($ poly rr $\circ$ rectpath a b) 0

- We don't need winding-number to finish the proof, but thanks to Cauthy's Index theorem (i.e., $\llbracket f i n i t e-R e Z$-segments ? $g ? z$; valid-path ? $g ; ? z \notin$ path-image ? $g$; pathfinish ? $g=$ pathstart ? $g \rrbracket \Longrightarrow$ winding-number ? $g$ ? $z=$ complex-of-real $(-$ cindex-pathE?g?z / 2)) we can make the proof shorter.


## proof -

have winding-number (poly rr $\circ$ rectpath $a b$ ) 0
$=-$ cindex-pathE (poly rr $\circ$ rectpath a b) $0 / 2$
proof (rule winding-number-cindex-pathE)
show finite-ReZ-segments (poly rr $\circ$ rectpath $a b$ ) 0 using finite-ReZ-segments-poly-rectpath .
show valid-path (poly rr $\circ$ rectpath $a b$ )
using valid-path-poly-rectpath .
show $0 \notin$ path-image (poly rr $\circ$ rectpath a b)
by (smt (z3) DiffE add.right-neutral add-diff-cancel-left' add-uminus-conv-diff
$\operatorname{assms}(1) \operatorname{assms(2)} \operatorname{assms(3)}$ basic-cqe-conv1(1) diff-add-cancel imageE
mult.right-neutral
mult-zero-right path-image-compose path-image-rectpath-cbox-minus-box poly-pCons rr-def)
show pathfinish (poly rr $\circ$ rectpath $a b$ ) $=$ pathstart ( poly rr $\circ$ rectpath a $b$ )
by (simp add: pathfinish-compose pathstart-compose)

```
    qed
    then show ?thesis by auto
    qed
    also have ... = 0
    proof -
    have winding-number (poly rr ○ rectpath a b) 0=0
    proof (rule winding-number-zero-outside)
        have path-image (poly rr 0 rectpath a b) = poly rr ' path-image (rectpath a b)
            using path-image-compose by simp
        also have ... = poly rr '(cbox a b-box a b)
            apply (subst path-image-rectpath-cbox-minus-box)
            using assms(1,2) by (simp|blast)+
            also have ...\subseteq(\lambdax. x-r)'cbox a b
                unfolding rr-def by (simp add: image-subset-iff)
            finally show path-image (poly rr o rectpath a b)\subseteq(\lambdax.x-r)'cbox a b .
            show 0 #(\lambdax.x - r)'cbox a b using assms(3) by force
            show path (poly rr ○ rectpath a b) by (simp add: path-poly-comp)
            show convex ((\lambdax.x-r)'cbox a b)
            using convex-box(1) convex-translation-subtract-eq by blast
            show pathfinish (poly rr 0 rectpath a b)= pathstart (poly rr \circ rectpath a b)
                by (simp add: pathfinish-compose pathstart-compose)
    qed
    then show ?thesis by simp
    qed
    finally show ?thesis unfolding rr-def by simp
qed
lemma cindexP-rectpath-add-one-root:
    assumes Re a<Reb Im a < Im b
        and not-rect-vertex rab
    and not-rect-vanishing pab
    shows cindexP-pathE ([:-r,1:]*p) (rectpath a b)=
            cindexP-pathE p (rectpath a b)
            + (if r\inbox a b then -2 else if r\inpath-image (rectpath a b) then - 1 else
0)
proof -
    define rr where rr = [:-r,1:]
    have rr-nzero:poly rr a\not=0 poly rr (Complex (Re b) (Im a))\not=0
                        poly rr b\not=0 poly rr (Complex (Re a) (Im b))\not=0
    using<not-rect-vertex r a b> unfolding rr-def not-rect-vertex-def by auto
    have p-nzero:poly p a\not=0 poly p(Complex (Re b) (Im a))\not=0
                    poly p b\not=0 poly p (Complex (Re a) (Im b))\not=0
    using <not-rect-vanishing p a b> unfolding not-rect-vanishing-def by auto
define cindp where cindp = (llpab.
                    cindexP-lineE p a (Complex (Re b) (Im a))
                        + cindexP-lineE p (Complex (Re b) (Im a)) b
                        + cindexP-lineE p b (Complex (Re a) (Im b))
```

$$
+ \text { cindexP-lineE } p(\text { Complex }(\operatorname{Re} a)(\operatorname{Im} b)) a
$$ )

define cdiff where cdiff $=\left(\begin{array}{ll}\lambda r r & p a b \\ \text {. }\end{array}\right.$
cdiff-aux rr pa(Complex (Re b) (Im a))

+ cdiff-aux rr $p($ Complex (Re b) (Im a) ) b
+ cdiff-aux rr pb(Complex (Re a) (Im b))
+ cdiff-aux rr $p($ Complex $(\operatorname{Re} a)(\operatorname{Im} b)) a$
)
have cindexP-pathE $(r r * p)($ rectpath a $b)=$
cindexP-pathE $(r r * p)$ (linepath a (Complex (Re b) (Im a) ))
+ cindexP-pathE $(r r * p)($ linepath $(C o m p l e x ~(R e ~ b) ~(I m ~ a)) ~ b) ~) ~$
+ cindexP-pathE $(r r * p)$ (linepath $b(C o m p l e x ~(R e a)(\operatorname{Im} b)))$
+ cindexP-pathE $(r r * p)($ linepath $(C o m p l e x ~(R e a)(\operatorname{Im} b)) a)$
unfolding rectpath-def Let-def
by ((subst cindex-poly-pathE-joinpaths
$\mid$ subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
( simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose) ?)+
also have $\ldots=$ cindexP-lineE $(r r * p)$ a (Complex (Re b) $(\operatorname{Im} a))$
+ cindexP-lineE (rr*p) (Complex (Re b) (Ima)) b
+ cindexP-lineE $(r r * p) b($ Complex $(R e a)(\operatorname{Im} b))$
+ cindexP-lineE $(r r * p)($ Complex (Re a) (Im b)) $a$
unfolding cindexP-lineE-def by simp
also have $\ldots=\operatorname{cindp}$ rr ab+cindp pab+cdiff rr pab/2
unfolding cindp-def cdiff-def
by (subst cindexP-lineE-times;
(use rr-nzero p-nzero one-complex.code imaginary-unit.code in simp)?)+
also have $\ldots=$ cindexP-pathE $p$ (rectpath a $b)+($ if $r \in b o x a b$ then -2 else
if $r \in$ path-image (rectpath a b) then -1 else 0 )
proof -
have cindp rr a $b=$ cindexP-path $E$ rr (rectpath a $b$ )
unfolding rectpath-def Let-def cindp-def cindexP-lineE-def
by ((subst cindex-poly-pathE-joinpaths
|subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?)+
also have $\ldots=($ if $r \in b o x$ a $b$ then -2 else
if $r \in$ path-image (rectpath a b) then -1 else 0 )
proof -
have ?thesis if $r \in b o x a b$
using cindexP-rectpath-interior-base rr-def that by presburger
moreover have ?thesis if $r \notin b o x a b r \in$ path-image (rectpath $a b$ ) using cindexP-rectpath-edge-base $[O F \operatorname{assms}(1,2,3)]$ that unfolding rr-def by auto
moreover have ?thesis if $r \notin$ box a b r£path-image (rectpath a b)
proof -
have $r \notin c b o x a b$
using that assms(1) assms(2) path-image-rectpath-cbox-minus-box by auto
then show ?thesis unfolding rr-def
using assms(1) assms(2) cindexP-rectpath-outside-base that(1) that(2)
by presburger
qed
ultimately show ?thesis by auto
qed
finally have cindp rr a $b=($ if $r \in b o x a b$ then -2 else
if $r \in$ path-image (rectpath $a b$ ) then -1 else 0).
moreover have cindp $p a b=$ cindexP-pathE $p$ (rectpath a $b$ )
unfolding rectpath-def Let-def cindp-def cindexP-lineE-def
by ((subst cindex-poly-pathE-joinpaths
|subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join
pathfinish-compose pathstart-compose poly-pcompose)?)+
moreover have cdiff rr pab=0
unfolding cdiff-def cdiff-aux-def by simp
ultimately show ?thesis by auto
qed
finally show ?thesis unfolding $r$ r-def .
qed
lemma proots-rect-cindexP-pathE:
assumes Re $a<\operatorname{Re} b \operatorname{Im} a<\operatorname{Im} b$
and not-rect-vanishing $p a b$

a b)) / 2
using 〈not-rect-vanishing pab〉
proof (induct prule:poly-root-induct-alt)
case 0
then have False unfolding not-rect-vanishing-def by auto
then show? case by simp
next
case (no-proots $p$ )
then obtain $c$ where $p c: p=[: c:] c \neq 0$
by (meson fundamental-theorem-of-algebra-alt)
have cindexP-pathE $p$ (rectpath a $b$ ) $=0$
using $p c$ by (auto intro:cindexP-pathE-const)
moreover have proots-rect pab=0 proots-rect-border pab=0
using pc proots-count-const
unfolding proots-rect-def proots-rect-border-def by auto
ultimately show ?case by auto
next
case (root rp)

```
define }rr\mathrm{ where }rr=[:-r,1:
```

```
have hyps:real (proots-rect p a b)=
                            -(proots-rect-border pab+ cindexP-pathE p (rectpath a b)) / 2
    apply (rule root(1))
    by (meson not-rect-vanishing-def poly-mult-zero-iff root.prems)
have cind-eq:cindexP-pathE (rr*p) (rectpath a b) =
        cindexP-pathE p (rectpath a b) +
            (if r box a b then - 2 else if r f path-image (rectpath a b) then - 1
else 0)
    proof (rule cindexP-rectpath-add-one-root[OF assms(1,2),of r p,folded rr-def])
    show not-rect-vertex r a b
        using not-rect-vanishing-def not-rect-vertex-def root.prems by auto
    show not-rect-vanishing p a b
        using not-rect-vanishing-def root.prems by force
    qed
    have rect-eq:proots-rect (rr * p) ab = proots-rect p ab
                                    + (if r\inbox a b then 1 else 0)
proof -
    have proots-rect (rr * p) ab
                = proots-count rr (box a b) + proots-rect p a b
        unfolding proots-rect-def
        apply (rule proots-count-times)
        by (metis not-rect-vanishing-def poly-0 root.prems rr-def)
    moreover have proots-count rr (box a b) = (if r\inbox a b then 1 else 0)
            using proots-count-pCons-1-iff rr-def by blast
    ultimately show ?thesis by auto
qed
have border-eq:proots-rect-border (rr * p) a b=
            proots-rect-border p a b
                + (if r f path-image (rectpath a b) then 1 else 0)
proof -
    have proots-rect-border (rr*p) a b = proots-count rr (path-image (rectpath a
b))
                        + proots-rect-border p a b
        unfolding proots-rect-border-def
        apply (rule proots-count-times)
        by (metis not-rect-vanishing-def poly-0 root.prems rr-def)
    moreover have proots-count rr (path-image (rectpath a b))
                =(if r path-image (rectpath a b) then 1 else 0)
        using proots-count-pCons-1-iff rr-def by blast
    ultimately show ?thesis by auto
qed
have ?case if r\in box a b
proof -
```

```
    have proots-rect (rr * p) ab = proots-rect p ab +1
            unfolding rect-eq using that by auto
    moreover have proots-rect-border (rr*p) ab= proots-rect-border p ab
        unfolding border-eq using that
        using assms(1) assms(2) path-image-rectpath-cbox-minus-box by auto
        moreover have cindexP-pathE (rr* p) (rectpath a b) = cindexP-pathE p
(rectpath a b) - 2
        using cind-eq that by auto
    ultimately show ?thesis using hyps
        by (fold rr-def) simp
    qed
    moreover have ?case if r& box a b r f path-image (rectpath a b)
    proof -
    have proots-rect (rr * p) ab= proots-rect p ab
        unfolding rect-eq using that by auto
    moreover have proots-rect-border (rr* p) ab = proots-rect-border pab+1
        unfolding border-eq using that
        using assms(1) assms(2) path-image-rectpath-cbox-minus-box by auto
        moreover have cindexP-pathE (rr*p)(rectpath a b) = cindexP-pathE p
(rectpath a b) - 1
            using cind-eq that by auto
    ultimately show ?thesis using hyps
            by (fold rr-def) auto
qed
moreover have ?case if r& box a b r & path-image (rectpath a b)
proof -
    have proots-rect (rr * p) a b = proots-rect p a b
            unfolding rect-eq using that by auto
    moreover have proots-rect-border (rr*p) a b = proots-rect-border p a b
            unfolding border-eq using that
            using assms(1) assms(2) path-image-rectpath-cbox-minus-box by auto
            moreover have cindexP-pathE (rr* p) (rectpath a b) = cindexP-pathE p
(rectpath a b)
            using cind-eq that by auto
    ultimately show ?thesis using hyps
            by (fold rr-def) auto
qed
ultimately show ?case by auto
qed
```


### 2.20 Code generation

lemmas Complex-minus-eq $=$ minus-complex.code
lemma cindexP-pathE-rect-smods:
fixes $p::$ complex poly and $l b u b::$ complex
assumes $a b$-le:Re $l b<\operatorname{Re} u b \operatorname{Im} l b<\operatorname{Im} u b$ and not-rect-vanishing $p l b \quad u b$
shows cindexP-pathE $p($ rectpath $l b u b)=$
(let p1 = pcompose $p[: l b$, Complex $($ Re $u b-$ Re lb) $0:]$;
$p R 1=$ map-poly Re $p 1 ; p I 1=$ map-poly $\operatorname{Im} p 1 ;$ gc1 $=$ gcd $p R 1 p I 1 ;$
$p 2=$ pcompose $p[:$ Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
$p R 2=$ map-poly Re p2; pI2 $=$ map-poly Im p2; gc2 $=$ gcd pR2 pI2;
p3 = pcompose $p$ [:ub, Complex (Re lb - Re ub) 0:];
$p R 3=$ map-poly Re p3; pI3 $=$ map-poly $\operatorname{Im} p 3 ;$ gc3 $=$ gcd $p R 3$ pI3;
$p_{4}=$ pcompose $p$ [:Complex (Re lb) (Im ub), Complex 0 (Im lb $-\operatorname{Im}$
$p_{4}=$ map-poly Re $p_{4} ; p I_{4}=$ map-poly $\operatorname{Im} p_{4} ; g c_{4}=$ gcd $p R_{4} p_{4}$ in
(changes-alt-itv-smods 01 ( $p$ R1 div gc1) (pI1 div gc1)

+ changes-alt-itv-smods 01 ( $p$ R2 div gc2) ( $\mathrm{pI2}$ div gc2)
+ changes-alt-itv-smods 01 ( $p R 3$ div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
) / 2) (is ? $L=$ ? $R$ )
proof -
have cindexP-pathE $p$ (rectpath $l b u b)=$
cindexP-lineE plb (Complex (Re ub) (Im lb))
+ cindexP-lineE ( $p$ ) (Complex (Re ub) (Im lb)) ub
+ cindexP-lineE ( $p$ ) ub (Complex (Re lb) (Im ub))
+ cindexP-lineE (p) (Complex (Re lb) (Im ub)) lb
unfolding rectpath-def Let-def cindexP-lineE-def
by ((subst cindex-poly-pathE-joinpaths
|subst finite-ReZ-segments-joinpaths
|intro path-poly-comp conjI);
(simp add:poly-linepath-comp finite-ReZ-segments-poly-of-real path-compose-join

```
pathfinish-compose pathstart-compose poly-pcompose)?)+
```

    also have \(\ldots=\) ? \(R\)
        apply (subst (1 23 4) cindexP-lineE-changes)
        subgoal using assms(3) not-rect-vanishing-def by fastforce
    subgoal by (smt (verit) assms(2) complex.sel(2))
    subgoal by (metis assms(1) complex.sel(1) order-less-irrefl)
    subgoal by (smt (verit) assms(2) complex.sel(2))
    subgoal by (metis assms(1) complex.sel(1) order-less-irrefl)
    subgoal unfolding Let-def by (simp-all add:Complex-minus-eq)
    done
    finally show ?thesis.
    qed
lemma open-segment-Im-equal:
assumes Re $x \neq \operatorname{Re} y \operatorname{Im} x=\operatorname{Im} y$
shows open-segment $x y=\{z \cdot \operatorname{Im} z=\operatorname{Im} x$
$\wedge \operatorname{Re} z \in$ open-segment $(\operatorname{Re} x)(\operatorname{Re} y)\}$
proof -
have open-segment $x y=\left(\lambda u .(1-u) *_{R} x+u *_{R} y\right)$ ' $\{0<. .<1\}$
unfolding open-segment-image-interval
using assms by auto

```
also have \(\ldots=(\lambda u\). Complex \((\operatorname{Re} x+u *(\operatorname{Re} y-\operatorname{Re} x))\)
                    \((\operatorname{Im} y)) \cdot\{0<. .<1\}\)
    apply (subst (1 2334 ) complex-surj[symmetric])
    using assms by (simp add:scaleR-conv-of-real algebra-simps)
also have \(\ldots=\{z \cdot \operatorname{Im} z=\operatorname{Im} x \wedge \operatorname{Re} z \in\) open-segment \((\operatorname{Re} x)(\operatorname{Re} y)\}\)
proof -
    have Re \(x+u *(\operatorname{Re} y-\operatorname{Re} x) \in\) open-segment \((\operatorname{Re} x)(\operatorname{Re} y)\)
    if Re \(x \neq \operatorname{Re} y \operatorname{Im} x=\operatorname{Im} y \quad 0<u u<1\) for \(u\)
    proof -
    define \(y x\) where \(y x=\operatorname{Re} y-\operatorname{Re} x\)
        have Re \(y=y x+\operatorname{Re} x y x>0 \vee y x<0\)
            unfolding \(y x\)-def using that by auto
    then show ?thesis
                unfolding open-segment-eq-real-ivl
                using that mult-pos-neg by auto
    qed
    moreover have \(z \in(\lambda x a\). Complex \((\operatorname{Re} x+x a *(\operatorname{Re} y-\operatorname{Re} x))(\operatorname{Im} y))\)
                        ' \(\{0<. .<1\}\)
        if \(\operatorname{Im} x=\operatorname{Im} y \operatorname{Im} z=\operatorname{Im} y \operatorname{Re} z \in\) open-segment \((\operatorname{Re} x)(\operatorname{Re} y)\) for \(z\)
        apply (rule rev-image-eqI[of (Rez-Rex)/(Rey-Rex)])
        subgoal
            using that unfolding open-segment-eq-real-ivl
            by (auto simp:divide-simps)
        subgoal using \(\langle R e x \neq R e y>~ c o m p l e x-e q-i f f\) that(2) by auto
        done
    ultimately show ?thesis using assms by auto
qed
finally show? ?thesis .
qed
lemma open-segment-Re-equal:
    assumes Re \(x=\operatorname{Re}\) y Im \(x \neq \operatorname{Im} y\)
    shows open-segment \(x y=\{z\). Re \(z=\operatorname{Re} x\)
                            \(\wedge \operatorname{Im} z \in\) open-segment \((\operatorname{Im} x)(\operatorname{Im} y)\}\)
proof -
    have open-segment \(x y=\left(\lambda u .(1-u) *_{R} x+u *_{R} y\right)\) ' \(\{0<. .<1\}\)
        unfolding open-segment-image-interval
        using assms by auto
    also have \(\ldots=(\lambda u\). Complex \((\operatorname{Re} y) \quad(\operatorname{Im} x+u *(\operatorname{Im} y-\operatorname{Im} x))\)
                        )' \(\{0<. .<1\}\)
    apply (subst (1 23 4) complex-surj[symmetric])
    using assms by (simp add:scaleR-conv-of-real algebra-simps)
also have \(\ldots=\{z \cdot \operatorname{Re} z=\operatorname{Re} x \wedge \operatorname{Im} z \in\) open-segment \((\operatorname{Im} x)(\operatorname{Im} y)\}\)
proof -
    have \(\operatorname{Im} x+u *(\operatorname{Im} y-\operatorname{Im} x) \in\) open-segment \((\operatorname{Im} x)(\operatorname{Im} y)\)
        if Im \(x \neq \operatorname{Im} y \operatorname{Re} x=\operatorname{Re} y \quad 0<u u<1\) for \(u\)
        proof -
            define \(y x\) where \(y x=\operatorname{Im} y-\operatorname{Im} x\)
            have \(\operatorname{Im} y=y x+\operatorname{Im} x y x>0 \vee y x<0\)
```

```
            unfolding yx-def using that by auto
            then show ?thesis
            unfolding open-segment-eq-real-ivl
            using that mult-pos-neg by auto
    qed
    moreover have z\in(\lambdaxa.Complex (Re y)(\operatorname{Im}x+xa*(\operatorname{Im}y-\operatorname{Im}x)))
                        '{0<..<1}
        if Rex=Re y Rez=Re y Im z\inopen-segment (Im x) (Im y) for z
        apply (rule rev-image-eqI[of (Im z - Im x)/(Im y - Im x)])
        subgoal
            using that unfolding open-segment-eq-real-ivl
            by (auto simp:divide-simps)
        subgoal using <Im x = Im y〉 complex-eq-iff that(2) by auto
        done
    ultimately show ?thesis using assms by auto
qed
finally show ?thesis .
qed
lemma Complex-eq-iff:
    x=Complex y z\longleftrightarrowRe }x=y\wedge\operatorname{Im}x=
    Complex y z=x \longleftrightarrowRe }x=y\wedge\operatorname{Im}x=
    by auto
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 -
    have p\not=0
        using not-rect-vanishing-def not-van order-root by blast
    define l1 l2 l3 l4 where l1 = open-segment lb (Complex (Re ub) (Im lb))
                            and l2 =open-segment (Complex (Re ub) (Im lb)) ub
                            and l3 =open-segment ub (Complex (Re lb) (Im ub))
                            and}\mp@subsup{l}{4}{}=\mathrm{ open-segment (Complex (Re lb) (Im ub)) lb
    have ll-eq:
        l1 ={z. Im z \in{Im lb} ^Rez R\in{Relb<..<Re ub}}
        l2 ={z.Rez 
        l3 ={z.Imz 
        l4}={z.Rez\in{Relb}\wedge\operatorname{Im}z\in{\operatorname{Im}lb<..<\operatorname{Im}ub}
        subgoal unfolding l1-def
            apply (subst open-segment-Im-equal)
            using assms unfolding open-segment-eq-real-ivl by auto
```

```
subgoal unfolding l2-def
    apply (subst open-segment-Re-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
subgoal unfolding l3-def
    apply (subst open-segment-Im-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
subgoal unfolding l4-def
    apply (subst open-segment-Re-equal)
    using assms unfolding open-segment-eq-real-ivl by auto
done
```

have ll-disj: $l 1 \cap l 2=\{ \} l 1 \cap l 3=\{ \} l 1 \cap l_{4}=\{ \}$
$12 \cap 13=\{ \} 12 \cap l_{4}=\{ \} 13 \cap l_{4}=\{ \}$
using assms unfolding $l l$-eq by auto
have proots-rect-border $p l b u b=$ proots-count $p$

$$
\begin{aligned}
& (\{z . \operatorname{Re} z \in\{\operatorname{Re} l b, \operatorname{Re} u b\} \wedge \operatorname{Im} z \in\{\operatorname{Im} \operatorname{lb} . \operatorname{Im} u b\}\} \cup \\
& \{z . \operatorname{Im} z \in\{\operatorname{Im} l b, \operatorname{Im} u b\} \wedge \operatorname{Re} z \in\{\operatorname{Re} l b . . \operatorname{Re} u b\}\})
\end{aligned}
$$

unfolding proots-rect-border-def
apply (subst path-image-rectpath)
using $\operatorname{assms}(1,2)$ by auto
also have $\ldots=$ proots-count $p$

$$
(\{z . \operatorname{Re} z \in\{\operatorname{Re} l b, \operatorname{Re} u b\} \wedge \operatorname{Im} z \in\{\operatorname{Im} l b<. .<\operatorname{Im} u b\}\} \cup
$$

$\{z . \operatorname{Im} z \in\{\operatorname{Im} l b, \operatorname{Im} u b\} \wedge \operatorname{Re} z \in\{\operatorname{Re} l b<. .<R e u b\}\}$
$\cup\{l b$, Complex (Re ub) (Im lb), ub, Complex (Re lb) (Im ub) $\}$ )
apply (rule arg-cong2[where $f=$ proots-count])
unfolding not-rect-vanishing-def using assms(1,2) complex.exhaust-sel
by (auto simp add:order.order-iff-strict intro:complex-eqI)
also have $\ldots=$ proots-count $p$
$(\{z . \operatorname{Re} z \in\{\operatorname{Re} l b, \operatorname{Re} u b\} \wedge \operatorname{Im} z \in\{\operatorname{Im} l b<. .<\operatorname{Im} u b\}\} \cup$
$\{z . \operatorname{Im} z \in\{\operatorname{Im} l b, \operatorname{Im} u b\} \wedge \operatorname{Re} z \in\{\operatorname{Re} l b<. .<\operatorname{Re} u b\}\})$

+ proots-count $p$
(\{lb,Complex (Re ub) (Im lb), ub, Complex (Re lb) (Im ub) \})
apply (subst proots-count-union-disjoint)
using $\langle p \neq 0$ 〉 by auto
also have $\ldots=$ proots-count $p$
$(\{z . \operatorname{Re} z \in\{\operatorname{Re} l b, \operatorname{Re} u b\} \wedge \operatorname{Im} z \in\{\operatorname{Im} l b<. .<\operatorname{Im} u b\}\} \cup$
$\{z \cdot \operatorname{Im} z \in\{\operatorname{Im} l b, \operatorname{Im} u b\} \wedge \operatorname{Re} z \in\{\operatorname{Re} l b<. .<\operatorname{Re} u b\}\})$
proof -
have proots-count $p$
$(\{l b$, Complex $($ Re $u b)(\operatorname{Im} l b), u b$, Complex $(\operatorname{Re} l b)(\operatorname{Im} u b)\})=0$
apply (rule proots-count-nzero)
using not-van unfolding not-rect-vanishing-def by auto
then show ?thesis by auto
qed
also have $\ldots=$ proots-count $p(l 1 \cup l 2 \cup l 3 \cup l 4)$
apply (rule arg-cong2[where $f=$ proots-count $]$ )
unfolding $l l-e q$ by auto
also have ... = proots-count p l1

```
+ proots-count p l2
+ proots-count p l3
+ proots-count pl4
    using ll-disj < p\not=0`
    by (subst proots-count-union-disjoint;
    (simp add:Int-Un-distrib Int-Un-distrib2 )?)+
    also have ... = proots-line plb (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
    unfolding proots-line-def l1-def l2-def l3-def l4-def by simp-all
    finally show ?thesis.
qed
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 plb 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 01 gc2 (pderiv gc2)
                    + changes-itv-smods-ext 0 1 gc3 (pderiv gc3)
                    + changes-itv-smods-ext 01 gc4 (pderiv gc4)
                    )) (is ? L=?R)
proof -
    have 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
    apply (rule proots-rect-border-eq-lines)
    by fact+
    also have ... = ?R
    proof -
    define p1 pR1 pI1 gc1 C1 where pp1:
        p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:]
        pR1 = map-poly Re p1
        pI1 = map-poly Im p1
        gc1 = gcd pR1 pI1
```

```
and
    C1=changes-itv-smods-ext 01 gc1 (pderiv gc1)
define p2 pR2 pI2 gc2 C2 where pp2:
    p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im lb):]
    pR2 = map-poly Re p2
    pI2 = map-poly Im p2
    gc2 = gcd pR2 pI2
    and
    C2=changes-itv-smods-ext 0 1 gc2 (pderiv gc2)
define p3 pR3 pI3 gc3 C3 where pp3:
    p3 =pcompose p [:ub, Complex (Re lb - Re ub) 0:]
    pR3 = map-poly Re p3
    pI3 = map-poly Im p3
    gc3 = gcd pR3 pI3
    and
    C3=changes-itv-smods-ext 01 gc3 (pderiv gc3)
define p4 pR4 pI4 gc4 C4 where pp4:
    p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im ub):]
    pR4 = map-poly Re p4
    pI4 = map-poly Im p4
    gc4 = gcd pR4 pI4
    and
    C4=changes-itv-smods-ext 0 1 gc4(pderiv gc4)
```

have poly gc1 $0 \neq 0$ poly gc1 $1 \neq 0$
poly gc2 $0 \neq 0$ poly gc2 $1 \neq 0$
poly gc3 $0 \neq 0$ poly gc3 $1 \neq 0$
poly gc4 $0 \neq 0$ poly gc4 $1 \neq 0$
unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff
using not-van[unfolded not-rect-vanishing-def]
by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+
have proots-line plb (Complex (Re ub) (Im lb)) $=$ nat C1
apply (subst proots-line-smods)
using not-van assms (1,2)
unfolding not-rect-vanishing-def C1-def pp1 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have proots-line $p$ (Complex (Re ub) (Im lb)) ub = nat C2
apply (subst proots-line-smods)
using not-van assms (1,2)
unfolding not-rect-vanishing-def C2-def pp2 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have proots-line pub (Complex (Re lb) (Im ub)) = nat C3
apply (subst proots-line-smods)
using not-van assms (1,2)
unfolding not-rect-vanishing-def C3-def pp3 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have proots-line $p$ (Complex (Re lb) (Im ub)) lb = nat C4
apply (subst proots-line-smods)
using not-van assms (1,2)
unfolding not-rect-vanishing-def C4-def pp4 Let-def
by (simp-all add:Complex-eq-iff Complex-minus-eq)
moreover have $C 1 \geq 0 C 2 \geq 0 C 3 \geq 0 C 4 \geq 0$
unfolding C1-def C2-def C3-def C4-def
by (rule changes-itv-smods-ext-geq- $0 ;($ fact $\mid$ simp $))+$
ultimately have proots-line plb (Complex (Re ub) (Im lb))

$$
\begin{aligned}
& \text { + proots-line } p \text { (Complex }(\text { Re ub) }(\text { Im lb) }) u b \\
& + \text { proots-line p ub (Complex }(\text { Re lb) }(\text { Im ub) }) \\
& + \text { proots-line } p(\text { Complex }(\text { Re lb) }(\text { Im ub)) lb } \\
& =\text { nat }(C 1+C 2+C 3+C 4)
\end{aligned}
$$

by linarith
also have ... $=$ ? $R$
unfolding C1-def C2-def C3-def C4-def pp1 pp2 pp3 pp4 Let-def
by $\operatorname{simp}$
finally show ?thesis .
qed
finally show ?thesis .
qed
lemma proots-rect-smods:
assumes Re $l b<\operatorname{Re} u b \operatorname{Im} l b<\operatorname{Im} u b$
and not-van:not-rect-vanishing $p l b u b$
shows proots-rect p lb $u b=($ let $p 1=$ pcompose $p[: l b$, Complex $($ Re ub - Re lb) $0:]$; $p R 1=$ 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
$l b):]$;
$p R 2=$ map-poly Re p2; pI2 $=$ map-poly Im p2; gc2 $=$ gcd pR2 pI2;
p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
$p R 3=$ map-poly Re p3; pI3 $=$ map-poly $\operatorname{Im} p 3 ;$ gc3 $=$ gcd pR3 pI3;
$p 4=$ pcompose $p$ [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
$u b):]$;
$p_{4}=$ map-poly Re p4; pI4 $=$ map-poly $\operatorname{Im} p_{4} ; g c_{4}=$ gcd $p R_{4} p_{4}$
in
nat (- (changes-alt-itv-smods 01 (pR1 div gc1) (pI1 div gc1)

+ changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 01 ( $p R 3$ div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
$+2 * c h a n g e s-i t v-s m o d s$-ext 01 gc1 (pderiv gc1)
$+2 *$ changes-itv-smods-ext 01 gc2 (pderiv gc2)
$+2 *$ changes-itv-smods-ext 01 gc3 (pderiv gc3)
$+2 *$ changes-itv-smods-ext 01 gc4 (pderiv gc4)) div 4)
)
proof -
define $p 1 p R 1$ pI1 gc1 C1 D1 where pp1:
$p 1=$ pcompose $p$ [:lb, Complex (Re ub-Re lb) 0:]
$p R 1=$ map-poly Re p1

$$
\begin{aligned}
p I 1 & =\text { map-poly } \operatorname{Im} p 1 \\
g c 1 & =\text { gcd pR1 pI1 }
\end{aligned}
$$

and $C 1=$ changes-itv-smods-ext 01 gc1 (pderiv gc1)
and D1 =changes-alt-itv-smods 01 (pR1 div gc1) (pI1 div gc1)
define p2 pR2 pI2 gc2 C2 D2 where pp2:
p2 $=$ pcompose $p[$ Complex $($ Re ub) (Im lb), Complex $0(\operatorname{Im} u b-\operatorname{Im} l b):]$
$p R 2=$ map-poly Re p2
$p I 2=$ map-poly Im $p 2$
$g c 2=g c d p R 2 p I 2$
and C2=changes-itv-smods-ext 01 gc2 (pderiv gc2)
and D2 = changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
define $p 3$ pR3 pI3 gc3 C3 D3 where pp3:
p3 = pcompose $p[: u b$, Complex (Re lb-Re ub) 0:]
$p R 3=$ map-poly Re $p 3$
pI3 $=$ map-poly Im p3
$g c 3=$ gcd $p R 3$ pI3
and C3=changes-itv-smods-ext 01 gc3 (pderiv gc3)
and $D 3=$ changes-alt-itv-smods 01 ( $p R 3$ div gc3) (pI3 div gc3)
define $p_{4} p R_{4} \mathrm{pI}_{4} g_{4} C_{4} D_{4}$ where $p_{4}$ :
$p_{4}=$ pcompose $p$ [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im ub):]
$p_{4}=$ map-poly Re $p_{4}$
pI4 $=$ map-poly $\operatorname{Im} p_{4}$
$g_{4}=$ gcd $p_{4}$ pI $_{4}$
and $C_{4}=$ changes-itv-smods-ext 01 gc4 (pderiv gc4)
and $D_{4}=$ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
have poly gc1 $0 \neq 0$ poly gc1 $1 \neq 0$
poly gc2 $0 \neq 0$ poly gc2 $1 \neq 0$
poly gc3 $0 \neq 0$ poly gc3 $1 \neq 0$
poly gc4 $0 \neq 0$ poly gc4 $1 \neq 0$
unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff
using not-van[unfolded not-rect-vanishing-def]
by (simp fip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+
have $C 1 \geq 0 \quad C 2 \geq 0 \quad C 3 \geq 0 \quad C 4 \geq 0$
unfolding $C 1-$ def $C 2$-def $C 3$-def $C 4$-def
by (rule changes-itv-smods-ext-geq- $0 ;($ fact $\mid$ simp $))+$
define $C C D D$ where $C C=C 1+C 2+C 3+C 4$

$$
\text { and } D D=D 1+D 2+D 3+D_{4}
$$

have real (proots-rect $p l b u b)=-($ real $($ proots-rect-border $p l b u b)$ + cindexP-pathE $p$ (rectpath $l b u b)$ ) / 2
apply (rule proots-rect-cindexP-pathE)
by fact+
also have $\ldots=-($ nat $C C+D D / 2) / 2$
proof -
have proots-rect-border plb ub=nat CC
apply (rule proots-rect-border-smods[ of $l b u b p$,

```
            unfolded Let-def,
            folded pp1 pp2 pp3 pp4,
            folded C1-def C2-def C3-def C4-def,
            folded CC-def])
        by fact+
    moreover have cindexP-pathE p (rectpath lb ub) = (real-of-int DD)/2
        apply (rule cindexP-pathE-rect-smods[
            of lb ub p,
            unfolded Let-def,
            folded pp1 pp2 pp3 pp4,
            folded D1-def D2-def D3-def D4-def,
            folded DD-def])
        by fact+
    ultimately show ?thesis by auto
qed
also have ... = - (DD + 2*CC)/4
    by (simp add: CC-def <0 \leqC1\rangle\langle0\leqC2\rangle\langle0\leqC3><0 \leqC4>)
finally have real (proots-rect p lb ub)
                        = real-of-int }(-(DD+2*CC))/4
    then have proots-rect plb ub=nat (- (DD +2*CC) div 4)
    by simp
then show ?thesis unfolding Let-def
    apply (fold pp1 pp2 pp3 pp4)
    apply (fold C1-def C2-def C3-def C4-def D1-def D2-def D3-def D4-def)
    by (simp add:CC-def DD-def)
qed
lemma proots-rect-code[code]:
proots-rect \(p l b u b=\)
(if Re \(l b<\operatorname{Re} u b \wedge \operatorname{Im} l b<\operatorname{Im} u b\) then
if not-rect-vanishing \(p l b u b\) then
(
let \(p 1=\) pcompose \(p[: l b\), Complex \((\) Re ub - Re lb) \(0:]\);
\(p R 1=\) map-poly Re p1; pI1 = map-poly \(\operatorname{Im} p 1 ; g c 1=\) gcd pR1 pI1;
\(p 2=\) pcompose \(p\) [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
\(p R 2=\) map-poly Re p2; pI2 \(=\) map-poly Im p2; gc2 \(=\) gcd pR2 pI2;
\(p 3=\) pcompose \(p[: u b\), Complex (Re lb - Re ub) 0:];
\(p R 3=\) map-poly Re p3; pI3 \(=\) map-poly \(\operatorname{Im} p 3 ;\) gc3 \(=\) gcd \(p\) R3 \(p I 3 ;\)
\(p_{4}=\) pcompose \(p\) [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
\(u b):\);
\(p_{4}=\) map-poly Re \(p_{4} ;\) pI \(_{4}=\) map-poly \(\operatorname{Im} p_{4} ; g c_{4}=\operatorname{gcd} p R_{4} p_{4}\) in
            nat (- (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 01 ( \(p R 3\) div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
\(+2 *\) changes-itv-smods-ext 01 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 01 gc4 (pderiv gc4)) div 4)
        )
        else Code.abort (STR ''proots-rect: the polynomial should not vanish
        at the four vertices for now'') (\lambda-. proots-rect p lb ub)
        else 0)
proof (cases Re lb<Re ub\wedgeIm lb<Im ub ^ not-rect-vanishing plb ub)
    case False
    have ?thesis if }\neg(\mathrm{ Re lb < Re ub) }\vee\neg(Im lb<Im ub
    proof -
    have box lb ub={} using that by (metis complex-box-ne-empty(2))
    then show ?thesis
        unfolding proots-rect-def
        using proots-count-emtpy that by fastforce
    qed
    then show ?thesis using False by auto
next
    case True
    then show ?thesis
        apply (subst proots-rect-smods)
        unfolding Let-def by simp-all
qed
lemma proots-rect-ll-rect:
    assumes Re lb < Re ub Im lb < Im ub
    and not-van:not-rect-vanishing p lb ub
    shows proots-rect-ll p lb ub = proots-rect p lb ub
    + proots-line plb (Complex (Re ub) (Im lb))
    + proots-line plb (Complex (Re lb) (Im ub))
proof -
    have p\not=0
        using not-rect-vanishing-def not-van order-root by blast
    define l1 l4 where l1 = open-segment lb (Complex (Re ub) (Im lb))
            and l4 = open-segment lb (Complex (Re lb) (Im ub))
    have ll-eq:
        l1 ={z. Im z \in{Imlb}^Rez\in{Relb<..<Re ub}}
        l4}={z.Rez\in{Re lb}\wedge\operatorname{Im}z\in{\operatorname{Im}lb<..<Imub}
        subgoal unfolding l1-def
            apply (subst open-segment-Im-equal)
            using assms unfolding open-segment-eq-real-ivl by auto
            subgoal unfolding l4-def
            apply (subst open-segment-Re-equal)
            using assms unfolding open-segment-eq-real-ivl by auto
            done
    have ll-disj:l1\cap l4 = {} box lb ub\cap{lb}={}
```

```
    box \(l b u b \cap l 1=\{ \}\) box \(l b u b \cap l_{4}=\{ \}\)
```

    \(l 1 \cap\{l b\}=\{ \} l 4 \cap\{l b\}=\{ \}\)
    using assms unfolding \(l l\)-eq
    by (auto simp:in-box-complex-iff)
    have proots-rect-ll plbub=proots-count \(p(b o x l b u b)\)
    $$
\begin{aligned}
& \text { + proots-count p }\{l b\} \\
& \text { + proots-count p l1 } \\
& + \text { proots-count p l4 }
\end{aligned}
$$

unfolding proots-rect-ll-def using $l l$-disj $\langle p \neq 0\rangle$
apply (fold l1-def l4-def)
by (subst proots-count-union-disjoint
;(simp add:Int-Un-distrib Int-Un-distrib2 del: Un-insert-right)?)+ also have $\ldots=$ proots-rect $p l b u b$

+ proots-line p lb (Complex (Re ub) (Im lb))
+ proots-line plb (Complex (Re lb) (Im ub))
proof -
have proots-count $p\{l b\}=0$
by (metis not-rect-vanishing-def not-van proots-count-nzero singleton-iff)
then show?thesis
unfolding proots-rect-def l1-def l4-def proots-line-def by simp
qed
finally show ?thesis .
qed
lemma proots-rect-ll-smods:
assumes Re lb $<$ Re ub Im $l b<\operatorname{Im} u b$
and not-van:not-rect-vanishing $p l b u b$
shows proots-rect-ll p lb ub $=($
let $p 1=$ pcompose $p$ [:lb, Complex (Re ub - Re lb) 0:];
$p R 1=$ map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1; $p 2=$ pcompose $p[:$ Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
$l b):]$;
$p R 2=$ map-poly Re p2; pI2 $=$ map-poly Im p2; gc2 $=$ gcd pR2 pI2;
$p 3=$ pcompose $p[: u b$, Complex (Re lb $-R e u b) 0:]$;
$p R 3=$ map-poly Re p3; pI3 $=$ map-poly Im p3; gc3 $=$ gcd $p R 3$ pI3;
$p_{4}=$ pcompose $p$ [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
$u b):]$;
$p_{4}=$ map-poly Re $p_{4} ;$ pI $_{4}=$ map-poly $\operatorname{Im} p_{4} ; g c_{4}=g c d p R_{4} p_{4}$ in
nat (- (changes-alt-itv-smods 01 ( $p$ R1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 01 ( $p$ R3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
- 2*changes-itv-smods-ext 01 gc1 (pderiv gc1)
$+2 *$ changes-itv-smods-ext 01 gc2 (pderiv gc2)
$+2 *$ changes-itv-smods-ext 01 gc3 (pderiv gc3)
- 2*changes-itv-smods-ext 01 gc4 (pderiv gc4)) div 4))
proof -
have $p \neq 0$
using not-rect-vanishing-def not-van order-root by blast
define $l 1 l_{4}$ where $l 1=$ open-segment $l b($ Complex $(\operatorname{Re} u b)(\operatorname{Im} l b))$
and $l_{4}=$ open-segment $l b$ (Complex (Re lb) (Im ub))
have $l_{4}$-alt:l4 $=$ open-segment $($ Complex $($ Re lb) $(\operatorname{Im} u b)) l b$
unfolding $l_{4}$-def by (simp add: open-segment-commute)
have $l l-e q$ :

$$
l 1=\{z . \operatorname{Im} z \in\{\operatorname{Im} l b\} \wedge \operatorname{Re} z \in\{\operatorname{Re} l b<. .<\operatorname{Re} u b\}\}
$$

$l_{4}=\{z . \operatorname{Re} z \in\{\operatorname{Re} l b\} \wedge \operatorname{Im} z \in\{\operatorname{Im} l b<. .<\operatorname{Im} u b\}\}$
subgoal unfolding l1-def
apply (subst open-segment-Im-equal)
using assms unfolding open-segment-eq-real-ivl by auto
subgoal unfolding $l_{4}$-def
apply (subst open-segment-Re-equal)
using assms unfolding open-segment-eq-real-ivl by auto
done
have $l l$-disj: $l 1 \cap l_{4}=\{ \}$ box $l b u b \cap\{l b\}=\{ \}$
box $l b u b \cap l 1=\{ \}$ box $l b u b \cap l_{4}=\{ \}$
$l 1 \cap\{l b\}=\{ \} l 4 \cap\{l b\}=\{ \}$
using assms unfolding $l l$-eq
by (auto simp:in-box-complex-iff)
define $p 1 p R 1$ pI1 gc1 C1 D1 where pp1: $p 1=$ pcompose $p$ [:lb, Complex (Re ub-Re lb) 0:]
pR1 = map-poly Re p1
$p I 1=$ map-poly $\operatorname{Im} p 1$
$g c 1=g c d p R 1 p I 1$
and $C 1=$ changes-itv-smods-ext 01 gc1 (pderiv gc1)
and D1=changes-alt-itv-smods 01 (pR1 div gc1) (pI1 div gc1)
define $p 2 p R 2$ pI2 gc2 C2 D2 where $p p 2$ : p2 $=$ pcompose $p[$ Complex (Re ub) (Im lb), Complex $0(\operatorname{Im} u b-\operatorname{Im} l b):]$ pR2 $=$ map-poly Re p2
$p I 2=$ map-poly Im $p 2$ gc2 $=$ gcd pR2 pI2
and C2=changes-itv-smods-ext 01 gc2 (pderiv gc2)
and D2 = changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
define $p 3$ pR3 pI3 gc3 C3 D3 where pp3:
$p 3=$ pcompose $p[: u b$, Complex $($ Re lb - Re ub) $0:]$
$p R 3=$ map-poly Re p3
$p I 3=$ map-poly Im p3
$g c 3=$ gcd pR3 pI3
and C3=changes-itv-smods-ext 01 gc3 (pderiv gc3)
and $D 3=$ changes-alt-itv-smods 01 (pR3 div gc3) (pI3 div gc3)
define $p_{4} p R_{4} \mathrm{pI}_{4} g_{4} C_{4} D_{4}$ where $p_{4}$ :
$p_{4}=$ pcompose $p[$ Complex (Re lb) (Im ub), Complex $0(\operatorname{Im} l b-\operatorname{Im} u b):]$
$p_{4}=$ map-poly Re $p_{4}$

$$
\begin{aligned}
p I_{4} & =\text { map-poly } \operatorname{Im} p_{4} \\
g c_{4} & =\text { gcd pR4 pI4 }
\end{aligned}
$$

and C4=changes-itv-smods-ext 01 gc4 (pderiv gc4)
and $D_{4}=$ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
have poly gc1 $0 \neq 0$ poly gc1 $1 \neq 0$
poly gc2 $0 \neq 0$ poly gc2 $1 \neq 0$
poly gc3 $0 \neq 0$ poly gc3 $1 \neq 0$
poly gc $40 \neq 0$ poly gc $41 \neq 0$
unfolding pp1 pp2 pp3 pp4 poly-gcd-0-iff
using not-van[unfolded not-rect-vanishing-def]
by (simp flip:Re-poly-of-real Im-poly-of-real add:poly-pcompose
; simp add: Complex-eq-iff zero-complex.code plus-complex.code)+
have CC-pos: $C 1 \geq 0 C 2 \geq 0 C 3 \geq 0 C 4 \geq 0$
unfolding C1-def C2-def C3-def C4-def
by (rule changes-itv-smods-ext-geq- $0 ;($ fact $\mid$ simp $))+$
define $C C D D$ where $C C=C 2+C 3-C 4-C 1$

$$
\text { and } D D=D 1+D 2+D 3+D 4
$$

define p1 p2 p3 p4 where pp:p1=proots-line plb(Complex (Re ub) (Im lb))

$$
p 2=\text { proots-line } p(\text { Complex }(\text { Re } u b)(\operatorname{Im} l b)) u b
$$

p3 $=$ proots-line $p u b$ (Complex (Re lb) (Im ub))
$p 4=$ proots-line $p($ Complex $($ Re lb) $(\operatorname{Im} u b)) l b$
have $p_{4}$-alt: $p_{4}=$ proots-line $p l b($ Complex $($ Re lb) $)(\operatorname{Im} u b))$
unfolding $p p$ by (simp add: proots-line-commute)
have real $\left(\right.$ proots-rect-ll plbub) $=$ real $\left(\right.$ proots-rect plbub) $+p 1+p_{4}$
unfolding $p p$ by (simp add: proots-rect-ll-rect[OF assms] proots-line-commute)
also have $\ldots=(p 1+p 4-$ real $p 2-$ real $p 3-$ cindexP-pathE $p$ (rectpath lb
ub)) / 2
proof -
have real (proots-rect plbub) $=-($ real (proots-rect-border $p l b u b)$

+ cindexP-pathE $p$ (rectpath $l b u b)$ ) / 2
apply (rule proots-rect-cindexP-pathE)
by fact+
also have $\ldots=-(p 1+p 2+p 3+p 4+$ cindexP-pathE $p($ rectpath $l b u b)) /$
2
using proots-rect-border-eq-lines[OF assms,folded pp] by simp
finally have real (proots-rect plbub) =

$$
\begin{aligned}
- & (\text { real }(p 1+p 2+p 3+p 4) \\
& + \text { cindexP-pathE } p(\text { rectpath } l b u b)) / 2 .
\end{aligned}
$$

then show ?thesis by auto
qed
also have $\ldots=($ nat $C 1+$ nat $C 4-$ real $($ nat C2 $)-$ real $($ nat C3 $)$
$-(($ real-of-int $D D) / 2)) / 2$
proof -
have $p 1=$ nat C1 p2 $=$ nat C2 p3 $=$ nat C3 p4 $=$ nat $C 4$
using not-van[unfolded not-rect-vanishing-def] $\operatorname{assms}(1,2)$
unfolding $p$ p C1-def pp1 C2-def pp2 C3-def pp3 C4-def pp4
by (subst proots-line-smods ;simp-all add:Complex-eq-iff Let-def Complex-minus-eq)+
moreover have cindexP-pathE $p($ rectpath $l b u b)=($ real-of-int $D D) / 2$
apply (rule cindexP-pathE-rect-smods[ of $l b u b p$,
unfolded Let-def,
folded pp1 pp2 pp3 pp4,
folded D1-def D2-def D3-def D4-def, folded $D D-d e f]$ )
by fact+
ultimately show ?thesis by presburger
qed
also have $\ldots=-(D D+2 * C C) / 4$
unfolding CC-def using CC-pos by (auto simp add:divide-simps algebra-simps)
finally have real (proots-rect-ll p lb ub)

$$
=\text { real-of-int }(-(D D+2 * C C)) / 4 .
$$

then have proots-rect-ll plb ub

$$
=\operatorname{nat}(-(D D+2 * C C) \operatorname{div} 4)
$$

by $\operatorname{simp}$
then show ?thesis
unfolding Let-def
apply (fold pp1 pp2 pp3 pp4)
apply (fold C1-def C2-def C3-def C4-def D1-def D2-def D3-def D4-def)
by (simp add:CC-def $D D-d e f)$
qed
lemma proots-rect-ll-code[code]:
proots-rect-ll plbub=
(if Re $l b<R e ~ u b \wedge I m l b<I m$ ub then
if not-rect-vanishing $p l b$ ub then
(
let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0:];
$p R 1=$ 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
$l b):] ;$
$p R 2=$ map-poly Re p2; pI2 $=$ map-poly Im p2; gc2 $=$ gcd pR2 pI2;
p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0:];
$p R 3=$ map-poly Re p3; pI3 $=$ map-poly $\operatorname{Im} p 3 ;$ gc3 $=$ gcd $p R 3$ pI3;
$p_{4}=$ pcompose $p$ [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
$u b):]$;
$p R_{4}=$ map-poly Re p4;pI4 $=$ map-poly Im p4; gc4 $=$ gcd pR4 pI4 in
nat (- (changes-alt-itv-smods 01 ( $p$ R1 div gc1) ( $p$ I1 div gc1)

+ changes-alt-itv-smods 01 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 01 ( $p$ R3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 01 (pR4 div gc4) (pI4 div gc4)
- 2*changes-itv-smods-ext 01 gc1 (pderiv gc1)
$+2 * c h a n g e s-i t v-s m o d s$-ext 01 gc2 (pderiv gc2)

```
            +2*changes-itv-smods-ext 01 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'') (\lambda-. proots-rect-ll p lb ub)
        else Code.abort (STR "proots-rect-ll: the box is improper")
            (\lambda-. proots-rect-ll p lb ub))
proof (cases Re lb < Re ub\wedgeIm lb<Im ub ^ not-rect-vanishing p lb ub)
    case False
    then show ?thesis using False by auto
next
    case True
    then show ?thesis
        apply (subst proots-rect-ll-smods)
        unfolding Let-def by simp-all
qed
end
```


## 3 Procedures to count the number of complex roots in various areas

## theory Count-Complex-Roots imports

Count-Half-Plane
Count-Line
Count-Circle
Count-Rectangle
begin
end

## 4 Some examples for complex root counting

theory Count-Complex-Roots-Examples
imports Count-Complex-Roots
begin
value proots-rect [:2*i, $0, \mathrm{i}:]($ Complex $(-1) 0)($ Complex 22$)$
value proots-rect $[:-1,-2 * \mathrm{i}, 1:]$
(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:]) 04
end

## 5 Acknowledgements

The work was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council and led by Professor Lawrence Paulson at the University of Cambridge, UK.

## References

[1] M. Eisermann. The fundamental theorem of algebra made effective: An elementary real-algebraic proof via Sturm chains. American Mathematical Monthly, 119(9):715-752, 2012.
[2] Q. I. Rahman and G. Schmeisser. Analytic theory of polynomials. Number 26. Oxford University Press, 2002.

