Count the Number of Complex Roots

Wenda Li
April 20, 2020

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 An alternative Sturm sequences

theory Extended-Sturm imports
Sturm-Tarski,Sturm-Tarski
Winding-Number-Eval,Cauchy-Index-Theorem
begin

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

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

hide-const Permutations.sign

1.1 Misc

lemma is-unit-pCons-ex-iff:
  fixes p::'a::field poly
  shows is-unit p \longleftrightarrow (\exists a. a\neq0 \land p=[::a:])
  using is-unit-poly-iff is-unit-triv by auto

lemma poly-gcd-iff:
  poly (gcd p q) x=0 \longleftrightarrow poly p x=0 \land poly q x=0
  by (simp add: poly-eq-0-iff-dvd)
lemma \textit{eventually-poly-nz-at-within}:
fixes \(x\) :: 'a::{idom,euclidean-space}
assumes \(p \neq 0\)
shows \(\text{eventually } (\lambda x. \text{poly } p \ x \neq 0) \ (\text{at } x \text{ within } S)\)
proof
\begin{itemize}
  \item have \(\text{eventually } (\lambda x. \text{poly } p \ x \neq 0) \ (\text{at } x \text{ within } S)\)
    \(= (\forall F \ x \in (\text{at } x \text{ within } S). \forall y \in \text{proots } p. \ x \neq y)\)
  \item apply (\text{rule eventually-subst,rule eventuallyI})
  \item by (\text{auto simp add:proots-def})
\end{itemize}
also have \(\ldots = (\forall y \in \text{proots } p. \forall F \ x \in (\text{at } x \text{ within } S), \ x \neq y)\)
apply (\text{subst eventually-ball-finite-distrib})
using \(\langle p \neq 0 \rangle\) by \text{auto}
also have \(\ldots\)
unfolding \text{eventually-at}
by (\text{metis gt-ex not-less-iff-gr-or-eq zero-less-dist-iff})
finally show \(\text{??thesis}\).
qed

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

lemma \textit{poly-divide-filterlim-at-top}:
fixes \(p \ q\) :: real poly
defines \(ll \equiv (\text{if degree } q<\text{degree } p \text{ then}\)
\(\text{at } 0\)
\(\text{else if degree } q=\text{degree } p \text{ then}\)
\(\text{nhds } (\text{lead-coeff } q / \text{lead-coeff } p)\)
\(\text{else if } \text{sgn-pos-inf } q * \text{sgn-pos-inf } p > 0 \text{ then}\)
\(\text{at-top}\)
\(\text{else}\)
\(\text{at-bot}\)
assumes \(p \neq 0 \ q \neq 0\)
shows \(\text{filterlim } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ ll \text{ at-top}\)
proof
\begin{itemize}
  \item define \(pp\) where \(pp=(\lambda x. \text{poly } p \ x / x^{\text{degree } p})\)
  \item define \(qq\) where \(qq=(\lambda x. \text{poly } q \ x / x^{\text{degree } q})\)
  \item define \(dd\) where \(dd=(\lambda x::\text{real}. \text{if degree } p>\text{degree } q \text{ then } 1/x^{\text{degree } p – \text{degree } q} \text{ else}\)
    \(x^{\text{degree } q – \text{degree } p})\)
  \item have divide-cong:\(\forall F \ x \in \text{at-top}. \text{poly } q \ x / \text{poly } p \ x = qq \ x / pp \ x * dd \ x\)
  \item proof (\text{rule eventually-at-top-linorderI[of 1]})
  \item fix \(x\) assume \(\langle x::\text{real}\rangle \geq 1\)
  \item then have \(x \neq 0\) by \text{auto}
  \item then show \(\text{poly } q \ x / \text{poly } p \ x = qq \ x / pp \ x * dd \ x\)
    unfolding \(qq\)-def \(pp\)-def \(dd\)-def \using\ \text{assms}
    by (\text{auto simp add:field-simps power-diff})
\end{itemize}
qed
have qpp-tendsto:\(\lambda x. \text{qq } x / \text{pp } x\) \(\longrightarrow\) lead-coeff q / lead-coeff p) at-top
proof – have \((\text{qq}\longrightarrow\text{lead-coeff q}) \text{ at-top}\)
unfolding \text{qq-def using poly-divide-tendsto-aux[of q]}
by (auto elim!:filterlim-mono simp:at-top-le-at-infinity)
moreover have \((\text{pp}\longrightarrow\text{lead-coeff p}) \text{ at-top}\)
unfolding pp-def using poly-divide-tendsto-aux[of p]
by (auto elim!:filterlim-mono simp:at-top-le-at-infinity)
ultimately show \(\text{thesis using }\langle p \neq 0\rangle\) by (auto intro!:tendsto-eq-intros)
qed

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

moreover have ?thesis when degree q > degree p 
  - sgn-pos-inf q * sgn-pos-inf p > 0
proof -
  have filterlim (λx. (qq x / pp x) * dd x) at-bot at-top
proof (subst filterlim-tendsto-neg-mult-at-bot-iff[OF qqpp-tendsto])
  show lead-coeff q / lead-coeff p < 0
  using that(2) : p≠0 ; q≠0; 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≡( if degree q < degree p then
    at 0
  else if degree q = degree p then
    nhds (lead-coeff q / lead-coeff p)
  else if sgn-neg-inf q * sgn-neg-inf p > 0 then
    at-top
  else
    at-bot)
  assumes p≠0 q≠0
  shows filterlim (λx. poly q x / poly p x) ll at-bot
proof -
  define pp where pp≡(λx. poly p x / x^(degree p))
  define qq where qq≡(λx. poly q x / x^(degree q))
  define dd where dd≡(λx::real. if degree p > degree q then 1 / x^(degree p - degree q) else
  x^(degree q - degree p))
  have divide-cong:\\forall x in at-bot. poly q x / poly p x = qq x / pp x * dd x
proof (rule eventually-at-bot-linorder1[of -1])
  fix x assume (x::real)≤-1
  then have x≠0 by auto
  then show poly q x / poly p x = qq x / pp x * dd x
    unfolding qq-def pp-def dd-def using assms
  by (auto simp add:field_simps power-diff)
qed
have qqpp-tendsto:((λx. qx x / pp x) ----> lead-coeff q / lead-coeff p) at-bot

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

have ?thesis when degree \(q < \text{degree } p\)
proof
  have \(\text{filterlim }((\lambda x. \text{poly } q \, x / \text{poly } p \, x) \, (at \, 0))\) at-bot
  proof (rule filterlim-atI)
    show \((\lambda x. \text{poly } q \, x / \text{poly } p \, x) \rightarrow 0\) at-bot
    using \(\text{poly-divide-tendsto-0-at-infinity[of that]}\)
    by (auto elim!:\text{filterlim-mono simp:at-bot-le-at-infinity})
    have \(\forall \, F \, x \, \text{in at-bot. } \text{poly } q \, x \neq 0\) \(\forall \, F \, x \, \text{in at-bot. } \text{poly } p \, x \neq 0\)
    using \(\text{poly-eventually-not-zero[of } q \neq 0\]\(\text{poly-eventually-not-zero[of } p \neq 0\]\)
    filter-leD[\(\text{OF at-bot-le-at-infinity}]\)
    by auto
    then show \(\forall \, F \, x \, \text{in at-bot. } \text{poly } q \, x / \text{poly } p \, x \neq 0\)
    by eventually-elim auto
  qed
then show ?thesis unfolding ll-def using that by auto
qed
moreover have ?thesis when degree \(q = \text{degree } p\)
proof
  have \((\lambda x. \text{poly } q \, x / \text{lead-coeff } q / \text{lead-coeff } p)\) at-bot
  using \(\text{divide-cong qpp-tendsto that}\) unfolding dd-def
  by (auto dest: \text{tendsto-cong})
  then show ?thesis using \(\langle p \neq 0 \rangle\) \(\langle q \neq 0 \rangle\)
  unfolding sgn-neg-inf-def cc-def zero-less-mult-iff
  divide-less-0-iff zero-less-divide-iff
  apply (simp add: if-split[of (<) 0] if-split[of (>) 0])
  by argo
qed

moreover have ?thesis when degree \(q > \text{degree } p\) \(\text{sgn-neg-inf } q \times \text{sgn-neg-inf } p > 0\)
proof
  define \(cc\) where \(cc = \text{lead-coeff } q / \text{lead-coeff } p\)
  have \((cc > 0 \land \text{even } (\text{degree } q - \text{degree } p)) \lor (cc < 0 \land \text{odd } (\text{degree } q - \text{degree } p))\)
  proof
    have even (degree \(q - \text{degree } p\) \(\iff\)
      \((\text{even } (\text{degree } q) \land \text{even } (\text{degree } p)) \lor (\text{odd } (\text{degree } q) \land \text{odd } (\text{degree } p))\)
    using \(\text{degree } q > \text{degree } p\) by auto
    then show ?thesis
      using that \(p \neq 0\) \(\langle q \neq 0 \rangle\) unfolding sgn-neg-inf-def cc-def zero-less-mult-iff
      divide-less-0-iff zero-less-divide-iff
      apply (simp add: if-split[of (<) 0] if-split[of (>) 0])
      by argo
  qed
qed
moreover have \( \text{filterlim} (\lambda x. (q q x / p p x) \ast dd x) \) at-top at-bot when \( cc > 0 \) even \( (\text{degree } q - \text{degree } p) \)

proof (subst \text{filterlim-tendsto-pos-mull-at-top-iff}[\text{OF qqqp-tendsto}])
show \( 0 < \text{lead-coeff } q / \text{lead-coeff } p \) using \( (cc > 0) \) unfolding cc-def by auto
show filterlim dd at-top at-bot
unfolding dd-def using \( (\text{degree } q > \text{degree } p) \) that(2)
by (auto intro!:filterlim-pow-at-bot-even simp:filterlim-ident)
qed

moreover have \( \text{filterlim} (\lambda x. (q q x / p p x) \ast dd x) \) at-top at-bot when \( cc < 0 \) odd \( (\text{degree } q - \text{degree } p) \)

proof (subst \text{filterlim-tendsto-neg-mull-at-top-iff}[\text{OF qqqp-tendsto}])
show \( 0 > \text{lead-coeff } q / \text{lead-coeff } p \) using \( (cc < 0) \) unfolding cc-def by auto
show filterlim dd at-bot at-bot
unfolding dd-def using \( (\text{degree } q > \text{degree } p) \) that(2)
by (auto intro!:filterlim-pow-at-bot-odd simp:filterlim-ident)
qed

ultimately have \( \text{filterlim} (\lambda x. (q q x / p p x) \ast dd x) \) at-top at-bot by blast
then have \( \text{LIM } x \) at-bot. \( poly q x / poly p x :> \) at-top using filterlim-cong[\text{OF - divide-cong}] by blast
then show \( ?\text{thesis} \) unfolding \( \text{ll-def} \) using that by auto
qed

moreover have \( ?\text{thesis} \) when \( \text{degree } q > \text{degree } p - \text{sgn-neg-inf } q \ast \text{sgn-neg-inf } p > 0 \)

proof –
define cc where \( cc = \text{lead-coeff } q / \text{lead-coeff } p \)
have \( (cc < 0 \land \text{even } (\text{degree } q - \text{degree } p)) \lor (cc > 0 \land \text{odd } (\text{degree } q - \text{degree } p)) \)

proof –
have even \( (\text{degree } q - \text{degree } p) \) \( \iff \)
\( (\text{even } (\text{degree } q) \land \text{even } (\text{degree } p)) \lor (\text{odd } (\text{degree } q) \land \text{odd } (\text{degree } p)) \)
using \( (\text{degree } q > \text{degree } p) \) by auto
then show \( ?\text{thesis} \)
using that \( (p \neq 0) \) unfolding \( \text{sgn-neg-inf-def cc-def zero-less-mult-iff divide-less-0-iff zero-less-divide-iff} \)
apply \( (\text{simp add:if-split[of } (<) 0]\ \text{if-split[of } (>) 0]}\)
by (metis leading-coeff-0-iff linorder-neqE-linordered-idom)
qed

moreover have \( \text{filterlim} (\lambda x. (q q x / p p x) \ast dd x) \) at-bot at-bot when \( cc < 0 \) even \( (\text{degree } q - \text{degree } p) \)

proof (subst \text{filterlim-tendsto-neg-mull-at-bot-iff}[\text{OF qqqp-tendsto}])
show \( 0 > \text{lead-coeff } q / \text{lead-coeff } p \) using \( (cc < 0) \) unfolding cc-def by auto
show filterlim dd at-bot at-bot
unfolding dd-def using \( (\text{degree } q > \text{degree } p) \) that(2)
by (auto intro!:filterlim-pow-at-bot-even simp:filterlim-ident)
qed

moreover have \( \text{filterlim} (\lambda x. (q q x / p p x) \ast dd x) \) at-bot at-bot when \( cc > 0 \) odd \( (\text{degree } q - \text{degree } p) \)

proof (subst \text{filterlim-tendsto-pos-mull-at-bot-iff}[\text{OF qqqp-tendsto}])
show 0 < lead-coeff q / lead-coeff p using ⟨cc>0⟩ unfolding cc-def by auto
show filterlim dd at-bot at-bot
  unfolding dd-def using ⟨degree q>degree p⟩ that(2)
  by (auto intro!:filterlim-pow-at-bot-odd simp;filterlim-ident)
qed
ultimately have filterlim (λx. (qq x / pp x) * dd x) at-bot at-bot
  by blast
then have LIM x at-bot. poly q x / poly p x :> at-bot
  using filterlim-corg[OF - - divide-cong] by blast
then show ?thesis unfolding ll-def using that by auto
qed
ultimately show ?thesis by linarith
qed

1.2 Alternative definition of cross

definition cross-alt :: real poly ⇒real poly ⇒ real ⇒ real ⇒ int where
cross-alt p q a b = |sign (poly p a) − sign (poly q a)| − |sign (poly p b) − sign(poly q b)|

lemma cross-alt-coprime-0:
  assumes coprime p q p=0∨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 simp
    then obtain a where a≠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 simp
    then obtain a where a≠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 (p=0∨q=0) by auto
qed

lemma cross-alt-0[simp]; cross-alt 0 0 a b=0 unfolding cross-alt-def by auto

lemma cross-alt-poly-commute:
  cross-alt p q a b = cross-alt q p a b
  unfolding cross-alt-def by auto

lemma cross-alt-clear-n:
  assumes coprime p q
shows \( \operatorname{cross-alt} p q a b = \operatorname{cross-alt} 1 (p \ast q) a b \)

proof 
- have \(|\operatorname{sign}(\operatorname{poly} p a) - \operatorname{sign}(\operatorname{poly} q a)| = |1 - \operatorname{sign}(\operatorname{poly} p a) \ast \operatorname{sign}(\operatorname{poly} q a)|\)
  proof (cases \(\operatorname{poly} p a = 0 \land \operatorname{poly} q a = 0\))
  case True
  then have \(\text{False using \text{assms using \text{coprime-poly-0}} by \text{blast}}\)
  then show \(\text{?thesis by \text{auto}}\)
next
  case False
  then show \(\text{?thesis}\)
    unfolding \(\text{Sturm-Tarski.sign-def}\)
    by force
qed
moreover have \(|\operatorname{sign}(\operatorname{poly} p b) - \operatorname{sign}(\operatorname{poly} q b)| = |1 - \operatorname{sign}(\operatorname{poly} p b) \ast \operatorname{sign}(\operatorname{poly} q b)|\)
  proof (cases \(\operatorname{poly} p b = 0 \land \operatorname{poly} q b = 0\))
  case True
  then have \(\text{False using \text{assms using \text{coprime-poly-0}} by \text{blast}}\)
  then show \(\text{?thesis by \text{auto}}\)
next
  case False
  then show \(\text{?thesis}\)
    unfolding \(\text{Sturm-Tarski.sign-def}\)
    by force
qed
ultimately show \(\text{?thesis}\)
  by (simp add: \text{cross-alt-def sign-times})
qed

1.3 Alternative sign variation sequencse

fun \(\text{changes-alt:: \('a ::\text{linordered-idom} list \Rightarrow \text{int}}\) where
changes-alt [] = 0
changes-alt [x] = 0
changes-alt (x1#x2#xs) = abs(sign x1 - sign x2) + changes-alt (x2#xs)

definition \(\text{changes-alt-poly-at::\('a ::\text{linordered-idom} poly list \Rightarrow \text{'a \Rightarrow \text{int}}}}\) poly list \(\Rightarrow \text{'a \Rightarrow \text{int}}\) where
changes-alt-poly-at ps a= changes-alt (map (\(\lambda p\). poly p a) ps)

definition \(\text{changes-alt-itv-smods:: real \Rightarrow real \Rightarrow real poly \Rightarrow real poly \Rightarrow \text{int}}\) 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 \(\text{changes-alt-itv-smods-rec}\):
  assumes \(a < b \text{ coprime p q}\)
  shows \(\text{changes-alt-itv-smods a b p q = \operatorname{cross-alt} p q a b + \text{changes-alt-itv-smods a b q} (- (p \text{mod q})}}\)
proof \((\text{cases } p = 0 \lor q = 0 \lor q \text{ dvd } p)\)

\textbf{case True}

moreover have \(p=0 \lor q=0 \implies \) \(?\text{thesis}\)

\textbf{using} \(\text{cross-alt-coprime-0} \langle OF \langle \text{coprime } p \ q \rangle \rangle\)

moreover have \([p\neq 0; q\neq 0; p \text{ mod } q = 0] \implies \) \(?\text{thesis}\)

\textbf{unfolding} \(\text{changes-alt-ite-smods-def} \ \text{changes-alt-poly-at-def} \ \text{cross-alt-def}\)

\textbf{by} \((\text{simp add:sign-times})\)

ultimately show \(?\text{thesis}\)

\textbf{by} \(\text{auto} \) \((\text{auto elim: dvdE})\)

\textbf{next}

\textbf{case False}

hence \(p \neq 0 \ \text{and} \ q \neq 0 \ \text{and} \ p \mod q \neq 0\) \(\text{by} \ \text{auto}\)

then obtain \(ps\) where \(ps\): \(\text{smods } p \ q = p \#q \#-(p \mod q)\#ps \ \text{smods} \ q \ -(p \mod q)\#ps\)

\textbf{by} \(\text{auto}\)

\textbf{define} \(\text{changes-diff} \ \text{where} \ \text{changes-diff} \equiv \lambda x. \ \text{changes-alt-poly-at} \ (p\#q\#-(p \mod q)\#ps)\ x\)

\textbf{by} \(\text{force}\)

qed

1.4 \textbf{jumpF on polynomials}

definition \(\text{jumpF-polyR}:: \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{poly} \Rightarrow \text{real} \Rightarrow \text{real} \ \text{where} \)

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

definition \(\text{jumpF-polyL}:: \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{poly} \Rightarrow \text{real} \Rightarrow \text{real} \ \text{where} \)

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

definition \(\text{jumpF-poly-top}:: \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{where} \)

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

definition \(\text{jumpF-poly-bot}:: \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{poly} \Rightarrow \text{real} \ \text{where} \)

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

lemma \(\text{jumpF-polyR-0}[\text{simp}]: \text{jumpF-polyR} \ 0 \ a = 0 \ \text{jumpF-polyR} \ q \ 0 \ a = 0\)

\textbf{unfolding} \(\text{jumpF-polyR-def} \ \text{by} \ \text{auto}\)

lemma \(\text{jumpF-polyL-0}[\text{simp}]: \text{jumpF-polyL} \ 0 \ a = 0 \ \text{jumpF-polyL} \ q \ 0 \ a = 0\)

\textbf{unfolding} \(\text{jumpF-polyL-def} \ \text{by} \ \text{auto}\)

lemma \(\text{jumpF-polyR-mul-cancel}: \)

\(\text{assumes } p \neq 0\)
shows \( \text{jumpF-polyR} \ (p' \ast q) \ (p' \ast p) \ a = \text{jumpF-polyR} \ q \ p \ a \)
unfolding \( \text{jumpF-polyR-def} \)
proof (rule jumpF-cong)
  obtain \( \text{ub} \) where \( a < \text{ub} \ \forall z. \ a < z \land z < \text{ub} \longrightarrow \text{poly} \ p' \ z \neq 0 \)
  using next-non-root-interval[OF \( \langle p' \neq 0 \rangle, of \ a \)] by auto
  then show \( \forall F \ x \ \text{in at-right a. poly} \ (p' \ast q) \ x / \text{poly} \ (p' \ast p) \ x = \text{poly} \ q \ x / \text{poly} \ p \ x \)
  apply (unfold eventually-at-right)
  apply (intro exI[where \( x = \text{ub} \)])
  by auto
qed simp

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

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

lemma \( \text{jumpF-polyR-coprime} \):
  assumes \( \text{coprime} \ p \ q \)
  shows \( \text{jumpF-polyR} \ q \ p \ x = (\text{if} \ p \neq 0 \ \land \ q \neq 0 \ \land \ \text{poly} \ p \ x = 0 \ \text{then} \)
  \( \text{if} \ \text{sign-r-pos} \ p \ x \longleftrightarrow \text{poly} \ q \ x > 0 \ \text{then} \ 1/2 \ \text{else} \ -1/2 \)
else 0)
proof (cases \( p = 0 \ \lor \ q = 0 \ \lor \ \text{poly} \ p \ x \neq 0 \))
  case True
  then show \( \text{thesis using} \ \text{jumpF-poly-root} \ \text{by fastforce} \)
next
  case False
  then have \( \text{asm:p\neq0 q\neq0 poly p x=0 by auto} \)
then have \( \text{poly } q \neq 0 \) using assms using coprime-poly-0 by blast
have \(?\text{thesis when sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0\)
proof
  have \((\text{poly } p \ \text{has-sgnx } \text{sgn } \ (\text{poly } q \ x)) \ (\text{at-right } x)\)
  by (metis False \((\text{poly } q \ x) \neq 0\) add.inverse-neutral has-sgnx-imp-sgnx less-not-sym

  \begin{align*}
    \text{neg-less-iff-less } \text{poly-has-sgnx-values}(2) \ \text{sgn-if sign-r-pos-sgnx-iff that} \\
    \text{trivial-limit-at-right-real zero-less-one}
  \end{align*}

then have \( \text{LIM } x \ \text{at-right } x. \ \text{poly } q \ x / \ \text{poly } p \ x :> \ \text{at-top} \)
apply \((\substack{\text{subst filterlim-divide-at-bot-at-top-iff} \ \text{of - poly } q \ x})\)
apply \((\text{auto simp add:poly q x\neq0})\)
by (metis asm(3) poly-tendsto(3))
then have \( \text{jumpF-polyR } q \ p \ x = 1/2 \)
unfolding \( \text{jumpF-polyR-def} \ \text{jumpF-def by auto} \)
then show \(?\text{thesis using that False by auto}\)
qed
moreover have \(?\text{thesis when } \neg (\text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0)\)
proof
  have \((\text{poly } p \ \text{has-sgnx } \neg \text{sgn } \ (\text{poly } q \ x)) \ (\text{at-right } x)\)
  proof
    have \( (0::\text{real}) < 1 \lor \neg (1::\text{real}) < 0 \land \text{sign-r-pos } p \ x \lor (\text{poly } p \ \text{has-sgnx } \neg \text{sgn } \ (\text{poly } q \ x)) \ (\text{at-right } x) \)
    by simp
  then show \(?\text{thesis}\)
  by (metis (no-types) False \((\text{poly } q \ x) \neq 0\) add.inverse-inverse has-sgnx-imp-sgnx

    \begin{align*}
      \text{neg-less-0-iff-less } \text{poly-has-sgnx-values}(2) \ \text{sgn-if sign-less sign-r-pos-sgnx-iff} \\
      \text{that trivial-limit-at-right-real}
    \end{align*}
  qed
then have \( \text{LIM } x \ \text{at-right } x. \ \text{poly } q \ x / \ \text{poly } p \ x :> \ \text{at-bot} \)
apply \((\substack{\text{subst filterlim-divide-at-bot-at-top-iff} \ \text{of - poly } q \ x})\)
apply \((\text{auto simp add:poly q x\neq0})\)
by (metis asm(3) poly-tendsto(3))
then have \( \text{jumpF-polyR } q \ p \ x = -1/2 \)
unfolding \( \text{jumpF-polyR-def} \ \text{jumpF-def by auto} \)
then show \(?\text{thesis using that False by auto}\)
qed
ultimately show \(?\text{thesis by auto}\)
qed

lemma \( \text{jumpF-polyL-coprime}: \)
assumes \( \text{coprime } p \ q \)
shows \( \text{jumpF-polyL } q \ p \ x = (\text{if } p \neq 0 \land q \neq 0 \land \text{poly } p \ x=0 \text{ then} \\
\text{if even (order } x \ p) \longleftrightarrow \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \text{ then } 1/2 \text{ else } 0) \)
proof \((\text{cases } p=0 \lor q=0 \lor \text{poly } p \ x\neq0)\)
case \text{True}
then show \(?\text{thesis using jumpF-poly-noroot by fastforce}\)
next

  case False
  then have \( \text{asm}: p \neq 0 \quad q \neq 0 \) poly \( p \ x = 0 \) by auto
  then have \( \text{poly } q \ x \neq 0 \) using assms using coprime-poly-0 by blast
  have ?thesis when even \( \text{order } x \ p \) \( \iff \) sign-r-pos \( p \ x \iff \) poly \( q \ x > 0 \)
  proof 
  consider \((lt)\) poly \( q \ x > 0 \) \(\mid\) \((gt)\) poly \( q \ x < 0 \) using \(\text{poly } q \ x \neq 0\) by linarith
  then have \( \text{sgnx} \ (\text{poly } p) \ (\text{at-left } x) = \text{sgn} \ (\text{poly } q \ x) \)
       apply cases
  subgoal using that \(\text{sign-r-pos} - \text{sgnx}\)-iff \(\text{poly} - \text{sgnx}\)-values[\(\text{OF } p \neq 0; af x\)]
      apply \(\text{subst} \ \text{poly} - \text{sgnx}-\text{left-right}\)[\(\text{OF } p \neq 0;]\])
      by auto
  subgoal using that \(\text{sign-r-pos} - \text{sgnx}\)-iff \(\text{poly} - \text{sgnx}\)-values[\(\text{OF } p \neq 0; af x\)]
      apply \(\text{subst} \ \text{poly} - \text{sgnx}-\text{left-right}\)[\(\text{OF } p \neq 0;]\])
      by auto
  done
  then have \( (\text{poly } p \ \text{has-sgxnx} \ - \text{sgn} \ (\text{poly } q \ x)) \ (\text{at-left } x) \)
       by \((\text{metis} \ \text{sgnx-able-poly}\ (2) \ \text{sgnx-able-sgxnx})\)
  then have \( \text{LIM} \ x \ \text{at-left } x. \ \text{poly } q \ x / \ \text{poly } p \ x : \to \ \text{at-top} \)
      apply \(\text{subst} \ \text{filterlim-divide-at-bot-at-top-iff}[\text{of} - \ \text{poly } q \ x]\)
      apply \(\text{auto} \ \text{simp add} : \ (\text{poly } q \ x \neq 0)\)
      by \((\text{metis} \ \text{asm}\ (3) \ \text{poly-tendsto}\ (2))\)
  then have \( \text{jumpF-polyL} \ q \ p \ x = 1/2 \)
      unfolding \(\text{jumpF-polyL-def} \ \text{jumpF-def}\) by auto
  then show ?thesis using that False by auto
  qed

moreover have ?thesis when \( \neg \ (\text{even } \text{order } x \ p \ \iff \ \text{sign-r-pos} \ p \ x \iff \ \text{poly} \ q \ x > 0) \)
  proof 
  consider \((lt)\) poly \( q \ x > 0 \) \(\mid\) \((gt)\) poly \( q \ x < 0 \) using \(\text{poly } q \ x \neq 0\) by linarith
  then have \( \text{sgnx} \ (\text{poly } p) \ (\text{at-left } x) = - \text{sgn} \ (\text{poly } q \ x) \)
       apply cases
  subgoal using that \(\text{sign-r-pos} - \text{sgnx}\)-iff \(\text{poly} - \text{sgnx}\)-values[\(\text{OF } p \neq 0; af x\)]
      apply \(\text{subst} \ \text{poly} - \text{sgnx}-\text{left-right}\)[\(\text{OF } p \neq 0;]\])
      by auto
  subgoal using that \(\text{sign-r-pos} - \text{sgnx}\)-iff \(\text{poly} - \text{sgnx}\)-values[\(\text{OF } p \neq 0; af x\)]
      apply \(\text{subst} \ \text{poly} - \text{sgnx}-\text{left-right}\)[\(\text{OF } p \neq 0;]\])
      by auto
  done
  then have \( (\text{poly } p \ \text{has-sgxnx} = - \text{sgn} \ (\text{poly } q \ x)) \ (\text{at-left } x) \)
       by \((\text{metis} \ \text{sgnx-able-poly}\ (2) \ \text{sgnx-able-sgxnx})\)
  then have \( \text{LIM} \ x \ \text{at-left } x. \ \text{poly } q \ x / \ \text{poly } p \ x : \to \ \text{at-bot} \)
      apply \(\text{subst} \ \text{filterlim-divide-at-bot-at-top-iff}[\text{of} - \ \text{poly } q \ x]\)
      apply \(\text{auto} \ \text{simp add} : \ (\text{poly } q \ x \neq 0)\)
      by \((\text{metis} \ \text{asm}\ (3) \ \text{poly-tendsto}\ (2))\)
  then have \( \text{jumpF-polyL} \ q \ p \ x = -1/2 \)
      unfolding \(\text{jumpF-polyL-def} \ \text{jumpF-def}\) by auto
  then show ?thesis using that False by auto
  qed
ultimately show \( ?\text{thesis} \) by \textit{auto} 

\textbf{qed}

\textbf{lemma} \textit{jumpF-times}: 
\textit{assumes} \( \text{tendsto} (f \longrightarrow c) \ \text{F and} \ c \neq 0 \ \text{F \# bot} \)
\textit{shows} \( \text{jumpF} (\lambda x. f \ x \ + \ g \ x) \ \text{F} = \text{sgn} \ c \ * \ \text{jumpF} \ g \ \text{F} \)
\textbf{proof} –
\begin{itemize}
  \item \( c > 0 \ \lor \ c < 0 \) \textit{using} \((c \neq 0)\) \textit{by auto}
  \item \textit{moreover have} \( ?\text{thesis} \) when \( c > 0 \)
  \begin{itemize}
    \item \textit{note} \( \text{filterlim-tendsto-pos-mult-at-top-iff} \ [\text{OF tendsto} \ (c > 0), \text{of g}] \)
    \item \textit{moreover note} \( \text{filterlim-tendsto-pos-mult-at-bot-iff} \ [\text{OF tendsto} \ (c > 0), \text{of g}] \)
    \item \textit{moreover have} \( \text{sgn} \ c = 1 \) \textit{using} \((c > 0)\) \textit{by auto}
  \end{itemize}
\end{itemize}
\textit{ultimately show} \( ?\text{thesis} \) \textit{unfolding} \textit{jumpF-def} \textit{by auto}
\textbf{qed}

\textit{moreover have} \( ?\text{thesis} \) when \( c < 0 \)
\textbf{proof} –
\begin{itemize}
  \item \textit{define} \textit{atbot} where \( \text{atbot} = \text{filterlim} \ g \ \text{at-bot} \ \text{F} \)
  \item \textit{define} \textit{attop} where \( \text{attop} = \text{filterlim} \ g \ \text{at-top} \ \text{F} \)
  \item \textit{have} \( \text{jumpF} (\lambda x. f \ x \ * \ g \ x) \ \text{F} = \text{(if} \ \text{atbot} \ \text{then} \ 1 / 2 \ \text{else if} \ \text{attop} \ \text{then} \ -1 / 2 \ \text{else} \ 0) \)
  \begin{itemize}
    \item \textit{proof} –
      \begin{itemize}
        \item \textit{note} \( \text{filterlim-tendsto-neg-mult-at-top-iff} \ [\text{OF tendsto} \ (c < 0), \text{of g}] \)
        \item \textit{moreover note} \( \text{filterlim-tendsto-neg-mult-at-bot-iff} \ [\text{OF tendsto} \ (c < 0), \text{of g}] \)
        \item \textit{ultimately show} \( ?\text{thesis} \) \textit{unfolding} \textit{jumpF-def atbot-def attop-def by auto}
      \end{itemize}
  \end{itemize}
\end{itemize}
\textit{also have} \( ... = - (\text{if} \ \text{attop} \ \text{then} \ 1 / 2 \ \text{else if} \ \text{atbot} \ \text{then} \ -1 / 2 \ \text{else} \ 0) \)
\textbf{proof} –
\begin{itemize}
  \item \textit{have} \( \text{False} \) \textit{when} \ \text{atbot} \ \text{attop} 
    \begin{itemize}
      \item \textit{using} \( \text{filterlim-at-top-at-bot} \ [\text{OF} - - (F \ # \ bot)] \) \textit{that} \textit{unfolding} \textit{atbot-def attop-def by auto}
      \item \textit{then show} \( ?\text{thesis} \) \textit{by fastforce}
    \end{itemize}
\end{itemize}
\textbf{qed}

\textit{also have} \( ... = \text{sgn} \ c \ * \ \text{jumpF} \ g \ \text{F} \)
\textbf{proof} \( (c < 0) \) \textit{unfolding} \textit{jumpF-def attop-def atbot-def by auto}
\textbf{finally show} \( ?\text{thesis} \).
\textbf{qed}

\textit{ultimately show} \( ?\text{thesis} \) \textit{by auto}
\textbf{qed}

\textbf{lemma} \textit{jumpF-polyR-inverse-add}:
\textit{assumes} \( \text{coprime} \ p \ q \)
\textit{shows} \( \text{jumpF-polyR} \ p \ q \ x + \text{jumpF-polyR} \ q \ p \ x = \text{jumpF-polyR} \ 1 \ (q \ * \ p) \ x \)
\textbf{proof} \( (\text{cases} \ p=0 \ \lor \ q=0) \)
\begin{itemize}
  \item \textit{case} \textit{True}
    \item \textit{then show} \( ?\text{thesis} \) \textit{by auto}
  \item \textit{next}
    \item \textit{case} \textit{False}
\end{itemize}
have \( \text{jumpF-add}: \)
\[ \text{jumpF-polyR} \; q \; p \; x = \text{jumpF-polyR} \; 1 \; (q*p) \; x \; \text{when} \; \text{poly} \; p \; x=0 \; \text{coprime} \; p \; q \; \text{for} \; p \; q \]

proof (cases p=0)

  case True
  then show ?thesis by auto
next

  case False
  have poly q x ≠ 0 using that coprime-poly-0 by blast
  then have \( q≠0 \) by auto
  moreover have \( \text{sign-r-pos} \; p \; x = (0 < \text{poly} \; q \; x) \iff \text{sign-r-pos} \; (q * p) \; x \)
  using \( \text{sign-r-pos-mult}[\text{OF} \; \langle \text{q≠0} \rangle \; \langle \text{p≠0} \rangle] \; \text{sign-r-pos-rec}[\text{OF} \; \langle \text{q≠0} \rangle \; \langle \text{poly} \; q \; x≠0 \rangle] \)
  by auto
  ultimately show ?thesis using \( \text{poly} \; p \; x=0 \)

  unfolding \( \text{jumpF-polyR-coprime}[\text{OF} \; \langle \text{coprime} \; p \; q \rangle] \; \langle \text{jumpF-polyR-coprime}[\text{OF} \; \langle \text{q≠0} \rangle \; \langle \text{poly} \; q \; x≠0 \rangle]\rangle \text{jumpF-polyR-coprime}[\text{OF} \; \langle \text{coprime} \; p \; q \rangle] \)
  by auto
qed

have False when \( \text{poly} \; p \; x=0 \; \text{poly} \; q \; x=0 \)
  using \( \text{coprime} \; p \; q \) that coprime-poly-0 by blast
moreover have ?thesis when \( \text{poly} \; p \; x=0 \; \text{poly} \; q \; x≠0 \)

proof —
  have \( \text{jumpF-polyR} \; p \; q \; x = 0 \) using \( \text{jumpF-poly-root}[\text{OF} \; \langle \text{poly} \; q \; x≠0 \rangle] \) by auto
  then show ?thesis using \( \text{jumpF-add}[\text{OF} \; \langle \text{poly} \; p \; x=0 \rangle \; \langle \text{coprime} \; p \; q \rangle] \) by auto
qed
moreover have ?thesis when \( \text{poly} \; p \; x≠0 \; \text{poly} \; q \; x=0 \)

proof —
  have \( \text{jumpF-polyR} \; p \; q \; x = 0 \) using \( \text{jumpF-poly-root}[\text{OF} \; \langle \text{poly} \; p \; x≠0 \rangle] \) by auto
  then show ?thesis using \( \text{jumpF-add}[\text{OF} \; \langle \text{poly} \; q \; x=0 \rangle \; \langle \text{of} \; p \rangle] \; \langle \text{coprime} \; p \; q \rangle \)
  by (simp add: ac-simps)
qed
moreover have ?thesis when \( \text{poly} \; p \; x≠0 \; \text{poly} \; q \; x≠0 \)
  by (simp add: \( \text{jumpF-poly-root}(2) \) that(1) that(2))
ultimately show ?thesis by auto
qed

lemma \( \text{jumpF-polyL-inverse-add}: \)
assumes \( \text{coprime} \; p \; q \)
sows \( \text{jumpF-polyL} \; p \; x + \text{jumpF-polyL} \; p \; q \; x = \text{jumpF-polyL} \; 1 \; (q*p) \; x \)

proof (cases p=0 ∨ q=0)

  case True
  then show ?thesis by auto
next

  case False
  have \( \text{jumpF-add}: \)
  \( \text{jumpF-polyL} \; q \; p \; x = \text{jumpF-polyL} \; 1 \; (q*p) \; x \) when \( \text{poly} \; p \; x=0 \; \text{coprime} \; p \; q \) for \( p \; q \)
proof (cases p=0)
  case True
  then show ?thesis by auto
next
  case False
  have poly q x≠0 using that coprime-poly-0 by blast
  then have q≠0 by auto
  moreover have sign-r-pos p x = (0 < poly q x) ←→ sign-r-pos (q * p) x
  using sign-r-pos-mult[of q≠0, p≠0] sign-r-pos-rec[of q≠0, poly q x≠0] by auto
  moreover have order x p = order x (q * p)
  by (metis (poly q x ≠ 0) add-cancel-right-left divisors-zero order-mult order-root)
  ultimately show ?thesis using ⟨poly p x=0⟩ by auto
qed

have False when poly p x=0 poly q x=0
  using ⟨coprime p q⟩ that coprime-poly-0 by blast
moreover have ?thesis when poly p x=0 poly q x≠0
proof
  have jumpF-polyL p q x = 0 using jumpF-poly-noroot[of poly q x≠0] by auto
  then show ?thesis using jumpF-add[of poly p x=0, coprime p q] by auto
qed
moreover have ?thesis when poly p x≠0 poly q x=0
proof
  have jumpF-polyL q p x = 0 using jumpF-poly-noroot[of poly p x≠0] by auto
  then show ?thesis using jumpF-add[of poly q x=0, of p, coprime p q]
    by (simp add: ac-simps)
  qed
moreover have ?thesis when poly p x≠0 poly q x≠0
  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) p x = 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 λ.. c, symmetric])
  by auto
lemma jumpF-polyR-smult-1:
jumpF-polyR (smult $c$ $q$) $p$ $x$ = sgn $c$ * jumpF-polyR $q$ $p$ $x$
proof (cases $c$=0)
  case True
  then show ?thesis by auto
next
case False
  then show ?thesis unfolding jumpF-polyR-def using False
    apply (subst jumpF-times[of $\lambda$.. $c$.symmetric])
    by auto
qed

lemma shows jumpF-polyR-mod: jumpF-polyR $q$ $p$ $x$ = jumpF-polyR ($q$ mod $p$) $p$ $x$
  and jumpF-polyL-mod: jumpF-polyL $q$ $p$ $x$ = jumpF-polyL ($q$ mod $p$) $p$ $x$
proof -
define $f$ where $f$ = ($\lambda$x. poly ($q$ div $p$) $x$)
define $g$ where $g$ = ($\lambda$x. poly ($q$ mod $p$) $x$ / poly $p$ $x$)
have jumpF-eq: jumpF ($\lambda$x. poly $q$ $x$ / poly $p$ $x$) (at $y$ within $S$) = jumpF $g$ (at $y$ within $S$)
  when $p$\#0 for $y$ $S$
proof -
  let $?F$ = at $y$ within $S$
  have $\forall$ $F$ $x$ in at $y$ within $S$. poly $p$ $x$ \#0
    using eventually-poly-nz-at-within[OF $p$\#0.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 $p$ $x$ \#0
    have poly $q$ $x$ = poly ($q$ div $p$ * $p$ + $q$ mod $p$) $x$
      by simp
    also have \ldots = $f$ $x$ * poly $p$ $x$ + poly ($q$ mod $p$) $x$
      by (simp only: poly-add poly-mult $f$-def $g$-def)
    moreover have poly ($q$ mod $p$) $x$ = $g$ $x$ * poly $p$ $x$
      using $P$ by (simp add: $g$-def)
    ultimately show poly $q$ $x$ / poly $p$ $x$ = $f$ $x$ + $g$ $x$
      using $P$ by simp
    qed
  then have jumpF ($\lambda$x. poly $q$ $x$ / poly $p$ $x$) $?F$ = jumpF ($\lambda$x. $f$ $x$ + $g$ $x$) $?F$
    by (intro jumpF-cong,auto)
  also have \ldots = jumpF $g$ $?F$
    proof -
    have ($f$ \longrightarrow $f$ $y$) (at $y$ within $S$)
      unfolding $f$-def by (intro tendsto-intros)
    from filterlim-tendsto-add-at-bot-iff[OF this.of $g$] filterlim-tendsto-add-at-top-iff[OF this.of $g$]

```
this, of j]
    show ?thesis unfolding jumpF-def by auto
qed
finally show ?thesis .
qed
show jumpF-polyR q p x = jumpF-polyR (q mod p) p x
  apply (cases p=0)
  subgoal by auto
  subgoal using jumpF-eq unfolding g-def jumpF-polyR-def by auto
done
show jumpF-polyL q p x = jumpF-polyL (q mod p) p x
  apply (cases p=0)
  subgoal by auto
  subgoal using jumpF-eq unfolding g-def jumpF-polyL-def by auto
done
qed

lemma jumpF-poly-top-0[simp]: jumpF-poly-top 0 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 \land q\neq 0 \land 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 \land q\neq 0 \land 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 jumpF (\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 sgn-pos-inf q * 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 jumpF (\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))

17
```
have \( P \) when \( p = 0 \lor q = 0 \)

unfolding \( P \)-def using that

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

moreover have \( P \) when \( p \neq 0 \land q \neq 0 \)

degree \( p > \) degree \( q \)

proof –

have \( \lim x \) at-top. \( \poly q x / \poly p x :> \) at-top

using poly-divide-filterlim-at-top[\( \operatorname{OF} \) that(1,2)] that(3) by auto

then show \( \text{thesis unfolding} \) \( P \)-def

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

qed

moreover have \( P \) when \( p \neq 0 \land q \neq 0 \)

degree \( p = \) degree \( q \)

proof –

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

using poly-divide-filterlim-at-top[\( \operatorname{OF} \) that(1,2)] using that by auto

then show \( \text{thesis unfolding} \) \( P \)-def

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

qed

ultimately have \( P \) using False by fastforce

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

unfolding \( \text{jumpF-def} \) \( P \)-def by auto

then show \( \text{thesis unfolding} \) \( \text{jumpF-poly-top-def} \) using False by presburger

qed

lemma \( \text{jumpF-poly-bot-code} \):

\( \text{jumpF-poly-bot} q p = (\text{if} p \neq 0 \land q \neq 0 \land \text{degree} q > \text{degree} p \) then

if \( \text{sgn.neg-inf} q \ast \text{sgn.neg-inf} p > 0 \) then \( 1/2 \) else \( -1/2 \) else \( 0 \)

proof (cases \( p \neq 0 \land q \neq 0 \land \text{degree} q > \text{degree} p \)

case \( \text{True} \)

have \( \text{thesis when} \) \( \text{sgn.neg-inf} q \ast \text{sgn.neg-inf} p > 0 \)

proof –

have \( \lim x \) at-bot. \( \poly q x / \poly p x :> \) at-bot

using poly-divide-filterlim-at-bot[\( \operatorname{of} p q \)] \( \text{True} \) that by auto

then have \( \text{jumpF} (\lambda x. \poly q x / \poly p x) \) at-bot = \( 1/2 \)

unfolding \( \text{jumpF-def} \) by auto

then show \( \text{thesis unfolding} \) \( \text{jumpF-poly-bot-def} \) using that \( \text{True} \) by auto

qed

moreover have \( \text{thesis when} \) \( \neg \text{sgn.neg-inf} q \ast \text{sgn.neg-inf} p > 0 \)

proof –

have \( \lim x \) at-bot. \( \poly q x / \poly p x :> \) at-bot

using poly-divide-filterlim-at-bot[\( \operatorname{of} p q \)] \( \text{True} \) that by auto

then have \( \text{jumpF} (\lambda x. \poly q x / \poly p x) \) at-bot = \( -1/2 \)

unfolding \( \text{jumpF-def} \) by auto

then show \( \text{thesis unfolding} \) \( \text{jumpF-poly-bot-def} \) using that \( \text{True} \) by auto

qed

ultimately show \( \text{thesis by auto} \)

next

\( \text{False} \)

define \( P \) where \( P = (\neg \text{lim x at-bot.} \poly q x / \poly p x :> \text{at-bot}) \)
∧ ¬ (LIM x at-bot. poly q x / poly p x ⊢ at-top))

have P when p=0 ∨ q=0

unfolding P-def using that

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

moreover have P when p≠0 q≠0 degree p > degree q

proof –

have LIM x at-bot. poly q x / poly p x ⊢ at 0

using poly-divide-filterlim-at-bot[OF (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≠0 q≠0 degree p = degree q

proof –

have ((λx. poly q x / poly p x) −−→ lead-coef q / lead-coef p) at-bot

using poly-divide-filterlim-at-bot[OF (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 (λx. 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

1.5 The extended Cauchy index on polynomials

definition cindex-polyE:: real ⇒ real ⇒ real poly ⇒ real poly ⇒ real where

cindex-polyE a b q p = jumpF-polyR q p a + cindex-poly a b q p - jumpF-polyL q p b

definition cindex-poly-ubd::real poly ⇒ real poly ⇒ int where

cindex-poly-ubd q p = (THE l. (∀ F r in at-top. cindexE (−r) r (λx. poly q x/poly p x) = of-int l))

lemma cindex-polyE-0[simp]: cindex-polyE a b 0 0 = 0 cindex-polyE a b q 0 = 0

unfolding cindex-polyE-def by auto

lemma cindex-polyE-mult-cancel:

fixes p q p':real poly

assumes p' ≠ 0

shows cindex-polyE a b (p' * q ) (p' * p) = cindex-polyE a b q p

unfolding cindex-polyE-def

using cindex-poly-mult[OF (p'≠0)] jumpF-polyL-mult-cancel[OF (p'≠0)]
jumpF-polyR-mult-cancel[OF (p'≠0)]

by simp

lemma cindexE-eq-cindex-polyE:

assumes a<b

shows cindexE a b (λx. poly q x/poly p x) = cindex-polyE a b q p
proof (cases p=0 ∨ q=0)
  case True
  then show ?thesis by (auto simp add: cindexE-constI)
next
  case False
  then have p̸=0 q̸=0 by auto
  define g where g=gcd p q
  define p′ q′ where p′=p div g and q′ = q div g
  have g̸=0 using False g-def by auto
  have pq-f: p = g ∗ p′ q = g ∗ q′ and coprime p′ q′
    unfolding g-def p′-def q′-def
  proof
    define f where f=(λx. poly q x/poly p x)
    define f′ where f′=(λx. poly q′ x/poly p′ x)
    have jumpF f (at-right x) = jumpF f′ (at-right x) for x
  proof (rule jumpF-cong)
    obtain ub where x < ub ∀ z. x < z ∧ z ≤ ub → poly g z ≠ 0
      using next-non-root-interval[OF ⟨g̸=0⟩,of x] by auto
    then show ∀ F x in at-right x. f x = f′ x
      unfolding eventually-at-right f-def f′-def pq-f
    apply (intro exI[where x=ub])
    by auto
  qed simp
  moreover have jumpF f (at-left x) = jumpF f′ (at-left x) for x
  proof (rule jumpF-cong)
    obtain lb where lb < x ∀ z. lb ≤ z ∧ z < x → poly g z ≠ 0
      using last-non-root-interval[OF ⟨g̸=0⟩,of x] by auto
    then show ∀ F x in at-left x. f x = f′ x
      unfolding eventually-at-left f-def f′-def pq-f
    apply (intro exI[where x=lb])
    by auto
  qed simp
  ultimately show ?thesis unfolding cindexE-def
  apply (fold f-def f′-def)
  by auto
  qed
  also have ... = jumpF f′ (at-right a) + real-at-int (cindex a b f′) − jumpF f′
  (at-left b)
  unfolding f′-def
  apply (rule cindex-eq-cindexE-divide)
  subgoal using ⟨a<b⟩.
  subgoal using False poly-roots-finite pq-f(1) pq-f(2) by fastforce
  subgoal using ⟨coprime p′ q′⟩ poly-gcd-iff by force
  subgoal by (auto intro:continuous-intros)
subgoal by (auto intro:continuous-intras)
done
also have ... = cindex-polyE a b q' p'
using cindex-eq-cindex-poly unfolding cindex-polyE-def jumpF-polyR-def jumpF-polyL-def
f'-def
  by auto
also have ... = cindex-polyE a b q p
using cindex-polyE-mult-cancel[OF (g\neq 0)] unfolding pq-f by auto
finally show \theta\text{thesis}.
qed

lemma cindex-polyE-cross:
  fixes p::real poly and a b::real
  assumes a < b
  shows cindex-polyE a b 1 p = \text{cross-alt 1 p a b} / 2
proof (induct degree p arbitrary:p rule:nat-less-induct)
case induct:
  have \theta\text{case when }p=0
    using that unfolding \text{cross-alt-def} by auto
  moreover have \theta\text{case when }p\neq 0 and noroot:{x. a < x \land x < b \land poly p x=0
  } = {}
  proof
    have cindex-polyE a b 1 p = jumpF-polyR 1 p a - jumpF-polyL 1 p b
    proof
      have cindex-poly a b 1 p = 0 unfolding cindex-poly-def
      apply (rule sum.neutral)
      using that by auto
      then show \theta\text{thesis unfolding cindex-polyE-def by auto}
    qed
  also have ... = cross-alt 1 p a b / 2
  proof
    define f where f = (\lambda x. 1 / poly p x)
    define ja where ja = jumpF f (at-right a)
    define jb where jb = jumpF f (at-left b)
    define right where right = (\lambda R. R ja (0::real) \lor \text{continuous (at-right a) f}
     \land R (poly p a) 0))
    define left where left = (\lambda R. R jb (0::real) \lor \text{continuous (at-left b) f} \land R
     (poly p b) 0))
    note ja-alt=jumpF-polyR-coprime[of p a,unfolded jumpF-polyR-def,simplified,folded f-def ja-def]
    note jb-alt=jumpF-polyL-coprime[of p b,unfolded jumpF-polyL-def,simplified,folded f-def jb-def]

    have [simp]:0 < ja \iff jumpF-polyR 1 p a = 1/2 0 > ja \iff jumpF-polyR
     1 p a = -1/2
    0 < jb \iff jumpF-polyL 1 p b = 1/2 0 > jb \iff jumpF-polyL 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 ≠ 0 ⇒ continuous (at-right a) f
  poly p b ≠ 0 ⇒ continuous (at-left b) f

unfolding f-def by (auto intro: continuous-intros)

have not-right-left: False when (right greater ∧ left less ∨ right less ∧ left greater)

proof

  have [simp]: f a > 0 ←→ poly p a > 0 f a < 0 ←→ poly p a < 0

  f b > 0 ←→ poly p b > 0 f b < 0 ←→ poly p b < 0

unfolding f-def by auto

have continuous-on {a<..<b} f

then have ∃x>a. x < b ∧ f x = 0

apply (elim jumpF-IVT[OF ⟨a<b⟩,of 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 ∧ poly p b > 0 ∨ poly p a < 0 ∧ poly p b < 0

using that jumpF-poly-noroot unfolding cross-alt-def by auto

moreover have False when poly p a > 0 ∧ poly p b < 0 ∨ poly p a < 0 ∧ 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 ∨ poly p a < 0 ∧ poly p b < 0

proof

  have ja > 0 ∨ ja < 0 using ja-alt ⟨p ≠ 0⟩ ⟨poly p a = 0⟩ by argo

moreover have False when ja > 0 ∧ poly p b < 0 ∨ ja < 0 ∧ poly p b > 0

apply (rule not-right-left)

unfolding right-def left-def using that by fastforce

moreover have ?thesis when ja > 0 ∧ poly p b > 0 ∨ ja < 0 ∧ poly p b < 0

using that jumpF-poly-noroot ⟨poly p a = 0⟩ unfolding cross-alt-def by auto

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 ∨ poly p a < 0

proof

  have jb > 0 ∨ jb < 0 using jb-alt ⟨p ≠ 0⟩ ⟨poly p b = 0⟩ by argo

moreover have False when jb > 0 ∧ poly p a < 0 ∨ jb < 0 ∧ poly p a > 0

apply (rule not-right-left)

unfolding right-def left-def using that by fastforce

moreover have ?thesis when jb > 0 ∧ poly p a > 0 ∨ jb < 0 ∧ poly p a < 0

using that jumpF-poly-noroot ⟨poly p b = 0⟩ unfolding cross-alt-def by auto

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

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

ultimately show ?thesis using that jumpF-poly-noroot unfolding cross-alt-def

by auto

qed
proof
have \( \text{ja} > 0 \lor \text{ja} < 0 \) using \( \text{ja-alt} \) \( \langle \text{pol} y \ p \ a = 0 \rangle \) by argo
moreover have \( \text{ja} > 0 \lor \text{ja} < 0 \) using \( \text{ja-alt} \) \( \langle \text{pol} y \ p \ a = 0 \rangle \) by argo
moreover have False when \( \text{ja} > 0 \land \text{ja} < 0 \lor \text{ja} < 0 \land \text{ja} > 0 \)
apply (rule not-right-left)
unfolding right-def left-def using that by fastforce
moreover have \( \text{thesis} \) when \( \text{ja} > 0 \land \text{ja} > 0 \lor \text{ja} < 0 \land \text{ja} > 0 \)
unfolding cross-alt-def by auto
ultimately show \( \text{thesis} \) by blast
qed
ultimately show \( \text{thesis} \) by argo
qed
finally show \( \text{thesis} \).
qed
moreover have \( \text{case when} \ p \neq 0 \) and no-empty:\( \{ x. \ a < x \land x < b \land \text{pol} y \ p \ x = 0 \} \neq \{ \} \)
proof
  define roots where \( \text{roots} = \{ x. \ a < x \land x < b \land \text{pol} y \ p \ x = 0 \} \)
  have finite roots unfolding roots-def using poly-roots-finite[\( \langle O\F \ p \neq 0 \rangle \)] by auto
  define max-r where \( \text{max-r} \equiv \text{Max} \ 	ext{roots} \)
  hence \( \text{poly} \ p \ \text{max-r}=0 \) and \( \text{a}<\text{max-r} \) and \( \text{max-r}<\text{b} \)
  using Max-in[\( \langle \text{finite roots} \rangle \)] no-empty unfolding roots-def by auto
  define max-rp where \( \text{max-rp} \equiv [\text{max-r},1:] \cdot \text{order} \ \text{max-r} \ p \)
  then obtain \( \text{p}' \) where \( \text{p}'-\text{def}:p=p' \text{max-rp} \) and \( \text{p}' \text{max-r}\equiv[\text{max-r},1:] \text{dvd} \ p' \)
  by (metis \( \langle \text{p}\neq 0 \rangle \) mult.commute order-decomp)
  hence \( \text{p}' \neq 0 \) and \( \text{max-rp}=0 \) and \( \text{max-r}<\text{p}' \text{max-r}<0 \)
  using \( \langle \text{p}\neq 0 \rangle \) by (auto simp add: dvd-iff-poly-eq-0)
  define max-r-sign where \( \text{max-r-sign}\equiv\text{if o}dd(\text{order} \ \text{max-r} \ p) \ \text{then} \ -1 \text{ else} \ 1::\text{int} \)
  define roots' where \( \text{roots'} = \{ x. \ a < x \land x < b \land \text{pol} y \ p' \ x = 0 \} \)
  have cindex-polyE a b 1 p = jumpF-polyR 1 p a + (\( \sum x \in \text{roots} \). jump-poly 1 p x) − jumpF-polyL 1 p b
  unfolding cindex-polyE-def cindex-poly-def roots-def by (simp,meson)
  also have \( \ldots = \text{max-r-sign} * \text{cindex-poly} a b 1 p' + \text{jump-poly} 1 p \ 	ext{max-r} \)
  + \( \text{max-r-sign} * \text{jumpF-polyR} 1 p' a - \text{jumpF-polyL} 1 p' b \)
  proof −
  have \( (\sum x \in \text{roots} \). jump-poly 1 p x) = \text{max-r-sign} * \text{cindex-poly} a b 1 p' + \text{jump-poly} 1 p \text{ max-r} \)
  proof −
  have \( (\sum x \in \text{roots}' \ . \ text{jump-poly} 1 p x) = (\sum x \in \text{roots}' \ . \ text{jump-poly} 1 p x) + \text{jump-poly} 1 p \text{ max-r} \)
  proof −
  have roots = insert max-r roots'
  unfolding roots-def roots'-def p'-def
  using \( \langle \text{poly} \ p \ \text{max-r}=0 \rangle \ \langle \text{a}<\text{max-r} \rangle \ \langle \text{max-r}<\text{b} \rangle \ \langle \text{p}\neq 0 \rangle \ \text{order-root} \)
apply (subst max-rp-def) 
by auto
moreover have finite roots'
  unfolding roots'-def using poly-roots-finite[OF \(p' \neq 0\)] by auto
moreover have max-r \(\notin\) roots'
  unfolding roots'-def using max-r-nz
  by auto
ultimately show \(\text{thesis}\) using sum.insert[of roots' max-r] by auto
qed
moreover have \((\sum x \in \text{roots'}, \: \text{jump-poly} 1 \: p \: x) = \text{max-r-sign} * \text{cindex-poly}\)
a b 1 \(p'
proof
have \((\sum x \in \text{roots'}, \: \text{jump-poly} 1 \: p \: x) = (\sum x \in \text{roots'}. \: \text{max-r-sign} * \: \text{jump-poly}\)
  I \(p' \: x)\nproof (rule sum.cong, rule refl)
fix \(x\) assume \(x \in \text{roots'}\)
hence \(x \neq \text{max-r}\) using max-r-nz unfolding roots'-def
  by auto
hence poly max-rp \(x \neq 0\) using poly-power-n-eq unfolding max-rp-def
by auto
hence order \(x \: \text{max-r} = 0\) by (metis order-root)
moreover have \(\text{jump-poly} 1 \: \text{max-r} \: x = 0\)
  using \(\text{poly} \: \text{max-rp} \: x \neq 0\) by (metis jump-poly-not-root)
moreover have \(x \in \text{roots}\)
  using \(x \in \text{roots'}\) unfolding roots-def roots'-def \(p'\)-def by auto
hence \(x < \text{max-r}\)
  using Max-ge[OF \(\text{finite roots'}, \: \text{of} \: x\) \(\: x \neq \text{max-r}\)] by (fold max-r-def,auto)
hence sign \(\text{poly} \: \text{max-rp} \: x\) = \text{max-r-sign}
  using \(\text{poly} \: \text{max-rp} \: x \neq 0\) 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 \(\text{jump-poly} 1 \: p \: x = \text{max-r-sign} * \: \text{jump-poly} 1 \: p' \: x\)
  using \(\text{jump-poly-1-mul}[\text{of} \: p' \: x \: \text{max-r}]\) unfolding \(p'\)-def
  by (simp add: \(\text{poly} \: \text{max-rp} \: x \neq 0\))
qed
also have \(\ldots = \text{max-r-sign} * (\sum x \in \text{roots'}. \: \text{jump-poly} 1 \: p' \: x)\)
  by (simp add: sum-distrib-left)
also have \(\ldots = \text{max-r-sign} * \: \text{cindex-poly} \: a \: b \: 1 \: p'\)
  unfolding cindex-poly-def roots'-def by meson
finally show \(\text{thesis}\)
qed
ultimately show \(\text{thesis}\) by simp
qed
moreover have \(\text{jumpF-polyR} 1 \: p \: a = \text{max-r-sign} * \: \text{jumpF-polyR} 1 \: p' \: a\)
proof
  define \(f\) where \(f = (\lambda x. \: 1 / \: \text{poly} \: \text{max-rp} \: x)\)
  define \(g\) where \(g = (\lambda x. \: 1 / \: \text{poly} \: p' \: x)\)
  have \(\text{jumpF-polyR} 1 \: p \: a = \text{jumpF} (\lambda x. \: f \: x * \: g \: x) (\text{at-right} a)\)
  unfolding jumpF-polyR-def f-def g-def \(p'\)-def
  by (auto simp add:field-simps)
also have ... = sgn (f a) * jumpF g (at-right a)
proof (rule jumpF-times)
  have [simp]: poly max-rp a ≠ 0
  unfolding max-rp-def using ⟨max-r>a⟩ by auto
  show (f −−−→ f a) (at-right a) f a ≠ 0
  unfolding f-def by (auto intro:tendsto-intros)
qed auto
also have ... = max-r-sign * jumpF-polyR 1 p' a
proof −
  have sgn (f a) = max-r-sign
  unfolding max-r-sign-def f-def max-rp-def using ⟨a<max-r⟩
  by (auto simp add:sgn-power)
  then show ?thesis unfolding jumpF-polyR-def g-def by auto
qed
finally show ?thesis.
qed
moreover have jumpF-polyL 1 p b = jumpF-polyL 1 p' b
proof −
  define f where f = (λx. 1 / poly max-rp x)
  define g where g = (λx. 1 / poly p' x)
  have jumpF-polyL 1 p b = jumpF (λx. f x * g x) (at-left b)
  unfolding jumpF-polyL-def f-def g-def p'-def
  by (auto simp add:field-simps)
also have ... = sgn (f b) * jumpF g (at-left b)
proof (rule jumpF-times)
  have [simp]: poly max-rp b ≠ 0
  unfolding max-rp-def using ⟨max-r<b⟩ by auto
  show (f −−−→ f b) (at-left b) f b ≠ 0
  unfolding f-def by (auto intro:tendsto-intros)
qed auto
also have ... = jumpF-polyL 1 p' b
proof −
  have sgn (f b) = 1
  unfolding max-r-sign-def f-def max-rp-def using ⟨b>max-r⟩
  by (auto simp add:sgn-power)
  then show ?thesis unfolding jumpF-polyL-def g-def by auto
qed
finally show ?thesis.
qed
ultimately show ?thesis by auto
qed
also have ... = max-r-sign * cindex-polyE a b 1 p' 1 p max-r
  + (max-r-sign − 1) * jumpF-polyL 1 p' b
  unfolding cindex-polyE-def roots'-def by (auto simp add:algebra-simps)
also have ... = max-r-sign * cross-alt 1 p' a b / 2 + jumpF-polyL 1 p' b
  + (max-r-sign − 1) * jumpF-polyL 1 p' b
proof −
  have degree max-rp>0 unfolding max-rp-def degree-linear-power
  using ⟨poly p max-r=0; order-root ⟨p≠0⟩ by blast
25
then have \( \text{degree } p' < \text{degree } p \) unfolding \( p' \)-def
using degree-mult-eq \( \langle p' \neq 0 \rangle \langle \text{max-rp} \neq 0 \rangle \) by auto
from induct[rule-format, \( \text{OF this} \)]
have \( \text{cindex-polyE a b 1 p} = \text{real-of-int } (\text{cross-alt } 1 \ p \ a \ b) \ / \ 2 \) by auto
then show \( \text{thesis by auto} \)
qed
also have ... = \( \text{real-of-int } (\text{cross-alt } 1 \ p \ a \ b) \ / \ 2 \)
proof –
  have \( \text{sjump-p:jump-poly } 1 \ p \ \text{max-r} = (\text{if odd } (\text{order } \text{max-r } p) \ \text{then sign } (\text{poly } p' \ \text{max-r}) \ \text{else } 0) \)
  proof –
    note \( \text{max-r-nz} \)
    moreover then have \( \text{poly max-rp max-r=0} \)
    using \( \langle \text{poly } p \ \text{max-r = 0} \rangle \langle p' \)-def by auto
    ultimately have \( \text{jump-poly } 1 \ p \ \text{max-r} = \text{sign } (\text{poly } p' \ \text{max-r}) \ \ast \text{jump-poly} 1 \ \text{max-rp max-r} \)
    unfolding \( p' \)-def using jump-poly-1-mult \[ \text{of } p' \ \text{max-r max-rp} \]
    by auto
    also have ... = (if odd \( \text{order max-r max-rp} \) \ \text{then sign } (\text{poly } p' \ \text{max-r}) \ \text{else } 0) \)
    proof –
      have \( \text{sign-r-pos max-rp max-r} \)
      unfolding max-rp-def using sign-r-pos-power by auto
      then show \( \text{thesis using } (\text{max-rp} \neq 0) \ \text{unfolding jump-poly-def by auto} \)
    qed
also have ... = (if odd \( \text{order max-r max-rp} \) \ \text{then sign } (\text{poly } p' \ \text{max-r}) \ \text{else } 0) \)
proof –
  have \( \text{order max-r p'=0} \) by \( \text{simp add: } \langle \text{poly } p' \ \text{max-r} \neq 0 \rangle \langle \text{order-0I} \rangle \)
  then have \( \text{order max-r max-rp = order max-r p} \)
  unfolding \( p' \)-def using \( \langle p' \neq 0 \rangle \langle \text{max-rp} \neq 0 \rangle \)
  apply \( \text{(subst order-mult)} \)
  by auto
  then show \( \text{thesis by auto} \)
  qed
finally show \( \text{thesis .} \)
qed
have \( \text{thesis when } \text{even } (\text{order max-r p}) \)
proof –
  have \( \text{sign } (\text{poly } p \ 0) = \text{sign } (\text{poly } p' \ 0) \ \text{when } \text{x=max-r} \ \text{for } x \)
  proof –
    have \( \text{sign } (\text{poly } p \ 0) = 1 \)
    unfolding max-rp-def using \( \text{even } (\text{order max-r p}) \) \ \text{that}
    apply \( \text{(simp add:sign-power )} \)
    by \( \text{(simp add: Sturm-Tarski.sign-def) } \)
  then show \( \text{thesis unfolding } p'\)-def by \( \text{(simp add:sign-times)} \)
  qed
from this[of a] this[of b] \( \langle a<\text{max-r} \rangle \langle \text{max-r}<b \rangle \)
have \( \text{cross-alt } 1 \ p' \ a \ b = \text{cross-alt } 1 \ p \ a \ b \)
unfolding cross-alt-def by auto
then show \( \emptyset \text{thesis using that unfolding max-r-sign-def sjump-p by auto} \)
moreover have \( \emptyset \text{thesis when odd (order max-r p)} \)
proof –
let \( \emptyset \text{thesis2 = sign (poly p' max-r) * 2 - cross-alt 1 p' a b - 4 * jumpF-polyL 1 p' b} \)
= \( \text{cross-alt 1 p a b} \)
have \( \emptyset \text{thesis2 when poly p' b = 0} \)
proof –
have \( \text{jumpF-polyL 1 p' b = 1/2} \lor \text{jumpF-polyL 1 p' b = -1/2} \)
using \( \text{jumpF-polyL-coprime[of p' 1 b,simplified]} \langle p'\neq0 \rangle \langle \text{poly p' b = 0} \rangle \) by auto
moreover have \( \emptyset \text{poly p' max-r > 0} \lor \text{poly p' max-r < 0} \)
using \( \text{max-r-nz} \) by auto
moreover have \( \emptyset \text{False when poly p' max-r > 0} \land \text{jumpF-polyL 1 p' b = -1/2} \)
\( \lor \) \( \text{poly p' max-r < 0} \land \text{jumpF-polyL 1 p' b = 1/2} \)
proof –
define \( \emptyset \text{f where f = (\lambda x. 1/poly p' x)} \)
have \( \emptyset \text{noroots: poly p' x \neq 0 when x\in {max-r<..<b}} \) for \( x \)
proof (rule ccontr)
assume \( \neg \text{poly p' x \neq 0} \)
then have \( \text{poly p x = 0} \) unfolding \( \emptyset \text{p'-def by auto} \)
then have \( \emptyset \text{x\in roots unfolding roots-def using that (a<max-r) by auto} \)
then have \( \emptyset \text{x\leq max-r using Max-ge[OF (finite roots)] unfolding max-r-def by auto} \)
moreover have \( \emptyset \text{x>max-r using that by auto} \)
ultimately show \( \emptyset \text{False by auto} \)
qed
have \( \emptyset \text{continuous-on \{max-r<..<b\} f} \)
unfolding \( \emptyset \text{f-def using noroots by (auto intro!:continuous-intros)} \)
moreover have \( \emptyset \text{continuous (at-right max-r) f} \)
unfolding \( \emptyset \text{f-def using max-r-nz} \)
by (auto intro!:continuous-intros)
moreover have \( \emptyset \text{f max-r > 0 \land jumpF f (at-left b) < 0} \)
\( \lor \) \( \text{f max-r < 0 \land jumpF f (at-left b) > 0} \)
using that unfolding \( \emptyset \text{f-def jumpF-polyL-def by auto} \)
ultimately have \( \exists x > max-r. x < b \land f x = 0 \)
apply (intro jumpF-IVT[OF \( \langle \text{max-r < b} \rangle \)])
by auto
then show \( \emptyset \text{False using noroots unfolding f-def by auto} \)
qed
moreover have \( \emptyset \text{thesis when poly p' max-r > 0 \land jumpF-polyL 1 p' b = 1/2} \)
\( \lor \) \( \text{poly p' max-r < 0 \land jumpF-polyL 1 p' b = -1/2} \)
proof –
have \( \text{poly max-rp a < 0 poly max-rp b > 0} \)
unfolding \( \emptyset \text{max-rp-def using (odd (order max-r p) (a<max-r) (max-r<b))} \)
by (simp-all add:zero-less-power-eq)
then have \( \text{cross-alt 1 p a b = - cross-alt 1 p' a b} \)
unfolding cross-alt-def p'-def using ⟨poly p' b=0⟩
apply (simp add:sign-times)
by (simp add: Sturm-Tarski.sign-def)
with that show ?thesis by auto
qed
ultimately show ?thesis by blast
qed
moreover have ?thesis2 when poly p' b≠0
proof –
have [simp]:jumpF-polyL 1 p' b = 0
using jumpF-polyL-coprime[of p' 1 b,simplified] (poly p' b≠0) by auto
have [simp]:poly max-rp a < 0 poly max-rp b>0
unfolding max-rp-def using ⟨odd (order max-r p)⟩ ⟨a<max-r)⟩ ⟨max-r<b)⟩
by (simp-all add:zero-less-power-eq)
have poly p' b>0 ∨ poly p' b<0
using ⟨poly p' b≠0⟩ by auto
moreover have poly p' max-r>0 ∨ poly p' max-r<0
using max-r-nz by auto
moreover have ?thesis when poly p' b>0 ∧ poly p' max-r>0
using that unfolding cross-alt-def p'-def
apply (simp add:sign-times)
by (simp add: Sturm-Tarski.sign-def)
moreover have ?thesis when poly p' b<0 ∧ poly p' max-r<0
using that unfolding cross-alt-def p'-def
apply (simp add:sign-times)
by (simp add: Sturm-Tarski.sign-def)
moreover have False when poly p' b>0 ∧ poly p' max-r<0 ∨ poly p'
b<0 ∧ poly p' max-r>0
proof –
have ∃x>max-r. x < b ∧ poly p' x = 0
apply (rule poly-IVT[OF ⟨max-r<b)⟩,of p'])
using that mult-less-0-iff by blast
then obtain x where max-r<x x<b poly p x=0 unfolding p'-def by auto
then have x∈roots using (a<max-r) unfolding roots-def by auto
then have x≤max-r unfolding max-r-def using Max-ge[OF (finite
roots)] by auto
then show False using (max-r<x) by auto
qed
ultimately show ?thesis by blast
qed
ultimately have ?thesis2 by auto
then show ?thesis unfolding max-r-def sjump-p using that by simp
qed
ultimately show ?thesis by auto
qed
finally show ?thesis .
qed
ultimately show ?case by fast
lemma cindex-polyE-inverse-add:
  fixes p q::real poly
  assumes cp: coprime p q
  shows cindex-polyE a b p q + cindex-polyE a b q p = cindex-polyE a b 1 (q*p)
  unfolding cindex-polyE-def
  using cindex-poly-inverse-add[OF cp,symmetric] jumpF-polyR-inverse-add[OF cp,symmetric]
  jumpF-polyL-inverse-add[OF cp,symmetric]
  by auto

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

lemma cindex-polyE-smult-1:
  fixes p q::real poly and c::real
  shows cindex-polyE a b q (smult c q p) = (sgn c) * cindex-polyE a b q p
  unfolding cindex-polyE-def jumpF-polyR-smult-1 jumpF-polyL-smult-1 cindex-poly-smult-1
  by (auto simp add:sgn-sign-eq[symmetric] algebra-simps)

lemma cindex-polyE-mod:
  fixes p q::real poly
  shows cindex-polyE a b q p = cindex-polyE a b (q 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 p q
  shows cindex-polyE a b q p = cross-alt q p a b/2 + cindex-polyE a b ((p mod q)) q
  proof -
    note cindex-polyE-inverse-add-cross[OF assms]
    moreover have cindex-polyE a b (− (p mod q)) q = − cindex-polyE a b p q
      using cindex-polyE-mod cindex-polyE-smult-1[of a b −1]
      by auto
    ultimately show ?thesis by (auto simp add:field-simps cross-alt-poly-commute)
lemma cindex-polyE-changes-alt-itu-mods:
assumes a< b coprime p q
shows cindex-polyE a b q p = changes-alt-itu-smods a b p q / 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-itu-smods-def changes-alt-poly-at-def)
next
case (Cons x xs)
then have p≠0 by auto
have ?case when q=0
using that by (simp add:changes-alt-itu-smods-def changes-alt-poly-at-def)
moreover have ?case when q≠0
proof –
define r where r≡− (p mod q)
obtain ps where ps:smods p q=q#p#ps smods q r=q#ps and xs=q#ps
unfolding r-def using ⟨q≠0⟩ ⟨p≠0⟩ ⟨x # xs = smods p q⟩
by (metis list.inject smods.simps)
from Cons.prems ⟨q ≠ 0⟩ have coprime q r
by (simp add: r-def ac-simps)
then have cindex-polyE a b r q = real-of-int (changes-alt-itu-smods a b q r) / 2
apply (rule-tac Cons.hyps(1))
using ps ⟨xs=q#ps⟩ by simp-all
moreover have changes-alt-itu-smods a b p q = cross-alt p q a b + changes-alt-itu-smods a b q r
using changes-alt-itu-smods-rec[OF ⟨a<b⟩ ⟨coprime p q⟩,folded r-def] .
moreover have cindex-polyE a b q p = real-of-int (cross-alt q p a b) / 2 + cindex-polyE a b r q
using cindex-polyE-rec[OF ⟨a<b⟩ ⟨coprime p q⟩,folded r-def] .
ultimately show ?case
by (auto simp add:field-simps cross-alt-poly-commute)
qed
ultimately show ?case by blast
qed

lemma cindex-poly-ubd-eventually:
shows ∀ F r in at-top. cindexE (−r) r (λx. poly q x/poly p x) = of-int (cindex-poly-ubd q p)
proof –
define f where f=(λx. poly q x/poly p x)
obtain R where R-def: R>0 proots p ⊆ {−R..<R}
if p≠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 ⊆ ball 0 r
by auto
have proots p ⊆ {−r..<r}
proof
fix x assume x ∈ proots p
then have x∈ball 0 r using r-ball by auto
then have abs x< r using mem-ball-0 by auto
then show x ∈ {−r..<r} using ⟨r>0⟩ by auto
qed
then show ?thesis using that[of r] False ⟨r>0⟩ by auto
qed
define l where l = (if p=0 then 0 else cindex-poly (−R) R q p)
define P where P = (λl. (∀F r in at-top. cindexE (−r) r f = of-int l))
have P l
proof (cases p=0 )
case True
then show ?thesis unfolding P-def f-def l-def using True
by (auto intro!: eventuallyI cindexE-constI)
next
case False
have P l unfolding P-def
proof (rule eventually-at-top-linorderI[of R])
fix r assume R ≤ r
then have cindexE (−r) r f = cindex-polyE (−r) r q p
unfolding f-def using R-def[of p≠0] by (auto intro: cindexE-eq-cindex-polyE)
also have ... = of-int (cindex-poly (−r) r q p)
proof –
have jumpF-polyR q p (−r) = 0
  apply (rule jumpF-poly-noroot)
  using ⟨R≤r⟩ R-def[of p≠0] by auto
moreover have jumpF-polyL q p r = 0
  apply (rule jumpF-poly-noroot)
  using ⟨R≤r⟩ R-def[of p≠0] by auto
ultimately show ?thesis unfolding cindex-polyE-def by auto
qed
also have ... = of-int (cindex-poly (−R) R q p)
proof –
define rs where rs={x. poly p x = 0 ∧ −r < x ∧ x < r}
define Rs where Rs={x. poly p x = 0 ∧ −R < x ∧ x < R}
have rs=Rs
using R-def[of p≠0] !:R≤r unfolding rs-def Rs-def by force
then show ?thesis
  unfolding cindex-poly-def by (fold rs-def Rs-def,auto)
also have also ... = of-int l unfolding l-def using False by auto
finally show cindexE (− r) r f = real-of-int l .
qed
then show ?thesis unfolding P-def by auto
qed
moreover have x = l when P x for x
proof —
  have ∀ r in at-top. cindexE (− r) r f = real-of-int x
  ∀ r in at-top. cindexE (− r) r f = real-of-int l
  using (P x) ⟨P l⟩ unfolding P-def by auto
  from eventually-conj[OF this]
  have ∀ r::real in at-top. real-of-int x = real-of-int l
    by (elim eventually-mono,auto)
  then have real-of-int x = real-of-int l by auto
  then show ?thesis by simp
qed
ultimately have P (THE x. P x) using theI[of P l] by blast
then show ?thesis unfolding P-def f-def cindex-poly-ubd-def by auto
qed

lemma cindex-poly-ubd-0:
  assumes p=0 ∨ q=0
  shows cindex-poly-ubd q p = 0
proof —
  have ∀ r in at-top. cindexE (− r) r (λx. poly q x/poly p x) = 0
    (rule eventuallyI)
    using assms by (auto intro:cindexE-constI)
  from eventually-conj[OF this cindex-poly-ubd-eventually[of q p]]
  have ∀ r::real in at-top. (cindex-poly-ubd q p) = (0::int)
    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≡smods p q
  have p∈set ps using ps-def ⟨p≠0⟩ by auto
  obtain lb where lb∩ p∈set ps. ∀ x. poly p x=0 → x>lb
    and lb-sgn:x≤lb. ∀ p∈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 \( ub: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \rightarrow x < ub \)
and \( \text{ub-sgn:} \forall x \geq ub. \forall p \in \text{set } ps. \ \text{sgn} (\text{poly } p \ x) = \text{sgn-pos-inf } p \)
and \( ub > 0 \)
using root-list-ub[\( \text{OF no-0-in-smods,of p,folded ps-def} \)]
by auto
define \( f \) where \( f = (\lambda t. \text{poly } q t/ \text{poly } p t) \)
define \( P \) where \( P = (\lambda l. (\forall r \ \text{in at-top. cindexE } (-r) \ r f = \text{of-int } l)) \)
have \( P \ (\text{changes-R-smods } p \ q) \) unfolding \( P \)-def
proof (rule eventually-at-top-linorderI[OF max \( |lb| \ |ub| + 1 \)])
fix \( r \) assume \( r-asn: r \geq \max \{|lb|, |ub| + 1\} \)
have \( \text{cindexE } (-r) \ r f = \text{cindex-polyE } (-r) \ r q p \)
unfolding \( f \)-def using \( r-asn \) by (auto intro: \( \text{cindexE-eq-cindex-polyE} \))
also have \( \ldots = \text{of-int } (\text{cindex-poly } (-r) \ r q p) \)
proof –
have \( \text{jumpF-polyR } q p (-r) = 0 \)
apply (rule \( \text{jumpF-poly-noroot} \))
using \( r-asn \) \( \text{lb}[\text{rule-format,OF } \langle \text{p \in set } ps, \text{of-r} \rangle] \) by linarith
moreover have \( \text{jumpF-polyL } q p r = 0 \)
apply (rule \( \text{jumpF-poly-noroot} \))
using \( r-asn \) \( \text{ub}[\text{rule-format,OF } \langle \text{p \in set } ps, \text{of-r} \rangle] \) by linarith
ultimately show \( ? \text{thesis unfolding } \text{cindex-polyE-def} \) by auto
qed
also have \( \ldots = \text{of-int } (\text{changes-itv-smods } (-r) \ r q p) \)
apply (rule \( \text{cindex-poly-changes-itv-modal}[\text{THEN arg-cong}] \))
using \( r-asn \) \( \text{lb}[\text{rule-format,OF } \langle \text{p \in set } ps, \text{of-r} \rangle] \) \( \text{ub}[\text{rule-format,OF } \langle \text{p \in set } ps, \text{of-r} \rangle] \) \( \text{by linarith} \)
also have \( \ldots = \text{of-int } (\text{changes-R-smods } p \ q) \)
proof –
have \( \text{map } (\text{sgn o } (\lambda p. \text{poly } p (-r))) \) \( ps = \text{map sgn-neg-inf } ps \)
and \( \text{map } (\text{sgn o } (\lambda p. \text{poly } p r)) \) \( ps = \text{map sgn-pos-inf } ps \)
using \( \text{lb-sgn} [\text{THEN spec,of-r,simplified}] \) \( \text{ub-sgn} [\text{THEN spec,of-r,simplified}] \)
\( \text{r-asn} \)
by auto
hence \( \text{changes-poly-at } ps (-r) = \text{changes-poly-neg-inf } ps \)
\( \wedge \text{changes-poly-at } ps r = \text{changes-poly-pos-inf } ps \)
unfolding \( \text{changes-poly-neg-inf-def} \) \( \text{changes-poly-at-def} \) \( \text{changes-poly-pos-inf-def} \)
by (subt (\( 1 \ 3 \)))\( \text{changes-map-sgn-eq,metis map-map} \)
thus \( ? \text{thesis unfolding } \text{changes-R-smods-def} \) \( \text{changes-itv-smods-def} \) \( ps-def \)
by metis
qed
finally show \( \text{cindexE } (-r) \ r f = \text{of-int } (\text{changes-R-smods } p \ q) \).
qed
moreover have \( x = \text{changes-R-smods } p \ q \) when \( P \ x \) for \( x \)
proof –
have \( \forall r \ \text{in at-top. cindexE } (-r) \ r f = \text{real-of-int } (\text{changes-R-smods } p \ q) \)
\( \forall r \ \text{in at-top. cindexE } (-r) \ r f = \text{real-of-int } x \)
using \( P \ (\text{changes-R-smods } p \ q) \) \( \langle P \ x \rangle \) unfolding \( P \)-def by auto
from eventually-conv[OF this]
have ∀F (r::real) in at-top. of-int x = of-int (changes-R-smods p q)
  by (elim eventually-mono auto)
thен 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 (λx. 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 mx mn where mx = Max {x. poly p x = 0} and mn = Min {x. poly p x=0}
define rr where rr = 1 + (max |mx| |mn|)
have rr:−rr < x ∧ x < rr when poly p x = 0 for x
proof −
  have finite {x. poly p x = 0} using ⟨p≠0⟩ poly-roots-finite by blast
  then have mn ≤ x x ≤ mx
    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=(λx. poly q x / poly p x)
have ∀F r in at-top. cindexE (− r) r f = cindexE-ubd f
proof (rule eventually-at-top-linorderI[of rr])
  fix r assume r≥rr
define R1 R2 where R1={x. jumpF f (at-right x) ≠ 0 ∧ −r ≤ x ∧ x < r} 
    and R2 = {x. jumpF f (at-right x) ≠ 0} 
define L1 L2 where L1={x. jumpF f (at-left x) ≠ 0 ∧ −r < x ∧ x ≤ r}
    and L2={x. jumpF f (at-left x) ≠ 0}
have R1=R2
proof −
  have jumpF f (at-right x) = 0 when ¬ (−r ≤ x ∧ x < r) for x
  proof −
    have jumpF f (at-right x) = jumpF-polyR q p x
      unfolding f-def jumpF-polyR-def by simp
    also have ... = 0
      apply (rule jumpF-poly-noroot)
    using that ⟨r≥rr⟩ by (auto dest:rr)
  finally show ?thesis .
qed
then show \( \text{thesis unfolding } R1\text{-def } R2\text{-def } \) by blast

qed

moreover have \( L1 = L2 \)

proof –

have \( \text{jumpF } f \text{ (at-left } x) = 0 \) when \( \neg (\neg r < x \land x \leq r) \) for \( x \)

proof –

have \( \text{jumpF } f \text{ (at-left } x) = \text{jumpF-polyL } q \ p \ x \)

unfolding \( f\text{-def } \text{jumpF-polyL-def } \) by simp

also have \( ... = 0 \)

apply (rule \( \text{jumpF-poly-noroot} \))

using that \( (r \geq rr) \) by (auto dest:rr)

finally show \( \text{thesis} \).

qed

then show \( \text{thesis unfolding } L1\text{-def } L2\text{-def } \) by blast

qed

ultimately show \( \text{cindexE } (- r) \ r \ f = \text{cindexE-ubd } f \)

unfolding \( \text{cindexE-def } \text{cindexE-ubd-def } \) apply (fold \( R1\text{-def } R2\text{-def } L1\text{-def } L2\text{-def } \))

by auto

qed

moreover have \( \forall \ F \ r \ in \ \text{at-top}. \ \text{cindexE } (- r) \ r \ f = \text{cindex-poly-ubd } q \ p \)

using \( \text{cindex-poly-ubd-eventually } \text{f-def } \) by auto

ultimately have \( \forall \ F \ r \ in \ \text{at-top}. \ \text{cindexE } (- r) \ r \ f = \text{cindexE-ubd } f \)

\( \land \ \text{cindexE } (- r) \ r \ f = \text{cindex-poly-ubd } q \ p \)

using \( \text{eventually-conj } \) by auto

then have \( \forall \ F \ (r::real) \ in \ \text{at-top}. \ \text{cindexE-ubd } f = \text{cindex-poly-ubd } q \ p \)

by (elim eventually-mono) auto

then show \( \text{thesis unfolding } f\text{-def } \) by auto

qed

end

2 More useful lemmas related polynomials

theory \( \text{More-Polynomials imports } \)

\( \text{Winding-Number-Eval. Missing-Algebraic } \)

\( \text{Winding-Number-Eval. Missing-Transcendental } \)

\( \text{Sturm-Tarski. PolyMisc } \)

\( \text{Budan-Fourier. BF-Misc } \)

begin

2.1 More about \( \text{order} \)

lemma order-normalize: \( \text{order x (normalize p) = order x p } \)

by (metis dvd-normalize-iff normalize-eq-0-iff order-1 order-2 order-unique-lemma)

lemma order-gcd: \( \text{order x (gcd p q) = min (order x p) (order x q) } \)

assumes \( p \neq 0 \ q \neq 0 \)

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

35
proof

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

obtain \(pp\) where \(pp : p = xx \ op * pp \ xx \ dvd pp\) using order-decomp [OF \(p\neq0\), of \(x\), folded xx-def op-def] by auto

obtain \(qq\) where \(qq : q = xx \ oq * qq \ xx \ dvd qq\) using order-decomp [OF \(q\neq0\), of \(x\), folded xx-def oq-def] by auto

define \(pq\) where \(pq = \gcd pp qq\)

have \(p\text{-unfold}: p = (pq * xx \ (\min op oq)) \ (* (pp \ div pp) \ xx \ (op - \min op oq))\) and [simp]: coprime \(xx (pp \ div pp)\) and \(pp\neq0\)

proof

have \(xx \ oq = xx \ (\min op oq) \ xx \ (op - \min op oq)\) by (simp flip: power-add)

moreover have \(pp = pq * (pp \ div pp)\)

unfolding \(pq\text{-def}\) by simp

ultimately show \(p = (pq * xx \ (\min op oq)) \ (* (pp \ div pp) \ xx \ (op - \min op oq))\)

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

show coprime \(xx (pp \ div pp)\)

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

using \(pp = pq * (pp \ div pp)\) pp(2) by auto

qed (use \(pp \neq 0\). in auto)

have \(q\text{-unfold}: q = (pq * xx \ (\min op oq)) \ (* (qq \ div qq) \ xx \ (oq - \min op oq))\) and [simp]: coprime \(xx (qq \ div qq)\)

proof

have \(xx \ oq = xx \ (\min op oq) \ xx \ (oq - \min op oq)\) by (simp flip: power-add)

moreover have \(qq = pq * (qq \ div qq)\)

unfolding \(pq\text{-def}\) by simp

ultimately show \(q = (pq * xx \ (\min op oq)) \ (* (qq \ div qq) \ xx \ (oq - \min op oq))\)

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

show coprime \(xx (qq \ div qq)\)

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

using \(qq = pq * (qq \ div qq)\) qq(2) by auto

qed

have gcd \(p q\)=normalize \((pq * xx \ (\min op oq))\)

proof

have coprime \((pp \ div pp * xx \ (op - \min op oq))\) \((qq \ div qq * xx \ (oq - \min op oq))\)

proof (cases \(op\geq q\))

case True

then have \(oq - \min op oq = 0\) by auto

moreover have coprime \((xx \ (op - \min op oq))\) \((qq \ div qq)\) by auto
moreover have coprime (pp div pq) (qq div pq)
apply (rule div-gcd-coprime[of pp qq folded pq-def])
using ⟨pp≠0⟩ by auto
ultimately show ?thesis by auto
next
case False
then have op - min op oq = 0 by auto
moreover have coprime (pp div pq) (xx ^ (oq - min op oq))
by (auto simp:coprime-commute)
moreover have coprime (pp div pq) (qq div pq)
apply (rule div-gcd-coprime[of pp qq folded pq-def])
using ⟨pp≠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 (gcd p q) = order x (xx ^ (min op oq))
apply simp
apply (subst order-mult)
using assms(1) p-unfold by auto
also have ... = order x (xx ^ (min op oq))
using pp(2) qq(2) unfolding pq-def xx-def
by (auto simp add: order-0I poly-eq-0-iff-dvd)
also have ... = min op oq
unfolding xx-def
by (rule order-power-n-n)
also have ... = min (order x p) (order x q)
unfolding op-def oq-def by simp
finally show ?thesis.
qed

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

lemma order-pderiv:
fixes p::('a::{idom,semiring-char-0}) poly
assumes p≠0 poly p x=0
shows order x p = Suc (order x (pderiv p)) using assms
proof -
define xx op where xx=[:- x, 1:] and op = order x p
have op ≠ 0 unfolding op-def using assms order-root by blast
obtain pp where pp:p = xx ^ op * pp ^ xx dvd pp
using order-decomp[OF ⟨op≠0,of xx 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 xx ^ (op - 1) dvd (pderiv p)

37
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 ¬ xx \circ op dvd (pderiv p)
proof
  assume xx \circ op dvd pderiv p
  then have xx \circ op dvd smult (of-nat op) (xx \circ (op \multimap -1) * pp)
    unfolding p-der by (simp add: dvd-add-left-iff)
  then have xx \circ op dvd (xx \circ (op \multimap -1)) * pp
    apply (elim dvd-monic [rotated])
    using (op\neq0) by (auto simp:lead-coeff-power xx-def)
  then have xx \circ op dvd (xx \circ (op \multimap -1))
    using (\neg xx dvd pp) by (simp add: :op \neq 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 \multimap -1 = order x (pderiv p)
  using order-uniform-lemma[of x op \multimap -1 pderiv p_folded xx-def] :op\neq0
  by auto
  then show \?thesis using (op\neq0); unfolding op-def by auto
qed

2.2 More about rsquarefree

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

lemma rsquarefree-times: assumes rsquarefree (p*q)
shows rsquarefree q using assms
proof (induct p rule:poly-root-induct-alt)
  case 0
  then show \?case by simp
next
  case (no-proots p)
  then have [simp]:p\neq0 q\neq0 \land a. order a p = 0
    using order-0I by auto
  have order a (p * q) = 0 \iff order a q = 0
    order a (p * q) = 1 \iff 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≠0 aa≠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-smult)  
from root(1)[OF this[unfolded pq-def]] show ?case .  
qed

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

lemma card-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 - λx. x∈s])  
  case 0  
  then have False by simp  
  then show ?case by simp  
next  
  case (no-roots p)  
  then show ?case by (metis all-not-in-conv card-empty proots-count-def proots-within-if 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, hide-lams) add.inverse-add.add.inverse-neutral minus-pCons proots-count-pCons-1-iff proots-count-uminus root.hyps(1))  
done  
also have ... = 1 + card (proots-within p s)  
proof –  
  have rsquarefree p using rsquarefree ([:a, −1:] * p))  
    by (elim rsquarefree-times)  
  from root(2)[OF this] show ?thesis by simp  
qed  
also have ... = card (proots-within ([:a, −1:] * p) s) unfolding proots-within-times  
proof (subst card-Un-disjoint)
have \([\text{simp}]:p \neq 0\) using \(\text{root.prems}\) by \text{auto}

show \(\text{finite} (\text{proots-within} [:a, -1:] s)\) \(\text{finite} (\text{proots-within} p s)\)
by \text{auto}

show \(1 + \text{card} (\text{proots-within} p s) = \text{card} (\text{proots-within} [:a, -1:] s) + \text{card} (\text{proots-within} p s)\)
using \(a \in s\)
apply \((\text{subst} \text{proots-within-pCons-1-iff})\)
by \text{simp}

have \(\text{poly} p a \neq 0\)
proof \((\text{rule ccontr})\)
assume \(\neg \text{poly} p a \neq 0\)
then have \(\text{order} a \leq 0\) by \((\text{simp add: order-root})\)
moreover have \(\text{order} a [:a,-1:] = 1\)
by \((\text{metis (no-types, hide-lams) add.inverse-inverse add.inverse-neutral minus-pCons order-power-n-n order-uminus power-one-right})\)
ultimately have \(\text{order} a [:a,-1:] \ast p > 1\)
apply \((\text{subst \text{order-mult}})\)
subgoal using \(\text{root.prems}\) by \text{auto}
subgoal by \text{auto}
done

then show \(\text{False}\) using \(\text{rsquarefree} [:a,-1:] \ast p)\)
unfolding \(\text{rsquarefree-def}\) using \((\text{gr-implies-not0 \text{less-not-refl2 by blast}})\)
qed
then show \(\text{proots-within} [:a, -1:] s \cap \text{proots-within} p s = {}\)
using \(\text{proots-within-pCons-1-iff}(2)\) by \text{auto}
qed

finally show \(?case .\)

qed

lemma \(\text{rsquarefree-gcd-pderiv}\):
fixes \(p::\{\text{factorial-ring-gcd, semiring-gcd-mul-normalize, \text{semiring-char-0}}\}\) \text{poly}
assumes \(p \neq 0\)
shows \(\text{rsquarefree} (p \text{ div} (\text{gcd} p (pderie p)))\)
proof \((\text{cases} \text{pderie} p = 0)\)

case True
have \(\text{poly} (\text{unit-factor} p) x \neq 0\) \(\text{for} x\)
using \(\text{unit-factor-is-unit}[\text{OF} \ p \neq 0]\)
by \((\text{meson asms dvd-trans order-decomp poly-eq-0-iff-dvd unit-factor-dvd})\)
then have \(\text{order} x (\text{unit-factor} p) = 0\) \(\text{for} x\)
using \(\text{order-0I by blast}\)
then show \(?\text{thesis using True}\ p \neq 0\) unfolding \(\text{rsquarefree-def}\) by \text{simp}\n
next

case False
define \(q\) where \(q = p \text{ div} (\text{gcd} p (\text{pderie} p))\)
have \(q \neq 0\) unfolding \(q\text{-def}\) by \((\text{simp add: asms dvd-div-eq-0-iff})\)

have \(\text{order-pq:order} x p = \text{order} x q + \text{min} (\text{order} x p) (\text{order} x (\text{pderie} p))\)
for \(x\)
proof
  
  have \( *: p = q * \gcd (p \operatorname{deriv} p) \)
  unfolding \( q\)-def by simp
  
  show \( ?\)thesis
  apply (subst \( *\))
  using \( q \neq 0 \) \( p \neq 0 \) \( p \operatorname{deriv} p \neq 0 \)
  by \( \text{simp add: order-mul order-gcd} \)
  qed

  have \( \text{order } x \ q = 0 \lor \text{order } x \ q = 1 \) for \( x \)
  proof
    (cases \( \text{poly } p \ x = 0 \))
    case True
    then obtain \( a \) where \( p = [\ a\ ] \)
    using \( \text{pderiv-iszero} \)
    by auto
    then show \( ?\)thesis
    by (auto simp add: \( \text{unit-factor-poly-def} \))
  
  next
    case False
    then have \( p \neq 0 \)
    by \( \text{simp add: order-0I} \)
    then have \( \text{order } x \ q = 0 \)
    using \( \text{order-pq[of } x \text{]} \)
    by simp
    then show \( ?\)thesis
    by simp
  qed

  then show \( ?\)thesis
  using \( q \neq 0 \)
  unfolding \( \text{rsquarefree-def} \ q\)-def
  by auto
  qed

lemma \( \text{poly-gcd-pderiv-iff} \):
  fixes \( p :: a \cdot \{ \text{semiring-char-0}, \text{factorial-ring-gcd}, \text{semiring-gcd-mult-normalize} \} \) \( \text{poly} \)
  shows \( \text{poly } (p \operatorname{div} (\gcd (p \operatorname{deriv} p))) \ x = 0 \leftrightarrow \text{poly } p \ x = 0 \)
  proof
    (cases \( \text{pderiv} p \))
    case True
    then obtain \( a \) where \( p = [\ a\ ] \)
    using \( \text{pderiv-iszero} \)
    by auto
    then show \( ?\)thesis
    by (auto simp add: \( \text{unit-factor-poly-def} \))
  
  next
    case False
    then have \( p \neq 0 \)
    using \( \text{pderiv-0} \)
    by blast
    define \( q \) where \( q = p \operatorname{div} (\gcd (p \operatorname{deriv} p)) \)
    have \( q \neq 0 \)
    unfolding \( q\)-def
    by \( \text{simp add: } q \neq 0 \text{ dvd-div-eq-0-iff} \)
    have \( \text{order-pq[of } x \text{]} = \text{order } x \ p \ + \ \text{min } (\text{order } x \ p) \) \( \text{order } x \ (p \operatorname{deriv} p) \) for \( x \)
    proof
      have \( *: p = q * \gcd (p \operatorname{deriv} p) \)
      unfolding \( q\)-def by simp
      
      show \( ?\)thesis
      apply (subst \( *\))
      using \( q \neq 0 \) \( p \neq 0 \) \( p \operatorname{deriv} p \neq 0 \)
      by \( \text{simp add: order-mul order-gcd} \)
      qed

  have \( \text{order } x \ q = 0 \leftrightarrow \text{order } x \ p = 0 \)
  proof
    (cases \( \text{poly } p \ x = 0 \))
    case True
    qed

  41
from order-pderiv[OF \langle p\neq 0 \rangle\ this]
have order x p = order x (pderiv p) + 1 by simp
then show \langle thesis \rangle 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 \langle thesis \rangle (order x p = 0) by simp
qed
then show \langle thesis \rangle
apply (fold q-def)
unfolding order-root using \langle p\neq 0 \rangle\langle q\neq 0 \rangle by auto
qed

2.3 Composition of a polynomial and a circular path

lemma poly-circlepath-tan-eq:
fixes z0::complex and r::real and p::complex poly
defines q1 \equiv fcompose p \[(z0+r)+i,z0-r;i,1;\] and q2 \equiv [i,1;] ^ degree p
assumes 0 \leq t < 1 t\neq 1/2
shows poly p (circlepath z0 r t) = poly q1 (tan (pi*t)) / poly q2 (tan (pi*t))
is \langle L = \langle R \rangle \rangle
proof –

have \langle L = poly p (z0 + r*exp (2 * of-real pi * i * t)) \rangle
unfolding circlepath by simp
also have ... = \langle R \rangle
proof –
define f where f = (poly p o (\lambda x::real. z0 + r * exp (i * x)))
have f-eq:f t = ((\lambda x::real. poly q1 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 (z0 + r * (cos t + i * sin t))
unfolding f-def exp-Euler by (auto simp add:cos-of-real sin-of-real)
also have ... = poly p ((\lambda x. ((z0-r)*x+(z0+r)*i) / (i+x)) (tan (t/2)))
proof –
define tt where tt=complex-of-real (tan (t / 2))
define rr where rr = complex-of-real r
have cos t = (1-tt*tt) / (1 + tt * tt)
sin t = 2*tt / (1 + tt * tt)
  unfolding sin-tan-half[of t/2,simplified] cos-tan-half[of t/2,OF that, simplified] tt-def
by (auto simp add:power2-eq-square)
moreover have 1 + tt * tt \neq 0 unfolding tt-def
apply (fold of-real-mult)
by (metis (no-types, hide-lams) 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 z0 + r * ( (cos t) + i * (sin t))
  = (z0+(1+tt*tt)+rr*(1-tt*tt)+i*rr*2*tt ) / (1 + tt * tt)
ultimately have z0 + r * ( (cos t) + i * (sin t))
  = (z0+(1+tt*tt)+rr*(1-tt*tt)+i*rr*2*tt ) / (1 + tt * tt)
ultimately have z0 + r * ( (cos t) + i * (sin t))
  = (z0+(1+tt*tt)+rr*(1-tt*tt)+i*rr*2*tt ) / (1 + tt * tt)
apply (fold r′-def, simp add: add-divide-distrib)
by (simp add: algebra-simps)
also have \ldots = \((z\bar{0} - r\bar{r}) + z\bar{0} * i + r\bar{r} * i) / (tt + i)
proof -
have tt + i \neq 0
  using (1 + tt * tt \neq 0)
  by (metis i-squared neg-eq-iff-add-eq-0 square-eq-iff)
then show ?thesis
  using (1 + tt * tt \neq 0)
  by (auto simp add: divide-simps algebra-simps)
qed
also have \ldots = ((z\bar{0} - r\bar{r}) * \bar{t}t + z\bar{0} * i + r\bar{r} * i) /
(t + i) .
then show ?thesis unfolding tt-def r′-def
  by (auto simp add: algebra-simps power2-eq-square)
qed
also have \ldots = (poly p o ((\lambda x. ((z\bar{0} - r) * \bar{t}x + (z\bar{0} + r) * i) / (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 q1 x / poly q2 x) o \lambda x. \tan
\((x/2))) ) 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
imaginary-unit.sel(1) plus-complex.sel(1) rcis-zero-arg rcis-zero-mod)
done
finally show ?thesis .
qed
have cos (pi * t) \neq 0 unfolding cos-zero-iff-int2
proof
  assume \exists i. pi * t = real-of-int i * pi + pi / 2
  then obtain i where pi * t = real-of-int i * pi + pi / 2 by auto
  then have pi * t = pi * (real-of-int i + 1 / 2) by (simp add: algebra-simps)
  then have t = real-of-int i + 1 / 2 by auto
  then show False using (0 \leq t) (t \leq 1) (t \neq 1/2) by auto
qed
from f-eq[of 2*pi*t,simplified,OF this]
show ?thesis
unfolding f-def comp-def by (auto simp add: algebra-simps)
qed
finally show ?thesis .
qed
end
3 Procedures to count the number of complex roots

theory Count-Complex-Roots imports
  Winding-Number-Eval, Winding-Number-Eval
  Extended-Sturm
  More-Polynomials
  Budan-Fourier.Sturm-Multiple-Roots
begin

3.1 Misc

corollary path-image-part-circlepath-subset:
  assumes r ≥ 0
  shows path-image(part-circlepath z r st tt) ⊆ sphere z r
proof (cases st ≤ tt)
  case True
  then show ?thesis
    by (auto simp: assms path-image-part-circlepath sphere-def dist-norm algebra-simps norm-mult)
next
  case False
  then have path-image(part-circlepath z r tt st) ⊆ sphere z r
    by (auto simp: assms path-image-part-circlepath sphere-def dist-norm algebra-simps norm-mult)
  moreover have path-image(part-circlepath z r 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 ∈ path-image(part-circlepath z r st tt) 0 ≤ r
  shows norm(w − z) = r
proof
  have w ∈ {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)

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 \((\text{path-image} (\text{circlepath} a r))\)
apply \((\text{rule simple-path-image-uncountable})\)
using simple-path-circlepath assms by simp
then have uncountable \((sphere a r)\)
using assms by simp
from uncountable-infinite[OF this] show \(\text{thesis}\) .
qed

lemma infinite-halfspace-Im-gt: infinite \(\{x. \operatorname{Im} x > b\}\)
apply \((\text{rule connected-uncountable}[\text{THEN uncountable-infinite, of -(b+1)* i (b+2)*i]})\)
by \((\text{auto intro!:convex-connected simp add: convex-halfspace-Im-gt})\)

lemma \((\text{in ring-1})\) Ints-minus2: \(- a \in \mathbb{Z} \Rightarrow a \in \mathbb{Z}\)
using Ints-minus[OF \(- a\)] by auto

lemma dvd-divide-Ints-iff:
\(b \operatorname{dvd} a \lor b=0 \iff \operatorname{of-int} a / \operatorname{of-int} b \in (\mathbb{Z} :: 'a :: \{\text{field, ring-char-0}\}\ set)\)
proof
assume \(\text{asm:} b \operatorname{dvd} a \lor b=0\)
let \(\text{thesis} = \operatorname{of-int} a / \operatorname{of-int} b \in (\mathbb{Z} :: 'a :: \{\text{field, ring-char-0}\}\ set)\)
have \(\text{thesis when} b \operatorname{dvd} a\)
proof –
obtain \(c\) where \(a=b * c\) using \(\langle \text{b dvd a} \rangle\) unfolding dvd-def by auto
then show \(\text{thesis by} \ (\text{auto simp add: field-simps})\)
qed
moreover have \(\text{thesis when} b=0\)
using \(\text{that by} \ \text{auto}\)
ultimately show \(\text{thesis using} \ \text{asm by} \ \text{auto}\)
next
assume \(\operatorname{of-int} a / \operatorname{of-int} b \in (\mathbb{Z} :: 'a :: \{\text{field, ring-char-0}\}\ set)\)
from Ints-cases[OF this] obtain \(c\) where \(*:(\operatorname{of-int}:::- \Rightarrow 'a) c= \operatorname{of-int} a / \operatorname{of-int} b\)
by metis
have \(b \operatorname{dvd} a\) when \(b\neq0\)
proof –
have \(\langle \text{of-int:::- \Rightarrow 'a} \rangle a = \operatorname{of-int} b \ast \operatorname{of-int} c\) using \(\text{that \ast by} \ \text{auto}\)
then have \(a = b \ast c\) using \(\text{of-int-eq-iff by} \ \text{fastforce}\)
then show \(\text{thesis unfolding} \ \text{dvd-def by} \ \text{auto}\)
qed
then show \(b \operatorname{dvd} a \lor b = 0\) by auto

45
lemma of-int-div-field:
assumes d dvd n
shows (of-int ::'a::field-char-0) (n div d) = of-int n / of-int d
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 ↔ a=1 ∨ b=0
proof
  assume a powr b = 1
  have b * ln a = 0
    using ⟨a powr b = 1⟩ ln-powr[of a b] assms by auto
  then have b = 0 ∨ ln a = 0 by auto
  then show a = 1 ∨ b = 0 using assms by auto
qed (insert assms, auto)

lemma tan-inj-pi:
  − (pi/2) < x ⇒ x < pi/2 ⇒ − (pi/2) < y ⇒ y < pi/2 ⇒ tan x = tan y
  y ⇒ x = y
by (metis arctan-tan)

lemma finite-ReZ-segments-poly-circlepath:
finite-ReZ-segments (poly p ◦ circlepath z0 r) 0
proof (cases ∀ t∈({0..1} − {1/2}). Re ((poly p ◦ circlepath z0 r) t) = 0)
case True
  have isCont (Re ◦ poly p ◦ circlepath z0 r) (1/2)
    by (auto intro!:continuous-intros simp:circlepath)
  moreover have (Re ◦ poly p ◦ circlepath z0 r)− 1/2 → 0
    proof –
      have ∀ x in at (1/2). (Re ◦ poly p ◦ circlepath z0 r) 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 Re ((poly p ◦ circlepath z0 r) (1/2)) = 0
  unfolding comp-def by (simp add: LIM-unique continuous-within)
then have ∀ t∈{0..1}. Re ((poly p ◦ circlepath z0 r) t) = 0
  using True by blast
then show ?thesis
  apply (rule_tac finite-ReZ-segments-congI[THEN finite-ReZ-segments-congE])
  by auto
next
  case False
define \( q_1 \) \( q_2 \) where \( q_1 = \text{compose } p \) \([\{(z0+r)\ast i, z0-r\} \mid i, 1] \) and \( q_2 = \{(i, 1) \ast \text{ degree } p \} \)

define \( q_1R \) \( q_1I \) where \( q_1R = \text{map-poly } \text{Re} q_1 \) and \( q_1I = \text{map-poly } \text{Im} q_1 \)
define \( q_2R \) \( q_2I \) where \( q_2R = \text{map-poly } \text{Re} q_2 \) and \( q_2I = \text{map-poly } \text{Im} q_2 \)
define \( qq \) where \( qq = q_1R \ast q_2R + q_1I \ast q_2I \)

have \( \text{poly-eq-Re} ((\text{poly } p \circ \text{circlepath } z0 r) t) = 0 \iff \text{poly } qq (\tan (pi \ast t)) = 0 \)

when \( 0 \leq t \leq 1 \) \( t \neq 1/2 \) for \( t \)

proof

- define \( tt \) where \( tt = \text{tan} (pi \ast t) \)

have \( \text{Re} ((\text{poly } p \circ \text{circlepath } z0 r) t) = 0 \iff \text{Re} (\text{poly } q_1 tt / \text{poly } q_2 tt) = 0 \)

unfolding \( \text{comp-def} \)

apply (\text{subst } \text{poly-circlepath-tan-eq}[\text{of } t p z0 r.] \text{folded } q_1-def \text{ } q_2-def \text{ tt-def})

using \( \text{that} \) by simp-all

also have ... \( \iff \text{poly } q_1R tt \ast \text{poly } q_2R tt + \text{poly } q_1I tt \ast \text{poly } q_2I tt = 0 \)

unfolding \( q_1I-def \) \( q_1R-def \) \( q_2R-def \) \( q_2I-def \)

by \( \text{simp add: Re-complex-div-eq-0 Re-poly-of-real Im-poly-of-real} \)

also have ... \( \iff \text{poly } qq tt = 0 \)

unfolding \( \text{qq-def} \) by simp

finally show \( ?\text{thesis} \) unfolding \( \text{tt-def} \).

qed

have \( \text{finite } \{ t. \text{Re} ((\text{poly } p \circ \text{circlepath } z0 r) t) = 0 \land 0 \leq t \land t \leq 1 \} \)

proof

- define \( P \) where \( P = (\lambda t. \text{Re} ((\text{poly } p \circ \text{circlepath } z0 r) t) = 0) \)

define \( A \) where \( A = \{(0, 1)\} :: \text{real set} \)

define \( S \) where \( S = \{t \in A - \{1, 1/2\} \cdot P t\} \)

have \( \text{finite } \{ t. \text{poly } qq (\tan (pi \ast t)) = 0 \land 0 \leq t \land t < 1 \land t \neq 1/2 \} \)

proof

- define \( A \) where \( A = \{t:: \text{real}. 0 \leq t \land t < 1 \land t \neq 1/2 \} \)

have \( \text{finite } ((\lambda t. \tan (pi \ast t)) \ast \{x. \text{poly } qq x = 0\} \cap A) \)

proof (rule finite-vimage-IntI)

have \( x = y \) when \( \tan (pi \ast x) \ast \text{tan} (pi \ast y) x \in A y \in A \) for \( x y \)

proof

- define \( x' \) where \( x' = (if x < 1/2 then x else x - 1) \)

 define \( y' \) where \( y' = (if y < 1/2 then y else y - 1) \)

 have \( x \ast pi = y \ast pi \)

 proof (rule tan-inj-pi)

 have \( \ast :: 1/2 < x' x' < 1/2 - 1/2 y' y' < 1/2 \)

 using \( \text{that}(2, 3) \) unfolding \( x' \ast y' \ast A \text{-def} \) by simp-all

 show \( (pi / 2) < x' \ast y' \ast pi \ast pi / 2 \ast (pi / 2) < y' \ast pi \ast pi \)

 by auto

 next

47
have $\tan(x' \pi) = \tan(x \pi)$

unfolding $x'$-def using tan-periodic-int[of ... - 1, simplified]

by (auto simp add: algebra-simps)

also have $\ldots = \tan(y \pi)$

using $\tan(pi \times x) = \tan(pi \times y)$ by (auto simp: algebra-simps)

also have $\ldots = \tan(y' \pi)$

unfolding $y'$-def using tan-periodic-int[of ... - 1, simplified]

by (auto simp add: algebra-simps)

finally show $\tan(x' \pi) = \tan(y' \pi)$.

qed

then have $x' = y'$ by auto

then show ?thesis

using that $(2,3)$ unfolding $x'$-def $y'$-def A-def by (auto split: if-splits)

qed

then show inj-on $(\lambda t. \tan(pi \times t)) A$

unfolding inj-on-def by blast

next

have $qq \neq 0$

proof (rule ccontr)

assume $\neg qq \neq 0$

then have $Re ((\text{poly } p \circ 	ext{circlepath z0 r}) t) = 0$ when $t \in \{0..1\} - \{1/2\}$

for $t$

apply (subst poly-eq)

using that by auto

then show False using False by blast

qed

then show finite $\{x. \text{poly } qq x = 0\}$ by (simp add: poly-roots-finite)

qed

then show ?thesis by (elim rev-finite-subset) (auto simp: A-def)

qed

moreover have $\{t. \text{poly } qq (\tan(pi \times t)) = 0 \land 0 \leq t \land t < 1 \land t \neq 1/2\} = S$

unfolding S-def P-def A-def using poly-eq by force

ultimately have finite $S$ by blast

then have finite $(S \cup \{t \in A. P \land 0 \leq t \land t < 1 \land t \neq 1/2\})$ by auto

moreover have $(S \cup \{t \in A. P \land 0 \leq t \land t < 1 \land t \neq 1/2\}) = \{t. P t \land 0 \leq t \land t \leq 1\}$

proof -

have $1 \in A \land 1/2 \in A$ unfolding A-def by auto

then have $(S \cup \{t \in A. P \land 0 \leq t \land t < 1 \land t \neq 1/2\}) = \{t. P t \land 0 \leq t \land t \leq 1\}$

unfolding S-def

apply auto

by (metis eq-divide-eq-numeral1 (1) zero-neq-numeral)+

also have $\ldots = \{t. P t \land 0 \leq t \land t \leq 1\}$

unfolding A-def by auto
finally show ?thesis.

qed

ultimately have finite \( \{ t . \ P t \land \theta \leq t \land t \leq 1 \} \) by auto

then show ?thesis unfolding \( P \)-def by simp

qed

then show ?thesis

apply (rule-tac finite-imp-finite-ReZ-segments)

by auto

qed

3.2 Some useful conformal/bij-betw properties

lemma bij-betw-plane-ball:bie-betw \( (\lambda x . (i - x) / (i + x)) \) \( \{ x . \ \text{Im} x > 0 \} \) (ball 0 1)

proof (rule bij-betw-imageI)

have neq: \( i + x \neq 0 \) when \( \text{Im} x > 0 \) for \( x \)

using that by (metis add-less-same-cancel2 add-uminus-conv-diff diff-0 diff-add-cancel

imaginary-unit.simps(2) not-one-less-zero uminus-complex.sel(2))

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

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

have cmod \((i - x) / (i + x)\) < 1 when \( 0 < \text{Im} x \) for \( x \)

proof

have cmod \((i - x)\) < cmod \((i + x)\)

unfolding norm-lt inner-complex-def using that by (auto simp add:algebra-simps)

then show ?thesis

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

qed

moreover have \( x \in (\lambda x . (i - x) / (i + x)) \)' \( \{ x . \ 0 < \text{Im} x \} \) when \( \text{cmod} x < 1 \) for \( x \)

proof (rule rev-image-eqI[of \( \text{i}*(1-\text{x})/(1+\text{x}) \)])

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

subgoal using that by (metis complex-mod-triangle-sub norm-one norm-zero not-le pth-7(1))

subgoal using that by (metis complex-i-not-zero div-mult-self4

mult-2

mult-zero-right nonzero-mult-cancel-left nonzero-mult-cancel-right

one-add-one zero-neq-numeral)

done

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

by (auto simp add:field-simps)

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

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

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

qed

ultimately show \( (\lambda x . (i - x) / (i + x)) \)' \( \{ x . \ 0 < \text{Im} x \} = \text{ball} 0 1 \)

by auto

qed
lemma bij-betw-axis-sphere: bij-betw \( \lambda x. (i-x)/(i+x)) \{ x. \text{Im } x = 0 \} \ (\text{sphere } 0 1 - \{-1\})
proof (rule bij-betw-imageI)
  have neq:i + x \neq 0 when \text{Im } x=0 for x
    using that
    by (metis add-diff-cancel-left' imaginary-unit.simps(2) minus-complex.simps(2)
      right-minus-eq zero-complex.simps(2) zero-neq-one)
  then show inj-on \( \lambda x. (i-x) / (i+x)) \{ x. \text{Im } x = 0 \}
    unfolding inj-on-def by (auto simp add:dividesims algebra-simps)
  have \( \text{cmod } ((i-x) / (i+x)) = 1 \ (i-x) / (i+x) \neq -1 \) when \text{Im } x = 0
  for x
  proof
    have \( \text{cmod } (i+x) = \text{cmod } (i-x) \)
      using that
    then show \( \text{cmod } ((i-x) / (i+x)) = 1 \)
      unfolding norm-divide using neq[OF that] by auto
  qed
  moreover have \( x \in \{ \lambda x. (i-x) / (i+x) \} \) \{ x. \text{Im } x = 0 \}
    when \text{cmod } x = 1 x\neq-1 for x
  proof (rule rev-image-eqI[of \( i*(1-x)/(1+x)\)])
    have 1 + x\neq 0 i * 2 + i * (x * 2) \neq 0
      subgoal using that(2) by algebra
    subgoal using that by (metis '1 + x \neq 0' complex-i-not-zero div-mult-self4
      mult-2
      mult-zero-right nonzero-mult-cancel-left nonzero-mult-cancel-right
      one-add-one zero-neq-numeral)
    done
    then show \( x = (i-i *(1-x) / (1+x)) / (i+i *(1-x) / (1+x)) \)
      by (auto simp add:fieldsims)
    show \( i * (1-x) / (1+x) \in \{ x. \text{Im } x = 0 \} \)
      apply (auto simp:algebra-simps \text{Im-complex-div-eq-0})
      using that(1) unfolding cmod-def by (auto simp:power2-eq-square)
    qed
    ultimately show \( \lambda x. (i-x) / (i+x) \) \{ x. \text{Im } x = 0 \} = \text{sphere } 0 1 - \{-1\} \)
      by force
  qed

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

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

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

lemma proots-sphere-axis-eq:
defines q1\(\equiv [i, -1:]\) and q2\(\equiv [i, 1:]\)
  assumes p\(\neq 0\)
  shows proots-count p (\text{sphere } 0 1 - \{ -1 \}) = proots-count (fcompose p q1 q2) \(\{x. 0 = \text{Im } x\}\)
  unfolding q1-def q2-def
proof (rule proots-fcompose-bij-eq[OF assms])
  show \(\forall x \in \{x. 0 = \text{Im } x\}. \text{poly } [i, 1:] x \neq 0\) by (simp add: Complex-eq-0
    plus-complex.code)
  show infinite (\text{UNIV}::\text{complex set}) by (simp add: infinite-UNIV-char-0)
qed (use bij-betw-axis-plane-ball in auto)
lemma proots-card-ball-plane-eq:
defines q1≡[i,−1:] and q2≡[i,1:]
assumes p≠0
shows card (proots-within p (ball 0 1)) = card (proots-within (fcompose p q1 q2) \{x. 0 < Im x\})
unfolding q1-def q2-def
proof (rule proots-card-fcompose-bij-eq [OF ⟨p≠0⟩])
show ∀x∈\{x. 0 < Im x\}. poly [i, 1:] x ≠ 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 q1≡[i,−1:] and q2≡[i,1:]
assumes p≠0
shows card (proots-within p (sphere 0 1 −{−1})) = card (proots-within (fcompose p q1 q2) \{x. 0 = Im x\})
unfolding q1-def q2-def
proof (rule proots-card-fcompose-bij-eq [OF ⟨p≠0⟩])
show ∀x∈\{x. 0 = Im x\}. poly [i, 1:] x ≠ 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 z0::complex and r::real
defines q≡[z0, of-real r::complex]
assumes p≠0 and r>0
shows proots-count p (ball z0 r) = proots-count (p ◦ p ◦ q) (ball 0 1)
proof –
  show ?thesis
    apply (rule proots-pcompose-bij-eq [OF ⟨p≠0⟩])
    subgoal unfolding q-def using bij-betw-ball-uball [OF ⟨r>0⟩, of z0] by (auto
     simp:algebra-simps)
    subgoal unfolding q-def using ⟨r>0⟩ by auto
    done
qed

lemma proots-card-uball-eq:
fixes z0::complex and r::real
defines q≡[z0, of-real r::complex]
assumes r>0
shows card (proots-within p (ball z0 r)) = card (proots-within (p ◦ p ◦ q) (ball 0 1))
proof –
  have ?thesis
    when p=0
    proof –
      have card (ball z0 r) = 0 card (ball (0::complex) 1) = 0
using infinite-ball[OF ⟨r>0,of z0⟩] infinite-ball[of 1 0::complex] by auto
then show ?thesis using that by auto
qed
moreover have ?thesis
when p≠0
apply (rule proots-card-pcompose-bij-eq[OF ⟨p≠0⟩])
subgoal unfolding q-def using bij-betw-ball-uball[OF ⟨r>0⟩,of z0] by (auto simp:algebra-simps)
subgoal unfolding q-def using (r>0) by auto
done
ultimately show ?thesis
by blast
qed

lemma proots-card-usphere-eq:
fixes z0::complex and r::real
defines q≡[z0,of-real r]
assumes r>0
shows card (proots-within p (sphere z0 r)) = card (proots-within (p ◦ p q) (sphere 0 1))
proof –
  have ?thesis
  when p=0
  proof –
    have card (sphere z0 r) = 0 card (sphere (0::complex) 1) = 0
    using infinite-sphere[OF ⟨r>0⟩,of z0] infinite-sphere[of 1 0::complex] by auto
    then show ?thesis using that by auto
  qed
moreover have ?thesis
when p≠0
apply (rule proots-card-pcompose-bij-eq[OF ⟨p≠0⟩])
subgoal unfolding q-def using bij-betw-sphere-usphere[OF ⟨r>0⟩,of z0]
by (auto simp:algebra-simps)
subgoal unfolding q-def using (r>0) by auto
done
ultimately show card (proots-within p (sphere z0 r)) = card (proots-within (p ◦ p q) (sphere 0 1))
by blast
qed

3.3 Combining two real polynomials into a complex one

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

lemma cpoly-of-vq-0-iff[iff]:
cpoly-of pR pI = 0 ⇔ pR = 0 ∧ pI = 0
proof –

53
have \( pR = 0 \land pI = 0 \) when \( \text{cpoly-of} \) \( pR pI = 0 \)
proof
  have complex-of-real \( (\text{coeff} pR \ n) + i \ast \text{complex-of-real} \ (\text{coeff} pI \ n) = 0 \) for \( n \)
    using that unfolding poly-eq-iff cpoly-of-def by (auto simp:coeff-map-poly)
  then have \( \text{coeff} pR \ n = 0 \land \text{coeff} pI \ n = 0 \) 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 \(?thesis\) unfolding poly-eq-iff by (auto simp:cpoly-of-def)
qed

lemma cpoly-of-decompose:
  \( p = \text{cpoly-of} \ (\text{map-poly} \ Re \ p) \ (\text{map-poly} \ Im \ p) \)
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:
  \( \text{cpoly-of} \ (pR\ast g) \ (pI\ast g) = \text{cpoly-of} \ pR \ pI \ast (\text{map-poly} \ of-real \ g) \)
unfolding cpoly-of-def by (simp add: distrib-right)

lemma poly-cpoly-of-real:
  \( \text{poly} \ (\text{cpoly-of} \ pR \ pI) \ (\text{of-real} \ x) = \text{Complex} \ (\text{poly} \ pR \ x) \ (\text{poly} \ pI \ x) \)
unfolding cpoly-of-def by (simp add: Complex-eq of-real-map-poly)

lemma poly-cpoly-of-real-iff:
  shows \( \text{poly} \ (\text{cpoly-of} \ pR \ pI) \ (\text{of-real} \ t) \neq 0 \longleftrightarrow \text{poly} \ pR \ t = 0 \land \text{poly} \ pI \ t = 0 \)
unfolding poly-cpoly-of-real using Complex-eq-0 by blast

lemma order-cpoly-gcd-eq:
  assumes \( pR \neq 0 \lor pI \neq 0 \)
  shows order \( t \) \( (\text{cpoly-of} \ pR \ pI) = \text{order} \ t \ (\text{gcd} \ pR \ pI) \)
proof
  define \( g \) where \( g = \text{gcd} \ pR \ pI \)
  have [simp]:\( g \neq 0 \) unfolding g-def using assms by auto
  obtain \( pr \ pi \) where \( pri: \ \text{pr} = pr \ast g \ \text{pi} = pi \ast g \ \text{coprime} \ \text{pr} \ \text{pi} \)
    unfolding g-def using assms(1) gcd-coprime-exists \( g \neq 0 \) g-def by blast
  then have \( pr \neq 0 \lor pi \neq 0 \) using assms mult-zero-left by blast
  have order \( t \) \( (\text{cpoly-of} \ pR \ pI) = \text{order} \ t \ (\text{cpoly-of} \ pr \ pi \ast (\text{map-poly} \ of-real \ g)) \)
      unfolding pri cpoly-of-dist-right by simp
  also have \( \ldots = \text{order} \ t \ (\text{cpoly-of} \ pr \ pi) + \text{order} \ t \ g \)
      apply (subst order-mult)
      using \( \ldots \neq 0 \lor \ pi \neq 0 \) by (auto simp:map-poly-order-of-real)
  also have \( \ldots = \text{order} \ t \ g \)
  proof
    have \( \text{poly} \ (\text{cpoly-of} \ pr \ pi) \ t \neq 0 \) 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

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

— 1 dimensional hyperplane
definition unbounded-line :: 'a::{real,vector} ⇒ 'a ⇒ 'a set where
  unbounded-line a b = \{x. ∃ u::real. x = (1 - u) *R a + u *R b\}
definition proots-line-card :: complex poly ⇒ complex ⇒ complex ⇒ nat where
  proots-line-card p st tt = card (proots-within p (open-segment st tt))
definition proots-unbounded-line-card :: complex poly ⇒ complex ⇒ complex ⇒ nat where
  proots-unbounded-line-card p st tt = card (proots-within p (unbounded-line st tt))
definition proots-unbounded-line :: complex poly ⇒ complex ⇒ complex ⇒ nat where
  proots-unbounded-line p st tt = proots-count p (unbounded-line st tt)

lemma card-proots-open-segments:
  assumes poly p st ≠ 0 poly p tt ≠ 0
  shows card (proots-within p (open-segment st tt)) =
    (let pc = pcompose p [:st, tt-st];
     pR = map-poly Re pc;
     pI = map-poly Im pc;
     g = gcd pR pI
     in changes-itv-smods 0 1 g (pderiv g)) (is ?L = ?R)
proof –
define pc pR pI g where
  pc = pcompose p [:st, tt-st] and
  pR = map-poly Re pc and
  pI = map-poly Im pc and
  g = gcd pR pI
have poly-iff:poly g t=0 ""→"" poly pc t =0 for t
proof –
have poly g t = 0 ""→"" poly pR t =0 ∧ poly pI t =0
  unfolding g-def using poly-gcd-iff by auto
also have ... ""→"" poly pc t =0
proof –
have cpoly-of pR pI = pc
  unfolding pc-def pR-def pI-def using cpoly-of-decompose by auto
then show ?thesis using poly-cpoly-of-real-iff by blast
qed
finally show ?thesis by auto

55
have \( ?R = \text{changes-itv-smods 0 1 g} \) (pderiv g)
unfolding pc-def g-def pI-def pR-def by (auto simp add:Let-def)
also have ... = card \( \{t. \text{poly g t} = 0 \land 0 < t \land t < 1\} \)
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 0 1 g] by auto
qed
also have ... = card \( \{t::\text{real}. \text{poly pc t} = 0 \land 0 < t \land t < 1\} \)
unfolding poly-iff by simp
also have ... = ?L
proof (cases st=tt)
  case True
  then show ?thesis unfolding pc-def poly-pcompose using \( \langle \text{poly p tt} \neq 0 \rangle \)
    by (auto simp add:algebra-simps)
next
  case False
  define ff where ff = \( \lambda t::\text{real}. st + t*(tt-st) \) (\lambda t::\text{real}. complex-of-real t)
  define ll where ll = \( \{t. \text{poly pc (complex-of-real t)} = 0 \land 0 < t \land t < 1\} \)
  have ff ' ll = proots-within p (open-segment st tt)
    proof (rule equalityI)
      show ff ' ll \( \subseteq \) 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 tt) \( \subseteq \) ff ' ll
        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 st + u *R (1 - st)
            by (auto simp add:algebra-simps scaleR-conv-of-real)
          moreover have x = ff u
            using ff-def using u by (auto simp add:algebra-simps scaleR-conv-of-real)
          ultimately show x \( \in \) ff ' ll by (rule rev-image-eqI[of u])
        qed
      qed
    qed
  moreover have inj-on ff ll
    unfolding ff-def using False inj-on-def by fastforce
  ultimately show ?thesis unfolding ll-def using card-image[of ff] by fastforce
qed
finally show \textit{thesis} by simp
qed

lemma \textit{unbounded-line-closed-segment}: closed-segment \( a \ b \subseteq \text{unbounded-line} \ a \ b \)
unfolding \textit{unbounded-line-def} \textit{closed-segment-def} by auto

lemma \textit{card-proots-unbounded-line}:
assumes \( \text{st} \neq \text{tt} \)
shows \( \text{card} \ (\text{proots-within} \ p \ (\text{unbounded-line} \ \text{st} \ \text{tt})) = \)
\( \text{(let} \ pc = \text{pcompose} \ p \ [\text{st}, \ tt - \text{st}], \)
\( \text{pR} = \text{map-poly} \ Re \ pc; \)
\( \text{pI} = \text{map-poly} \ Im \ pc; \)
\( g = \text{gcd} \ pR \ pI \)
\( \text{in} \ \text{nat} \ (\text{changes-R-smods} \ g \ (\text{pderiv} \ g))) \ (\text{is} \ ?L = ?R) \)
proof –
define \( pc \ pR \ pI \ g \) where
\( pc = \text{pcompose} \ p \ [\text{st}, \ tt - \text{st}]; \)
\( pR = \text{map-poly} \ Re \ pc \) and
\( pI = \text{map-poly} \ Im \ pc \) and
\( g = \text{gcd} \ pR \ pI \)
have \text{poly-iff}:\text{poly} \ g \ t = 0 \longleftrightarrow \text{poly} \ pc \ t = 0 \text{ for } t
proof –
have \text{poly} \ g \ t = 0 \longleftrightarrow \text{poly} \ pR \ t = 0 \land \text{poly} \ pI \ t = 0
unfolding \textit{g-def} using \textit{poly-gcd-iff} by auto
also have ... \longleftrightarrow \text{poly} \ pc \ t = 0
proof –
have \text{cpoly-of} \ pR \ pI = pc
unfolding \textit{pc-def} \textit{pR-def} \textit{pI-def} using \textit{cpoly-of-decompose} by auto
then show \textit{thesis} using \textit{poly-cpoly-of-real-iff} by blast
qed
finally show \textit{thesis} by auto
qed

have ?R = \text{nat} \ (\text{changes-R-smods} \ g \ (\text{pderiv} \ g))
unfolding \textit{pc-def} \textit{g-def} \textit{pI-def} \textit{pR-def} by simp \textit{add:Let-def}
also have ... = \text{card} \ \{t. \ \text{poly} \ g \ t = 0\}
using \textit{sturm-R[of} \ g] \text{by simp}
also have ... = \text{card} \ \{t::real. \ \text{poly} \ pc \ t = 0\}
unfolding \textit{poly-iff} by simp
also have ... = ?L
proof (cases \textit{st=tt})
case True
then show \textit{thesis} unfolding \textit{pc-def} \textit{poly-pcompose} \textit{unbounded-line-def} using \textit{assms}
by (auto simp \textit{add:proots-within-def})
next
case False
define \( ff \) where \( ff = (\lambda t::real. \ \text{st} + t*(tt-st)) \)
define \( ll \) where \( ll = \{t. \ \text{poly} \ pc \ (\text{complex-of-real} \ t) = 0\} \)

57
have \( \mathcal{F} \cdot \mathcal{L} = \text{proots-within } p \) (unbounded-line st tt)

proof (rule equalityI)
show \( \mathcal{F} \cdot \mathcal{L} \subseteq \text{proots-within } p \) (unbounded-line st tt)
unfolding \( \mathcal{L} \)-def \( \mathcal{F} \)-def \( p \)-def \( \text{poly-compose} \)
by (auto simp add:unbounded-line-def False scaleR-conv-of-real algebra-simps)

next
show \( \text{proots-within } p \) (unbounded-line st tt) \( \subseteq \mathcal{F} \cdot \mathcal{L} \)
proof clarify
fix \( x \)
assume \( \text{asm} : x \in \text{proots-within } p \) (unbounded-line st tt)
then obtain \( u \)
where \( u = (1 - u) * R \text{ st } + u * R \text{ tt} \)
by (auto simp add:unbounded-line-def)
then have \( \text{poly } p \) \((1 - u) * R \text{ st } + u * R \text{ tt}\) = 0 using \( \text{asm} \) by simp
then have \( u \in \mathcal{L} \)
unfolding \( \mathcal{L} \)-def \( p \)-def \( \text{poly-compose} \)
by (simp add:scaleR-conv-of-real algebra-simps unbounded-line-def)
moreover have \( x = \mathcal{F} u \)
unfolding \( \mathcal{F} \)-def using \( \text{u} \) by (auto simp add:algebra-simps scaleR-conv-of-real)
ultimately show \( x \in \mathcal{F} \cdot \mathcal{L} \) by (rule rev-image-eqI[of \( u \)])
qed

qed

moreover have \( \text{inj-on } \mathcal{F} \cdot \mathcal{L} \)
unfolding \( \mathcal{F} \)-def using \( \text{False inj-on-def} \) by fastforce
ultimately show \(?\text{thesis}\) unfolding \( \mathcal{L} \)-def
using card-image[of \( \mathcal{F} \)] by metis
qed

finally show \(?\text{thesis}\) by simp
qed

lemma proots-unbounded-line:
assumes \( \text{st} \neq \text{tt} \) \( p \neq 0 \)
shows \((\text{proots-count } p \) (unbounded-line st tt)) =
\((\text{let } pc = \text{pcompose } p \) [:st, tt - st:];
\( pR = \text{map-poly Re } pc; \)
\( pI = \text{map-poly Im } pc; \)
\( g = \text{gcd } pR \) \( pI \)
in \( \text{nat } \) \( \text{changes-R-smods-ext } g \) \((\text{pderiv } g))\) \((\text{is } ?L = ?R)\)

proof –
define \( pc \) \( pR \) \( pI \) \( g \) where
\( pc = \text{pcompose } p \) [:st, tt - st:]; and
\( pR = \text{map-poly Re } pc \) and
\( pI = \text{map-poly Im } pc \) and
\( g = \text{gcd } pR \) \( pI \)
have \([\text{simp}]\): \( g \neq 0 \) \( pc \neq 0 \)
proof –
show \( pc \neq 0 \) using \( \text{assms}(1) \) \( \text{assms}(2) \) \( p \)-def \( \text{pcompose-eq-0} \) by fastforce
then have \( pR \neq 0 \) \( \lor pI \neq 0 \) unfolding \( p \)-def \( p \)-def by (metis \( \text{cpoly-of-decompose} \) \( \text{map-poly-0} \))
then show \( g \neq 0 \) unfolding \( g \)-def by simp
qed

58
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≠0). 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≠0⟩] by auto
also have ... = proots-count (map-poly complex-of-real g) (af-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 ⟨g≠0⟩ complex-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 - x
    apply (rule-tac x=Re x in exI)
    apply (simp add:algebra-simps)
    by (simp add: mult.commute scaleR-complex.code times-complex.code)
  subgoal for - u
    apply (rule rev-image-eqI[of of-real u])
    by (auto simp:scaleR-conv-of-real algebra-simps)
done
then show ?thesis
unfolding pc-def
apply (subst proots-pcompose)
using ⟨p≠0⟩ ⟨st≠tt⟩ by auto
qed
finally show ?thesis by simp
qed

lemma proots-unbounded-line-card-code[code]:
  proots-unbounded-line-card p st tt =
  (if st≠tt then
    (let pc = pcompose p [:st, tt - st:] ;
      pR = map-poly Re pc;
      pI = map-poly Im pc ;
    ...
\[ g = \gcd p_R p_I \]
\[ \text{in } \mathbb{N} \left( \text{changes-R-smods} \ g \ (pderiv \ g) \right) \]

else

\begin{center}
\texttt{Code.abort (STR \ "proots-unbounded-line-card fails due to invalid hyperplanes."\)}
\end{center}

\begin{center}
\texttt{(\lambda-. \ proots-unbounded-line-card \ p \ st \ tt)}
\end{center}

\textbf{unfolding} \texttt{proots-unbounded-line-card-def using card-proots-unbounded-line[of st tt p]} \textbf{by auto}

\textbf{lemma} \texttt{proots-unbounded-line-code[code]}:
\begin{align*}
\text{proots-unbounded-line} \ p \ st \ tt &= \ \\
&\ (\text{if } st \neq tt \text{ then} \ \\
&\quad \text{if } p \neq 0 \text{ then} \ \\
&\quad \text{(let } pc = pcompose p \ [st, \ tt - st]; \ \\
&\quad \quad p_R = \text{map-poly Re pc;} \ \\
&\quad \quad p_I = \text{map-poly Im pc;} \ \\
&\quad \quad g = \gcd p_R p_I \ \\
&\quad \text{in } \mathbb{N} \left( \text{changes-R-smods-ext} \ g \ (pderiv \ g) \right) \ \\
&\text{else} \ \\
&\text{Code.abort (STR \ "proots-unbounded-line fails due to p=0"\)} \ \\
&\text{(\lambda-. \ proots-unbounded-line \ p \ st \ tt)} \ \\
&\text{else} \ \\
&\text{Code.abort (STR \ "proots-unbounded-line fails due to invalid hyperplanes."\)} \ \\
&\text{(\lambda-. \ proots-unbounded-line \ p \ st \ tt)} \ \\
&\text{unfolding} \texttt{proots-unbounded-line-def using proots-unbounded-line by auto}
\end{align*}

\subsection*{3.5 Checking if there a polynomial root on a closed segment}

\textbf{definition} \texttt{no-proots-line::complex \ poly \Rightarrow \ complex \Rightarrow \ complex \Rightarrow bool \ where}
\begin{align*}
\text{no-proots-line} \ p \ st \ tt &= (\text{proots-within} \ p \ (\text{closed-segment} \ st \ tt) = \{\})
\end{align*}

\textbf{lemma} \texttt{no-proots-line-code[code]}: \texttt{no-proots-line} \ p \ st \ tt = (\text{if } poly \ p \ st \neq 0 \land poly \ p \ tt \neq 0 \text{ then} \ \\
\quad \text{(let } pc = pcompose p \ [st, \ tt - st]; \ \\
\quad \quad p_R = \text{map-poly Re pc;} \ \\
\quad \quad p_I = \text{map-poly Im pc;} \ \\
\quad \quad g = \gcd p_R p_I \ \\
\quad \text{in if changes-ite-smods 0 1 g (pderiv g) = 0 then True else False) \ \\
\text{else False)} \ \\
\quad (is \ ?L = ?R) \ \\
\text{proof (cases } poly \ p \ st \neq 0 \land poly \ p \ tt \neq 0) \ \\
\quad \text{case False} \ \\
\quad \text{thus } ?\text{thesis unfolding no-proots-line-def by auto} \ \\
\text{next} \ \\
\quad \text{case True} \ \\
\quad \text{then have } poly \ p \ st \neq 0 \ poly \ p \ tt \neq 0 \text{ by auto} \ \\
\quad \text{define } pc \ p_R p_I g \text{ where}
\[\begin{align*}
    pc &= \text{pcompose } p \; [\text{st}, \text{tt}-\text{st}] \quad \text{and} \\
    pR &= \text{map-poly } Re \; pc \quad \text{and} \\
    pI &= \text{map-poly } Im \; pc \quad \text{and} \\
    g &= \text{gcd } pR \; pI \\
    \text{have poly-iff} &: \text{poly } g \; t = 0 \iff \text{poly } pc \; t = 0 \text{ for } t \\
    \text{proof} - \\
    &\text{have poly } g \; t = 0 \iff \text{poly } pR \; t = 0 \land \text{poly } pI \; t = 0 \\
    &\quad \text{unfolding } g\text{-def using poly-gcd-iff by auto} \\
    &\quad \text{also have } ... \iff \text{poly } pc \; t = 0 \\
    &\quad \text{proof} - \\
    &\quad \text{have } \text{cpoly-of } pR \; pI = pc \\
    &\quad \quad \text{unfolding } pc\text{-def } pR\text{-def } pI\text{-def using cpoly-of-decompose by auto} \\
    &\quad \quad \text{then show } \text{thesis using poly-cpoly-of-real-iff by blast} \\
    &\quad \text{qed} \\
    &\text{finally show } \text{thesis by auto} \\
    &\text{qed} \\
    \text{have } \?R &= (\text{changes-itv-smods } 0 \; t \; 1 \; g \; (p\text{deriv } g) = 0) \\
    &\quad \text{using True unfolding } pc\text{-def } g\text{-def } pR\text{-def } pI\text{-def} \\
    &\quad \quad \text{by (auto simp add:Let-def)} \\
    &\quad \text{also have } ... = (\text{card } \{x. \text{poly } g \; x = 0 \land 0 < x \land x < 1\} = 0) \\
    &\quad \text{proof} - \\
    &\quad \text{have poly } g \; 0 \neq 0 \\
    &\quad \quad \text{using poly-iff[of ] True unfolding } pc\text{-def by (auto simp add:poly-pcompose)} \\
    &\quad \quad \text{moreover have poly } g \; 1 \neq 0 \\
    &\quad \quad \text{using poly-iff[of ] True unfolding } pc\text{-def by (auto simp add:poly-pcompose)} \\
    &\quad \quad \text{ultimately show } \text{thesis using sturm-interval[of ] by auto} \\
    &\quad \text{qed} \\
    &\text{also have } ... = (\{x. \text{poly } g \; x = 0 \land 0 < x \land x < 1\} = \{\}) \\
    &\text{proof} - \\
    &\quad \text{have } g \neq 0 \\
    &\quad \quad \text{proof (rule ccontr)} \\
    &\quad \quad \quad \text{assume } \neg \; g \neq 0 \\
    &\quad \quad \quad \quad \text{then have poly } pc \; 0 = 0 \\
    &\quad \quad \quad \quad \quad \text{using poly-iff[of ] by auto} \\
    &\quad \quad \quad \quad \text{then show } \text{False using True unfolding } pc\text{-def by (auto simp add:poly-pcompose)} \\
    &\quad \quad \quad \quad \text{qed} \\
    &\text{from poly-roots-finite[of this] have finite } \{x. \text{poly } g \; x = 0 \land 0 < x \land x < 1\} \\
    &\quad \quad \text{by auto} \\
    &\quad \quad \text{then show } \text{thesis using card-eq-0-iff by auto} \\
    &\quad \text{qed} \\
    &\text{also have } ... = \?L \\
    &\text{proof} - \\
    &\quad \text{have } (\exists t. \text{poly } g \; t = 0 \land 0 < t \land t < 1) \iff (\exists t::\text{real. poly } pc \; t = 0 \land 0 < t \land t < 1) \\
    &\quad \quad \text{using poly-iff by auto} \\
    &\quad \text{also have } ... \iff (\exists x. x \in \text{closed-segment } st \; tt \land \text{poly } p \; x = 0) \\
    &\text{proof} \\
    &\quad \text{assume } \exists t. \text{poly } pc \; (\text{complex-of-real } t) = 0 \land 0 < t \land t < 1 \\
    &\quad \text{then obtain } t \text{ where } *:\text{poly } pc \; (\text{of-real } t) = 0 \land 0 < t \land t < 1 \text{ by auto} \\
\end{align*}\]
define \( x \) where \( x = \text{poly } [\text{st}, \text{tt} - \text{st}] \) \( t \)

have \( x \in \text{closed-segment } st \text{ tt} \) 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 \( \text{poly } p \ x = 0 \) using * unfolding pc-def x-def
by (auto simp add:poly-pcompose)

ultimately show \( \exists x. x \in \text{closed-segment } st \text{ tt} \land \text{poly } p \ x = 0 \) by auto

next
assumes \( \exists x. x \in \text{closed-segment } st \text{ tt} \land \text{poly } p \ x = 0 \)
then obtain \( t :: \text{real} \) where \( \text{poly } p \ (\text{complex-of-real } t) = 0 \) unfolding in-segment by auto

ultimately show \( \exists t. \text{poly } p \ (\text{complex-of-real } t) = 0 \land 0 < t \land t < 1 \) by auto
qed

finally show \?thesis
unfolding no-proots-line-def proots-within-def
by blast

qed

3.6 Counting roots in a rectangle

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

lemma closed-segment-imp-Re-Im:
fixes \( x :: \text{complex} \)
assumes \( x \in \text{closed-segment } lb \ ub \)
shows \( \text{Re } lb \leq \text{Re } ub \Rightarrow \text{Re } lb \leq \text{Re } x \land \text{Re } x \leq \text{Re } ub \)
\( \text{Im } lb \leq \text{Im } ub \Rightarrow \text{Im } lb \leq \text{Im } x \land \text{Im } x \leq \text{Im } ub \)

proof –

obtain \( u \) where \( x-u=x=(1-u) \ast_R lb + u \ast_R ub \) and \( 0 \leq u u \leq 1 \)
using assms unfolding closed-segment-def by auto

have \( \text{Re } lb \leq \text{Re } x \) when \( \text{Re } lb \leq \text{Re } ub \)
proof –

have \( \text{Re } x = \text{Re } ((1-u) \ast_R lb + u \ast_R ub) \)
using \( x-u \) by blast
also have \( \ldots = \text{Re } (lb + u \ast_R (ub - lb)) \) by (auto simp add:algebra-simps)
also have \( \ldots \geq \text{Re } lb \) using \( \langle u \geq 0 \rangle \langle \text{Re } lb \leq \text{Re } ub \rangle \) by auto

finally show \?thesis .

qed

moreover have \( \text{Im } lb \leq \text{Im } x \) when \( \text{Im } lb \leq \text{Im } ub \)

62
proof
  have \( \text{Im } x = \text{Im } ((1 - u) \ast_R \text{lb} + u \ast_R \text{ub}) \)
  using \( x-u \) by blast
  also have \( \ldots = \text{Im } (\text{lb} + u \ast_R (\text{ub} - \text{lb})) \) by (auto simp add: algebra-simps)
  also have \( \ldots = \text{Im } \text{lb} + u \ast (\text{Im } \text{ub} - \text{Im } \text{lb}) \) by auto
  also have \( \ldots \geq \text{Im } \text{lb} \) using \( \langle u \geq 0 \rangle \langle \text{Im } \text{lb} \leq \text{Im } \text{ub} \rangle \) by auto
  finally show \( ?\text{thesis} \).
qed

moreover have \( \text{Re } x \leq \text{Re } \text{ub} \) when \( \text{Re } \text{lb} \leq \text{Re } \text{ub} \)
proof
  have \( \text{Re } x = \text{Re } ((1 - u) \ast_R \text{lb} + u \ast_R \text{ub}) \)
  using \( x-u \) by blast
  also have \( \ldots = (1 - u) \ast \text{Re } \text{lb} + u \ast \text{Re } \text{ub} \) by auto
  also have \( \ldots \leq (1 - u) \ast \text{Re } \text{ub} + u \ast \text{Re } \text{ub} \)
  using \( \langle u \leq 1 \rangle \langle \text{Re } \text{lb} \leq \text{Re } \text{ub} \rangle \) by (auto simp add: mult-left-mono)
  also have \( \ldots = \text{Re } \text{ub} \) by (auto simp add: algebra-simps)
  finally show \( ?\text{thesis} \).
qed

moreover have \( \text{Im } x \leq \text{Im } \text{ub} \) when \( \text{Im } \text{lb} \leq \text{Im } \text{ub} \)
proof
  have \( \text{Im } x = \text{Im } ((1 - u) \ast_R \text{lb} + u \ast_R \text{ub}) \)
  using \( x-u \) by blast
  also have \( \ldots = (1 - u) \ast \text{Im } \text{lb} + u \ast \text{Im } \text{ub} \) by auto
  also have \( \ldots \leq (1 - u) \ast \text{Im } \text{ub} + u \ast \text{Im } \text{ub} \)
  using \( \langle u \leq 1 \rangle \langle \text{Im } \text{lb} \leq \text{Im } \text{ub} \rangle \) by (auto simp add: mult-left-mono)
  also have \( \ldots = \text{Im } \text{ub} \) by (auto simp add: algebra-simps)
  finally show \( ?\text{thesis} \).
qed

ultimately show
  \( \text{Re } \text{lb} \leq \text{Re } \text{ub} \imp \text{Re } \text{lb} \leq \text{Re } x \land \text{Re } x \leq \text{Re } \text{ub} \)
  \( \text{Im } \text{lb} \leq \text{Im } \text{ub} \imp \text{Im } \text{lb} \leq \text{Im } x \land \text{Im } x \leq \text{Im } \text{ub} \)
  by auto
qed

lemma closed-segment-degen-complex:
  \[
  \begin{align*}
  \text{Re } \text{lb} = \text{Re } \text{ub}; \text{Im } \text{lb} \leq \text{Im } \text{ub} & \implies x \in \text{closed-segment } \text{lb} \text{ ub} \iff \text{Re } x = \text{Re } \text{lb} \land \text{Im } \text{lb} \leq \text{Im } x \land \text{Im } x \leq \text{Im } \text{ub} \\
  \text{Im } \text{lb} = \text{Im } \text{ub}; \text{Re } \text{lb} \leq \text{Re } \text{ub} & \implies x \in \text{closed-segment } \text{lb} \text{ ub} \iff \text{Im } x = \text{Im } \text{lb} \land \text{Re } \text{lb} \leq \text{Re } x \land \text{Re } x \leq \text{Re } \text{ub}
  \end{align*}
\]
proof
  show \( x \in \text{closed-segment } \text{lb} \text{ ub} \iff \text{Re } x = \text{Re } \text{lb} \land \text{Im } \text{lb} \leq \text{Im } x \land \text{Im } x \leq \text{Im } \text{ub} \)
  when \( \text{Re } \text{lb} = \text{Re } \text{ub} \) \( \text{Im } \text{lb} \leq \text{Im } \text{ub} \)
  proof
    show \( \text{Re } x = \text{Re } \text{lb} \land \text{Im } \text{lb} \leq \text{Im } x \land \text{Im } x \leq \text{Im } \text{ub} \) when \( x \in \text{closed-segment } \text{lb} \text{ ub} \)
    using closed-segment-imp-Re-Im[OF that] \( \langle \text{Re } \text{lb} = \text{Re } \text{ub} \rangle \langle \text{Im } \text{lb} \leq \text{Im } \text{ub} \rangle \)
by fastforce

next
assume \( \text{asm: } \Re x = \Re lb \land \Im lb \leq \Im x \land \Im x \leq \Im ub \)
define \( u \) where \( u = (\Im x - \Im lb) / (\Im ub - \Im lb) \)
have \( x = (1 - u) * R lb + u * R ub \)
unfolding \( u\)-def using \( \text{asm: } \Re lb = \Re ub \land \Im lb \leq \Im ub \)
apply (intro complex-eqI)
apply (auto simp add: field-simps)
apply (cases \( \Im ub - \Im lb = 0 \))
apply (auto simp add: field-simps)
done
moreover have \( 0 \leq u \leq 1 \) unfolding \( u\)-def
using \( \text{asm: } \Im lb \leq \Im ub \)
by cases \( \Re ub = \Re lb = 0 \), auto simp add: field-simps+
ultimately show \( x \in \text{closed-segment lb ub} \) unfolding \( \text{closed-segment-def} \) by
auto
qed

show \( x \in \text{closed-segment lb ub} \iff \Im x = \Im lb \land \Re lb \leq \Re x \land \Re x \leq \Re ub \)
when \( \Im lb = \Im ub \) \( \Re lb \leq \Re ub \)
proof
show \( \Im x = \Im lb \land \Re lb \leq \Re x \land \Re x \leq \Re ub \) when \( x \in \text{closed-segment lb ub} \)
using \( \text{closed-segment-imp-Re-Im} \) (OF that) \( \Im lb = \Im ub \land \Re lb \leq \Re ub \)
by fastforce
next
assume \( \text{asm: } \Im x = \Im lb \land \Re lb \leq \Re x \land \Re x \leq \Re ub \)
define \( u \) where \( u = (\Re x - \Re lb) / (\Re ub - \Re lb) \)
have \( x = (1 - u) * R lb + u * R ub \)
unfolding \( u\)-def using \( \text{asm: } \Im lb = \Im ub \land \Re lb \leq \Re ub \)
apply (intro complex-eqI)
apply (auto simp add: field-simps)
apply (cases \( \Re ub - \Re lb = 0 \))
apply (auto simp add: field-simps)
done
moreover have \( 0 \leq u \leq 1 \) unfolding \( u\)-def
using \( \text{asm: } \Re lb \leq \Re ub \)
by cases \( \Re ub - \Re lb = 0 \), auto simp add: field-simps+
ultimately show \( x \in \text{closed-segment lb ub} \) unfolding \( \text{closed-segment-def} \) by
auto
qed
qed

lemma \( \text{complex-box-ne-empty} \):
fixes \( a \) \( b :: \text{complex} \)
shows
\( \text{cbox } a \neq \{ \} \iff (\Re a \leq \Re b \land \Im a \leq \Im b) \)
\( \text{box } a \neq \{ \} \iff (\Re a < \Re b \land \Im a < \Im b) \)
by (auto simp add: box-ne-empty Basis-complex-def)
lemma proots-rectangle-code1:
proots-rectangle p lb ub = (if Re lb < Re ub ∧ Im lb < Im ub then
  if p ≠ 0 then
    if no-proots-line p lb (Complex (Re ub) (Im lb))
    ∧ no-proots-line p (Complex (Re ub) (Im lb)) ub
    ∧ no-proots-line p ub (Complex (Re lb) (Im ub))
    ∧ no-proots-line p (Complex (Re lb) (Im ub)) lb then
      (let p1 = pcompose p [lb, Complex (Re ub − Re lb) 0];
       pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
       p2 = pcompose p [Complex (Re ub) (Im lb), Complex 0 (Im ub − Im lb)];
       pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
       p3 = pcompose p [ub, Complex (Re lb − Re ub) 0];
       pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
       p4 = pcompose p [Complex (Re lb) (Im ub), Complex 0 (Im lb − Im ub)];
       pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
       in
       nat (− (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1))
          + changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
          + changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
          + changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)) div 4)
    ) else Code.abort (STR "proots-rectangle fails when there is a root on the border.")
  ) else Code.abort (STR "proots-rectangle fails when p = 0.")
else 0)
proof
  have ?thesis when ¬ (Re lb < Re ub ∧ Im lb < Im ub)
proof
  have box lb ub = {} using complex-box-ne-empty[of lb ub] that by auto
  then have proots-rectangle p lb ub = 0 unfolding proots-rectangle-def by auto
  then show ?thesis by (simp add:that)
qed
moreover have ?thesis when Re lb < Re ub ∧ Im lb < Im ub p=0
  using that by simp
moreover have ?thesis when
  Re lb < Re ub Im lb < Im ub p≠0
  and no-proots:
    no-proots-line p lb (Complex (Re ub) (Im lb))
    no-proots-line p (Complex (Re ub) (Im lb)) ub
    no-proots-line p ub (Complex (Re lb) (Im ub))
    no-proots-line p (Complex (Re lb) (Im ub)) lb
proof
  define l1 where l1 = linepath lb (Complex (Re ub) (Im lb))
define \( l_2 \) where \( l_2 = \text{linepath} \left( \text{Complex} \left( \text{Re} \ ub \right) \left( \text{Im} \ lb \right) \right) \) ub
define \( l_3 \) where \( l_3 = \text{linepath} \left( \text{ub} \left( \text{Complex} \left( \text{Re} \ lb \right) \left( \text{Im} \ ub \right) \right) \right) \)
define \( l_4 \) where \( l_4 = \text{linepath} \left( \text{Complex} \left( \text{Re} \ lb \right) \left( \text{Im} \ ub \right) \right) \) lb
define \( \text{rec} \) where \( \text{rec} = l_1 \text{+++} l_2 \text{+++} l_3 \text{+++} l_4 \)

have \( \text{valid} \left[ \text{simp} \right] : \text{valid-path} \ \text{rec} \) and loop \( \text{valid} \left[ \text{simp} \right] : \text{pathfinish} \ \text{rec} = \text{pathstart} \ \text{rec} \)

unfolding \( \text{rec-def l1-def l2-def l3-def l4-def} \) by auto

have \( \text{valid} \left[ \text{simp} \right] : \text{path-no-proots} \ \text{path-image} \ \text{rec} \) and loop \( \text{valid} \left[ \text{simp} \right] : \text{pathfinish} \ \text{rec} = \text{pathstart} \ \text{rec} \)

unfolding \( \text{g1-def g2-def g3-def g4-def l1-def l2-def l3-def l4-def} \) by auto

define \( g_1 \) where \( g_1 = \text{poly} \ \text{p} \circ \ l_1 \)
define \( g_2 \) where \( g_2 = \text{poly} \ \text{p} \circ \ l_2 \)
define \( g_3 \) where \( g_3 = \text{poly} \ \text{p} \circ \ l_3 \)
define \( g_4 \) where \( g_4 = \text{poly} \ \text{p} \circ \ l_4 \)

have \( \text{valid} \left[ \text{simp} \right] : \text{path} \ \text{g1 path} \ \text{g2 path} \ \text{g3 path} \ \text{g4} \)

pathfinish \( g_1 = \text{pathstart} \ \text{g2 pathfinish} \ \text{g2} = \text{pathstart} \ \text{g3 pathfinish} \ \text{g3} = \text{pathstart} \ \text{g4} \)

unfolding \( \text{g1-def g2-def g3-def g4-def l1-def l2-def l3-def l4-def} \) by auto

have \( \text{valid} \left[ \text{simp} \right] : \text{finite-ReZ-segments} \ \text{g1 0 finite-ReZ-segments} \ \text{g2 0} \)

finite-ReZ-segments \( g_3 \) \( 0 \) finite-ReZ-segments \( g_4 \) \( 0 \)

unfolding \( \text{g1-def g2-def g3-def g4-def l1-def l2-def l3-def l4-def poly-linepath-comp} \) by auto

define \( p_1 \ pR_1 \ pI_1 \ gc_1 \)
\( p_2 \ pR_2 \ pI_2 \ gc_2 \)
\( p_3 \ pR_3 \ pI_3 \ gc_3 \)
\( p_4 \ pR_4 \ pI_4 \ gc_4 \)

where \( p_1 = \text{pcompose} \ \text{p [:} \text{lb, Complex} \left( \text{Re} \ ub - \text{Re} \ lb \right) \ 0:] \)
and \( pR_1 = \text{map-poly} \ \text{Re} \ p_1 \) and \( pI_1 = \text{map-poly} \ \text{Im} \ p_1 \) and \( gc_1 = \text{gcd} \ \text{pR} \text{R}_1 \) \( pI_1 \)
and \( p_2 = \text{pcompose} \ \text{p [:} \text{Complex} \left( \text{Re} \ ub \right) \left( \text{Im} \ lb \right), \text{Complex} \ 0 \left( \text{Im} \ ub - \text{Im} \ lb \right) :] \)
and \( pR_2 = \text{map-poly} \ \text{Re} \ p_2 \) and \( pI_2 = \text{map-poly} \ \text{Im} \ p_2 \) and \( gc_2 = \text{gcd} \ \text{pR} \text{R}_2 \) \( pI_2 \)
and \( p_3 = \text{pcompose} \ \text{p [:} \text{ub, Complex} \left( \text{Re} \ lb - \text{Re} \ ub \right) \ 0:] \)
and \( pR_3 = \text{map-poly} \ \text{Re} \ p_3 \) and \( pI_3 = \text{map-poly} \ \text{Im} \ p_3 \) and \( gc_3 = \text{gcd} \ \text{pR} \text{R}_3 \) \( pI_3 \)
and \( p_4 = \text{pcompose} \ \text{p [:} \text{Complex} \left( \text{Re} \ lb \right) \left( \text{Im} \ ub \right), \text{Complex} \ 0 \left( \text{Im} \ lb - \text{Im} \ ub \right) :] \)
and \( pR_4 = \text{map-poly} \ \text{Re} \ p_4 \) and \( pI_4 = \text{map-poly} \ \text{Im} \ p_4 \) and \( gc_4 = \text{gcd} \ \text{pR} \text{R}_4 \) \( pI_4 \)

have \( gc_1 \neq 0 \) \( gc_2 \neq 0 \) \( gc_3 \neq 0 \) \( gc_4 \neq 0 \)

proof
  show \( gc_1 \neq 0 \)
  proof (rule ccontr)

66
assume $\neg gc1 \neq 0$
then have $pI1 = 0$ $pR1 = 0$ unfolding $gc1$-def by auto
then have $pI1 = 0$ unfolding $pI1$-def $pR1$-def
by (metis cpoly-of-decompose map-poly-0)
then have $p = 0$ unfolding $pI1$-def using ($Re lb < Re ub$)
by (auto elim!:pcompose-eq-0 simp add:Complex-eq-0)
then show False using $\langle p \neq 0 \rangle$ by simp
qed

show $gc2 \neq 0$
proof (rule ccontr)
assume $\neg gc2 \neq 0$
then have $pI2 = 0$ $pR2 = 0$ unfolding $gc2$-def by auto
then have $pI2 = 0$ unfolding $pI2$-def $pR2$-def
by (metis cpoly-of-decompose map-poly-0)
then have $p = 0$ unfolding $pI2$-def using ($Im lb < Im ub$)
by (auto elim!:pcompose-eq-0 simp add:Complex-eq-0)
then show False using $\langle p \neq 0 \rangle$ by simp
qed

show $gc3 \neq 0$
proof (rule ccontr)
assume $\neg gc3 \neq 0$
then have $pI3 = 0$ $pR3 = 0$ unfolding $gc3$-def by auto
then have $pI3 = 0$ unfolding $pI3$-def $pR3$-def
by (metis cpoly-of-decompose map-poly-0)
then have $p = 0$ unfolding $pI3$-def using ($Re lb < Re ub$)
by (auto elim!:pcompose-eq-0 simp add:Complex-eq-0)
then show False using $\langle p \neq 0 \rangle$ by simp
qed

show $gc4 \neq 0$
proof (rule ccontr)
assume $\neg gc4 \neq 0$
then have $pI4 = 0$ $pR4 = 0$ unfolding $gc4$-def by auto
then have $pI4 = 0$ unfolding $pI4$-def $pR4$-def
by (metis cpoly-of-decompose map-poly-0)
then have $p = 0$ unfolding $pI4$-def using ($Im lb < Im ub$)
by (auto elim!:pcompose-eq-0 simp add:Complex-eq-0)
then show False using $\langle p \neq 0 \rangle$ by simp
qed

qed

define sms where

\[
\text{sms} = (\text{changes-alt-itv-smods} 0 1 (pR1 \text{ div } gc1) (pI1 \text{ div } gc1) \\
+ \text{changes-alt-itv-smods} 0 1 (pR2 \text{ div } gc2) (pI2 \text{ div } gc2) \\
+ \text{changes-alt-itv-smods} 0 1 (pR3 \text{ div } gc3) (pI3 \text{ div } gc3) \\
+ \text{changes-alt-itv-smods} 0 1 (pR4 \text{ div } gc4) (pI4 \text{ div } gc4))
\]

have proots-rectangle p lb ub = ($\sum r \in \text{proots } p. \ \text{winding-number } \text{rec } r * (\text{order } r \ p)$)
proof
have winding-number rec x * of-nat (order x p) = 0
when $x \in \text{proots } p$ at proots-within p (box lb ub) for x
proof

have \textasteriskcentered cbox \( lb \) \( ub \) = box \( lb \) \( ub \) \( \cup \) path-image \( \text{rec} \)

proof

have \( x \in cbox \) \( lb \) \( ub \) when \( x \in \) box \( lb \) \( ub \) \( \cup \) path-image \( \text{rec} \) for \( x \)

using that \( \text{Re} \) \( lb \) \( < \) \( \text{Re} \) \( ub \) \( \land \) \( \text{Im} \) \( lb \) \( < \) \( \text{Im} \) \( ub \)

unfolding box-def cbox-def Basis-complex-def rec-def l1-def l2-def l3-def

l4-def

apply (auto simp add: path-image-join closed-segment-degen-complex)

apply (subst (asm) closed-segment-commute, simp add: closed-segment-degen-complex)+
done

moreover have \( x \in \) box \( lb \) \( ub \) \( \cup \) path-image \( \text{rec} \) when \( x \in \) cbox \( lb \) \( ub \) for \( x \)

using that

unfolding box-def cbox-def Basis-complex-def rec-def l1-def l2-def l3-def

l4-def

apply (auto simp add: path-image-join closed-segment-degen-complex)

apply (subst (asm) (1 2) closed-segment-commute, simp add: closed-segment-degen-complex)+
done

ultimately show \( \text{thesis} \) by auto

qed

moreover have \( x \notin \) path-image \( \text{rec} \)

using path-no-proots that

ultimately have \( x \notin \) cbox \( lb \) \( ub \) using that by simp

from winding-number-zero-outside[OF valid-path-imp-path[OF valid] - loop

this,simplified]*

have winding-number \( \text{rec} \) \( x \) = 0 by auto

then show \( \text{thesis} \) by auto

qed

moreover have of-nat \( (\text{order} \ x \ p) = \) winding-number \( \text{rec} \) \( x \) \( \times \) of-nat \( (\text{order} \ x \ p) \) when

\( x \in \) proots-within \( p \) \( (\text{box} \ lb \ ub) \) for \( x \)

proof

have \( x \in \) box \( lb \) \( ub \) using that unfolding proots-within-def by auto

then have order-asms: \( \text{Re} \ lb < \text{Re} \ x \ \text{Re} \ x < \text{Re} \ ub \) \( \text{Im} \ lb < \text{Im} \ x \ \text{Im} \ x \) \( \text{Im} \ ub \)

by (auto simp add: box-def Basis-complex-def)

have winding-number \( \text{rec} \) \( x \) = 1

unfolding rec-def l1-def l2-def l3-def l4-def

proof eval-winding

let \( ?l1 = \) linepath \( lb \) (Complex \( \text{Re} \) \( ub \) \( \text{(Im} \ lb) \))

and \( ?l2 = \) linepath (Complex \( \text{Re} \) \( ub \) \( \text{(Im} \ lb) \)) \( lb \)

and \( ?l3 = \) linepath ub (Complex \( \text{Re} \) \( lb \) \( \text{(Im} \ ub) \)) ub

and \( ?l4 = \) linepath (Complex \( \text{Re} \) \( lb \) \( \text{(Im} \ ub) \)) \( lb \)

show \( l1: \ x \notin \) path-image \( ?l1 \) and \( l2: \ x \notin \) path-image \( ?l2 \) and

\( l3: \ x \notin \) path-image \( ?l3 \) and \( l4: \ x \notin \) path-image \( ?l4 \)

using no-proots that unfolding no-proots-line-def by auto

show \( - \) of-real \( (\text{cindex-pathE} \ ?l1 \ x + (\text{cindex-pathE} \ ?l2 \ x + (\text{cindex-pathE} \ ?l3 \ x + \text{cindex-pathE} \ ?l4 \ x))) = 2 * 1 \)

proof

have \((Im \ x - Im \ ub) * (Re \ ub - Re \ lb) < 0\)
using mult-less-0-iff order-asms(1) order-asms(2) order-asms(4) by fastforce
then have cindex-pathE \(?l3 \ x = \cdash\)
apply (subst cindex-pathE-linepath)
using l3 by (auto simp add: algebra-simps order-asms)
moreover have \((Im \ lb - Im \ x) * (Re \ ub - Re \ lb) < 0\)
using mult-less-0-iff order-asms(1) order-asms(2) order-asms(3) by fastforce
then have cindex-pathE \(?l1 \ x = \cdash\)
apply (subst cindex-pathE-linepath)
using l1 by (auto simp add: algebra-simps order-asms)
moreover have cindex-pathE \(?l2 \ x = 0\)
apply (subst cindex-pathE-linepath)
using l2 order-asms by auto
moreover have cindex-pathE \(?l4 \ x = 0\)
apply (subst cindex-pathE-linepath)
using l4 order-asms by auto
ultimately show \(?thesis\) by auto
qed

ultimately show \(?thesis\) by auto
qed

ultimately show \(?thesis\) using \(\langle p \neq 0 \rangle\)
unfolding proots-rectangle-def proots-count-def
by (auto intro!: sum.mono-neutral-cong-left[where \(a=\text{complex}\)] of proots p proots-within p (box lb ub))

also have \(\ldots = 1/(2 * \text{of-real } pi * i) * \text{ contour-integral } (\lambda x. \text{ deriv } (\text{poly } p) x / \text{poly } p x)\)
proof -
  have \(\text{ contour-integral } (\lambda x. \text{ deriv } (\text{poly } p) x / \text{poly } p x) = 2 * \text{ of-real } pi * i * (\sum x | \text{poly } p x = 0. \text{ winding-number } \text{rec } x * \text{ of-int } (\text{order } (\text{poly } p) x))\)
  proof (rule argument-principle[of UNIV poly p \{\} \ldash 1 rec,simplified])
      show connected (UNIV::complex set) using connected-UNIV[where \(a=\text{complex}\)]
      .
      show path-image rec \subseteq UNIV - \{x. poly p x = 0\}
      using path-no-proots unfolding proots-within-def by auto
      show finite \(\{x. poly p x = 0\}\) by (simp add: poly-roots-finite that(3))
  qed
also have \(\ldots = 2 * \text{ of-real } pi * i * (\sum x \in \text{proots } p. \text{ winding-number } \text{rec } x * (\text{order } x p))\)
unfolding proots-within-def
apply (auto intro!: sum.cong simp add: order-root order-zorder[OF \(p\neq 0\)] )
by (metis nat-eq-iff2 of-nat-nat order-root order-root order-zorder that(3))
finally show \(?thesis\) by auto
qed
also have \( \ldots = \text{winding-number} (\text{poly } p \circ \text{rec}) 0 \)

proof

have \( 0 \notin \text{path-image} (\text{poly } p \circ \text{rec}) \)

using path-no-proots unfolding path-image-compose proots-within-def by fastforce

from winding-number-comp[OF - poly-holomorphic-on - - this[of UNIV,simplified]]

show \(?thesis by auto

qed

also have winding-eq\(\ldots = - \text{cindex-pathE} (\text{poly } p \circ \text{rec}) 0 / 2 \)

proof (rule winding-number-cindex-pathE)

show \(\text{finite-ReZ-segments} (\text{poly } p \circ \text{rec}) 0 \)

unfolding rec-def path-compose-join

apply (fold g1-def g2-def g3-def g4-def)

by (auto intro!: finite-ReZ-segments-joinpaths path-join-imp)

show valid-path (\text{poly } p \circ \text{rec})

by (rule valid-path-compose-holomorphic[where S=UNIV]) auto

show \(0 \notin \text{path-image} (\text{poly } p \circ \text{rec}) \)

using path-no-proots unfolding path-image-compose proots-def by fastforce

show pathfinish (\text{poly } p \circ \text{rec}) = pathstart (\text{poly } p \circ \text{rec})

unfolding rec-def pathstart-compose pathfinish-compose by (auto simp add:11-def l4-def)

qed

also have cindex-pathE-eq\(\ldots = \text{of-int} (- \text{sms}) / \text{of-int 4} \)

proof

have cindex-pathE (\text{poly } p \circ \text{rec}) 0 = cindex-pathE (g1+++g2+++g3+++g4)

0

unfolding rec-def path-compose-join g1-def g2-def g3-def g4-def by simp

also have \(\ldots = \text{cindex-pathE} g1 0 + \text{cindex-pathE} g2 0 + \text{cindex-pathE} g3 0 + \text{cindex-pathE} g4 0 \)

by (subst cindex-pathE-joinpaths,auto intro!:finite-ReZ-segments-joinpaths)+

also have \(\ldots = \text{cindex-polyE} 0 1 (pI1 div gc1) (pR1 div gc1)

+ \text{cindex-polyE} 0 1 (pI2 div gc2) (pR2 div gc2)

+ \text{cindex-polyE} 0 1 (pI3 div gc3) (pR3 div gc3)

+ \text{cindex-polyE} 0 1 (pI4 div gc4) (pR4 div gc4)

proof

have \(\text{cindex-pathE} g1 0 = \text{cindex-polyE} 0 1 (pI1 div gc1) (pR1 div gc1) \)

proof

have \(\ast:g1 = \text{poly } p1 o \text{of-real} \)

unfolding g1-def p1-def l1-def poly-linepath-comp

by (subst (5) complex-surf[ symmetric],simp)

then have \(\text{cindex-pathE} g1 0 = \text{cindexE} 0 1 (\lambda t. \text{poly } p1 t / \text{poly } pR1 t) \)

unfolding cindex-pathE-def pR1-def pI1-def

by (simp add:Im-poly-of-real Re-poly-of-real)

also have \(\ldots = \text{cindex-polyE} 0 1 pI1 pR1 \)

using cindexE-eq-cindex-polyE by auto

also have \(\ldots = \text{cindex-polyE} 0 1 (pI1 div gc1) (pR1 div gc1) \)

using \(gc1 \neq 0 \)

apply (subst (2) cindex-polyE-mult-cancel[of gc1,symmetric])

by (simp-all add: gc1-def)
finally show \(\textit{thesis}\).

\textbf{qed}

moreover have \(\text{cindex-pathE } g2 \ 0 = \text{cindex-polyE } 0 \ 1 \ (pI2 \ div \ gc2) \ (pR2 \ div \ gc2)\)

\textbf{proof} –

have \(g2 = \text{poly } p2 \ o \ \text{of-real}\)

unfolding \(g2\text{-def} \ p2\text{-def} \ l2\text{-def} \ \text{poly-linepath-comp}\)

by (subst (5) complex-surj[symmetric],simp)

then have \(\text{cindex-pathE } g2 \ 0 = \text{cindexE } 0 \ 1 \ (\lambda t. \ \text{poly } pI2 t / \ \text{poly } pR2 t)\)

unfolding \(\text{cindex-pathE-def } pR2\text{-def} \ pI2\text{-def}\)

by (simp add:Im-poly-of-real Re-poly-of-real)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ pI2 \ pR2\)

using \(\text{cindexE-eq-cindex-polyE by auto}\)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ (\lambda t. \ \text{poly } pR2 t / \ \text{poly } pI2 t)\)

unfolding \(\text{cindex-polyE-mult-cancel[of gc2,symmetric]}\)

by (simp-all add: gc2-def)

finally show \(\textit{thesis}\).

\textbf{qed}

moreover have \(\text{cindex-pathE } g3 \ 0 = \text{cindex-polyE } 0 \ 1 \ (pI3 \ div \ gc3) \ (pR3 \ div \ gc3)\)

\textbf{proof} –

have \(g3 = \text{poly } p3 \ o \ \text{of-real}\)

unfolding \(g3\text{-def} \ p3\text{-def} \ l3\text{-def} \ \text{poly-linepath-comp}\)

by (subst (5) complex-surj[symmetric],simp)

then have \(\text{cindex-pathE } g3 \ 0 = \text{cindexE } 0 \ 1 \ (\lambda t. \ \text{poly } pI3 t / \ \text{poly } pR3 t)\)

unfolding \(\text{cindex-pathE-def } pR3\text{-def} \ pI3\text{-def}\)

by (simp add:Im-poly-of-real Re-poly-of-real)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ pI3 \ pR3\)

using \(\text{cindexE-eq-cindex-polyE by auto}\)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ (\lambda t. \ \text{poly } pR3 t / \ \text{poly } pI3 t)\)

using \(\langle gc3 \neq 0 \rangle\)

apply (subst (2) \text{cindex-polyE-mult-cancel[of gc3,symmetric]})

by (simp-all add: gc3-def)

finally show \(\textit{thesis}\).

\textbf{qed}

moreover have \(\text{cindex-pathE } g4 \ 0 = \text{cindex-polyE } 0 \ 1 \ (pI4 \ div \ gc4) \ (pR4 \ div \ gc4)\)

\textbf{proof} –

have \(g4 = \text{poly } p4 \ o \ \text{of-real}\)

unfolding \(g4\text{-def} \ p4\text{-def} \ l4\text{-def} \ \text{poly-linepath-comp}\)

by (subst (5) complex-surj[symmetric],simp)

then have \(\text{cindex-pathE } g4 \ 0 = \text{cindexE } 0 \ 1 \ (\lambda t. \ \text{poly } pI4 t / \ \text{poly } pR4 t)\)

unfolding \(\text{cindex-pathE-def } pR4\text{-def} \ pI4\text{-def}\)

by (simp add:Im-poly-of-real Re-poly-of-real)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ pI4 \ pR4\)

using \(\text{cindexE-eq-cindex-polyE by auto}\)

also have \(\ldots = \text{cindex-polyE } 0 \ 1 \ (\lambda t. \ \text{poly } pR4 t / \ \text{poly } pI4 t)\)

using \(\langle gc4 \neq 0 \rangle\)

71
apply (subst (2) cindex-polyE-mult-cancel[of gc4,symmetric])
by (simp-all add: gc4-def)
finally show ?thesis.

qed
ultimately show ?thesis by auto

qed
also have ... = sms / 2

proof –
  have cindex-polyE 0 1 (pI1 div gc1) (pR1 div gc1)
    = changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1) / 2
  apply (rule cindex-polyE-changes-alt-itv-mods)
  using ⟨gc1 ≠ 0⟩ unfolding gc1-def
  by (auto intro: div-gcd-coprime)

  moreover have cindex-polyE 0 1 (pI2 div gc2) (pR2 div gc2)
    = changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2) / 2
  apply (rule cindex-polyE-changes-alt-itv-mods)
  using ⟨gc2 ≠ 0⟩ unfolding gc2-def
  by (auto intro: div-gcd-coprime)

  moreover have cindex-polyE 0 1 (pI3 div gc3) (pR3 div gc3)
    = changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3) / 2
  apply (rule cindex-polyE-changes-alt-itv-mods)
  using ⟨gc3 ≠ 0⟩ unfolding gc3-def
  by (auto intro: div-gcd-coprime)

  moreover have cindex-polyE 0 1 (pI4 div gc4) (pR4 div gc4)
    = changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4) / 2
  apply (rule cindex-polyE-changes-alt-itv-mods)
  using ⟨gc4 ≠ 0⟩ unfolding gc4-def
  by (auto intro: div-gcd-coprime)

ultimately show ?thesis unfolding sms-def
by auto

qed
finally have *:cindex-pathE (poly p ◦ rec) 0 = real-of-int sms / 2.
show ?thesis
  apply (subst *)
  by auto

qed

finally have (of-nat::→ complex) (proots-rectangle p lb ub) = of-int (− sms)
/ of-int 4.

moreover have 4 dvd sms

proof –
  have winding-number (poly p ◦ rec) 0 ∈ ℤ
  proof (rule integer-winding-number)
    show path (poly p ◦ rec)
      by (auto intro: valid-path-compose-holomorphic[where S=UNIV] valid-path-imp-path)
    show pathfinish (poly p ◦ rec) = pathstart (poly p ◦ rec)
      unfolding rec-def path-compose-join
      by (auto simp add:l1-def l4-def pathfinish-compose pathstart-compose)
    show 0 ∉ path-image (poly p ◦ rec)
      using path-no-proots unfolding path-image-compose proots-def by fastforce
    qed
  then have of-int (− sms) / of-int 4 ∈ (ℤ::complex set)
    by (simp only: winding-eq cindex-pathE-eq)
  then show ?thesis
    by (subst (asm) dvd-divide-Ints-iff[symmetric],auto)
  qed

72
ultimately have \( \text{proots-rectangle } p \text{ lb ub } = \text{ nat } (\text{-- sms div 4}) \)
apply (subst (asm) of-int-div-field[symmetric])
by (simp,metis nat-int of-int-eq-iff of-int-of-nat-eq)
then show ?thesis
unfolding Let-def
apply (fold p1-def p2-def p3-def p4-def pI1-def pR1-def pI2-def pR2-def pI3-def pR3-def)
pI4-def pR4-def gc1-def gc2-def gc3-def gc4-def
apply (fold sms-def)
using that by auto
qed
ultimately show ?thesis
by fastforce
qed

lemma \( \text{proots-rectangle-code2}[\text{code}]: \)
\( \text{proots-rectangle } p \text{ lb ub } = (\text{if Re lb} < \text{Re ub} \land \text{Im lb} < \text{Im ub} \text{ then}) \)
\( \text{if } p \neq 0 \text{ then} \)
\( \text{if } \text{poly } p \text{ lb } \neq 0 \land \text{poly } p \text{ (Complex (Re ub) (Im lb)) } \neq 0 \land \text{poly } p \text{ ub } \neq 0 \land \text{poly } p \text{ (Complex (Re ub) (Im ub)) } \neq 0 \text{ then} \)
\( \text{(let } p1 = \text{pcompose } p [:: \text{lb, Complex (Re ub - Re lb) 0}]; \)
pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
p2 = \text{pcompose } p [:: \text{Complex (Re ub) (Im lb)}], \text{Complex 0 (Im ub - Im lb)}]; \)
pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
p3 = \text{pcompose } p [:: \text{ub, Complex (Re lb - Re ub) 0}]; \)
pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
p4 = \text{pcompose } p [:: \text{Complex (Re lb) (Im ub)}, \text{Complex 0 (Im lb - Im ub)}]; \)
pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
\( \text{if } \text{changes-ite-smods 0 1 gc1 (pderiv gc1)} = 0 \land \text{changes-ite-smods 0 1 gc2 (pderiv gc2)} = 0 \land \text{changes-ite-smods 0 1 gc3 (pderiv gc3)} = 0 \land \text{changes-ite-smods 0 1 gc4 (pderiv gc4)} = 0 \text{ then} \)
nat (\text{-- (changes-alt-ite-smods 0 1 (pR1 div gc1) (pI1 div gc1) + changes-alt-ite-smods 0 1 (pR2 div gc2) (pI2 div gc2) + changes-alt-ite-smods 0 1 (pR3 div gc3) (pI3 div gc3) + changes-alt-ite-smods 0 1 (pR4 div gc4) (pI4 div gc4)) div 4}) \text{ else Code.abort (STR "proots-rectangle fails when there is a root on the border.")}
(\lambda-. \text{proots-rectangle p lb ub})
else Code.abort (STR "proots-rectangle fails when there is a root on the border.")
(\lambda-. \text{proots-rectangle p lb ub})
else Code.abort (STR "proots-rectangle fails when p=0."")
(\lambda-. \text{proots-rectangle p lb ub})
proof
-  
define p1 pR1 pI1 gc1
  p2 pR2 pI2 gc2
  p3 pR3 pI3 gc3
  p4 pR4 pI4 gc4
  where p1 = pcompose p [:lb, Complex (Re ub − Re lb) 0:]
    and pR1 = map-poly Re p1 and pI1 = map-poly Im p1 and gc1 =
    gcd pR1 pI1
    and p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub −
    Im lb):]
    and pR2 = map-poly Re p2 and pI2 = map-poly Im p2 and gc2 =
    gcd pR2 pI2
    and p3 = pcompose p [:ub, Complex (Re lb − Re ub) 0:]
    and pR3 = map-poly Re p3 and pI3 = map-poly Im p3 and gc3 =
    gcd pR3 pI3
    and p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb −
    Im ub):]
    and pR4 = map-poly Re p4 and pI4 = map-poly Im p4 and gc4 =
    gcd pR4 pI4
  define sms where
    sms = (− (changes-alt-iv-smods 0 1 (pR1 div gc1) (pI1 div gc1)) +
    changes-alt-iv-smods 0 1 (pR2 div gc2) (pI2 div gc2) +
    changes-alt-iv-smods 0 1 (pR3 div gc3) (pI3 div gc3) +
    changes-alt-iv-smods 0 1 (pR4 div gc4) (pI4 div gc4)) div
  have more-folds:
    p1 = p ◦ p [:lb, Complex (Re ub) (Im lb) − lb:]
    p2 = p ◦ p [:Complex (Re ub) (Im lb), ub − Complex (Re ub) (Im lb):]
    p3 = p ◦ p [:ub, Complex (Re lb) (Im ub) − ub:]
    p4 = p ◦ p [:Complex (Re lb) (Im ub), lb − Complex (Re lb) (Im ub):]
  subgoal unfolding p1-def
    by (subst (10) complex-surj[symmetric],auto simp add:minus-complex.code)
  subgoal unfolding p2-def by (subst (10) complex-surj[symmetric],auto)
  subgoal unfolding p3-def by (subst (10) complex-surj[symmetric],auto simp
    add:minus-complex.code)
  subgoal unfolding p4-def by (subst (10) complex-surj[symmetric],auto)
  done
  show ?thesis
  apply (subst proots-rectangle-code1)
  apply (unfold no-proots-line-code Let-def)
  apply (fold p1-def p2-def p3-def p4-def pI1-def pR1-def pI2-def pR2-def pI3-def
    pR3-def pI4-def pR4-def gc1-def gc2-def gc3-def gc4-def more-folds)
  apply (fold sms-def)
  by presburger
qed
3.7 Polynomial roots on the upper half-plane

— Roots counted WITH multiplicity

**definition** proots-upper :: complex poly ⇒ nat where
proots-upper p = proots-count p {z. Im z > 0}

— Roots counted WITHOUT multiplicity

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

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

moreover have ∀ F x in at-top. f x = g x
  unfolding f-def g-def using (a≠0)
  apply (subst Im-Ln-eq)
  subgoal for x using Complex-eq-0 by blast
  subgoal unfolding eventually-at-top-linorder by auto
  done
  ultimately show ?thesis
  using tendsto-cong[of f g at-top] unfolding g-def by auto
next
  case True
show ?thesis
  apply (rule tendsto-eventually)
  apply (rule eventually-at-top-linorderI[of 1])
  using True by (subst Im-Ln-eq, auto simp add: Complex-eq-0)
qed

lemma Im-Ln-tendsto-at-bot: ((λx. Im (Ln (Complex a x))) ----> - pi/2) at-bot

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

moreover have ∀ F x in at-bot. f x = g x
  unfolding f-def g-def using (a≠0)
  apply (subst Im-Ln-eq)
  subgoal for x using Complex-eq-0 by blast
  subgoal unfolding eventually-at-bot-linorder by (auto intro:exI[where x=-1])
  done
  ultimately show ?thesis
    using tendsto-cong[of f g at-bot] unfolding g-def by auto
next
  case True
  show ?thesis
    apply (rule tendsto-eventually)

76
apply \((\text{rule eventually-at-hot-linorderI[of } -1])\)
using True by (subst Im-Ln-eq,auto simp add:Complex-eq-0)
qed

lemma Re-winding-number-tendsto-part-circlepath:
  shows \((\lambda r. \text{Re} (\text{winding-number (part-circlepath } z_0 r 0 \pi ) a)) \longrightarrow 1/2 )\)
at-top
proof (cases Im z_0\leq Im a)
case True
  define g_1 where g_1=(\lambda r. \text{part-circlepath } z_0 r 0 \pi)
  define g_2 where g_2=(\lambda r. \text{part-circlepath } z_0 r \pi (2*\pi))
  define f_1 where f_1=(\lambda r. \text{Re (winding-number (g_1 r ) a)})
  define f_2 where f_2=(\lambda r. \text{Re (winding-number (g_2 r ) a)})
  have \((f_2 \longrightarrow 1/2 )\) at-top
  proof
  define h_1 where h_1=(\lambda r. \text{Im (Ln (Complex (Im a - Im z_0) (Re z_0 - Re a + r)))})
  define h_2 where h_2=(\lambda r. \text{Im (Ln (Complex (Im a - Im z_0) (Re z_0 - Re a - r))))})
  have \(\forall x \text{ in at-top. } f_2 x = (h_1 x - h_2 x) / (2 * \pi)\)
  proof (rule eventually-at-top-linorderI[of cmod (a-z_0) + 1])
  fix r assume asm: \(r \geq \text{cmod (a-z_0) + 1}\)
  have Im p \leq Im a when p\in\text{path-image (g_2 r) for p}
  proof
  obtain t where p-def:p:=z_0 + of-real r * exp (i * of-real t) and pi\leq t
  t\leq2*pi
  using \(p\in\text{path-image (g_2 r)}\);
  unfolding g_2-def path-image-part-circlepath[of pi 2*pi,simplified]
  by auto
  then have Im p:=Im z_0 + sin t * r by (auto simp add:Im-exp)
  also have ... \(\leq Im z_0\)
  proof
  have sin t\leq0 using (pi\leq t) (t\leq2*pi) 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 ... \(\leq Im a\) using True .
  finally show ?thesis .
  qed
  moreover have valid-path (g_2 r) unfolding g_2-def by auto
  moreover have a \notin\ text{path-image (g_2 r)}
  unfolding g_2-def
  apply (rule not-on-circlepathI)
  using asm by auto
  moreover have [symmetric]:Im (Ln (i * pathfinish (g_2 r) - i * a)) = h_1 r
  unfolding h_1-def g_2-def

77
apply (simp only: pathfinish-pathstart-partcirclepath-simps)
apply (subst (4 10) complex-eq)
by (auto simp add: algebra-simps Complex-eq)

moreover have [symmetric]: \( \text{Im} \left( \text{Ln} (i * \text{pathstart} \ (g2 \ r) - i * a) \right) = h2 \ r \)
unfolding h2-def g2-def
apply (simp only: pathfinish-pathstart-partcirclepath-simps)
apply (subst (4 10) complex-eq)
by (auto simp add: algebra-simps Complex-eq)

ultimately show \( f2 \ r = (h1 \ r - h2 \ r) / (2 * \pi) \)
unfolding f2-def
apply (subst Re-winding-number-half-lower)
by (auto simp add: exp-Euler algebra-simps)

qed

moreover have \( \forall F \ r \in \text{at-top} \ . \ f2 \ r = 1 - f1 \ r \)
proof –
have \( h1 \longrightarrow \pi / 2 \) at-top
unfolding h1-def
apply (subst filterlim-at-top-linear-iff[of 1 - Re a - Re z0, simplified, symmetric])

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

using \( \text{Im-Ln-tendsto-at-bot} \) by (simp del: Complex-eq)
ultimately have \( (\lambda x. \ (h1 \ x - h2 \ x) / (2 * \pi)) \longrightarrow 1 / 2 \ ) \ at-top
then show \( \neg \)thesis
by (auto intro: tendsto-eq-intros)

qed

ultimately show \( \neg \)thesis by (auto dest: tendsto-cong)

qed
have winding-number (circlepath z0 r) a = 1
apply (rule winding-number-circlepath)
using asm by auto
then show ?thesis by auto
qed
finally have f1 r + f2 r = 1 .
then show f2 r = 1 - f1 r by auto
qed
ultimately have ((λr. 1 - f1 r) ----> 1/2) at-top
using tendsto-cong[of f2 λr. 1 - f1 r at-top] by auto
then have (f1 ----> 1/2) at-top
apply (rule-tac tendsto-minus-cancel)
apply (subst tendsto-add-const-iff[of 1,symmetric])
by auto
then show ?thesis unfolding f1-def g1-def by auto
next
case False
define g where g = (λr. part-circlepath z0 r 0 pi)
define f where f = (λr. Re (winding-number (g r) a))
have (f ----> 1/2) at-top
proof
define h1 where h1 = (λr. Im (Ln (Complex (Im z0 - Im a) (Re a - Re z0 + r))))
define h2 where h2 = (λr. Im (Ln (Complex (Im z0 - Im a) (Re a - Re z0 - r))))
have ∀ p x in at-top. f x = (h1 x - h2 x) / (2 * pi)
proof (rule eventually-at-top-inorder[of cmod (a - z0) + 1])
fix r assume asm: r ≥ cmod (a - z0) + 1
have Im p ≥ Im a when p ∈ path-image (g r) for p
proof
obtain t where p-def:p = z0 + of-real r * exp (i * of-real t) and 0 ≤ t ≤ pi
using (p ∈ path-image (g r))
unfolding g-def path-image-part-circlepath[of 0 pi,simplified]
by auto
then have Im p = Im z0 + sin t * r by (auto simp add:Im-exp)
moreover have sin t * r ≥ 0
proof
have sin t ≥ 0 using (0 ≤ t) (t ≤ pi) sin-ge-zero by fastforce
moreover have r ≥ 0
using asm by (metis add.inverse-inverse add.left-neutral add-uminus-conv-diff
diff-ge-0-iff-ge norm-ge-zero order-trans zero-le-one)
ultimately have sin t * r ≥ 0 by simp
then show ?thesis by auto
qed
ultimately show ?thesis using False by auto
qed
moreover have valid-path (g r) unfolding g-def by auto
moreover have a ∉ path-image (g r)
unfolding g-def
apply (rule not-on-circlepath1)
using asm by auto
moreover have [symmetric]: Im (Ln (i * a − i * pathfinish (g r))) = h1 r
  unfolding h1-def g-def
  apply (simp only: pathfinish-pathstart-partcirclepath-simps)
  apply (subst (4 9) complex-eq)
  by (auto simp add: algebra-simps Complex-eq)
moreover have [symmetric]: Im (Ln (i * a − i * pathstart (g r))) = h2 r
  unfolding h2-def g-def
  apply (simp only: pathfinish-pathstart-partcirclepath-simps)
  apply (subst (4 9) complex-eq)
  by (auto simp add: algebra-simps Complex-eq)
ultimately show f r = (h1 r − h2 r) /
  (2 * pi)
  unfolding f-def
  apply (subst Re-winding-number-half-upper)
  by (auto simp add: exp-Euler algebra-simps)
qed
moreover have ((λx. (h1 x − h2 x)) /
  (2 * pi)) −→ 1 / 2
proof
  have (h1 −→ pi/2) at-top
  unfolding h1-def
  apply (subst filterlim-at-top-linear-iff[of 1 - − Re a + Re z0 , simplified, symmetric])
  using Im-Ln-tendsto-at-top by (simp del: Complex-eq)
moreover have (h2 −→ − pi/2) at-top
  unfolding h2-def
  apply (subst filterlim-at-bot-linear-iff[of − 1 - Re a - Re z0 , simplified, symmetric])
  using Im-Ln-tendsto-at-bot by (simp del: Complex-eq)
ultimately have ((λx. h1 x - h2 x) −→ pi) at-top
  by (auto intro: tendsto-eq-intros)
then show ‹thesis›
  by (auto intro: tendsto-eq-intros)
qed
ultimately show ‹thesis› by (auto dest: tendsto-cong)
qed
then show ‹thesis› unfolding f-def g-def by auto
qed

lemma not-image-at-top-poly-part-circlepath:
assumes degree p>0
shows ∀ F r 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 and R-ball:proots (p [-[b]:]) ⊆ ball z0 R by auto
  show ‹thesis›
proof (rule eventually-at-top-linorderI[of R])
fix r assume r≥R
show b /∈ path-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 ∈ proots (p - [;b;]) unfolding proots-def by auto
then have x ∈ ball z0 r using R-ball ⟨r≥R⟩ by auto
then have cmod (x - z0) < r
  by (simp add: dist-commute dist-norm)
moreover have cmod (x - z0) = r
  using asm(2) in-path-image-part-circlepath ⟨R≥0⟩ ⟨r≥R⟩ by auto
ultimately show False by auto
qed
qed

lemma not-image-poly-part-circlepath:
assumes degree p>0
shows ∃ r>0. b /∈ path-image (poly p o part-circlepath z0 r st tt)
proof –
  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;]) ⊆ ball z0 r by auto
  have b /∈ path-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 ∈ proots (p - [;b;]) unfolding proots-def by auto
    then have x ∈ ball 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 ⟨R≥0⟩ ⟨r≥R⟩ by auto
    ultimately show False by auto
  qed
then show ?thesis using ⟨r>0⟩ by blast
qed

lemma Re-winding-number-poly-part-circlepath:
assumes degree p>0
shows (∀ r. Re (winding-number (poly p o part-circlepath z0 r 0 pi) 0)) ——→ degree p/2 ) at-top
using assms
proof (induct rule:poly-root-induct-alt)
case 0
then show ?case by auto
next
case (no-proots p)
  then have False
  using Fundamental-Theorem-Algebra.fundamental-theorem-of-algebra constant-degree
  neg0-conv
    by blast
  then show ?case by auto
next
case (root a p)
define g where g = (\(\lambda r.\) part-circlepath z0 r 0 pi)
define q where q = [:- a, 1:] * p
define w where w = (\(\lambda r.\) winding-number (poly q \circ g r) 0)
  have ?case when degree p = 0
  proof
    obtain pc where pc-def: p = [pc:] using (degree p = 0: degree-eq-zeroE by blast
    then have pc \neq 0 using root(2) by auto
    have \(\forall r \in\) at-top. Re (w r) = Re (winding-number (g r) a)
    proof (rule eventually-at-top-linorderI[of cmod ((pc * a) / pc - z0) + 1])
      fix r::real assume asm: cmod ((pc * a) / pc - z0) + 1 \leq r
      have w r = winding-number ((\(\lambda x.\) pc*x - pc+a) \circ (g r)) 0
      unfolding w-def pc-def g-def
      apply auto
      by (metis (no-types, hide-lams) add.right-neutral mult.commute mult-zero-right
        poly-0 poly-pCons uminus-add-conv-diff)
      also have .. = winding-number (g r) a
      apply (subst winding-number-comp-linear[where b=-pc*a,simplified])
      subgoal using (:pc\neq0) .
      subgoal unfolding g-def by auto
      subgoal unfolding g-def
      apply (rule not-on-circlepathI)
      using asm by auto
      subgoal using :pc\neq0 by (auto simp add:field-simps)
      done
      finally have w r = winding-number (g r) a .
      then show Re (w r) = Re (winding-number (g r) a) by simp
    qed
  moreover have ((\(\lambda r.\) Re (winding-number (g r) a)) \longrightarrow 1/2) at-top
    using Re-winding-number-tendsto-part-circlepath unfolding g-def by auto
  ultimately have ((\(\lambda r.\) Re (w r)) \longrightarrow 1/2) at-top
    by (auto dest!:tendsto-cong)
  moreover have degree ([:- a, 1:] * p) = 1 unfolding pc-def using :pc\neq0
    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 \(\forall r \in\) at-top. 0 \notin path-image (poly q \circ g r)
    unfolding g-def

82
apply (rule not-image-at-top-poly-part-circlepath)
unfolding q-def using root.prems by blast
then have \( \forall r \in \text{at-top}. \Re (w r) = \Re (\text{winding-number} (g r) a) + \Re (\text{winding-number} (\text{poly} p \circ g r) 0) \)
proof (rule eventually-mono)
fix r assume asm:0 \( \notin \) path-image (poly q \circ g r)
define cc where cc= 1 / (of-real (2 * pi) * i)
define pf where pf=(\(\lambda x\). deriv (poly p) w / poly p w)
define af where af=(\(\lambda x\). 1/(w−a))
have w r = cc * contour-integral (g r) (\(\lambda x\). deriv (poly q) w / poly q w)
unfolding w-def
apply (subst winding-number-comp[of UNIV,simplified])
using asm unfolding q-def cc-def by auto
also have ... = cc * contour-integral (g r) (\(\lambda x\). deriv (poly p) w / poly p w
+ 1/(w−a))
proof –
have contour-integral (g r) (\(\lambda x\). deriv (poly q) w / poly q w)
= contour-integral (g r) (\(\lambda x\). deriv (poly p) w / poly p w + 1/(w−a))
proof (rule contour-integral-eq)
fix x assume x \( \in \) path-image (g r)
have deriv (poly q) x = deriv (poly p) x * (x−a) + poly p x
proof –
have poly q = (\(\lambda x\). (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 \(\lambda x\). x− a - poly p])
by (auto intro: derivative-intros)
qed
moreover have poly p x\(\neq\)0 \(\land\) x−a\(\neq\)0
proof (rule ccontr)
assume \(\neg\) (poly p x \(\neq\) 0 \(\land\) x − a \(\neq\) 0)
then have poly q x=0 unfolding q-def by auto
then have 0\(\in\)poly q \(\circ\) path-image (g r)
using (x \(\in\) path-image (g r)) by auto
then show False using (\(\notin\) \(\in\) path-image (poly q \circ g 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) (\(\lambda x\). deriv (poly p) w / poly p w
+ cc * contour-integral (g r) (\(\lambda x\). 1/(w−a))
proof (subst contour-integral-add)
have continuous-on (path-image (g r)) (\(\lambda x\). deriv (poly p) w)
unfolding deriv-pderiv by (intro continuous-intros)
moreover have ∀ w∈path-image (g r). poly p w ≠ 0
using asm unfolding q-def path-image-compose by auto
ultimately show (λw. deriv (poly p) w / poly p w) contour-integrable-on

unfolding g-def
by (auto intro: contour-integrable-continuous-part-circlepath continuous-intros)
show (λw. 1 / (w − a)) contour-integrable-on g r
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 ...

proof –
have winding-number (poly p o g r) 0
= cc * contour-integral (g r) (λw. deriv (poly p) w / poly p w)
apply (subst winding-number-comp[of UNIV,simplified])
using (0 ∉ path-image (poly q o g r)) unfolding path-image-compose q-def
g-def cc-def
by auto
moreover have winding-number (g r) a = cc * contour-integral (g r) (λw. 1/(w−a))
apply (subst winding-number-valid-path)
using (0 ∉ path-image (poly q o g r)) unfolding path-image-compose q-def

ultimately show ?thesis by auto

qed
also have ...

proof –
have winding-number (poly p o g r) 0
= cc * contour-integral (g r) (λw. deriv (poly p) w / poly p w)
apply (subst winding-number-comp[of UNIV,simplified])
using (0 ∉ path-image (poly q o g r)) unfolding path-image-compose q-def
g-def cc-def
by auto
moreover have (λr. Re (winding-number (g r) a)
+ Re (winding-number (poly p o g r) 0)) ----> degree q / 2) at-top

proof –
have ((λr. Re (winding-number (g r) a)) ----> 1 / 2) at-top

ultimately show ?thesis by (rule Re-winding-number-tendsto-part-circlepath)
moreover have ((λr. Re (winding-number (poly p o g r) 0)) ----> degree p / 2) at-top

ultimately have (λr. Re (w r)) ----> 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 ≡ (λr. poly pp o linepath (−r) (of-real r))
  assumes lead-coeff pp=1 and no-real-zero:∀ x∈proots pp. Im x≠0
  shows ((λr. 2*Re (winding-number (g r) 0) + cindex-pathE (g r) 0 ) −−→ 0
   ) at-top
proof –
  define p where p=map-poly Re pp
define q where q=map-poly Im pp
define f where f=(λt. 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 (g r) for r
proof (rule ccontr)
  assume ¬ 0 ∉ path-image (g r)
  then obtain x where poly pp x=0 x∈closed-segment (− of-real r) (of-real r)
    unfolding g-def path-image-compose of-real-linepath by auto
  then have Im x=0 x∈proots pp
    using closed-segment-imp-Re-Im(2) unfolding proots-def by fastforce+
  then show False using (∀ x∈proots pp. Im x≠0) by auto
  qed
  have arctan-f-tendsto:((λr. (arctan (f r) − arctan (f (−r))) / πi) −−→ 0)
    at-top
proof (cases degree p>0)
  case True
  have degree p>degree q
    proof –
    have degree p=degree pp
      unfolding p-def using (lead-coeff pp=1)
      by (auto intro:map-poly-degree-eq)
    moreover then have degree q<degree pp
      unfolding q-def using (lead-coeff pp=1) True
      by (auto intro!:map-poly-degree-less)
    ultimately show ?thesis by auto
    qed
  then have (f −−→ 0) at-infinity
    unfolding f-def using poly-divide-tendsto-0-at-infinity by auto
  then have (f −−→ 0) at-bot (f −−→ 0) at-top
    by (auto elim!:filterlim-mono simp add:at-top-le-at-infinity at-bot-le-at-infinity)
  then have ((λr. arctan (f r)) −−→ 0) at-top ((λr. arctan (f (−r))) −−→ 0)
    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 simp add:at-bot-mirror[symmetric])

next

  case False

  obtain c where f=(λr. c)

  proof
    −
      have degree p=0 using False by auto
      moreover have degree q≤degree p
      proof
        −
          have degree p=degree pp
            unfolding p-def using ⟨lead-coeff pp=1⟩
            by (auto intro:map-poly-degree-eq)
          moreover have degree q≤degree pp
            unfolding q-def by simp
          ultimately show ?thesis by auto
        qed
    qed ultimately have degree q=0 by simp
    then obtain pa qa where p=[:pa:] q=[:qa:]
      using ⟨degree p=0⟩ by (meson degree-eq-zeroE)
    then show ?thesis using that unfolding f-def by auto
    qed
  then show ?thesis by auto
  qed

have [simp]:valid-path (g r) path (g r) finite-ReZ-segments (g r) 0 for r

proof −
  show valid-path (g r) unfolding g-def
    apply (rule valid-path-compose-holomorphic[where S=UNIV])
    by (auto simp add:of-real-linepath)
  then show path (g r) using valid-path-imp-path by auto
  show finite-ReZ-segments (g r) 0
    unfolding g-def of-real-linepath using finite-ReZ-segments-poly-linepath by simp
  qed

have g-f-eq:Im (g r t) / Re (g r t) = (f o (λx. 2*r*x - r)) t for r t

proof −
  have Im (g r t) / Re (g r t) = Im ((poly pp o of-real o (λx. 2*r*x - r)) t)
    / Re ((poly pp o of-real o (λx. 2*r*x - r)) t)
    unfolding g-def linepath-def comp-def
    by (auto simp add:algebra-simps)
  also have ... = (f o (λx. 2*r*x - r)) t
  unfolding comp-def
    by (simp only:Im-poly-of-real diff-0-right Re-poly-of-real f-def q-def p-def)
  finally show ?thesis .
  qed

86
have \( \forall \) \( r \in \text{at-top} \). 2 \* Re (winding-number (\( g \) \( r \)) 0) + cindex-pathE (\( g \) \( r \)) 0 = (arctan (\( f \) \( r \)) - arctan (\( f \) (-\( r \)))) / \pi

proof (rule eventually-at-top-linorder[af 1])

fix \( r :: \text{real} \)

have image-pos: \( \forall p \in \text{path-image} (\( g \) \( r \)). 0 < Re p \)

proof

(\( \text{rule eventually-at-top-linorderI[of 1]} \))

fix \( r :: \text{real} \)

assume \( r \geq 1 \)

have \( \forall p \in \text{path-image} (\( g \) \( r \)). 0 < Re p \)

proof

(\( \text{rule ccontr} \))

assume \( \neg (\forall p \in \text{path-image} (\( g \) \( r \)). 0 < Re p) \)

then obtain \( t \) where \( \text{poly} p \leq 0 \)

unfolding \( g \)-def \( \text{path-image-compose} \) \( \text{of-real-linepath} \) \( p \)-def

using \( \text{Re-poly-of-real} \)

apply (simp add: \( \text{closed-segment-def} \))

by (metis not-less \( \text{of-real-def} \) \( \text{real-vector} \).

moreover have \( \text{False} \) when \( \text{poly} p \leq 0 \)

proof

(\( \text{have} \ sgnx (\( \text{poly} p \)) (\text{at-right} \( t \)) = -1 \))

using \( sgnx-poly-nz \) that

then obtain \( x \) where \( x > t \) \( \text{poly} p \leq 0 \)

using \( sgnx-at-top-IVT[\text{of} \( p \) \( t \)] \) \( sgnx-top \) by auto

then have \( x \in \text{proots} p \)

unfolding \( \text{proots-def} \)

by auto

qed

moreover have \( \text{False} \) when \( \text{poly} p = 0 \)

proof

(\( \text{have} \ sgnx (\( \text{poly} p \)) (\text{at-right} \( t \)) = 0 \))

using \( sgnx-poly-nz \) that

then obtain \( x \) where \( x > t \) \( \text{poly} p = 0 \)

using \( sgnx-at-top-IVT[\text{of} \( p \) \( t \)] \) \( sgnx-top \) by auto

then have \( x \in \text{proots} p \)

unfolding \( \text{proots-def} \) by auto

then show \( \text{False} \) using \( (\text{proots} p = 0) \) by auto

qed

ultimately show \( \text{False} \) by linarith

qed

have \( \text{Re} (\text{winding-number} (\( g \) \( r \)) 0) = (\text{Im} (\text{Ln} (\text{pathfinish} (\( g \) \( r \)))) - \text{Im} (\text{Ln} (\text{pathstart} (\( g \) \( r \)))) \)/ (2 \* \( \pi \)) \)

apply (rule \( \text{Re-winding-number-half-right[of} \( g \) \( r \) 0,simplified} \))

subgoal by \( \text{image-pos by auto} \)

subgoal by (auto simp add: \( \text{not-g-image} \))

done

also have \( ... = (\text{arctan} (\( f \) \( r \)) - \text{arctan} (\( f \) (-\( r \))))/(2 \* \( \pi \)) \)

proof

(\( \text{have} \ \text{Im} (\text{Ln} (\text{pathfinish} (\( g \) \( r \)))) = \text{arctan} (\( f \) \( r \)) \))

proof

(\( \text{have} \ \text{Re} (\text{pathfinish} (\( g \) \( r \))) > 0 \))

by (auto intro: \( \text{image-pos[rule-format]} \))

then have \( \text{Im} (\text{Ln} (\text{pathfinish} (\( g \) \( r \)))) = \text{arctan} (\text{Im} (\text{pathfinish} (\( g \) \( r \)) \)/ \text{Re} (\text{pathfinish} (\( g \) \( r \))))) \)

by (subst \( \text{Im-Ln-eq,auto} \))

also have \( ... = \text{arctan} (\( f \) \( r \)) \)

unfolding \( \text{path-defs} \) by (subst \( g-f-eq,auto \))

finally show \( \text{thesis} \).

qed
moreover have \( \text{Im} \ (L \ (\text{pathstart} \ (g \ r))) = \arctan \ (f \ (-r)) \)

proof –
  have \( \text{Re} \ (\text{pathstart} \ (g \ r)) > 0 \)
    by (auto intro: image-pos[rule-format])
  then have \( \text{Im} \ (L \ (\text{pathstart} \ (g \ r))) = \arctan \ (\text{Im} \ (\text{pathstart} \ (g \ r)) / \text{Re} \ (\text{pathstart} \ (g \ r))) \)
    by (subst Im-Ln-eq)auto
  also have \( \ldots = \arctan \ (f \ (-r)) \)
    unfolding path-defs by (subst g-f-eq)auto
  finally show \( \text{thesis} \).

qed
ultimately show \( ?\text{thesis} \) by auto

finally have \( \text{cindex-pathE} \ (g \ r) 0 = 0 \)

proof –
  have \( \text{cindex-pathE} \ (g \ r) 0 = \text{cindex-pathE} \ (\text{poly pp o of-real o} \ (\lambda x. 2*r*x - r)) 0 \)
    unfolding g-def linepath-def comp-def
    by (auto simp add: algebra-simps)
  also have \( \ldots = \text{cindexE} \ 0 \ 1 \ (f \ o \ (\lambda x. 2*r*x - r)) \)
    unfolding cindex-pathE-def comp-def
    by (simp only: Re-poly-of-real f-def q-def p-def)
  also have \( \ldots = \text{cindexE} \ (-r) \ r f \)
    apply (subst cindexE-linear-comp[of 2*r 0 1 - -r, simplified])
    using \( r \geq 1 \) by auto
  also have \( \ldots = 0 \)
  proof –
    have \( \text{jumpF} \ f \ (\text{at-left} \ x) \ = 0 \ \text{jumpF} \ f \ (\text{at-right} \ x) \ = 0 \ \text{when} \ x \in \{-r..r\} \)
      for \( x \)
        proof –
          have \( \text{poly} \ p \ x \neq 0 \) using \( \{ \text{proots} \ p = \{ \} \) unfolding proots-def by auto
          then show \( \text{jumpF} \ f \ (\text{at-left} \ x) \ = 0 \ \text{jumpF} \ f \ (\text{at-right} \ x) \ = 0 \ \text{when} \ x \in \{-r..r\} \)
            using \( r \geq 1 \) by auto
        qed
      finally have \( ?\text{thesis} \) unfolding cindexE-def by auto
  qed
  finally show \( ?\text{thesis} \).

qed
ultimately show \( 2 * \text{Re} \ (\text{winding-number} \ (g \ r) 0) + \text{cindex-pathE} \ (g \ r) 0 \)
  = \( \arctan \ (f \ r) - \arctan \ (f \ (-r)) \) / \( \pi \)
  unfolding path-defs by (auto simp add: field-simps)

qed
with \( \arctan-f\text{-tendsto} \) show \( ?\text{thesis} \) by (auto dest: tendsto-cong)

moreover have \( ?\text{thesis} \) when \( \text{proots} \ p \neq \{ \} \)

proof –
  define \( \text{max-r} \) where \( \text{max-r} = \text{Max} \ (\text{proots} \ p) \)
define min-r where min-r = \text{Min} (\text{proots } p)

have max-r ∈ \text{proots } p \text{ } \text{min-r ≤ max-r} \text{ and }\n\text{min-max-bound} : \forall p ∈ \text{proots } p, p ∈ \{min-r..max-r\}

proof -
  have p ≠ 0
  proof -
    have (0::real) ≠ 1
      by simp
      then show ?thesis
      by (metis (full-types) \( p \equiv \text{map-poly Re pp} \) \( \text{coeff-0 coeff-map-poly one-complex..simps(1)} \) zero-complex.sel(1))
  qed
  then have finite (\text{proots } p) by auto
  then show max-r ∈ \text{proots } p min-r ∈ \text{proots } p\text{ min-r ≤ max-r}
    using Min-in Max-in that unfolding \( \text{max-r-def min-r-def} \) by fast+
    then show ∀ p ∈ \text{proots } p.\ p ∈ \{\text{min-r..max-r}\}
      using Min-le Max-ge \( \text{finite (proots } p) \) unfolding \( \text{max-r-def min-r-def} \) by auto
    then show \( \text{min-r ≤ max-r} \) using \( \text{max-r ∈ proots } p \) by auto
  qed

have ∀ F r in at-top.\ 2 * Re (winding-number (g r) 0) + cindex-pathE (g r) 0
  = (\text{arctan (f r)} − \text{arctan (f (-r))}) / pi
proof (rule eventually-at-top-linorderI [of max (norm max-r) (norm min-r) + 1])
  fix r assume r-asm: max (norm max-r) (norm min-r) + 1 ≤ r
  then have r ≠ 0 min-r ≥ -r max-r ≤ r by auto
  define u where u = (min-r + r) / (2*r)
  define v where v = (max-r + r) / (2*r)
  have uv: u ∈ \{0..1\} v ∈ \{0..1\} u ≤ v
    unfolding u-def v-def using r-asm \( \text{min-r ≤ max-r} \)
    by (auto simp add: field-simps)
  define g1 where g1 = subpath 0 u (g r)
  define g2 where g2 = subpath u v (g r)
  define g3 where g3 = subpath v 1 (g r)
  have path g1 path g2 path g3 valid-path g1 valid-path g2 valid-path g3
    unfolding g1-def g2-def g3-def using uv
    by (auto intro!: path-subpath valid-path-subpath)
  define wc-add where wc-add = (λg. 2 * Re (winding-number g 0) + cindex-pathE g 0)
  have wc-add (g r) = wc-add g1 + wc-add g2 + wc-add g3
  proof -
    have winding-number (g r) 0 = winding-number g1 0 + winding-number g2 0 + winding-number g3 0
      unfolding g1-def g2-def g3-def using (u ∈ \{0..1\}) (v ∈ \{0..1\}) not-g-image
      by (subt winding-number-subpath-combine simp-all)+
    moreover have cindex-pathE (g r) 0 = cindex-pathE g1 0 + cindex-pathE g2 0 + cindex-pathE g3 0
      unfolding g1-def g2-def g3-def using (u ∈ \{0..1\}) (v ∈ \{0..1\}) (u ≤ v)
      not-g-image
  qed
ultimately show "thesis unfolding wc-add-def by auto qed
moreover have wc-add g2 = 0
proof
  have 2 * Re (winding-number g2 0) = - cindex-pathE g2 0
    unfolding g2-def
    apply (rule winding-number-cindex-pathE-aux)
    subgoal using wv by (auto intro:finite-ReZ-segments-subpath)
    subgoal using wv by (auto intro:valid-path-subpath)
    subgoal using Path-Connected.path-image-subpath-subset (\A 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)
    done
    then show "thesis unfolding wc-add-def by auto qed
moreover have wc-add g1 = - arctan (f (-r)) / pi
proof
  have g1-pq:
    Re (g1 t) = poly p (min-r*t+r*t-r)
    Im (g1 t) = poly q (min-r*t+r*t-r)
    Im (g1 t)/Re (g1 t) = (f o (\lambda x. (min-r+r)*x - r)) t
    for t
    proof
      have g1 t = poly pp (of-real (min-r*t+r*t-r))
        using (r\neq 0) unfolding g1-def g-def linepath-def subpath-def u-def p-def
        by (auto simp add:field-simps)
      then show
        Re (g1 t) = poly p (min-r*t+r*t-r)
        Im (g1 t) = poly q (min-r*t+r*t-r)
        unfolding p-def q-def
        by (simp only:Re-poly-of-real Im-poly-of-real)+
      then show Im (g1 t)/Re (g1 t) = (f o (\lambda x. (min-r+r)*x - r)) t
        unfolding f-def by (auto simp add:algebra-simps)
    qed
  have Re(g1 1) = 0
    using (r\neq 0) Re-poly-of-real (min-r\in proots p)
    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 (\A r. path (g r))
      atLeastAtMost iff g1-def le-less not-g-image subsetCE uv(1) zero-le-one)
have \( wc-add-pos : wc-add \ h = - \arctan \ (\text{poly} \ q \ (-r) / \text{poly} \ p \ (-r)) / \pi \)
when
\[
\begin{align*}
\text{Re-pos} &: \forall x \in \{0..<1\}, \ 0 < (\text{Re} \circ \ h) x \\
\text{hp} &: \forall t. \ \text{Re} \ (h \ t) = c*\text{poly} \ p \ (\min-r*t+r*t-r) \\
\text{hq} &: \forall t. \ \text{Im} \ (h \ t) = c*\text{poly} \ q \ (\min-r*t+r*t-r) \\
\text{[simp]} &: c \neq 0 \\
\end{align*}
\]
and \( \text{Re} \ (h \ 1) = 0 \)
and \( \text{valid-path} \ h \)
and \( \text{h-img} \ : 0 \notin \text{path-image} \ h \)
for \( h \ c \)

proof –

define \( f \) where \( f = (\lambda t. \ c*\text{poly} \ q \ t / (c*\text{poly} \ p \ t)) \)

define \( \text{farg} \) where \( \text{farg} = (\text{if} \ 0 < \text{Im} \ (h \ 1) \ \text{then} \ pi / 2 \ \text{else} -pi / 2) \)

have \( \text{Re} \ (\text{winding-number} \ h \ 0) = (\text{Im} \ (\text{Ln} \ (\text{pathfinish} \ h))) \)
\( - (\text{Im} \ (\text{Ln} \ (\text{pathstart} \ h)))) / (2*\pi) \)

apply (rule Re-winding-number-half-right[of h 0,simplified])

subgoal using that \( \text{Re} \ (h \ 1) = 0 \) unfolding path-image-def
by \( (\text{auto simp add:le-less}) \)

subgoal using \( \text{valid-path} \ h \).

subgoal using \( \text{h-img} \).
done

also have \( ... = (\text{farg} - \arctan \ (f \ (-r))) / (2 * \pi) \)

proof –

have \( \text{Im} \ (\text{Ln} \ (\text{pathfinish} \ h)) = \text{farg} \)
using \( \text{Re}(h \ 1) = 0 \) unfolding farg-def path-defs
apply (subst Im-Ln-eq)

subgoal using \( \text{h-img} \) unfolding path-defs by fastforce

subgoal by simp
done

moreover have \( \text{Im} \ (\text{Ln} \ (\text{pathstart} \ h)) = \arctan \ (f \ (-r)) \)

proof –

have \( \text{pathstart} \ h \neq 0 \)
using \( \text{h-img} \)
by \( (\text{metis pathstart-in-path-image}) \)
then have \( \text{Im} \ (\text{Ln} \ (\text{pathstart} \ h)) = \arctan \ (\text{Im} \ (\text{pathstart} \ h) / \text{Re} \ (\text{pathstart} \ h)) \)
using \( \text{Re-pos}[rule-format,of 0] \)
by \( (\text{simp add: \text{Im-Ln-eq path-defs}) \)
also have \( ... = \arctan \ (f \ (-r)) \)
unfolding f-def path-defs \( \text{hp}[rule-format] \ \text{hq}[rule-format] \)
by simp
finally show \( \text{thesis} \).

qed

ultimately show \( \text{thesis} \) by auto

qed

finally have \( \text{Re} \ (\text{winding-number} \ h \ 0) = (\text{farg} - \arctan \ (f \ (-r))) / (2 * \pi) \).

moreover have \( \text{cindex-pathE} \ h \ 0 = (-\text{farg}/\pi) \)
proof
  have cindex-pathE h 0 = cindexE 0 1 \( f \circ (\lambda x. (\text{min}-r + r) \cdot x - r) \)
    unfolding cindex-pathE-def using \( c \neq 0 \)
    by (auto simp add:hp hq f-def comp-def algebra-simps)
  also have \( \ldots \) = cindexE \( -r \) \( \text{min}-r \) \( f \)
    apply (sub: cindexE-linear-comp[where \( b=-r \),simplified])
    using r-asm by auto
  also have \( \ldots \) = \( -\text{jumpF} f \) \( \text{at-left} \) \( \text{min}-r \)
  proof
    define right where right = \( \{ x. \text{jumpF} f \) \( \text{at-right} \) \( x \) \( \neq 0 \) \& \( -r \leq x \) \& \( x < \text{min}-r \} \)
    define left where left = \( \{ x. \text{jumpF} f \) \( \text{at-left} \) \( x \) \( \neq 0 \) \& \( -r < x \) \& \( x \leq \text{min}-r \} \)
    have \( \star \): \text{jumpF} f \) \( \text{at-right} \) \( x \) = 0 \text{jumpF} f \) \( \text{at-left} \) \( x \) = 0
    when \( x \in \{-r..<\text{min}-r\} \) for \( x \)
    proof
      have False when \( \text{poly} p \) \( x =0 \)
      proof
        have \( x \geq \text{min}-r \)
          using \( \text{min-max-bound} \) [rule-format.of \( x \)] that by auto
        then show False using \( \{ x \in \{-r..<\text{min}-r\} \} \) by auto
        qed
      then show \text{jumpF} f \) \( \text{at-right} \) \( x \) = 0 \text{jumpF} f \) \( \text{at-left} \) \( x \) = 0
        unfolding f-def by (auto intro!::jumpF-not-infinity continuous-intros)
      qed
      then have right = {} \text{right-def by force}
      moreover have left = \( \{ \text{if} \text{jumpF} f \) \( \text{at-left} \) \( \text{min}-r \) = 0 \text{then} \) \( \{ \} \) \text{else} \( \{ \text{min}-r \}\) \text{left-def le-less using * r-asm by force}
      ultimately show \( \exists \text{thesis} \)
        unfolding cindexE-def by \( \{ \text{fold left-def right-def,auto} \}\)
      qed
      also have \( \ldots \) = \( -\text{farg}/\pi \)
      proof
        have \( \pi\text{-pos:} c*\text{poly} p \) \( x > 0 \) \text{when} \( x \in \{-r..<\text{min}-r\} \) for \( x \)
        proof
          define hh where hh = \( \lambda t. \text{min}-r*t+r*t-r \)
          have \( (x+r)/(\text{min}-r+r) \in \{0..<1\} \)
            using that r-asm by (auto simp add:field-simps)
          then have \( 0 < c*\text{poly} p \) \( \text{hh} \) (\( (x+r)/(\text{min}-r+r) \))
            apply (drule-tac Re-pos[rule-format])
          unfolding \text{comp-def hp rule-format} \( \text{hq} \) [rule-format] \text{hh-def} .
          moreover have \( \text{hh} \) (\( (x+r)/(\text{min}-r+r) \)) = \( x \)
            unfolding \text{hh-def using} \( \text{min}-r>-r \)
            apply (auto simp add:divide-simps)
            by (auto simp add:algebra-simps)
          ultimately show \( \exists \text{thesis} \) by simp
      qed
      qed
qed

have \( c \cdot \text{poly} q \text{ min-r} \neq 0 \)
  using no-real-zero \( (c \neq 0) \)
  by (metis Im-complex-of-real UNIV-I \( (\text{min-r} \in \text{proots p}) \cdot \text{cpoly-of-decompose} \)
  \( \text{mult-eq-0-iff p-def poly-cpoly-of-real-iff proots-within-iff q-def} \))

moreover have \( \text{thesis when } c \cdot \text{poly} q \text{ min-r} > 0 \)
proof -
  have \( 0 < \text{Im} (h 1) \) unfolding \( hq \text{[rule-format]} \) \( hp \text{[rule-format]} \) using that by auto
moreover have \( \text{jumpF f} (\text{at-left min-r}) = 1/2 \)
proof -
  have \( ((\lambda t. c \cdot \text{poly} p t) \text{ has-sgnx 1}) \) (at-left min-r)
    unfolding has-sgnx-def
    apply (rule eventually-at-leftI \([\text{of } -r]\))
    using p-pos \( (\text{min-r} > -r) \) by auto
  then have \( \text{filterlim f at-top} \) (at-left min-r)
    unfolding f-def
    apply (subst filterlim-divide-at-bot-at-top-iff \([\text{of } -c \cdot \text{poly} q \text{ min-r}]\))
    using that \( (\text{min-r} \in \text{proots p}) \) by (auto intro!: tendsto-eq-intros)
  then show \( \text{thesis unfolding jumpF-def by auto} \)
qed
ultimately show \( \text{thesis unfolding farg-def by auto} \)
qed

moreover have \( \text{thesis when } c \cdot \text{poly} q \text{ min-r} < 0 \)
proof -
  have \( 0 > \text{Im} (h 1) \) unfolding \( hq \text{[rule-format]} \) \( hp \text{[rule-format]} \) using that by auto
moreover have \( \text{jumpF f} (\text{at-left min-r}) = -1/2 \)
proof -
  have \( ((\lambda t. c \cdot \text{poly} p t) \text{ has-sgnx 1}) \) (at-left min-r)
    unfolding has-sgnx-def
    apply (rule eventually-at-leftI \([\text{of } -r]\))
    using p-pos \( (\text{min-r} > -r) \) by auto
  then have \( \text{filterlim f at-bot} \) (at-left min-r)
    unfolding f-def
    apply (subst filterlim-divide-at-bot-at-top-iff \([\text{of } -c \cdot \text{poly} q \text{ min-r}]\))
    using that \( (\text{min-r} \in \text{proots p}) \) by (auto intro!: tendsto-eq-intros)
  then show \( \text{thesis unfolding jumpF-def by auto} \)
qed
ultimately show \( \text{thesis unfolding farg-def by auto} \)
qed

ultimately show \( \text{thesis unfolding wc-add-def f-def by (auto simp} \)
have \( \forall x \in \{0..<1\}, (Re \circ g1) x \neq 0 \)

proof (rule econtr)
  assume \( \neg (\forall x \in \{0..<1\}, (Re \circ g1) x \neq 0) \)
  then obtain \( t \) where \( t \)-def: \( Re \circ (g1 \circ t) = 0 \ t \in \{0..<1\} \)
  unfolding path-image-def by fastforce

define \( m \) where \( m = \min-r \ast t + \ast r \ast t - \ast r \)

have \( poly p m = 0 \)

proof
  have \( Re \circ (g1 \circ t) = Re \circ (poly pp \circ (of-real m)) \)
  unfolding m-def g1-def g-def linepath-def subpath-def u-def
  by (auto simp add: algebra-simps)
then show \( ?thesis \) using \( t \)-def unfolding Re-poly-of-real p-def by auto
qed

moreover have \( m < \min-r \)

proof
  have \( \min-r + \ast r > 0 \) using \( r \)-asm by simp
then have \( (\min-r + \ast r) \ast (t - 1) < 0 \) 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

moreover have \( ?thesis \) when \( \forall x \in \{0..<1\}, 0 < (Re \circ g1) x \) \( \lor \forall x \in \{0..<1\}, (Re \circ g1) x < 0 \)

apply (elim continuous-on-neq-split)
using \( \{path g1\} \) unfolding path-def
by (auto intro!: continuous-intros elim: continuous-on-subset)

moreover have \( ?thesis \) when \( \forall x \in \{0..<1\}, (Re \circ g1) x < 0 \)

proof
  have \( wc-add \circ (uminus \circ g1) = - \arctan (f (- r)) / pi \)
  unfolding f-def
  apply (rule wc-add-pos[of - 1])
  using g1-pq that \( \min-r \in\text{proots p} \) \( \text{valid-path g1} \) \( 0 \notin \text{path-image g1} \)
  by (auto simp add: path-image-compose)
moreover have \( wc-add \circ (uminus \circ g1) = wc-add g1 \)
  unfolding wc-add-def cindex-pathE-def
  apply (subst winding-number-uminus-comp)
  using \( \text{valid-path g1} \) \( 0 \notin \text{path-image g1} \) by auto
ultimately show \( ?thesis \) by auto
qed

moreover have \( ?thesis \) when \( \forall x \in \{0..<1\}, (Re \circ g1) x > 0 \)

proof
  have \( wc-add \circ (uminus \circ g1) = - \arctan (f (- r)) / pi \)
  unfolding f-def
  apply (rule wc-add-pos[of - 1])
  using g1-pq that \( \min-r \in\text{proots p} \) \( \text{valid-path g1} \) \( 0 \notin \text{path-image g1} \)
  by (auto simp add: path-image-compose)

ultimately show False using min-max-bound unfolding proots-def by auto

ultimately show False using min-max-bound unfolding proots-def by auto

94
ultimately show thesis by blast
qed
moreover have wc-add g3 = arctan (f r) / pi
proof
have g3-pq:
  Re (g3 t) = poly p ((r - max-r) * t + max-r)
  Im (g3 t) = poly q ((r - max-r) * t + max-r)
  Im (g3 t) / Re (g3 t) = (f o (λx. (r - max-r) * x + max-r)) t
  for t
proof
  have g3 t = poly pp (of-real ((r - max-r) * t + max-r))
  using (r ≠ 0) ⟨max-r < r⟩ unfolding g3-def g-def linepath-def subpath-def
v-def p-def
  by (auto simp add: algebra-simps)
then show
  Re (g3 t) / Re (g3 t) = (f o (λx. (r - max-r) * x + max-r)) t
  unfolding f-def by (auto simp add: algebra-simps)
qed
have Re(g3 0) = 0
  using (r ≠ 0) Re-poly-of-real ⟨max-r ∈ proots p⟩
  unfolding g3-def subpath-def v-def g-def linepath-def
by (auto simp add: field-simps p-def)
have 0 ∉ path-image g3
proof
  have ⟨1::real⟩ ∈ {0..<1}
  by auto
then show thesis
  using Path-Connected.path-image-subpath-subset ⟨\ r. path (g r): g3-def
not-g-image uv ⟨2⟩⟩ by blast
qed

have wc-add-pos:wc-add h = arctan (poly q r / poly p r) / pi when
  Re-pos:∀ x∈{0..<1}. 0 < (Re o h) x
and hp:∀ t. Re (h t) = c * poly p ((r - max-r) * t + max-r)
and hq:∀ t. Im (h t) = c * poly q ((r - max-r) * t + max-r)
and [simp]:c ≠ 0

and Re (h 0) = 0
and valid-path h
and h-img:0 ∉ path-image h
for h c
proof
  define f where f = (λt. c * poly q t / (c * poly p t))
  define farg where farg = (if 0 < Im (h 0) then pi / 2 else - pi / 2)
have $\Re (\text{winding-number } h \ 0) = (\Im (\text{Ln (pathfinish } h)))$
  $-$ $\Im (\text{Ln (pathstart } h))) \div (2 \ast \pi)$
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 ...
  $\ast\ast (\arctan (f \ r) - \text{farg}) \div (2 \ast \pi)$
proof
  have $\Im (\text{Ln (pathstart } h)) = \text{farg}$
    using $\langle \Re (h \ 0) = 0 \rangle$
    unfolding farg-def path-defs
    apply (auto)
subgoal using $\langle \text{valid-path } h \rangle$.
proof
  have $\text{pathfinish } h \neq 0$
    using $\langle c \neq 0 \rangle$
    by (auto simp add: hp hq f-def comp-def algebra-simps)
also have ...
  $\ast\ast \text{jumpF } f$ (at-right $\max-r$)
proof
  define right where right $= \{x. \text{jumpF } f$ (at-right $x) \neq 0 \land \max-r \leq x \land x < r\}$
define left where left $= \{x. \text{jumpF } f$ (at-left $x) \neq 0 \land \max-r < x \land x \leq r\}$
  have $\ast\ast\ast \text{jumpF } f$ (at-right $x) = 0 \text{ jumpF } f$ (at-left $x) = 0$ when $x \in \{\max-r < \ldots r\}$ for $x$
proof 
  have False when poly p x = 0
  proof 
    have \( x \leq \text{max-r} \)
      using min-max-bound[rule-format, of x] that by auto
    then show False using \( \langle x \in \{\text{max-r} \leq \ldots \text{r} \} \rangle \) by auto
  qed
  then show \( \text{jumpF f \ (at-right} x) = 0 \) \( \text{jumpF f \ (at-left} x) = 0 \)
    unfolding f-def by (auto intro:jumpF-not-infinity continuous-intros)
  qed
  then have \( \text{left} = \{\} \)
    unfolding left-def by force
  moreover have \( \text{right} = (\text{if} \text{jumpF f \ (at-right} \text{max-r}) = 0 \then \{\} \else \{\text{max-r}\} \rangle \)
    unfolding right-def le-less using \( \ast\) r-asm by force
  ultimately show \( ?\text{thesis} \)
    unfolding cindexE-def by (fold left-def right-def,auto)
  qed

also have \( \ldots = \text{farg}/\pi \)
proof 
  have \( \text{p-pos; c*poly p x > 0 when} x \in \{\text{max-r} \leq \ldots \text{r} \} \) for \( x \)
  proof 
    define \( \text{hh where} \) \( \text{hh} = (\lambda t. (r - \text{max-r}) \ast t + \text{max-r}) \)
    have \( (x - \text{max-r})/(r - \text{max-r}) \in \{0 \leq \ldots 1\} \)
      using that \( \) by auto
    then have \( \emptyset < c \ast \text{poly p} \ (h hh ((x - \text{max-r})/(r - \text{max-r}))) \)
      apply (drule-tac Re-pos[rule-format])
    unfolding comp-def hp[rule-format] hp[rule-format] hh-def \).
    moreover have \( hh \ ((x - \text{max-r})/(r - \text{max-r})) = x \)
      unfolding hh-def using \( \{\text{max-r} \leq \text{r}\} \)
    by (auto simp add:divide-simps)
  ultimately show \( ?\text{thesis} \) by simp
  qed

have \( c \ast \text{poly q \text{max-r} \neq 0} \)
  using no-real-zero \( (c \neq 0) \)
by (metis Im-complex-of-real UNIV-I \( \{\text{max-r} \in \text{proots p} \} \) poly-of-decompose
  mult-eq-0-iff p-def poly-cpoly-of-real-iff proots-within-iff q-def)

moreover have \( ?\text{thesis} \) when \( c \ast \text{poly q \text{max-r} > 0} \)
proof 
  have \( 0 < \text{Im} \ (h 0) \) unfolding \( \text{hq[rule-format]} \) \( \text{hp[rule-format]} \) using 
  that by auto
moreover have \( \text{jumpF f \ (at-right} \text{max-r}) = 1/2 \)
proof 
  have \( ((\lambda t. c \ast \text{poly p} \ t) \text{has-sgnx 1}) \) \( \text{at-right} \text{max-r} \)
    unfolding has-sgnx-def
97
apply (rule eventually-at-rightI[of - r])
using p-pos (max-r<r) by auto
then have filterlim f at-top (at-right max-r)
  unfolding f-def
  apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q max-r])
  using that (max-r∈proots p) by (auto intro!:tendsto-eq-intros)
then show ?thesis unfolding jumpF-def by auto
qed
ultimately show ?thesis unfolding farg-def by auto
qed
moreover have ?thesis when c*poly q max-r < 0
proof
  have 0 > Im (h 0) unfolding hq[rule-format] hp[rule-format] using
    that by auto
  moreover have jumpF f (at-right max-r) = − 1/2
  proof
    have ((∀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 (max-r<r) by auto
    then have filterlim f at-bot (at-right max-r)
      unfolding f-def
      apply (subst filterlim-divide-at-bot-at-top-iff[of - c*poly q max-r])
      using that (max-r∈proots p) by (auto intro!:tendsto-eq-intros)
    then show ?thesis unfolding jumpF-def by auto
  qed
  ultimately show ?thesis unfolding farg-def by auto
  qed
  ultimately show ?thesis by linarith
  qed
  finally show ?thesis .
  qed
ultimately show ?thesis unfolding wc-add-def f-def by (auto simp add:field-simps)
  qed

  have ∀x∈{0<..1}. (Re o g3) x ≠ 0
  proof (rule ccontr)
    assume ∼ (∀x∈{0<..1}. (Re o g3) x ≠ 0)
    then obtain t where t-def:Re (g3 t) =0 t∈{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 Re (g3 t) = Re (poly pp (of-real m))
        unfolding m-def g3-def g-def linepath-def subpath-def v-def using (r≠0)
        by (auto simp add:algebra-simps)
      then show ?thesis using t-def unfolding Re-poly-of-real p-def by auto

  98
qed
moreover have \( m > \max-r \)
proof –
have \( r - \max-r > 0 \) using r-asm by simp
then have \( (r - \max-r) \cdot t > 0 \) 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 \( \forall x \in \{0 < .. 1\}. \ (Re \circ g3) x < 0 \)
proof –
have \( wc-add (uminus o g3) = arctan (f r) / pi \)
  unfolding f-def
  apply (rule wc-add-pos[of - 1])
  using g3-pq that \( \max-r \in \text{proots } p \) (valid-path g3) \( 0 \notin \text{path-image } g3 \)
  by (auto simp add: path-image-compose)
moreover have \( wc-add (uminus o g3) = wc-add g3 \)
  unfolding wc-add-def cindex-pathE-def
  apply (subst winding-number-uminus-comp)
  using (valid-path g3) \( 0 \notin \text{path-image } g3 \) by auto
ultimately show ?thesis by auto
qed
moreover have ?thesis when \( \forall x \in \{0 < .. 1\}. \ (Re \circ g3) x > 0 \)
proof –
have \( wc-add (g r) = (arctan (f r) - arctan (f (\neg r))) / pi \)
  by (auto simp add: field-simps)
then show \( 2 \ast Re (\text{winding-number } (g r) 0) + \text{cindex-pathE } (g r) 0 \)
  = \( (arctan (f r) - arctan (f (\neg r))) / pi \)
  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: \( \forall x \in \text{proots } p. \ Im \ x \neq 0 \)
shows \textit{proots-upper} \( p = \)

\[
(\text{degree } p - \text{cindex-poly-ubd} \ (\text{map-poly } \text{Im } p \ (\text{map-poly } \text{Re } p)) \ / 2
\]

\textbf{proof} (\textit{cases degree } p = 0)

\textbf{case} True

then obtain \( c \) \textit{where} \( p = [c:] \) \textit{using} \textit{degree-eq-zeroE} \textit{by} \textit{blast}

then have \( p \text{-def} : p = [1:] \) \textit{using} \langle \text{lead-coeff } p \rangle \textit{by} \textit{simp}

have \textit{proots-count} \( p \{x. \text{Im } x > 0\} = 0 \) \textit{unfolding} \textit{p-def} \textit{proots-count-def} \textit{by} \textit{auto}

moreover have \textit{cindex-poly-ubd} \( (\text{map-poly } \text{Im } p) \ (\text{map-poly } \text{Re } p) = 0 \)

apply \( (\text{subst cindex-poly-ubd-code}) \)

unfolding \textit{p-def} \textit{by} \textit{(auto simp add: map-poly-pCons changes-R-smods-def changes-poly-neg-inf-def}

changes-poly-pos-inf-def)

ultimately show \( ?\text{thesis} \) \textit{using} \( \text{True} \) \textit{unfolding} \textit{proots-upper-def} \textit{by} \textit{auto}

next

\textbf{case} False

then have \( \text{degree } p > 0 \ p \neq 0 \) \textit{by} \textit{auto}

\textbf{define} \textit{w1} \textit{where} \textit{w1} = \( (\lambda r. \text{Re} \ (\text{winding-number} \ (\text{poly } p \circ \ (\lambda x. \text{complex-of-real} \ (\text{linepath} (- r) \ (\text{of-real} r) x)))) 0)) \)

\textbf{define} \textit{w2} \textit{where} \textit{w2} = \( (\lambda r. \text{Re} \ (\text{winding-number} \ (\text{poly } p \circ \ (\text{part-circlepath} 0 r 0 \ \pi))) 0)) \)

\textbf{define} \textit{cp} \textit{where} \textit{cp} = \( (\lambda r. \text{cindex-pathE} \ (\text{poly } p \circ \ (\lambda x. \text{of-real} \ (\text{linepath} (- r) \ (\text{of-real} r) x)))) 0)) \)

\textbf{define} \textit{ci} \textit{where} \textit{ci} = \( (\lambda r. \text{cindexE} \ (-r) r \ (\lambda x. \text{poly} \ (\text{map-poly } \text{Im } p) x / \text{poly} \ (\text{map-poly } \text{Re } p) x)) \)

\textbf{define} \textit{cubd} \textit{where} \textit{cubd} = \textit{cindex-poly-ubd} \( (\text{map-poly } \text{Im } p) \ (\text{map-poly } \text{Re } p) \)

obtain \( R \) \textit{where} \textit{proots} \( p \subseteq \text{ball} 0 R \text{ and } R > 0 \)

using \( (p \neq 0) \) \textit{finite-ball-include[of proots p 0] by auto}

have \( ((\lambda r. \text{w1} r + \text{w2} r + \text{cp} r / 2 - \text{ci} r / 2)) \rightarrow \text{real} \ (\text{degree } p) / 2 - \text{of-int} \ \text{cubd} / 2) \atop \text{at-top} \)

proof –

have \textit{t1} : \( ((\lambda r. 2 * \text{w1} r + \text{cp} r) \rightarrow \text{0}) \atop \text{at-top} \)

using \( \text{Re-winding-number-poly-linepth[OF assms]} \) \textit{unfolding} \textit{w1-def cp-def}

by \textit{auto}

have \textit{t2} : \( \text{w2} \rightarrow \text{real} \ (\text{degree } p) / 2) \atop \text{at-top} \)

using \( \text{Re-winding-number-poly-part-circlepath[OF \langle \text{degree } p > 0\rangle, of 0]} \) \textit{unfolding} \textit{w2-def by auto}

have \textit{t3} : \( \text{ci} \rightarrow \text{of-int} \ \text{cubd} \) \atop \text{at-top} \)

apply \( (\text{rule tendsto-eventually}) \)

using \( \text{cindex-poly-ubd-eventually[of map-poly } \text{Im } p \text{ map-poly } \text{Re } p] \)

unfolding \( \text{ci-def cubd-def by auto} \)

from \textit{tendsto-add[OF tendsto-add[OF tendsto-mult-left[OF \langle \text{degree } t3, of } -1/2, simplified]]}

\( \text{tendsto-mul-left[OF } \langle \text{t1, of } 1/2, simplified]} \)

\( \text{t2} \)

show \( ?\text{thesis} \) \textit{by} \( (\text{simp add: algebra-simps}) \)

qed

moreover have \( \forall r \in \text{at-top}. \text{w1} r + \text{w2} r + \text{cp} r / 2 - \text{ci} r / 2 = \text{proots-count} \)
proof (rule eventually-at-top-linorderI[of R])
fix r assume r ≥ R
then have r-ball:proots p ⊆ ball 0 r and r > 0
using ⟨R > 0⟩ proots p ⊆ ball 0 R by auto
define ll where ll = linepath (− complex-of-real r) r
define rr where rr = part-circlepath 0 r 0 pi
define lr where lr = ll ++ rr
have img-ll: path-image ll ⊆ − proots p and img-rr: path-image rr ⊆ − proots p

unfolding ll-def using ⟨0 < r⟩ closed-segment-degen-complex(2) no-real-roots by auto
unfolding rr-def using in-path-image-part-circlepath ⟨0 < r⟩ r-ball by fastforce

have proots-count p {x. Im x > 0} = (∑x ∈ proots p. winding-number lr x) * (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 < Im x} ⊆ proots p
using ⟨p ≠ 0⟩ by auto
next
have winding-number lr x=0 when x ∈ proots p = proots-within p {x. 0 < Im x} for x
unfolding lr-def ll-def rr-def
proof (eval-winding,simp-all)
show *: x ∉ closed-segment (− complex-of-real r) (complex-of-real r)
using img-ll that unfolding ll-def by auto
show x ∉ path-image (part-circlepath 0 r 0 pi)
using img-rr that unfolding rr-def by auto
have xq: x > Im x − r < Re x Re x < r cmod x < r
proof —
have Im x ≤ 0 using that by auto
moreover have Im x ≠ 0 using no-real-roots that by blast
ultimately show 0 > Im x by auto
next
have cmod x < r using that r-ball by auto
then have |Re x| < r
using abs-Re-le-cmod[of x] by argo
then show \(-r < \Re x \Re x < r\) by linarith+
qed
then have \(\text{cindex-pathE} \ll x = 1\)
  using \(\langle r > 0 \rangle\) unfolding \(\text{cindex-pathE-linepath}[OF \, \ast] \ll\text{def}\)
  by (auto simp add: mult-pos-neg)
moreover have \(\text{cindex-pathE} \rr x = -1\)
  unfolding \(\rr\text{-def}\) using \(r\text{-ball}\) that
  by (auto intro!: \(\text{cindex-pathE-circlepath-upper}\))
ultimately show \(-\text{cindex-pathE} (\text{linepath} (\text{of-real} r) (\text{of-real} r)) x =\)
  \(\text{cindex-pathE} (\text{part-circlepath} 0 r 0 \pi) x\)
  unfolding \(\ll\text{-def} \rr\text{-def}\) by auto
qed
then show \(\forall i \in \text{proots p} - \text{proots-within p} \{ x. \, 0 < \Im x \}\).
  \(\text{winding-number} \, \text{lr} \, i \ast \text{of-nat} (\text{order} \, i \, p) = 0\)
  by auto
next
fix \(x\) assume \(x\text{-asm}: x \in \text{proots-within p} \{ x. \, 0 < \Im x \}\)
have \(\text{winding-number} \, \text{lr} \, x = 1\)
  unfolding \(\ll\text{-def} \rr\text{-def} \text{lr-def}\)
proof (eval-winding, simp-all)
  show \(*: x \notin \text{closed-segment} (\text{of-real} \, r) (\text{complex-of-real} \, r)\)
    using \(\text{img-ll} \, x\text{-asm}\) unfolding \(\ll\text{-def}\) by auto
  show \(x \notin \text{path-image} (\text{part-circlepath} 0 r 0 \pi)\)
    using \(\text{img-rr} \, x\text{-asm}\) unfolding \(\rr\text{-def}\) by auto
  have \(x\text{-facts}: 0 < \Im x \, -r < \Re x \Re x < r \, \text{cmod} x < r\)
    proof
      show \(0 < \Im x\) using \(x\text{-asm}\) by auto
      show \(\text{cmod} x < r\) using \(x\text{-asm}\) \(r\text{-ball}\) by auto
      then have \(\Re x < r\)
        using \(\text{abs-Re-le-cmod}[of \, x]\) by argo
      then show \(-r < \Re x \Re x < r\) by linarith+
    qed
then have \(\text{cindex-pathE} \ll x = -1\)
  using \(\langle r > 0 \rangle\) unfolding \(\text{cindex-pathE-linepath}[OF \, \ast] \ll\text{def}\)
  by (auto simp add: mult-less-0-iff)
moreover have \(\text{cindex-pathE} \rr x = -1\)
  unfolding \(\rr\text{-def}\) using \(r\text{-ball} \, x\text{-asm}\)
  by (auto intro!: \(\text{cindex-pathE-circlepath-upper}\))
ultimately show \(-\text{of-real} (\text{cindex-pathE} (\text{linepath} (\text{of-real} r)) (\text{of-real} \, r)) x =\)
  \(\text{of-real} (\text{cindex-pathE} (\text{part-circlepath} 0 r 0 \pi) x) = 2\)
  unfolding \(\ll\text{-def} \rr\text{-def}\) by auto
qed
then show \(\text{of-nat} (\text{order} \, x \, p) = \text{winding-number} \, \text{lr} \, x \ast \text{of-nat} (\text{order} \, x \, p)\) by auto
qed
also have \(\ldots = 1 / (2 \ast \pi \ast i) \ast \text{contour-integral} \, \text{lr} \, (\lambda x. \text{deriv} (\text{poly} \, p) \, x) / (\text{poly} \, p \, x)\)
apply (subst \(\text{argument-principle-poly}[of \, p \, \text{lr}]\))

102
using \( p \neq 0 \)
\[ \text{img-ll img-rr unfolding } \text{lr-def ll-def rr-def} \]
by (auto simp add: path-image-join)
also have \( \ldots = \text{winding-number} (\text{poly } p \circ \text{lr} ) \ 0 \)
apply (subst winding-number-comp[of UNIV poly p lr 0])
using \( p \neq 0 \)
\[ \text{img-ll img-rr unfolding } \text{lr-def ll-def rr-def} \]
by (auto simp add: path-image-join path-image-compose)
also have \( \ldots = \text{Re} (\text{winding-number} (\text{poly } p \circ \text{lr} ) \ 0) \)
proof
have \( \text{winding-number} (\text{poly } p \circ \text{lr} ) \ 0 \in \text{Ints} \)
apply (rule integer-winding-number)
using \( p \neq 0 \)
\[ \text{img-ll img-rr unfolding } \text{lr-def} \]
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 \( \ldots = \text{Re} (\text{winding-number} (\text{poly } p \circ \text{ll} ) \ 0) + \text{Re} (\text{winding-number} (\text{poly } p \circ \text{rr} ) \ 0) \)
unfolding lr-def path-compose-join
apply (subst winding-number-join)
by (auto simp add: valid-path-imp-path path-image-compose pathstart-compose pathfinish-compose)
also have \( \ldots = w_1 r + w_2 r \)
unfolding w1-def w2-def ll-def rr-def of-real-linepath
by auto
finally have \( \text{of-nat} (\text{proots-count } p \ { x. \ 0 < \text{Im } x}) = \text{complex-of-real} (w_1 r + w_2 r) \).
then have \( \text{proots-count } p \ ( x. \ 0 < \text{Im } x) = w_1 r + w_2 r \)
using of-real-eq-iff by fastforce
moreover have \( cp r = ci r \)
proof
define \( f \) where \( f = (\lambda x. \text{Im} (\text{poly } p \ (\text{of-real } x))/\text{Re} (\text{poly } p x)) \)
have \( cp r = \text{cindex-pathE} (\text{poly } p \circ (\lambda x. 2*r*x-r)) \ 0 \)
unfolding cp-def linepath-def by (auto simp add: algebra-simps)
also have \( \ldots = \text{cindexE} \ 0 \ 1 (f \circ (\lambda x. 2*r*x-r)) \)
unfolding cp-def ci-def cindex-pathE-def f-def comp-def by auto
also have \( \ldots = \text{cindexE} (-r) \ r f \)
apply (subst cindexE-linear-comp[of 2*r 0 1 f -r,simplified])
using \( r>0 \) by auto
also have \( \ldots = ci r \)
unfolding ci-def f-def Im-poly-of-real Re-poly-of-real by simp
finally show ?thesis .
qed
ultimately show \( w_1 r + w_2 r + cp r / 2 - ci r / 2 = \text{real} (\text{proots-count } p \ { x. \ 0 < \text{Im } x}) \)
by auto
qed
ultimately have \( ((\lambda r::\text{real} \ . \ \text{real} (\text{proots-count } p \ { x. \ 0 < \text{Im } x})) ) \)
\( \longrightarrow \text{real} (\text{degree } p) / 2 - \text{of-int cubd} / 2 \) at-top
by (auto dest: tendsto-cong)
then show ?thesis .

103
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 ≡ linepath a (a + of-real s)
  assumes pqr:p = q * r and r-monic:lead-coeff r=1 and r-proots:∀ x∈proots r.
  Im x=Im a
  shows cindexE lb ub (λt. Im ((poly p ◦ g) t) / Re ((poly p ◦ g) t)) =
  cindexE lb ub (λt. Im ((poly q ◦ g) t) / Re ((poly q ◦ g) t))
  using assays
  proof (induct r arbitrary:p rule:poly-root-induct-alt)
  case 0
  then have False
  by (metis Im-complex-of-real UNIV-I imaginary-unit.simps(2) proots-within-0 zero-neq-one)
  then show ?case by simp
  next
  case (no-proots r)
  then obtain b where b̸=0 r=[:b:]
  using fundamental-theorem-of-algebra-alt by blast
  then have r=1 using ⟨lead-coeff r=1⟩ by simp
  with ⟨p=q∗r⟩ 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-constsI)
  moreover have ?case when s≠0
  proof
  define qrg where qrg = poly (q*r) ◦ g
  have cindexE lb ub (λt. Im ((poly p ◦ g) t) / Re ((poly p ◦ g) t)) =
  cindexE lb ub (λt. Im ((poly q ◦ g) t) / Re ((poly q ◦ g) t))
  unfolding qrg-def p = q * ([− b, 1:] * r) comp-def
  by (simp add:algebra-simps)
  also have ... = cindexE lb ub
  (λt. ((Re a + t * s − Re b )* Im (qrg t)) / ((Re a + t * s − Re b )* Re (qrg t)))
  proof
  have Im b = Im a
  using ∀x∈proots ([− b, 1:] * r). Im x = Im a by auto
  then show ?thesis
  unfolding cindex-pathE-def g-def linepath-def
  by (simp add:algebra-simps)
  qed
  also have ... = cindexE lb ub (λt. Im (qrg t) / Re (qrg t))
  proof (rule cindexE-cong[of {t. Re a + t * s − Re b = 0}])
  show finite {t. Re a + t * s − Re b = 0}
  proof (cases Re a= Re b)
case True
then have \{ t \cdot \Re a + t \cdot s - \Re b = 0 \} = \{ 0 \}
  using \( s \neq 0 \) by auto
then show \(?thesis\) by auto
next
  case False
then have \{ t \cdot \Re a + t \cdot s - \Re b = 0 \} = \{ (\Re b - \Re a) / s \}
  using \( s \neq 0 \) by (auto simp add:field-simps)
then show \(?thesis\) by auto
qed
next
fix \( x \) assume asm: \( x / \in \{ t \cdot \Re a + t \cdot s - \Re b = 0 \} \)
define \( tt \) where \( tt = \Re a + x \cdot s - \Re b \)
have \( tt \neq 0 \) using asm unfolding \( tt\)-def by auto
then show \( tt \cdot \Im (qrg x) / (tt \cdot \Re (qrg x)) = \Im (qrg x) / \Re (qrg x) \)
  by auto
qed
also have \( ... = \text{cindexE lb ub} (\lambda t. \Im ((\text{poly q \circ g}) t) / \Re ((\text{poly q \circ g}) t)) \)
  unfolding \( qrg\)-def
proof (rule root(1))
  show \( \text{lead-coeff r} = 1 \)
    by (metis \( \text{lead-coeff-mult} \text{ lead-coeff-pCons} (1) \text{ mult-cancel-left2} \text{ one-poly-eq-simps} (2) \)

  root.prems(2) zero-neq-one)
qed (use root in simp-all)
finally show \(?thesis\).
qed
ultimately show \(?case\) by auto
qed

lemma \( \text{poly-decompose-by-proots} \):
  fixes \( p : a::\text{idom poly} \)
  assumes \( p \neq 0 \)
  shows \( \exists q r. \ p = q \cdot r \land \text{lead-coeff q}=1 \land (\forall x \in \text{proots q}. q P x) \land (\forall x \in \text{proots r}. \neg P x) \) using \( \text{assms} \)
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 \( pqr:p = q \cdot r \) and \( \text{leadq}\cdot\text{lead-coeff q}=1 \)

105
and qball:∀ a∈roots q. P a and rball:∀ x∈roots r. ¬ P x

using mult-zero-right by blast
have ?case when P a
  apply (rule-tac x=[− a, 1:] * q in exI)
  apply (rule-tac x=r in exI)
  using pqr qball rball that leadq unfolding lead-coeff-mult
  by (auto simp add: algebra-simps)
moreover have ?case when ¬ P a
  apply (rule-tac x=q in exI)
  apply (rule-tac x=[− a, 1:] * r in exI)
  using pqr qball rball that leadq unfolding lead-coeff-mult
  by (auto simp add: algebra-simps)
ultimately show ?case by blast
qed

lemma proots-upper-cindex-eq':
  assumes lead-coeff p=1
  shows 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≠0 using assms by auto
  from poly-decompose-by-roots[OF this,of λx. Im x≠0]
  obtain q r where pqr:p = q * r and leadq:lead-coeff q=1
                and qball:∀ x∈roots q. Im x ≠0 and rball:∀ x∈roots r. Im x =0
  by auto
  have real-of-int (proots-upper p) = proots-upper q + proots-upper r
    using (p≠0) 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}
    unfolding pqr
  apply (rule degree-mult-eq)
  using (p ≠ 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 ∈ proots-within p {x. |Im x| = 0}
then have Im x=0 by auto
then have order x q = 0
using qball order-0I by blast
then show order x r = order x p
using \(p \neq 0\), unfolding pqr 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 iq rq ip rp where iq = map-poly Im q and rq=map-poly Re q
and ip=map-poly Im p and rp = map-poly Re p
have cindexE \((−x)\) x (λx. poly iq x / poly rq x)
= cindexE \((−x)\) x (λx. poly ip x / poly rp x) for x
proof –
have lead-coeff r = 1
using \(p \neq 0\) \(\langle lead-coeff p=1 \rangle\) by (simp add: coeff-degree-mult)
then have cindexE \((−x)\) x (λt. Im (poly p (t * R 1)) / Re (poly p (t * R 1)))
= cindexE \((−x)\) x (λt. Im (poly q (t * R 1)) / Re (poly q (t * R 1)))
using cindexE-roots-on-horizontal-border[OF pqr,of 0 -x x 1,unfolded linepath-def comp-def,simplified] rball by simp
then show ?thesis
unfolding scaleR-conv-of-real iq-def ip-def rq-def rp-def
(by simp add:Im-poly-of-real Re-poly-of-real)
qed
then have \(∀ F \ r::real in at-top. \)
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 \( \text{card} \ (\text{proots-within} \ p \ \{x. \ \text{Im} \ x > 0\}) = (\text{let} \ pp = \text{smult} \ (\text{inverse} \ (\text{lead-coeff} \ p)) \ p; \ pI = \text{map-poly} \ \text{Im} \ pp; \ pR = \text{map-poly} \ \text{Re} \ pp; \ g = \text{gcd} \ pR \ pI \ in \ \text{nat} \ ((\text{degree} \ p - \text{changes-R-smods} \ g \ (\text{pderiv} \ g) - \text{changes-R-smods} \ pR \ pI) \ \text{div} \ 2) \) 

proof
\[
\begin{align*}
&\text{define } pp \text{ where } pp = \text{smult} \ (\text{inverse} \ (\text{lead-coeff} \ p)) \ p \\
&\text{define } pI \text{ where } pI = \text{map-poly} \ \text{Im} \ pp \\
&\text{define } pR \text{ where } pR = \text{map-poly} \ \text{Re} \ pp \\
&\text{define } g \text{ where } g = \text{gcd} \ pR \ pI \\
&\text{have } \text{card} \ (\text{proots-within} \ p \ \{x. \ \text{Im} \ x > 0\}) = \text{proots-upper} \ p \\
&\quad \text{unfolding } \text{proots-upper-def} \ \text{using } \text{card-proots-within-rsquarefree}\ [\text{OF } \text{assms}] \ \text{by auto} \\
&\text{also have } ... = \text{proots-upper} \ pp \\
&\quad \text{unfolding } \text{proots-upper-def} \ pp-def \\
&\quad \text{apply } (\text{subst } \text{proots-count-smult}) \\
&\quad \text{using } \text{assms by auto} \\
&\text{also have } ... = (\text{degree} \ pp - \text{proots-count} \ pp \ \{x. \ \text{Im} \ x = 0\} - \text{cindex-poly-ubd} \ pI \ pR) \ \text{div} \ 2 \\
&\quad \text{proof } (\text{subst } \text{real-of-int-div}) \\
&\quad \text{define } tt \text{ where } tt = \text{int} \ (\text{degree} \ pp - \text{rr} - \text{cpp}) \ - \text{cpp} \\
&\quad \text{have } \text{real-of-int} \ tt = 2 + \text{proots-upper} \ pp \\
&\quad \text{by } (\text{simp add;[folded } tt\text{-def]}) \\
&\quad \text{then show } \text{even} \ tt \ \text{by } (\text{metis } \text{ded-triv-left} \ \text{even-of-nat} \ \text{of-int-eq-iff} \ \text{of-int-of-nat-eq}) \\
&\text{qed simp} \\
&\text{finally show } ?\text{thesis} \ \text{unfolding} \ \text{rr-def} \ \text{cpp-def by simp} \\
&\text{qed} \\
&\text{also have } ... = (\text{degree} \ pp - \text{changes-R-smods} \ g \ (\text{pderiv} \ g) \\
&\quad - \text{cindex-poly-ubd} \ pI \ pR) \ \text{div} \ 2 \\
&\quad \text{proof } \\
&\quad \text{have } \text{rsquarefree} \ pp \\
&\quad \text{using } \text{assms } \text{rsquarefree-smult-iff} \ \text{unfolding} \ pp-def \\
&\quad \text{by } (\text{metis } \text{inverse-eq-imp-eq} \ \text{inverse-zero} \ \text{leading-coeff-neq-0} \ \text{rsquarefree-0}) \\
&\text{from } \text{card-proots-within-rsquarefree}[\text{OF } \text{this}] \\
&\text{have } \text{proots-count} \ pp \ \{x. \ \text{Im} \ x = 0\} = \text{card} \ (\text{proots-within} \ pp \ \{x. \ \text{Im} \ x = 0\}) \\
&\quad \text{by simp} \\
&\text{also have } ... = \text{card} \ (\text{proots-within} \ pp \ (\text{unbounded-line} \ 0 \ 1)) \\
&\quad \text{proof } \\
\end{align*}
\]

108
\[ \text{have } \{ x. \Im x = 0 \} = \text{unbounded-line } 0 1 \]

**unfolding** \text{unbounded-line-def} \\
**apply** \text{auto} \\
**subgoal for** \( x \) \\
**apply** (\text{rule-tac } x=\text{Re } x \text{ in exI}) \\
**by** (\text{metis complex-is-Real-iff of-real-Re of-real-def}) \\
**done**

**then show** \( ?\text{thesis} \) by \text{simp} \\
**qed**

**also have** \( ... \) = \text{changes-R-smods } g (\text{pderiv } g) \\
**unfolding** \text{card-proots-unbounded-line[of 0 1 pp,simplified.folded pl-def pR-def]} \\
**g-def** \\
**by** (\text{auto simp add:Let-def sturm-R}) \\
**finally have** \text{proots-count } pp \( \{ x. \Im x = 0 \} = \text{changes-R-smods } g (\text{pderiv } g) \).

**moreover have** \text{degree } pp \( \geq \text{proots-count } pp \( \{ x. \Im x = 0 \} \) \\
**by** (\text{metis } \langle \text{rsquarefree } pp \rangle \text{proots-count-leq-degree rsquarefree-0}) \\
**ultimately show** \( ?\text{thesis} \) by \text{auto} \\
**qed**

**also have** \( ... \) = \text{(degree } p \text{ - changes-R-smods } g (\text{pderiv } g) \)

**− changes-R-smods } pR pl) \text{ div 2} \\
**using** \text{cindex-poly-ubd-code unfolding pp-def by simp} \\
**finally have** \text{card (proots-within } p \{ x. \theta < \Im x \}) = \text{(degree } p \text{ - changes-R-smods } g (\text{pderiv } g) \)

**− changes-R-smods } pR pl) \text{ div 2} . \\
**then show** \( ?\text{thesis} \) unfolding \text{Let-def} \\
**apply** (\text{fold pp-def pR-def pl-def g-def}) \\
**by** (\text{simp add: pp-def}) \\
**qed**

**lemma** \text{proots-upper-code1[code]}:

\text{proots-upper } p = \\
(if p \neq 0 then \\
(let pp=smult (inverse (\text{lead-coeff } p)) p; \\
pI=map-poly \text{Im } pp; \\
pR=map-poly \text{Re } pp; \\
g = \text{gcd pI pR} \\
in \\
\text{nat } ((\text{degree } p \text{ - nat \text{(changes-R-smods-ext } g (\text{pderiv } g) \text{)} - changes-R-smods }) \\
pR pl) \text{ div 2} ) \\
else \\
\text{Code.abort (STR "proots-upper fails when p=0." ) (\lambda . \text{proots-upper } p))} \\
**proof** – \\
**define** pp where pp = smult (inverse (\text{lead-coeff } p)) p \\
**define** pl where pl = map-poly \text{Im } pp \\
**define** pR where pR=map-poly \text{Re } pp \\
**define** g where g = \text{gcd pI pR} \\
**have** \( ?\text{thesis} \) when p=0 \\
\[ \text{109} \]
using that by auto
moreover have \( \text{thesis when } p \neq 0 \)

proof –
  have \( pp \neq 0 \) unfolding \( pp \)-def using that by auto
define \( rr \) where \( rr = \text{int}(\deg pp - \text{proots-count } pp \{ x. \text{Im } x = 0 \}) - \text{cindex-poly-ubd } pI \ pR \)
  have \( \text{lead-coeff } p \neq 0 \) using \( \langle p \neq 0 \rangle \) by simp
then have \( \text{proots-upper } pp = rr / 2 \) unfolding \( rr \)-def
  apply (rule-tac \( \text{proots-upper-cindex-eq} \) [of \( pp \), folded \( pI \)-def \( pR \)-def])
unfolding \( pp \)-def \( \text{lead-coeff-smult} \) by auto
then have \( \text{proots-upper } pp = \text{nat}(rr \div 2) \) by linarith
moreover have
  \( rr = \deg p - \text{nat}(\text{changes-R-smods-ext } g (pderiv g)) - \text{changes-R-smods } pR \ pI \)
proof –
have \( \deg pp = \deg p \) unfolding \( pp \)-def by auto
moreover have \( \text{proots-count } pp \{ x. \text{Im } x = 0 \} = \text{nat}(\text{changes-R-smods-ext } g (pderiv g)) \)
proof –
  have \( \{ x. \text{Im } x = 0 \} = \text{unbounded-line } 0 \ 1 \)
  unfolding \( \text{unbounded-line-def} \) by (simp add: \( \text{complex-eq-iff} \))
then show \( \text{thesis} \)
  using \( \text{proots-unbounded-line} [of 0 \ 1 \ pp, simplified, folded } pI \text{-def } pR \text{-def} \)
(\( pp \neq 0 \))
  by (auto simp: Let-def \( g \)-def \( \text{gcd} \).commute)
qed
moreover have \( \text{cindex-poly-ubd } pI \ pR = \text{changes-R-smods } pR \ pI \)
  using \( \text{cindex-poly-ubd-code} \) by auto
ultimately show \( \text{thesis unfolding } rr \)-def by auto
qed
moreover have \( \text{proots-upper } pp = \text{proots-upper } p \)
unfolding \( pp \)-def \( \text{proots-upper-def} \)
apply (subst \( \text{proots-count-smult} \))
using that by auto
ultimately show \( \text{thesis} \)
unfolding Let-def using that
apply (fold \( pp \)-def \( pI \)-def \( pR \)-def \( g \)-def)
by argo
qed
ultimately show \( \text{thesis by blast} \)
qed

lemma \( \text{proots-upper-card-code}[\text{code}]: \)
\( \text{proots-upper-card } p = (\text{if } p = 0 \text{ then } 0 \text{ else} \)
  (let
    \( pf = p \div (\gcd p (pderiv p)) \);
    \( pp = \text{smult} (\text{inverse } (\text{lead-coeff } pf)) pf \);
    \( pI = \text{map-poly Im } pp \);
    \( pR = \text{map-poly Re } pp \);
  )
\[ g = \gcd pR pI \]
\[ \text{in } \nat (\text{degree pf} - \text{changes-R-smods g (pderiv g)} - \text{changes-R-smods pR pI}) \div 2) \]

**proof** (cases \( p=0 \))
- **case** \( \text{True} \)
  - then show \( ?\text{thesis unfolding proots-upper-card-def using infinite-halfspace-Im-gt by auto} \)
- **next**
- **case** \( \text{False} \)
  - define \( pf pp pI pR g \) where
    - \( pf = p \div (\gcd p (pderiv p)) \)
    - \( pp = \text{smult (inverse (lead-coeff pf))) pf} \)
    - \( pl = \text{map-poly Im pp} \)
    - \( pR = \text{map-poly Re pp} \)
    - \( g = \gcd pR pI \)
  - have \( \text{proots-upper-card p = proots-upper-card pf} \)
    - **proof**
      - have \( \text{proots-within p \{x . 0 < \text{Im x}\} = proots-within pf \{x . 0 < \text{Im x}\}} \)
        - unfolding \( \text{proots-within-def using poly-gcd-pderiv-iff[of p,folded pf-def]} \)
        - by auto
      - then show \( ?\text{thesis unfolding proots-upper-card-def by auto} \)
    - also have \( ... = \nat ((\text{degree pf} - \text{changes-R-smods g (pderiv g)} - \text{changes-R-smods pR pI}) \div 2) \)
      - using \( \text{proots-within-upper-squarefree[of rsquarefree-gcd-pderiv[of \( p\neq0\) \],unfolded Let-def,folded pf-def,folded pp-def pl-def pR-def g-def]} \)
      - unfolding \( \text{proots-upper-card-def by blast} \)
    - finally show \( ?\text{thesis unfolding Let-def} \)
      - apply (fold pf-def,fold pp-def pl-def pR-def g-def)
      - using False by auto
  - qed

### 3.8 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** \( \text{proots-half ::complex poly => complex => complex => nat where} \)
\( \text{proots-half p a b} = \text{proots-count p \{w. Im ((w-a) / (b-a)) > 0\}} \)

**lemma** \( \text{proots-half-empty:} \)
- assumes \( a=b \)
- shows \( \text{proots-half p a b = 0} \)
- unfolding \( \text{proots-half-def using assms by auto} \)

**lemma** \( \text{proots-half-proots-upper:} \)
- assumes \( a\neq b \ p\neq0 \)

111
shows \( \text{proots-half} \ p \ a \ b = \text{proots-upper} \ (\text{pcompose} \ p \ [:a, (b-a):]) \)

proof –
define \( q \) where \( q = [a, (b - a);] \)
define \( f \) where \( f = (\lambda x. (b-a) * x + a) \)
have \( (\sum r \in \text{proots-within} \ p \ \{ w. \ \text{Im} ((w-a) / (b-a)) > 0 \}, \ \text{order} \ r \ p) = (\sum r \in \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \}, \ \text{order} \ r (p \circ_p q)) \)
proof (rule sum.reindex-cong[of \( f \)])
  have \( \text{inj} f \)
  using \( \text{assms} \) unfolding \( f\)-def \( \text{inj-on-def} \) by \( \text{fastforce} \)
then show \( \text{inj-on} \ f \ (\text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \}) \)
  by (elim \( \text{inj-on-subset} \), \( \text{auto} \))
next
show \( \text{proots-within} \ p \ \{ w. \ \text{Im} ((w-a) / (b-a)) > 0 \} = f ^ \prime \ \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \} \)
proof safe
fix \( x \) assume \( x\)-asm: \( x \in \text{proots-within} \ p \ \{ w. \ \text{Im} ((w-a) / (b-a)) > 0 \} \)
define \( xx \) where \( xx = (x - a) / (b - a) \)
have \( \text{poly} \ (p \circ_p q) \ xx = 0 \)
  unfolding \( q\)-def \( xx\)-def \( \text{poly-pcompose} \) using \( \text{assms} \) \( x\)-asm by \( \text{auto} \)
moreover have \( \text{Im} xx > 0 \)
  unfolding \( xx\)-def using \( x\)-asm by \( \text{auto} \)
ultimately have \( xx \in \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \} \) by \( \text{auto} \)
then show \( x \in f ^ \prime \ \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \} \)
  apply (intro \( \text{rev-image-eqI} \) [of \( xx \)])
  unfolding \( f\)-def \( xx\)-def using \( \text{assms} \) by \( \text{auto} \)
qed
next
fix \( x \) assume \( x \in \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \} \)
then show \( f \ x \in \text{proots-within} \ p \ \{ w. \ 0 < \text{Im} ((w-a) / (b-a)) \} \)
  unfolding \( f\)-def \( xx\)-def using \( \text{assms} \)
  apply (auto simp add: \( \text{poly-pcompose} \))
  by (auto simp add: \( \text{algebra-simps} \))
qed
next
fix \( x \) assume \( x \in \text{proots-within} \ (p \circ_p q) \ \{ z. \ 0 < \text{Im} z \} \)
show \( \text{order} \ (f \ x) \ p = \text{order} \ x \ (p \circ_p q) \) using \( p \neq 0 \);
proof (induct \( p \) rule: \( \text{poly-root-induct-alt} \))
case \( 0 \)
then show \( ?\)case by \( \text{simp} \)
next
case \( \text{no-proots} \ p \)
  have \( \text{order} \ (f \ x) \ p = 0 \)
    apply (rule \( \text{order-0I} \))
    using \( \text{no-proots} \) by \( \text{auto} \)
  moreover have \( \text{order} \ x \ (p \circ_p q) = 0 \)
    apply (rule \( \text{order-0I} \))
    unfolding \( \text{poly-pcompose} \) \( q\)-def using \( \text{no-proots} \) by \( \text{auto} \)
ultimately show \( ?\)case by \( \text{auto} \)
next
case \( \text{root} \ c \ p \)
have \( \text{order} \ (f \ x) \ (\left[ -c, 1: \right] \ast p) = \text{order} \ (f \ x) \ (\left[ -c, 1: \right]) + \text{order} \ (f \ x) \ p \)
apply (\text{subst order-mult})
using \text{root by auto}
also have \( \ldots = \text{order} \ x \ (\left[ -c, 1: \right] \circ_p q) + \text{order} \ (p \circ_p q) \)
proof
have \( \text{order} \ (f \ x) \ (\left[ -c, 1: \right]) = \text{order} \ x \ (\left[ -c, 1: \right] \circ_p q) \)
proof (cases \( f \ x = c \))
case True
have \( \left[ -c, 1: \right] \circ_p q = \text{smult} \ (b-a) \ (\left[ -x, 1: \right]) \)
using True unfolding \text{q-def f-def pcompose-pCons by auto}
then have \( \text{order} \ x \ (\left[ -c, 1: \right] \circ_p q) = \text{order} \ x \ (\text{smult} \ (b-a) \ (\left[ -x, 1: \right])) \)
by auto
then have \( \text{order} \ x \ (\left[ -c, 1: \right] \circ_p q) = 1 \)
apply (\text{subst (asm) order-smult})
using \text{assms order-power-n-n[of - 1,simplified] by auto}
moreover have \( \text{order} \ (f \ x) \ (\left[ -c, 1: \right]) = 1 \)
using True order-power-n-n[of - 1,simplified] by auto
ultimately show \( \text{?thesis by auto} \)
qed
moreover have \( \text{order} \ (f \ x) \ p = \text{order} \ x \ (p \circ_p q) \)
apply (\text{rule root})
using \text{root by auto}
ultimately show \( \text{?thesis by auto} \)
qed
also have \( \ldots = \text{order} \ x \ (\left[ -c, 1: \right] \ast p) \circ_p q) \)
unfolding \text{pcompose-mult}
apply (\text{subst order-mult})
subgoal unfolding \text{q-def using assms(1) pcompose-eq-0 root.prems by fastforce} 
by simp
finally show \( \text{?case} \).
qed
qed
then show \( \text{?thesis unfolding proots-half-def proots-upper-def proots-count-def q-def} \)
by auto
qed

lemma proots-half-code1[code]:
proots-half \ p \ a \ b = (\text{if} \ a \neq b \text{ then} \ldots)
if \( p \neq 0 \) then \( \text{proots-upper} \ (p \circ_p [z_0, b - a]) \)
else Code.abort (STR "proots-half fails when \( p=0.\))
(\( \lambda \cdot \text{proots-half} \ p \ a \ b \))
else 0

proof –

have \(?\text{thesis}\) when \( a=b \)
  using \( \text{proots-half-empty} \) that by auto
moreover have \(?\text{thesis}\) when \( a\neq b \ p=0 \)
  using that by auto
moreover have \(?\text{thesis}\) when \( a\neq b \ p\neq 0 \)
  using \( \text{proots-half-proots-upper} \) that by auto
ultimately show \(?\text{thesis}\) by auto
qed

3.9 Polynomial roots within a circle (open ball)
— Roots counted WITH multiplicity

definition \( \text{proots-ball} :: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{real} \Rightarrow \text{nat} \) where
\( \text{proots-ball} \ p \ z0 \ r = \text{proots-count} \ p \ (\text{ball} \ z0 \ r) \)

— Roots counted WITHOUT multiplicity

definition \( \text{proots-ball-card} :: \text{complex poly} \Rightarrow \text{complex} \Rightarrow \text{real} \Rightarrow \text{nat} \) where
\( \text{proots-ball-card} \ p \ z0 \ r = \text{card} \ (\text{proots-within} \ p \ (\text{ball} \ z0 \ r)) \)

lemma \( \text{proots-ball-code1\}[\text{code}]\):\
\( \text{proots-ball} \ p \ z0 \ r = \)
  \( (\text{if} \ r \leq 0 \ \text{then} \ 0) \)
  else if \( p\neq 0 \) then
    \( \text{proots-upper} \ (f\text{compose} \ (p \circ_p [z0, \text{of-real} \ r]) \ [i, -1:] [i, I:]) \)
  else
    Code.abort (STR "proots-ball fails when \( p=0.\))
(\( \lambda \cdot \text{proots-ball} \ p \ z0 \ r \))
proof (cases \( p=0 \lor r\leq0 \))
case False
have \( \text{proots-ball} \ p \ z0 \ r = \text{proots-count} \ (p \circ_p [z0, \text{of-real} \ r]) \ (\text{ball} \ 0 \ 1) \)
  unfolding \( \text{proots-ball-def} \)
  apply (rule \( \text{proots-uball-eq}[\text{THEN} \ \text{arg-cong}] \))
  using False by auto
also have \( ... = \text{proots-upper} \ ((\text{fcompose} \ (p \circ_p [z0, \text{of-real} \ r]) \ [i, -1:] [i, I:]) \)
  unfolding \( \text{proots-upper-def} \)
  apply (rule \( \text{proots-ball-plane-eq}[\text{THEN} \ \text{arg-cong}] \))
  using False \( \text{fcompose-eq-0}[\text{of} \ [z0, \text{of-real} \ r]] \) by auto
finally show \(?\text{thesis}\) using False by auto
qed (auto simp:proots-ball-def ball-empty)

lemma \( \text{proots-ball-card-code1\}[\text{code}]\):
\( \text{proots-ball-card} \ p \ z0 \ r = \)
  \( (\text{if} \ r \leq 0 \lor p=0 \ \text{then} \)
else

proots-upper-card (fcompose (p ◦ p [:z0, of-real r:]) [i, −1:] [i, 1:])
)

proof (cases p=0 ∨ r≤0)
  case True
  moreover have ?thesis when r≤0
  proof –
    have proots-within p (ball z0 r) = {}
    by (simp add: ball-empty that)
    then show ?thesis unfolding proots-ball-card-def using that by auto
  qed
  moreover have ?thesis when r>0 p=0
  unfolding proots-ball-card-def using that infinite-ball[of r z0]
  by auto
  ultimately show ?thesis by argo
next
  case False
  then have p≠0 r>0 by auto

  have proots-ball-card p z0 r = card (proots-within (p ◦ p [:z0, of-real r:]) (ball 0 1))
  unfolding proots-ball-card-def
  by (rule proots-card-uball-eq[OF ⟨r>0⟩, THEN arg-cong])
  also have ... = proots-upper-card (fcompose (p ◦ p [:z0, 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 auto
  finally show ?thesis using False by auto
qed

3.10 Polynomial roots on a circle (sphere)
— Roots counted WITH multiplicity

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

— Roots counted WITHOUT multiplicity

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

lemma proots-sphere-card-code1[code]:
proots-sphere-card p z0 r =
  ( if r=0
    then (if poly p z0=0 then 1 else 0)
    else if r < 0 ∨ p=0 then
      0
    else
      (if poly p (z0−r) =0 then 1 else 0) +
proof
  have \( \text{thesis when } r=0 \)
  proof
    have proots-within \( p \{z0\} = (\text{if poly } p \cdot z0 = 0 \text{ then } \{z0\} \text{ else } \{\}) \)
    by auto
    then show \( \text{thesis unfolding proots-sphere-card-def using that by simp} \)
    qed
  moreover have \( \text{thesis when } r\neq 0 \text{ and } r < 0 \lor p=0 \)
  proof
    have \( \text{thesis when } r < 0 \)
    proof
      have proots-within \( p \) (sphere \( z0 \) \( r \)) = \{\}
      by (auto simp add: ball-empty that)
      then show \( \text{thesis unfolding proots-sphere-card-def using that by auto} \)
      qed
  moreover have \( \text{thesis when } r > 0 \text{ and } p=0 \)
  unfolding proots-sphere-card-def using that infinite-sphere [of \( r \) \( z0 \)]
  by auto
  ultimately show \( \text{thesis using that by argo} \)
  qed
moreover have \( \text{thesis when } r > 0 \text{ and } p\neq 0 \)
proof
  define \( pp \) where \( pp = p \circ p \) [of \( z0 \), of-real \( r \)]
  define \( ppp = \text{fcompose } pp \) [\( i, -1 : \) \( i, 1 : \)]
  have \( pp \neq 0 \) unfolding pp-def using that pcompose-eq-0 by fastforce
  have proots-sphere-card \( p \) \( z0 \) \( r \) = \( \text{card (proots-within } pp \text{ (sphere } 0 \ 1) \)
  unfolding proots-sphere-card-def pp-def
  by (rule proots-card-usphere-eq [OF \( r > 0 \), THEN arg-cong])
  also have ... = \( \text{card (proots-within } pp \{-1\} \cup \text{proots-within } pp \text{ (sphere } 0 \ 1 - \{-1\}) \)
  by (simp add: insert-absorb proots-within-union)
  also have ... = \( \text{card (proots-within } pp \{-1\}) + \text{card (proots-within } pp \text{ (sphere } 0 \ 1 - \{-1\}) \)
  apply (rule card-Un-disjoint)
  using \( pp \neq 0 \) by auto
  also have ... = \( \text{card (proots-within } pp \{-1\}) + \text{card (proots-within } ppp \{x. 0 \}
  = \text{Im } x\})
  using proots-card-sphere-axis-eq [OF \( pp \neq 0 \), folded ppp-def] by simp
  also have ... = (\text{if poly } p (z0-r) = 0 \text{ then } 1 \text{ else } 0) + \text{proots-unbounded-line-card } ppp 0 1
  proof
    have proots-within \( pp \{-1\} = (\text{if poly } p (z0-r) = 0 \text{ then } \{-1\} \text{ else } \{\}) \)
    unfolding pp-def by (auto simp: poly-pcompose)

116
then have \( \text{card} \left( \text{proots-within pp \{-1\}} \right) = (\text{if poly p (z0-r) =0 then 1 else 0}) \)
by auto

moreover have \( \{x. \operatorname{Im} x = 0\} = \operatorname{unbounded-line} 0 1 \)
unfolding unbounded-line-def
apply auto
by (metis complex-is-Real-iff of-real-Re of-real-def)
then have \( \text{card} \left( \text{proots-within ppp \{0. 0 = \operatorname{Im} x\}} \right) = \text{proots-unbounded-line-card ppp 0 1} \)
unfolding proots-unbounded-line-card-def by simp
ultimately show \(?\text{thesis}\) by auto
qed

finally show \(?\text{thesis}\) by auto
using that by auto
qed

3.11 Polynomial roots on a closed ball

— Roots counted WITH multiplicity

\textbf{definition} proots-cball :: complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat \ where
\( \text{proots-cball p z0 r} = \text{proots-count p (cball z0 r)} \)

— Roots counted WITHOUT multiplicity

\textbf{definition} proots-cball-card :: complex poly \Rightarrow complex \Rightarrow real \Rightarrow nat \ where
\( \text{proots-cball-card p z0 r} = \text{card (proots-within p (cball z0 r))} \)

\textbf{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 \lor p=0 then
 0
 else
 ( let pp=fcompose (p o_p [z0, of-real r:]) [i1,-1:] [i1,1:] in
  ( if poly p (z0-r) =0 then 1 else 0)
  + proots-unbounded-line-card pp 0 1
  + proots-upper-card pp
 )
 )

\textbf{proof} –
have \(?\text{thesis}\) when \(r=0\)
proof –
have \(\text{proots-within p \{z0\}} = (\text{if poly p z0 = 0 then \{z0\} else \{\}})\)
by auto

117
then show \( \text{thesis} \) unfolding \( \text{proots-cball-card-def} \) using that by simp

qed

moreover have \( \text{thesis} \) when \( r \neq 0 \) \( r < 0 \lor p = 0 \)

proof
   have \( \text{thesis} \) when \( r < 0 \)
      proof
         have \( \text{proots-within} \ p \ (\text{cball} \ z0 \ r) = \{\} \)
            by (auto simp add: \( \text{ball-empty} \) that)
         then show \( \text{thesis} \) unfolding \( \text{proots-cball-card-def} \) using that by auto
      qed
   moreover have \( \text{thesis} \) when \( r > 0 \) \( p = 0 \)
      unfolding \( \text{proots-cball-card-def} \) using \( \text{infinite-cball} \) \([\text{of} \ r \ z0]\)
      by auto
   ultimately show \( \text{thesis} \) using that by argo
  qed

moreover have \( \text{thesis} \) when \( p \neq 0 \) \( r > 0 \)
  proof
    define \( pp \) where \( pp = \text{fcompose} \ (p \circ \text{of-real} \ r : [i, -1:] \ [i, 1:]) \)
    have \( \text{proots-cball-card} \ p \ z0 \ r = \text{card} \ (\text{proots-within} \ p \ (\text{sphere} \ z0 \ r)) \)
      \( \cup \text{proots-within} \ p \ (\text{ball} \ z0 \ r) \))
      unfolding \( \text{proots-cball-card-def} \)
      apply (simp add: \( \text{proots-within-union} \))
      by (metis \( \text{Diff-partition} \) \( \text{cball-diff-sphere} \) \( \text{sphere-cball} \))
    also have \( \ldots = \text{card} \ (\text{proots-within} \ p \ (\text{sphere} \ z0 \ r)) + \text{card} \ (\text{proots-within} \ p \ (\text{ball} \ z0 \ r)) \))
      apply (rule \( \text{card-Un-disjoint} \))
      using \( p \neq 0 \) by auto
    also have \( \ldots = (\text{if} \ \text{poly} \ p \ (z0 - r) = 0 \text{ then} 1 \text{ else} 0) + \text{proots-unbounded-line-card} \)
      \( pp \ 0 \ 1 \)
      + \text{proots-upper-card} \( pp \)
      using \( \text{proots-sphere-card-code1} \) \([\text{of} \ p \ z0 \ r \ \text{folded} \ pp\text{-def} \ \text{unfolded} \text{proots-sphere-card-def}] \)
      that
      by simp
    finally show \( \text{thesis} \)
      apply (fold \( pp\text{-def} \))
      using that by auto
  qed

ultimately show \( \text{thesis} \) by auto

qed

end

4 Some examples for complex root counting

theory Count-Complex-Roots-Examples
imports Count-Complex-Roots
begin

value proots-rectangle [:2*i,0,i:] (Complex (−1) 0) (Complex 2 2)

value proots-rectangle [:-1,−2*i,1:]
  (Complex (−1) 0) (Complex 2 2)

value proots-half [:1−i,2−i,1:]
  0 (Complex 0 1)

value proots-half [:1−i,2−i,1:] (Complex 0 1) 0

value [code] proots-ball ([−2,1]:*[−2,1]:*[−3,1]:) 0 4

value [code] proots-ball-card ([−2,1]:*[−2,1]:*[−3,1]:) 0 3

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
