

# The Sturm–Tarski Theorem

Wenda Li

May 26, 2024

## Abstract

We have formalised the Sturm–Tarski theorem (also referred as the Tarski theorem): Given polynomials  $p, q \in \mathbb{R}[x]$ , the Sturm–Tarski theorem computes the sum of the signs of  $q$  over the roots of  $p$  by calculating some remainder sequences. Note, the better-known Sturm theorem is an instance of the Sturm–Tarski theorem when  $q = 1$ . The proof follows the classic book by Basu et al. [1] and Cyril Cohen’s work in Coq [2]. With the Sturm–Tarski theorem proved, it is possible to further build a quantifier elimination procedure for real numbers as Cohen did in Coq. Another application of the Sturm–Tarski theorem is to build sign determination procedures for polynomials at real algebraic points, as described in our formalisation of real algebraic numbers [3].

## 1 Misc polynomial lemmas for the Sturm–Tarski theorem

```
theory PolyMisc imports  
  HOL–Computational-Algebra.Polynomial-Factorial  
begin
```

```
lemma coprime-poly-0:  
  poly p x ≠ 0 ∨ poly q x ≠ 0 if coprime p q  
  for x :: 'a :: field  
proof (rule ccontr)  
  assume ¬ (poly p x ≠ 0 ∨ poly q x ≠ 0)  
  then have [:-x, 1:] dvd p [:-x, 1:] dvd q  
    by (simp-all add: poly-eq-0-iff-dvd)  
  with that have is-unit [:-x, 1:]  
    by (rule coprime-common-divisor)  
  then show False  
    by (auto simp add: is-unit-pCons-iff)  
qed
```

```
lemma smult-cancel:  
  fixes p::'a::idom poly  
  assumes c≠0 and smult: smult c p = smult c q
```

```

shows p=q
proof -
  have smult c (p-q)=0 using smult by (metis diff-self smult-diff-right)
  thus ?thesis using ‹c≠0› by auto
qed

lemma dvd-monic:
  fixes p q:: 'a :: idom poly
  assumes monic:lead-coeff p=1 and p dvd (smult c q) and c≠0
  shows p dvd q using assms
proof (cases q=0 ∨ degree p=0)
  case True
  thus ?thesis using assms
    by (auto elim!: degree-eq-zeroE simp add: const-poly-dvd-iff)
next
  case False
  hence q≠0 and degree p≠0 by auto
  obtain k where k:smult c q = p*k using assms dvd-def by metis
  hence k≠0 by (metis False assms(3) mult-zero-right smult-eq-0-iff)
  hence deg-eq:degree q=degree p + degree k
    by (metis False assms(3) degree-0 degree-mult-eq degree-smult-eq k)
  have c-dvd:∀ n≤degree k. c dvd coeff k (degree k - n)
  proof (rule,rule)
    fix n assume n ≤ degree k
    thus c dvd coeff k (degree k - n)
  proof (induct n rule:nat-less-induct)
    case (1 n)
    define T where T≡(λi. coeff p i * coeff k (degree p+degree k - n - i))
    have c * coeff q (degree q - n) = (∑ i≤degree q - n. coeff p i * coeff k
      (degree q - n - i))
      using coeff-mult[of p k degree q - n] k coeff-smult[of c q degree q - n] by
      auto
    also have ...=(∑ i≤degree p+degree k - n. T i)
      using deg-eq unfolding T-def by auto
    also have ...=(∑ i∈{0..

```

**proof** –  
**have**  $\{0..<degree\ p\} \neq \{degree\ p\}$   
**by** (*metis atLeast0LessThan insertI1 lessThan-iff less-imp-not-eq*)  
**moreover have**  $\{degree\ p\} \neq \{degree\ p + 1..degree\ p + degree\ k - n\}$   
**by** (*metis add.commute add-diff-cancel-right' atLeastAtMost-singleton-iff*  
*diff-self-eq-0 eq-imp-le not-one-le-zero*)  
**moreover have**  $\{0..<degree\ p\} \neq \{degree\ p + 1..degree\ p + degree\ k - n\}$   
**using**  $\langle degree\ k \geq n \rangle \langle degree\ p \neq 0 \rangle$  **by** *fastforce*  
**ultimately show** *?thesis unfolding C-def by auto*  
**qed**  
**ultimately show** *?thesis by auto*  
**qed**  
**also have**  $... = (\sum i \in \{0..<degree\ p\}. T\ i) + coeff\ k\ (degree\ k - n)$   
**proof** –  
**have**  $\forall x \in \{degree\ p + 1..degree\ p + degree\ k - n\}. T\ x = 0$   
**using** *coeff-eq-0[of p] unfolding T-def by simp*  
**hence**  $sum\ T\ \{degree\ p + 1..degree\ p + degree\ k - n\} = 0$  **by** *auto*  
**moreover have**  $T\ (degree\ p) = coeff\ k\ (degree\ k - n)$   
**using** *monic by (simp add: T-def)*  
**ultimately show** *?thesis by auto*  
**qed**  
**finally have**  $c \cdot coeff\ q\ (degree\ q - n) = sum\ T\ \{0..<degree\ p\}$   
 $+ coeff\ k\ (degree\ k - n)$ .  
**moreover have**  $n \neq 0 \implies c\ dvd\ sum\ T\ \{0..<degree\ p\}$   
**proof** (*rule dvd-sum*)  
**fix**  $i$  **assume**  $i \in \{0..<degree\ p\}$  **and**  $n \neq 0$   
**hence**  $(n + i - degree\ p) \leq degree\ k$  **using**  $\langle n \leq degree\ k \rangle$  **by** *auto*  
**moreover have**  $n + i - degree\ p < n$  **using**  $i \langle n \neq 0 \rangle$  **by** *auto*  
**ultimately have**  $c\ dvd\ coeff\ k\ (degree\ k - (n + i - degree\ p))$   
**using** *1(1) by auto*  
**hence**  $c\ dvd\ coeff\ k\ (degree\ p + degree\ k - n - i)$   
**by** (*metis add-diff-cancel-left' deg-eq diff-diff-left dvd-0-right le-degree*  
*le-diff-conv add.commute ordered-cancel-comm-monoid-diff-class.diff-diff-right*)  
**thus**  $c\ dvd\ T\ i$  **unfolding** *T-def by auto*  
**qed**  
**moreover have**  $n = 0 \implies ?case$   
**proof** –  
**assume**  $n = 0$   
**hence**  $\forall i \in \{0..<degree\ p\}. coeff\ k\ (degree\ p + degree\ k - n - i) = 0$   
**using** *coeff-eq-0[of k] by simp*  
**hence**  $c * coeff\ q\ (degree\ q - n) = coeff\ k\ (degree\ k - n)$   
**using** *c-coeff unfolding T-def by auto*  
**thus** *?thesis by (metis dvdI)*  
**qed**  
**ultimately show** *?case by (metis dvd-add-right-iff dvd-triv-left)*  
**qed**  
**qed**  
**hence**  $\forall n. c\ dvd\ coeff\ k\ n$   
**by** (*metis diff-diff-cancel dvd-0-right le-add2 le-add-diff-inverse le-degree*)

**then obtain  $f$  where  $f:\forall n. c * f n = \text{coeff } k n$  unfolding  $\text{dvd-def}$  by  $\text{metis}$**   
**have  $\forall_{\infty} n. f n = 0$**   
**by ( $\text{metis}$  ( $\text{mono-tags}$ ,  $\text{lifting}$ )  $\text{MOST-coeff-eq-0}$   $\text{MOST-mono-assms}(3)$   $f \text{mult-eq-0-iff}$ )**  
**hence  $\text{smult } c (\text{Abs-poly } f) = k$**   
**using  $f \text{smult.abs-eq}$ [of  $c$   $\text{Abs-poly } f$ ]  $\text{Abs-poly-inverse}$ [of  $f$ ]  $\text{coeff-inverse}$ [of  $k$ ]**  
**by  $\text{simp}$**   
**hence  $q = p * \text{Abs-poly } f$  using  $k \langle c \neq 0 \rangle \text{smult-cancel}$  by  $\text{auto}$**   
**thus  $?thesis$  unfolding  $\text{dvd-def}$  by  $\text{auto}$**   
**qed**

**lemma  $\text{poly-power-n-eq}$ :**  
**fixes  $x::'a :: \text{idom}$**   
**assumes  $n \neq 0$**   
**shows  $\text{poly } ([: - a, 1:]^{\wedge} n) x = 0 \iff (x = a)$  using  $\text{assms}$**   
**by ( $\text{induct } n, \text{auto}$ )**

**lemma  $\text{poly-power-n-odd}$ :**  
**fixes  $x a:: \text{real}$**   
**assumes  $\text{odd } n$**   
**shows  $\text{poly } ([: - a, 1:]^{\wedge} n) x > 0 \iff (x > a)$  using  $\text{assms}$**   
**proof -**

**have  $\text{poly } ([: - a, 1:]^{\wedge} n) x \geq 0 = (\text{poly } [: - a, 1:] x \geq 0)$**   
**unfolding  $\text{poly-power}$  using  $\text{zero-le-odd-power}$ [OF  $\langle \text{odd } n \rangle$ ] by  $\text{blast}$**   
**also have  $(\text{poly } [: - a, 1:] x \geq 0) = (x \geq a)$  by  $\text{fastforce}$**   
**finally have  $\text{poly } ([: - a, 1:]^{\wedge} n) x \geq 0 = (x \geq a)$ .**  
**moreover have  $\text{poly } ([: - a, 1:]^{\wedge} n) x = 0 = (x = a)$  by ( $\text{rule } \text{poly-power-n-eq}$ ,  $\text{metis}$   $\text{assms even-zero}$ )**  
**ultimately show  $?thesis$  by  $\text{linarith}$**   
**qed**

**lemma  $\text{gcd-coprime-poly}$ :**  
**fixes  $p q::'a::\{\text{factorial-ring-gcd, semiring-gcd-mult-normalize}\}$   $\text{poly}$**   
**assumes  $\text{nz}: p \neq 0 \vee q \neq 0$  and  $p': p = p' * \text{gcd } p q$  and**  
 **$q': q = q' * \text{gcd } p q$**   
**shows  $\text{coprime } p' q'$**   
**using  $\text{gcd-coprime nz } p' q'$  by  $\text{auto}$**

**lemma  $\text{poly-mod}$ :**  
 **$\text{poly } (p \text{ mod } q) x = \text{poly } p x$  if  $\text{poly } q x = 0$**   
**proof -**  
**from that have  $\text{poly } (p \text{ mod } q) x = \text{poly } (p \text{ div } q * q) x + \text{poly } (p \text{ mod } q) x$**   
**by  $\text{simp}$**   
**also have  $\dots = \text{poly } p x$**   
**by ( $\text{simp only: poly-add [symmetric]}$ )  $\text{simp}$**   
**finally show  $?thesis$ .**  
**qed**

**lemma  $\text{pseudo-divmod-0[simp]}$ :  $\text{pseudo-divmod } f 0 = (0, f)$**   
**unfolding  $\text{pseudo-divmod-def}$  by  $\text{auto}$**

**lemma** *map-poly-eq-iff*:  
**assumes**  $f \neq 0$  *inj*  $f$   
**shows**  $\text{map-poly } f \ x = \text{map-poly } f \ y \iff x = y$   
**using** *assms*  
**by** (*auto simp: poly-eq-iff coeff-map-poly dest:injD*)

**lemma** *pseudo-mod-0[simp]*:  
**shows**  $\text{pseudo-mod } p \ 0 = p \ \text{pseudo-mod } 0 \ q = 0$   
**unfolding** *pseudo-mod-def pseudo-divmod-def* **by** (*auto simp add: length-coeffs-degree*)

**lemma** *pseudo-mod-mod*:  
**assumes**  $g \neq 0$   
**shows**  $\text{smult } (\text{lead-coeff } g \ ^{\wedge} (\text{Suc } (\text{degree } f) - \text{degree } g)) \ (f \ \text{mod } g) = \text{pseudo-mod } f \ g$   
**proof** –  
**define**  $a$  **where**  $a = \text{lead-coeff } g \ ^{\wedge} (\text{Suc } (\text{degree } f) - \text{degree } g)$   
**have**  $a \neq 0$  **unfolding** *a-def* **by** (*simp add: assms*)  
**define**  $r$  **where**  $r = \text{pseudo-mod } f \ g$   
**define**  $r'$  **where**  $r' = \text{pseudo-mod } (\text{smult } (1/a) \ f) \ g$   
**obtain**  $q$  **where** *pdm*:  $\text{pseudo-divmod } f \ g = (q, r)$  **using** *r-def[unfolded pseudo-mod-def]*  
**apply** (*cases pseudo-divmod f g*)  
**by** *auto*  
**obtain**  $q'$  **where** *pdm'*:  $\text{pseudo-divmod } (\text{smult } (1/a) \ f) \ g = (q', r')$  **using** *r'-def[unfolded pseudo-mod-def]*  
**apply** (*cases pseudo-divmod (smult (1/a) f) g*)  
**by** *auto*  
**have**  $\text{smult } a \ f = q * g + r$  **and**  $\text{deg-r}: r = 0 \vee \text{degree } r < \text{degree } g$   
**using** *pseudo-divmod[OF assms pdm]* **unfolding** *a-def* **by** *auto*  
**moreover** **have**  $f = q' * g + r'$  **and**  $\text{deg-r'}: r' = 0 \vee \text{degree } r' < \text{degree } g$   
**using**  $\langle a \neq 0 \rangle$  *pseudo-divmod[OF assms pdm']* **unfolding** *a-def degree-smult-eq*  
  
**by** *auto*  
**ultimately** **have** *gr*:  $(\text{smult } a \ q' - q) * g = r - \text{smult } a \ r'$   
**by** (*auto simp add: smult-add-right algebra-simps*)  
**have**  $\text{smult } a \ r' = r$  **when**  $r = 0 \ r' = 0$   
**using** *that* **by** *auto*  
**moreover** **have**  $\text{smult } a \ r' = r$  **when**  $r \neq 0 \vee r' \neq 0$   
**proof** –  
**have**  $\text{smult } a \ q' - q = 0$   
**proof** (*rule ccontr*)  
**assume** *asm*:  $\text{smult } a \ q' - q \neq 0$   
**have**  $\text{degree } (r - \text{smult } a \ r') < \text{degree } g$   
**using** *deg-r deg-r' degree-diff-less that* **by** *force*  
**also** **have**  $\dots \leq \text{degree } ((\text{smult } a \ q' - q) * g)$   
**using** *degree-mult-right-le[OF asm, of g]* **by** (*simp add: mult.commute*)  
**also** **have**  $\dots = \text{degree } (r - \text{smult } a \ r')$   
**using** *gr* **by** *auto*  
**finally** **have**  $\text{degree } (r - \text{smult } a \ r') < \text{degree } (r - \text{smult } a \ r')$  .

```

    then show False by simp
  qed
  then show ?thesis using gr by auto
  qed
  ultimately have smult a r' = r by argo
  then show ?thesis unfolding r-def r'-def a-def mod-poly-def
    using assms by (auto simp add:field-simps)
  qed

lemma poly-pseudo-mod:
  assumes poly q x=0 q≠0
  shows poly (pseudo-mod p q) x = (lead-coeff q ^ (Suc (degree p) - degree q)) * poly p x
  proof -
    define a where a=coeff q (degree q) ^ (Suc (degree p) - degree q)
    obtain f r where fr:pseudo-divmod p q = (f, r) by fastforce
    then have smult a p = q * f + r r = 0 ∨ degree r < degree q
      using pseudo-divmod[OF ‹q≠0›] unfolding a-def by auto
    then have poly (q*f+r) x = poly (smult a p) x by auto
    then show ?thesis
      using assms(1) fr unfolding pseudo-mod-def a-def
      by auto
  qed

lemma degree-less-timesD:
  fixes q::'a::idom poly
  assumes q*g=r and deg:r=0 ∨ degree g>degree r and g≠0
  shows q=0 ∧ r=0
  proof -
    have ?thesis when r=0
      using assms(1) assms(3) no-zero-divisors that by blast
    moreover have False when r≠0
    proof -
      have degree r < degree g
        using deg that by auto
      also have ... ≤ degree (q*g)
        by (metis assms(1) degree-mult-right-le mult.commute mult-not-zero that)
      also have ... = degree r
        using assms(1) by simp
      finally have degree r < degree r .
      then show False by auto
    qed
  qed
  ultimately show ?thesis by auto
  qed

end

```

## 2 Sturm–Tarski Theorem

**theory** *Sturm-Tarski*

**imports** *Complex-Main PolyMisc HOL-Computational-Algebra.Field-as-Ring*  
**begin**

### 2.1 Misc

**lemma** *eventually-at-right*:

**fixes**  $x::'a::\{\text{archimedean-field, linorder-topology}\}$

**shows** *eventually*  $P$  (at-right  $x$ )  $\longleftrightarrow$   $(\exists b > x. \forall y > x. y < b \longrightarrow P y)$

**proof** –

**obtain**  $y$  **where**  $y > x$  **using** *ex-less-of-int* **by** *auto*

**thus** *?thesis* **using** *eventually-at-right[OF <y>x]* **by** *auto*

**qed**

**lemma** *eventually-at-left*:

**fixes**  $x::'a::\{\text{archimedean-field, linorder-topology}\}$

**shows** *eventually*  $P$  (at-left  $x$ )  $\longleftrightarrow$   $(\exists b < x. \forall y > b. y < x \longrightarrow P y)$

**proof** –

**obtain**  $y$  **where**  $y < x$

**using** *linordered-field-no-lb* **by** *auto*

**thus** *?thesis* **using** *eventually-at-left[OF <y<x]* **by** *auto*

**qed**

**lemma** *eventually-neg*:

**assumes**  $F \neq \text{bot}$  **and** *eve*:*eventually*  $(\lambda x. P x)$   $F$

**shows**  $\neg$  *eventually*  $(\lambda x. \neg P x)$   $F$

**proof** (*rule ccontr*)

**assume**  $\neg \neg$  *eventually*  $(\lambda x. \neg P x)$   $F$

**hence** *eventually*  $(\lambda x. \neg P x)$   $F$  **by** *auto*

**hence** *eventually*  $(\lambda x. \text{False})$   $F$  **using** *eventually-conj[OF eve, of  $(\lambda x. \neg P x)$ ]*

**by** *auto*

**thus** *False* **using**  $\langle F \neq \text{bot} \rangle$  *eventually-False* **by** *auto*

**qed**

**lemma** *poly-tendsto[simp]*:

$(\text{poly } p \longrightarrow \text{poly } p x)$  (at  $(x::\text{real})$ )

$(\text{poly } p \longrightarrow \text{poly } p x)$  (at-left  $(x::\text{real})$ )

$(\text{poly } p \longrightarrow \text{poly } p x)$  (at-right  $(x::\text{real})$ )

**using** *isCont-def* **where**  $f = \text{poly } p$  **by** (*auto simp add: filterlim-at-split*)

**lemma** *not-eq-pos-or-neg-iff-1*:

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

**shows**  $(\forall z. \text{lb} < z \wedge z \leq \text{ub} \longrightarrow \text{poly } p z \neq 0) \longleftrightarrow$

$(\forall z. \text{lb} < z \wedge z \leq \text{ub} \longrightarrow \text{poly } p z > 0) \vee (\forall z. \text{lb} < z \wedge z \leq \text{ub} \longrightarrow \text{poly } p z < 0)$  (**is** *?Q*  $\longleftrightarrow$  *?P*)

**proof** (*rule, rule ccontr*)

**assume** *?Q*  $\neg$  *?P*

**then obtain**  $z1 z2$  **where**  $z1: \text{lb} < z1$   $z1 \leq \text{ub}$   $\text{poly } p z1 \leq 0$

```

      and z2:lb<z2 z2≤ub poly p z2≥0
    by auto
  hence  $\exists z. lb < z \wedge z \leq ub \wedge poly\ p\ z = 0$ 
  proof (cases poly p z1 = 0  $\vee$  poly p z2 = 0  $\vee$  z1=z2)
    case True
      thus ?thesis using z1 z2 by auto
    next
      case False
        hence poly p z1 < 0 and poly p z2 > 0 and z1 ≠ z2 using z1(3) z2(3) by auto
        hence  $(\exists z > z1. z < z2 \wedge poly\ p\ z = 0) \vee (\exists z > z2. z < z1 \wedge poly\ p\ z = 0)$ 
          using poly-IVT-neg poly-IVT-pos by (subst (asm) linorder-class.neq-iff, auto)

        thus ?thesis using z1(1,2) z2(1,2) by (metis less-eq-real-def order.strict-trans2)
      qed
    thus False using ‹?Q› by auto
  next
    assume ?P
    thus ?Q by auto
  qed

lemma not-eq-pos-or-neg-iff-2:
  fixes p::real poly
  shows  $(\forall z. lb \leq z \wedge z < ub \longrightarrow poly\ p\ z \neq 0)$ 
     $\longleftrightarrow (\forall z. lb \leq z \wedge z < ub \longrightarrow poly\ p\ z > 0) \vee (\forall z. lb \leq z \wedge z < ub \longrightarrow poly\ p\ z < 0)$  (is ?Q  $\longleftrightarrow$  ?P)
  proof (rule, rule ccontr)
    assume ?Q  $\neg$  ?P
    then obtain z1 z2 where z1:lb≤z1 z1<ub poly p z1≤0
      and z2:lb≤z2 z2<ub poly p z2≥0
      by auto
    hence  $\exists z. lb \leq z \wedge z < ub \wedge poly\ p\ z = 0$ 
    proof (cases poly p z1 = 0  $\vee$  poly p z2 = 0  $\vee$  z1=z2)
      case True
        thus ?thesis using z1 z2 by auto
      next
        case False
          hence poly p z1 < 0 and poly p z2 > 0 and z1 ≠ z2 using z1(3) z2(3) by auto
          hence  $(\exists z > z1. z < z2 \wedge poly\ p\ z = 0) \vee (\exists z > z2. z < z1 \wedge poly\ p\ z = 0)$ 
            using poly-IVT-neg poly-IVT-pos by (subst (asm) linorder-class.neq-iff, auto)

          thus ?thesis using z1(1,2) z2(1,2) by (meson dual-order.strict-trans not-le)
        qed
      thus False using ‹?Q› by auto
    next
      assume ?P
      thus ?Q by auto
    qed

lemma next-non-root-interval:
  fixes p::real poly

```



```

assumes  $p \neq 0$ 
obtains  $ub$  where  $ub > lb$  and  $(\forall z. lb < z \wedge z \leq ub \longrightarrow poly\ p\ z \neq 0)$ 
proof (cases  $(\exists r. poly\ p\ r = 0 \wedge r > lb)$ )
  case False
    thus ?thesis by (intro that[of lb+1],auto)
next
  case True
    define  $lr$  where  $lr \equiv Min\ \{r . poly\ p\ r = 0 \wedge r > lb\}$ 
    have  $\forall z. lb < z \wedge z < lr \longrightarrow poly\ p\ z \neq 0$  and  $lr > lb$ 
      using True  $lr$ -def poly-roots-finite[OF  $\langle p \neq 0 \rangle$ ] by auto
    thus ?thesis using that[of (lb+lr)/2] by auto
qed

```

```

lemma last-non-root-interval:
  fixes  $p :: real\ poly$ 
  assumes  $p \neq 0$ 
  obtains  $lb$  where  $lb < ub$  and  $(\forall z. lb \leq z \wedge z < ub \longrightarrow poly\ p\ z \neq 0)$ 
proof (cases  $(\exists r. poly\ p\ r = 0 \wedge r < ub)$ )
  case False
    thus ?thesis by (intro that[of ub - 1]) auto
next
  case True
    define  $mr$  where  $mr \equiv Max\ \{r . poly\ p\ r = 0 \wedge r < ub\}$ 
    have  $\forall z. mr < z \wedge z < ub \longrightarrow poly\ p\ z \neq 0$  and  $mr < ub$ 
      using True  $mr$ -def poly-roots-finite[OF  $\langle p \neq 0 \rangle$ ] by auto
    thus ?thesis using that[of (mr+ub)/2]  $\langle mr < ub \rangle$  by auto
qed

```

## 2.2 Sign

```

definition  $sign :: 'a :: \{zero, linorder\} \Rightarrow int$  where
   $sign\ x \equiv (if\ x > 0\ then\ 1\ else\ if\ x = 0\ then\ 0\ else\ -1)$ 

```

```

lemma sign-simps[simp]:
   $x > 0 \implies sign\ x = 1$ 
   $x = 0 \implies sign\ x = 0$ 
   $x < 0 \implies sign\ x = -1$ 
unfolding sign-def by auto

```

```

lemma sign-cases [case-names neg zero pos]:
   $(sign\ x = -1 \implies P) \implies (sign\ x = 0 \implies P) \implies (sign\ x = 1 \implies P) \implies P$ 
unfolding Sturm-Tarski.sign-def by argo

```

```

lemma sign-times:
  fixes  $x :: 'a :: linordered-ring-strict$ 
  shows  $sign\ (x * y) = sign\ x * sign\ y$ 
unfolding Sturm-Tarski.sign-def
by (auto simp add: zero-less-mult-iff)

```

**lemma** *sign-power*:  
**fixes**  $x::'a::\text{linordered-idom}$   
**shows**  $\text{sign } (x \hat{=} n) = (\text{if } n=0 \text{ then } 1 \text{ else if even } n \text{ then } |\text{sign } x| \text{ else } \text{sign } x)$   
**by** (*simp add: Sturm-Tarski.sign-def zero-less-power-eq*)

**lemma** *sgn-sign-eq*:  $\text{sgn} = \text{sign}$   
**unfolding** *sign-def sgn-if* **by** *auto*

**lemma** *sign-sgn[simp]*:  $\text{sign } (\text{sgn } x) = \text{sign } (x::'b::\text{linordered-idom})$   
**by** (*simp add: sign-def*)

**lemma** *sign-uminus[simp]*:  $\text{sign } (- x) = - \text{sign } (x::'b::\text{linordered-idom})$   
**by** (*simp add: sign-def*)

## 2.3 Bound of polynomials

**definition** *sgn-pos-inf* ::  $('a :: \text{linordered-idom}) \text{ poly} \Rightarrow 'a$  **where**  
 $\text{sgn-pos-inf } p \equiv \text{sgn } (\text{lead-coeff } p)$

**definition** *sgn-neg-inf* ::  $('a :: \text{linordered-idom}) \text{ poly} \Rightarrow 'a$  **where**  
 $\text{sgn-neg-inf } p \equiv \text{if even } (\text{degree } p) \text{ then } \text{sgn } (\text{lead-coeff } p) \text{ else } -\text{sgn } (\text{lead-coeff } p)$

**lemma** *sgn-inf-sym*:  
**fixes**  $p::\text{real poly}$   
**shows**  $\text{sgn-pos-inf } (p \text{ compose } p [:0, -1:]) = \text{sgn-neg-inf } p$  (**is**  $?L=?R$ )  
**proof** –  
**have**  $?L = \text{sgn } (\text{lead-coeff } p * (-1) \hat{=} \text{degree } p)$   
**unfolding** *sgn-pos-inf-def* **by** (*subst lead-coeff-comp, auto*)  
**thus** *thesis* **unfolding** *sgn-neg-inf-def*  
**by** (*metis mult.right-neutral mult-minus1-right neg-one-even-power neg-one-odd-power sgn-minus*)  
**qed**

**lemma** *poly-pinfy-gt-lc*:  
**fixes**  $p::\text{real poly}$   
**assumes**  $\text{lead-coeff } p > 0$   
**shows**  $\exists n. \forall x \geq n. \text{poly } p x \geq \text{lead-coeff } p$  **using** *assms*  
**proof** (*induct p*)  
**case**  $0$   
**thus** *case* **by** *auto*  
**next**  
**case**  $(p \text{ Cons } a p)$   
**have**  $\llbracket a \neq 0; p = 0 \rrbracket \Longrightarrow ?\text{case}$  **by** *auto*  
**moreover** **have**  $p \neq 0 \Longrightarrow ?\text{case}$   
**proof** –  
**assume**  $p \neq 0$   
**then** **obtain**  $n1$  **where** *gte-lcoeff*:  $\forall x \geq n1. \text{lead-coeff } p \leq \text{poly } p x$  **using** *that*  
*pCons* **by** *auto*

```

have gt-0:lead-coeff p > 0 using pCons(3) ⟨p≠0⟩ by auto
define n where n≡max n1 (1 + |a|/(lead-coeff p))
show ?thesis
proof (rule-tac x=n in exI,rule,rule)
  fix x assume n ≤ x
  hence lead-coeff p ≤ poly p x
    using gte-lcoeff unfolding n-def by auto
  hence |a|/(lead-coeff p) ≥ |a|/(poly p x) and poly p x > 0 using gt-0
    by (intro frac-le,auto)
  hence x ≥ 1 + |a|/(poly p x) using ⟨n≤x⟩[unfolded n-def] by auto
  thus lead-coeff (pCons a p) ≤ poly (pCons a p) x
    using ⟨lead-coeff p ≤ poly p x⟩ ⟨poly p x > 0⟩ ⟨p≠0⟩
    by (auto simp add:field-simps)
  qed
qed
ultimately show ?case by fastforce
qed

lemma poly-sgn-eventually-at-top:
  fixes p::real poly
  shows eventually (λx. sgn (poly p x) = sgn-pos-inf p) at-top
proof (cases p=0)
  case True
  thus ?thesis unfolding sgn-pos-inf-def by auto
next
  case False
  obtain ub where ub:∀ x≥ub. sgn (poly p x) = sgn-pos-inf p
  proof (cases lead-coeff p > 0)
    case True
    thus ?thesis
      using that poly-pinfty-gt-lc[of p] unfolding sgn-pos-inf-def by fastforce
  next
    case False
    hence lead-coeff (-p) > 0 and lead-coeff p < 0 unfolding lead-coeff-minus
      using leading-coeff-neq-0[OF ⟨p≠0⟩]
      by (auto simp add:not-less-iff-gr-or-eq)
    then obtain n where ∀ x≥n. lead-coeff p ≥ poly p x
      using poly-pinfty-gt-lc[of -p] unfolding lead-coeff-minus by auto
    thus ?thesis using ⟨lead-coeff p < 0⟩ that[of n] unfolding sgn-pos-inf-def by
      fastforce
  qed
  thus ?thesis unfolding eventually-at-top-linorder by auto
qed

lemma poly-sgn-eventually-at-bot:
  fixes p::real poly
  shows eventually (λx. sgn (poly p x) = sgn-neg-inf p) at-bot
using
  poly-sgn-eventually-at-top[of pcompose p [:0,-1:],unfolded poly-pcompose sgn-inf-sym,simplified]

```

*eventually-filtermap*[of - uminus at-bot::real filter,folded at-top-mirror]  
**by auto**

**lemma root-ub:**

**fixes**  $p$ : real poly

**assumes**  $p \neq 0$

**obtains**  $ub$  **where**  $\forall x. \text{poly } p \ x = 0 \longrightarrow x < ub$

**and**  $\forall x \geq ub. \text{sgn } (\text{poly } p \ x) = \text{sgn-pos-inf } p$

**proof** –

**obtain**  $ub1$  **where**  $ub1: \forall x. \text{poly } p \ x = 0 \longrightarrow x < ub1$

**proof** (cases  $\exists r. \text{poly } p \ r = 0$ )

**case** *False*

**thus** ?thesis **using** *that* **by auto**

**next**

**case** *True*

**define**  $max-r$  **where**  $max-r \equiv \text{Max } \{x . \text{poly } p \ x = 0\}$

**hence**  $\forall x. \text{poly } p \ x = 0 \longrightarrow x \leq max-r$

**using** *poly-roots-finite*[*OF*  $\langle p \neq 0 \rangle$ ] *True* **by auto**

**thus** ?thesis **using** *that*[of  $max-r+1$ ]

**by** (*metis add commute add-strict-increasing zero-less-one*)

**qed**

**obtain**  $ub2$  **where**  $ub2: \forall x \geq ub2. \text{sgn } (\text{poly } p \ x) = \text{sgn-pos-inf } p$

**using** *poly-sgn-eventually-at-top*[*unfolded eventually-at-top-linorder*] **by auto**

**define**  $ub$  **where**  $ub \equiv \text{max } ub1 \ ub2$

**have**  $\forall x. \text{poly } p \ x = 0 \longrightarrow x < ub$  **using**  $ub1$   $ub\text{-def}$

**by** (*metis eq-iff less-eq-real-def less-linear max.bounded-iff*)

**thus** ?thesis **using** *that*[of  $ub$ ]  $ub2$   $ub\text{-def}$  **by auto**

**qed**

**lemma root-lb:**

**fixes**  $p$ : real poly

**assumes**  $p \neq 0$

**obtains**  $lb$  **where**  $\forall x. \text{poly } p \ x = 0 \longrightarrow x > lb$

**and**  $\forall x \leq lb. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$

**proof** –

**obtain**  $lb1$  **where**  $lb1: \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb1$

**proof** (cases  $\exists r. \text{poly } p \ r = 0$ )

**case** *False*

**thus** ?thesis **using** *that* **by auto**

**next**

**case** *True*

**define**  $min-r$  **where**  $min-r \equiv \text{Min } \{x . \text{poly } p \ x = 0\}$

**hence**  $\forall x. \text{poly } p \ x = 0 \longrightarrow x \geq min-r$

**using** *poly-roots-finite*[*OF*  $\langle p \neq 0 \rangle$ ] *True* **by auto**

**thus** ?thesis **using** *that*[of  $min-r - 1$ ] **by** (*metis lt-ex order.strict-trans2 that*)

**qed**

**obtain**  $lb2$  **where**  $lb2: \forall x \leq lb2. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$

**using** *poly-sgn-eventually-at-bot*[*unfolded eventually-at-bot-linorder*] **by auto**

**define**  $lb$  **where**  $lb \equiv \text{min } lb1 \ lb2$

**have**  $\forall x. \text{poly } p \ x=0 \longrightarrow x>lb$  **using** *lb1 lb-def*  
**by** (*metis (poly-guards-query) less-not-sym min-less-iff-conj neq-iff*)  
**thus** *?thesis* **using** *that[of lb] lb2 lb-def* **by** *auto*  
**qed**

## 2.4 Variation and cross

**definition** *variation* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{int}$  **where**  
*variation*  $x \ y = (\text{if } x * y \geq 0 \text{ then } 0 \text{ else if } x < y \text{ then } 1 \text{ else } -1)$

**definition** *cross* ::  $\text{real poly} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{int}$  **where**  
*cross*  $p \ a \ b = \text{variation} \ (\text{poly } p \ a) \ (\text{poly } p \ b)$

**lemma** *variation-0[simp]*:  $\text{variation } 0 \ y = 0$   $\text{variation } x \ 0 = 0$   
**unfolding** *variation-def* **by** *auto*

**lemma** *variation-comm*:  $\text{variation } x \ y = - \text{variation } y \ x$  **unfolding** *variation-def*  
**by** (*auto simp add: mult.commute*)

**lemma** *cross-0[simp]*:  $\text{cross } 0 \ a \ b = 0$  **unfolding** *cross-def* **by** *auto*

**lemma** *variation-cases*:

$\llbracket x > 0; y > 0 \rrbracket \Longrightarrow \text{variation } x \ y = 0$   
 $\llbracket x > 0; y < 0 \rrbracket \Longrightarrow \text{variation } x \ y = -1$   
 $\llbracket x < 0; y > 0 \rrbracket \Longrightarrow \text{variation } x \ y = 1$   
 $\llbracket x < 0; y < 0 \rrbracket \Longrightarrow \text{variation } x \ y = 0$

**proof** –

**show**  $\llbracket x > 0; y > 0 \rrbracket \Longrightarrow \text{variation } x \ y = 0$  **unfolding** *variation-def* **by** *auto*  
**show**  $\llbracket x > 0; y < 0 \rrbracket \Longrightarrow \text{variation } x \ y = -1$  **unfolding** *variation-def*  
**using** *mult-pos-neg* **by** *fastforce*  
**show**  $\llbracket x < 0; y > 0 \rrbracket \Longrightarrow \text{variation } x \ y = 1$  **unfolding** *variation-def*  
**using** *mult-neg-pos* **by** *fastforce*  
**show**  $\llbracket x < 0; y < 0 \rrbracket \Longrightarrow \text{variation } x \ y = 0$  **unfolding** *variation-def*  
**using** *mult-neg-neg* **by** *fastforce*

**qed**

**lemma** *variation-congr*:

**assumes**  $\text{sgn } x = \text{sgn } x' \ \text{sgn } y = \text{sgn } y'$   
**shows**  $\text{variation } x \ y = \text{variation } x' \ y'$  **using** *assms*

**proof** –

**have**  $0 \leq x * y = (0 \leq x' * y')$  **using** *assms* **by** (*metis Real-Vector-Spaces.sgn-mult zero-le-sgn-iff*)

**moreover hence**  $\neg 0 \leq x * y \Longrightarrow x < y = (x' < y')$  **using** *assms*

**by** (*metis less-eq-real-def mult-nonneg-nonneg mult-nonpos-nonpos not-le order.strict-trans2*

*zero-le-sgn-iff*)

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

**qed**

**lemma** *variation-mult-pos*:  
**assumes**  $c > 0$   
**shows**  $\text{variation } (c*x) y = \text{variation } x y$  **and**  $\text{variation } x (c*y) = \text{variation } x y$   
**proof** –  
**have**  $\text{sgn } (c*x) = \text{sgn } x$  **using**  $\langle c > 0 \rangle$   
**by** (*simp add: Real-Vector-Spaces.sgn-mult*)  
**thus**  $\text{variation } (c*x) y = \text{variation } x y$  **using** *variation-congr* **by** *blast*  
**next**  
**have**  $\text{sgn } (c*y) = \text{sgn } y$  **using**  $\langle c > 0 \rangle$   
**by** (*simp add: Real-Vector-Spaces.sgn-mult*)  
**thus**  $\text{variation } x (c*y) = \text{variation } x y$  **using** *variation-congr* **by** *blast*  
**qed**

**lemma** *variation-mult-neg-1*:  
**assumes**  $c < 0$   
**shows**  $\text{variation } (c*x) y = \text{variation } x y + (\text{if } y=0 \text{ then } 0 \text{ else } \text{sign } x)$   
**apply** (*cases x rule:linorder-cases[of 0]*)  
**apply** (*cases y rule:linorder-cases[of 0]*), *auto simp add:*  
*variation-cases mult-neg-pos[OF \langle c < 0 \rangle, of x] mult-neg-neg[OF \langle c < 0 \rangle, of x]* +  
**done**

**lemma** *variation-mult-neg-2*:  
**assumes**  $c < 0$   
**shows**  $\text{variation } x (c*y) = \text{variation } x y + (\text{if } x=0 \text{ then } 0 \text{ else } - \text{sign } y)$   
**unfolding** *variation-comm*[*of x c\*y, unfolded variation-mult-neg-1*[*OF \langle c < 0 \rangle, of y x*]]  
**by** (*auto, subst variation-comm, simp*)

**lemma** *cross-no-root*:  
**assumes**  $a < b$  **and** *no-root*:  $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p x \neq 0$   
**shows** *cross p a b = 0*  
**proof** –  
**have**  $\llbracket \text{poly } p a > 0; \text{poly } p b < 0 \rrbracket \implies \text{False}$  **using** *poly-IVT-neg*[*OF \langle a < b \rangle*] *no-root*  
**by** *auto*  
**moreover** **have**  $\llbracket \text{poly } p a < 0; \text{poly } p b > 0 \rrbracket \implies \text{False}$  **using** *poly-IVT-pos*[*OF \langle a < b \rangle*] *no-root* **by** *auto*  
**ultimately** **have**  $0 \leq \text{poly } p a * \text{poly } p b$   
**by** (*metis less-eq-real-def linorder-neqE-linordered-idom mult-less-0-iff*)  
**thus** *?thesis* **unfolding** *cross-def variation-def* **by** *simp*  
**qed**

## 2.5 Tarski query

**definition** *taq* ::  $'a::\text{linordered-idom set} \Rightarrow 'a \text{ poly} \Rightarrow \text{int}$  **where**  
 $\text{taq } s \ q \equiv \sum_{x \in s. \text{sign } (\text{poly } q \ x)}$

## 2.6 Sign at the right

**definition** *sign-r-pos* ::  $\text{real poly} \Rightarrow \text{real} \Rightarrow \text{bool}$   
**where**

$sign-r-pos\ p\ x \equiv (eventually\ (\lambda x. poly\ p\ x > 0))\ (at-right\ x)$

**lemma** *sign-r-pos-rec*:

**fixes**  $p$ : *real poly*

**assumes**  $p \neq 0$

**shows**  $sign-r-pos\ p\ x = (if\ poly\ p\ x = 0\ then\ sign-r-pos\ (pderiv\ p)\ x\ else\ poly\ p\ x > 0)$   
)

**proof** (*cases poly p x = 0*)

**case** *True*

**have**  $sign-r-pos\ (pderiv\ p)\ x \implies sign-r-pos\ p\ x$

**proof** (*rule ccontr*)

**assume**  $sign-r-pos\ (pderiv\ p)\ x \wedge \neg sign-r-pos\ p\ x$

**obtain**  $b$  **where**  $b > x$  **and**  $b: \forall z. x < z \wedge z < b \implies 0 < poly\ (pderiv\ p)\ z$

**using**  $\langle sign-r-pos\ (pderiv\ p)\ x \rangle$  **unfolding** *sign-r-pos-def eventually-at-right*

**by** *auto*

**have**  $\forall b > x. \exists z > x. z < b \wedge \neg 0 < poly\ p\ z$  **using**  $\langle \neg sign-r-pos\ p\ x \rangle$

**unfolding** *sign-r-pos-def eventually-at-right* **by** *auto*

**then obtain**  $z$  **where**  $z > x$  **and**  $z < b$  **and**  $poly\ p\ z \leq 0$

**using**  $\langle b > x \rangle$  **by** *auto*

**hence**  $\exists z' > x. z' < z \wedge poly\ p\ z' = (z - x) * poly\ (pderiv\ p)\ z'$

**using** *poly-MVT[OF <z>x]* **True** **by** (*metis diff-0-right*)

**hence**  $\exists z' > x. z' < z \wedge poly\ (pderiv\ p)\ z' \leq 0$

**using**  $\langle poly\ p\ z \leq 0 \rangle \langle z > x \rangle$  **by** (*metis leD le-iff-diff-le-0 mult-le-0-iff*)

**thus** *False* **using**  $b$  **by** (*metis <z < b> dual-order.strict-trans not-le*)

**qed**

**moreover have**  $sign-r-pos\ p\ x \implies sign-r-pos\ (pderiv\ p)\ x$

**proof** –

**assume**  $sign-r-pos\ p\ x$

**have**  $pderiv\ p \neq 0$  **using**  $\langle poly\ p\ x = 0 \rangle \langle p \neq 0 \rangle$

**by** (*metis monoid-add-class.add.right-neutral monom-0 monom-eq-0 mult-zero-right*)

*pderiv-iszero poly-0 poly-pCons*)

**obtain**  $ub$  **where**  $ub > x$  **and**  $ub: (\forall z. x < z \wedge z < ub \implies poly\ (pderiv\ p)\ z > 0)$

$\vee (\forall z. x < z \wedge z < ub \implies poly\ (pderiv\ p)\ z < 0)$

**using** *next-non-root-interval[OF <pderiv p ≠ 0>, of x,unfolding not-eq-pos-or-neg-iff-1]*

**by** (*metis order.strict-implies-order*)

**have**  $\forall z. x < z \wedge z < ub \implies poly\ (pderiv\ p)\ z < 0 \implies False$

**proof** –

**assume** *assm*:  $\forall z. x < z \wedge z < ub \implies poly\ (pderiv\ p)\ z < 0$

**obtain**  $ub'$  **where**  $ub' > x$  **and**  $ub': \forall z. x < z \wedge z < ub' \implies 0 < poly\ p\ z$

**using**  $\langle sign-r-pos\ p\ x \rangle$  **unfolding** *sign-r-pos-def eventually-at-right* **by** *auto*

**obtain**  $z'$  **where**  $x < z'$  **and**  $z' < (x + (\min\ ub'\ ub)) / 2$

**and**  $z': poly\ p\ ((x + \min\ ub'\ ub) / 2) = ((x + \min\ ub'\ ub) / 2 - x) * poly\ (pderiv$

$p)\ z'$

**using** *poly-MVT[of x (x + min ub' ub) / 2 p] <ub'>x <ub>x* **True** **by** *auto*

**moreover have**  $0 < poly\ p\ ((x + \min\ ub'\ ub) / 2)$

**using**  $ub'$  [*THEN HOL.spec, of (x + (min ub' ub)) / 2*]  $\langle z' < (x + \min\ ub'\ ub) / 2 \rangle$

$\langle x < z' \rangle$

by *auto*  
 moreover have  $(x + \min ub' ub) / 2 - x > 0$  using  $\langle ub' > x \rangle \langle ub > x \rangle$  by *auto*  
 ultimately have  $\text{poly } (pderiv\ p)\ z' > 0$  by *(metis zero-less-mult-pos)*  
 thus *False* using *asm[THEN spec, of z']*  $\langle x < z' \rangle \langle z' < (x + (\min ub' ub)) / 2 \rangle$   
 by *auto*  
 qed  
 hence  $\forall z. x < z \wedge z < ub \longrightarrow \text{poly } (pderiv\ p)\ z > 0$  using *ub* by *auto*  
 thus *sign-r-pos*  $(pderiv\ p)\ x$  unfolding *sign-r-pos-def eventually-at-right*  
 using  $\langle ub > x \rangle$  by *auto*  
 qed  
 ultimately show *?thesis* using *True* by *auto*  
 next  
 case *False*  
 have *sign-r-pos*  $p\ x \implies \text{poly } p\ x > 0$   
 proof (rule *ccontr*)  
 assume *sign-r-pos*  $p\ x \neg 0 < \text{poly } p\ x$   
 then obtain *ub* where  $ub > x$  and *ub*:  $\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p\ z$   
 unfolding *sign-r-pos-def eventually-at-right* by *auto*  
 hence  $\text{poly } p\ ((ub + x) / 2) > 0$  by *auto*  
 moreover have  $\text{poly } p\ x < 0$  using  $\langle \neg 0 < \text{poly } p\ x \rangle$  *False* by *auto*  
 ultimately have  $\exists z > x. z < (ub + x) / 2 \wedge \text{poly } p\ z = 0$   
 using *poly-IVT-pos[of x ((ub + x) / 2) p]*  $\langle ub > x \rangle$  by *auto*  
 thus *False* using *ub* by *auto*  
 qed  
 moreover have  $\text{poly } p\ x > 0 \implies \text{sign-r-pos } p\ x$   
 unfolding *sign-r-pos-def*  
 using *order-tendstoD(1)[OF poly-tendsto(1), of 0 p x]* *eventually-at-split* by  
*auto*  
 ultimately show *?thesis* using *False* by *auto*  
 qed  
  
 lemma *sign-r-pos-0[simp]*:  $\neg \text{sign-r-pos } 0\ (x :: \text{real})$   
 using *eventually-False[of at-right x]* unfolding *sign-r-pos-def* by *auto*  
  
 lemma *sign-r-pos-minus*:  
 fixes  $p :: \text{real poly}$   
 assumes  $p \neq 0$   
 shows *sign-r-pos*  $p\ x = (\neg \text{sign-r-pos } (-p)\ x)$   
 proof -  
 have *sign-r-pos*  $p\ x \vee \text{sign-r-pos } (-p)\ x$   
 unfolding *sign-r-pos-def eventually-at-right*  
 using *next-non-root-interval[OF <p≠0>, unfolded not-eq-pos-or-neg-iff-1]*  
 by *(metis (erased, opaque-lifting) le-less minus-zero neg-less-iff-less poly-minus)*  
 moreover have *sign-r-pos*  $p\ x \implies \neg \text{sign-r-pos } (-p)\ x$  unfolding *sign-r-pos-def*  
  
 using *eventually-neg[OF trivial-limit-at-right-real, of λx. poly p x > 0 x]*  
*poly-minus*  
 by *(metis (lifting) eventually-mono less-asym neg-less-0-iff-less)*  
 ultimately show *?thesis* by *auto*



qed

**lemma** *sign-r-pos-smult*:

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

**assumes**  $c \neq 0 \ p \neq 0$

**shows**  $\text{sign-r-pos (smult c p) } x = (\text{if } c > 0 \text{ then sign-r-pos } p \ x \text{ else } \neg \text{sign-r-pos } p \ x)$

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

**proof** (*cases*  $c > 0$ )

**assume**  $c > 0$

**hence**  $\forall x. (0 < \text{poly (smult c p) } x) = (0 < \text{poly } p \ x)$

**by** (*subst poly-smult,metis mult-pos-pos zero-less-mult-pos*)

**thus**  $?thesis$  **unfolding** *sign-r-pos-def* **using**  $\langle c > 0 \rangle$  **by** *auto*

**next**

**assume**  $\neg(c > 0)$

**hence**  $\forall x. (0 < \text{poly (smult c p) } x) = (0 < \text{poly } (-p) \ x)$

**by** (*subst poly-smult,metis assms(1) linorder-neqE-linordered-idom mult-neg-neg mult-zero-right*

*neg-0-less-iff-less poly-minus zero-less-mult-pos2*)

**hence**  $\text{sign-r-pos (smult c p) } x = \text{sign-r-pos } (-p) \ x$

**unfolding** *sign-r-pos-def* **using**  $\langle \neg c > 0 \rangle$  **by** *auto*

**thus**  $?thesis$  **using** *sign-r-pos-minus[OF  $\langle p \neq 0 \rangle$ , of x]*  $\langle \neg c > 0 \rangle$  **by** *auto*

qed

**lemma** *sign-r-pos-mult*:

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

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

**shows**  $\text{sign-r-pos (p*q) } x = (\text{sign-r-pos } p \ x \longleftrightarrow \text{sign-r-pos } q \ x)$

**proof** –

**obtain**  $ub$  **where**  $ub > x$

**and**  $ub: (\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p \ z) \vee (\forall z. x < z \wedge z < ub \longrightarrow \text{poly } p \ z < 0)$

**using** *next-non-root-interval[OF  $\langle p \neq 0 \rangle$ , of x, unfolded not-eq-pos-or-neg-iff-1]*

**by** (*metis order.strict-implies-order*)

**obtain**  $ub'$  **where**  $ub' > x$

**and**  $ub': (\forall z. x < z \wedge z < ub' \longrightarrow 0 < \text{poly } q \ z) \vee (\forall z. x < z \wedge z < ub' \longrightarrow \text{poly } q \ z < 0)$

**using** *next-non-root-interval[OF  $\langle q \neq 0 \rangle$ , unfolded not-eq-pos-or-neg-iff-1]*

**by** (*metis order.strict-implies-order*)

**have**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p \ z) \implies (\forall z. x < z \wedge z < ub' \longrightarrow 0 < \text{poly } q \ z) \implies ?thesis$

**proof** –

**assume**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p \ z) (\forall z. x < z \wedge z < ub' \longrightarrow 0 < \text{poly } q \ z)$

**hence**  $\text{sign-r-pos } p \ x$  **and**  $\text{sign-r-pos } q \ x$  **unfolding** *sign-r-pos-def* *eventually-at-right*

**using**  $\langle ub > x \rangle \ \langle ub' > x \rangle$  **by** *auto*

**moreover** **hence** *eventually*  $(\lambda z. \text{poly } p \ z > 0 \wedge \text{poly } q \ z > 0)$  (*at-right*  $x$ )

**unfolding** *sign-r-pos-def* **using** *eventually-conj-iff[of - - at-right x]* **by** *auto*

**hence** *sign-r-pos*  $(p*q)$   $x$   
**unfolding** *sign-r-pos-def poly-mult*  
**by** (*metis (lifting, mono-tags) eventually-mono mult-pos-pos*)  
**ultimately show** *?thesis by auto*  
**qed**  
**moreover have**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 > \text{poly } p \ z) \Longrightarrow (\forall z. x < z \wedge z < ub' \longrightarrow 0 < \text{poly } q \ z)$   
 $\Longrightarrow$  *?thesis*  
**proof** –  
**assume**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 > \text{poly } p \ z) (\forall z. x < z \wedge z < ub' \longrightarrow 0 < \text{poly } q \ z)$   
**hence** *sign-r-pos*  $(-p)$   $x$  **and** *sign-r-pos*  $q$   $x$  **unfolding** *sign-r-pos-def eventually-at-right*  
**using**  $\langle ub > x \rangle \langle ub' > x \rangle$  **by** *auto*  
**moreover hence** *eventually*  $(\lambda z. \text{poly } (-p) \ z > 0 \wedge \text{poly } q \ z > 0)$  (*at-right*  $x$ )  
**unfolding** *sign-r-pos-def using eventually-conj-iff[of - - at-right x]* **by** *auto*  
**hence** *sign-r-pos*  $(-p*q)$   $x$   
**unfolding** *sign-r-pos-def poly-mult*  
**by** (*metis (lifting, mono-tags) eventually-mono mult-pos-pos*)  
**ultimately show** *?thesis*  
**using** *sign-r-pos-minus*  $\langle p \neq 0 \rangle \langle q \neq 0 \rangle$  **by** (*metis minus-mult-left no-zero-divisors*)  
**qed**  
**moreover have**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p \ z) \Longrightarrow (\forall z. x < z \wedge z < ub' \longrightarrow 0 > \text{poly } q \ z)$   
 $\Longrightarrow$  *?thesis*  
**proof** –  
**assume**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 < \text{poly } p \ z) (\forall z. x < z \wedge z < ub' \longrightarrow 0 > \text{poly } q \ z)$   
**hence** *sign-r-pos*  $p$   $x$  **and** *sign-r-pos*  $(-q)$   $x$  **unfolding** *sign-r-pos-def eventually-at-right*  
**using**  $\langle ub > x \rangle \langle ub' > x \rangle$  **by** *auto*  
**moreover hence** *eventually*  $(\lambda z. \text{poly } p \ z > 0 \wedge \text{poly } (-q) \ z > 0)$  (*at-right*  $x$ )  
**unfolding** *sign-r-pos-def using eventually-conj-iff[of - - at-right x]* **by** *auto*  
**hence** *sign-r-pos*  $(p * (-q))$   $x$   
**unfolding** *sign-r-pos-def poly-mult*  
**by** (*metis (lifting, mono-tags) eventually-mono mult-pos-pos*)  
**ultimately show** *?thesis*  
**using** *sign-r-pos-minus*  $\langle p \neq 0 \rangle \langle q \neq 0 \rangle$   
**by** (*metis minus-mult-right no-zero-divisors*)  
**qed**  
**moreover have**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 > \text{poly } p \ z) \Longrightarrow (\forall z. x < z \wedge z < ub' \longrightarrow 0 > \text{poly } q \ z)$   
 $\Longrightarrow$  *?thesis*  
**proof** –  
**assume**  $(\forall z. x < z \wedge z < ub \longrightarrow 0 > \text{poly } p \ z) (\forall z. x < z \wedge z < ub' \longrightarrow 0 > \text{poly } q \ z)$   
**hence** *sign-r-pos*  $(-p)$   $x$  **and** *sign-r-pos*  $(-q)$   $x$   
**unfolding** *sign-r-pos-def eventually-at-right using*  $\langle ub > x \rangle \langle ub' > x \rangle$  **by** *auto*  
**moreover hence** *eventually*  $(\lambda z. \text{poly } (-p) \ z > 0 \wedge \text{poly } (-q) \ z > 0)$  (*at-right*  $x$ )

**unfolding** *sign-r-pos-def* **using** *eventually-conj-iff[of - - at-right x]* **by** *auto*  
**hence** *sign-r-pos*  $(p * q) x$   
**unfolding** *sign-r-pos-def poly-mult poly-minus*  
**apply** (*elim eventually-mono[of - at-right x]*)  
**by** (*auto intro:mult-neg-neg*)  
**ultimately show** *?thesis*  
**using** *sign-r-pos-minus*  $\langle p \neq 0 \rangle \langle q \neq 0 \rangle$  **by** *metis*  
**qed**  
**ultimately show** *?thesis* **using** *ub ub'* **by** *auto*  
**qed**

**lemma** *sign-r-pos-add*:  
**fixes**  $p q :: \text{real poly}$   
**assumes** *poly p x=0 poly q x≠0*  
**shows** *sign-r-pos*  $(p+q) x = \text{sign-r-pos } q x$   
**proof** (*cases poly (p+q) x=0*)  
**case** *False*  
**hence**  $p+q \neq 0$  **by** (*metis poly-0*)  
**have** *sign-r-pos*  $(p+q) x = (\text{poly } q x > 0)$   
**using** *sign-r-pos-rec[OF*  $\langle p+q \neq 0 \rangle$  *False poly-add*  $\langle \text{poly } p x=0 \rangle$  **by** *auto*  
**moreover have** *sign-r-pos*  $q x = (\text{poly } q x > 0)$   
**using** *sign-r-pos-rec[of q x]*  $\langle \text{poly } q x \neq 0 \rangle$  *poly-0* **by** *force*  
**ultimately show** *?thesis* **by** *auto*  
**next**  
**case** *True*  
**hence** *False* **using**  $\langle \text{poly } p x=0 \rangle \langle \text{poly } q x \neq 0 \rangle$  *poly-add* **by** *auto*  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *sign-r-pos-mod*:  
**fixes**  $p q :: \text{real poly}$   
**assumes** *poly p x=0 poly q x≠0*  
**shows** *sign-r-pos*  $(q \text{ mod } p) x = \text{sign-r-pos } q x$   
**proof** –  
**have** *poly*  $(q \text{ div } p * p) x = 0$  **using**  $\langle \text{poly } p x=0 \rangle$  *poly-mult* **by** *auto*  
**moreover hence** *poly*  $(q \text{ mod } p) x \neq 0$  **using**  $\langle \text{poly } q x \neq 0 \rangle$   
**by** (*simp add: assms(1) poly-mod*)  
**ultimately show** *?thesis*  
**by** (*metis div-mult-mod-eq sign-r-pos-add*)  
**qed**

**lemma** *sign-r-pos-pderiv*:  
**fixes**  $p :: \text{real poly}$   
**assumes** *poly p x=0 p≠0*  
**shows** *sign-r-pos*  $(\text{pderiv } p * p) x$   
**proof** –  
**have** *pderiv p*  $\neq 0$   
**by** (*metis assms(1) assms(2) monoid-add-class.add.right-neutral mult-zero-right pCons-0-0*)

```

    pderiv-iszero poly-0 poly-pCons)
  have ?thesis = (sign-r-pos (pderiv p) x  $\longleftrightarrow$  sign-r-pos p x)
    using sign-r-pos-mult[OF  $\langle$ pderiv p  $\neq$  0 $\rangle$   $\langle$ p $\neq$ 0 $\rangle$ ] by auto
  also have ...=((sign-r-pos (pderiv p) x  $\longleftrightarrow$  sign-r-pos (pderiv p) x))
    using sign-r-pos-rec[OF  $\langle$ p $\neq$ 0 $\rangle$ ]  $\langle$ poly p x=0 $\rangle$  by auto
  finally show ?thesis by auto
qed

```

**lemma** *sign-r-pos-power*:

```

  fixes p: real poly and a::real
  shows sign-r-pos ([: - a, 1:] ^ n) a
proof (induct n)
  case 0
  thus ?case unfolding sign-r-pos-def eventually-at-right by (simp, metis gt-ex)
next
  case (Suc n)
  have pderiv ([: - a, 1:] ^ Suc n) = smult (Suc n) ([: - a, 1:] ^ n)
  proof -
    have pderiv [: - a, 1::real:] = 1 by (simp add: pderiv.simps)
    thus ?thesis unfolding pderiv-power-Suc by (metis mult-cancel-left1)
  qed
  moreover have poly ([: - a, 1:] ^ Suc n) a=0 by (metis old.nat.distinct(2)
poly-power-n-eq)
  hence sign-r-pos ([: - a, 1:] ^ Suc n) a = sign-r-pos (smult (Suc n) ([: - a, 1:] ^ n))
  a
    using sign-r-pos-rec by (metis (erased, opaque-lifting) calculation pderiv-0)
  hence sign-r-pos ([: - a, 1:] ^ Suc n) a = sign-r-pos ([: - a, 1:] ^ n) a
    using sign-r-pos-smult by auto
  ultimately show ?case using Suc.hyps by auto
qed

```

## 2.7 Jump

**definition** *jump-poly* :: real poly  $\Rightarrow$  real poly  $\Rightarrow$  real  $\Rightarrow$  int

where

```

  jump-poly q p x  $\equiv$  (if p $\neq$ 0  $\wedge$  q $\neq$ 0  $\wedge$  odd((order x p) - (order x q)) then
    if sign-r-pos (q*p) x then 1 else -1
    else 0 )

```

**lemma** *jump-poly-not-root*: poly p x $\neq$ 0  $\implies$  jump-poly q p x=0

unfolding jump-poly-def by (metis even-zero order-root zero-diff)

**lemma** *jump-poly0*[simp]:

jump-poly 0 p x = 0

jump-poly q 0 x = 0

unfolding jump-poly-def by auto

**lemma** *jump-poly-smult-1*:

fixes p q::real poly and c::real

```

shows jump-poly (smult c q) p x = sign c * jump-poly q p x (is ?L=?R)
proof (cases c=0 ∨ q=0)
  case True
  thus ?thesis unfolding jump-poly-def by auto
next
  case False
  hence c≠0 and q≠0 by auto
  thus ?thesis unfolding jump-poly-def
    using order-smult[OF ‹c≠0›] sign-r-pos-smult[OF ‹c≠0›, of q*p x] ‹q≠0›
    by auto
qed

lemma jump-poly-mult:
  fixes p q p'::real poly
  assumes p'≠0
  shows jump-poly (p'*q) (p'*p) x = jump-poly q p x
proof (cases q=0 ∨ p=0)
  case True
  thus ?thesis unfolding jump-poly-def by fastforce
next
  case False
  then have q≠0 p≠0 by auto
  have sign-r-pos (p' * q * (p' * p)) x = sign-r-pos (q * p) x
  proof (unfold sign-r-pos-def, rule eventually-subst, unfold eventually-at-right)
    obtain b where b>x and b:∀ z. x < z ∧ z < b ⟶ poly (p' * p') z > 0
    proof (cases ∃ z. poly p' z = 0 ∧ z > x)
      case True
      define lr where lr ≡ Min {r . poly p' r = 0 ∧ r > x}
      have ∀ z. x < z ∧ z < lr ⟶ poly p' z ≠ 0 and lr > x
        using True lr-def poly-roots-finite[OF ‹p'≠0›] by auto
      hence ∀ z. x < z ∧ z < lr ⟶ 0 < poly (p' * p') z
        by (metis not-real-square-gt-zero poly-mult)
      thus ?thesis using that[OF ‹lr>x›] by auto
    next
      case False
      have ∀ z. x < z ∧ z < x+1 ⟶ poly p' z ≠ 0 and x+1 > x
        using False poly-roots-finite[OF ‹p'≠0›] by auto
      hence ∀ z. x < z ∧ z < x+1 ⟶ 0 < poly (p' * p') z
        by (metis not-real-square-gt-zero poly-mult)
      thus ?thesis using that[OF ‹x+1>x›] by auto
    qed
  qed
  show ∃ b > x. ∀ z > x. z < b ⟶ (0 < poly (p' * q * (p' * p)) z) = (0 < poly (q
  * p) z)
  proof (rule-tac x=b in exI, rule conjI[OF ‹b>x›], rule allI, rule impI, rule impI)
    fix z assume x < z z < b
    hence 0 < poly (p'*p') z using b by auto
    have (0 < poly (p' * q * (p' * p)) z) = (0 < poly (p'*p') z * poly (q*p) z)
      by (simp add: mult.commute mult.left-commute)
    also have ... = (0 < poly (q*p) z)

```

```

    using ⟨0 < poly (p' * p') z⟩ by (metis mult-pos-pos zero-less-mult-pos)
    finally show (0 < poly (p' * q * (p' * p)) z) = (0 < poly (q * p) z) .
qed
qed
moreover have odd (order x (p' * p) - order x (p' * q)) = odd (order x p -
order x q)
  using False ⟨p' ≠ 0⟩ ⟨p ≠ 0⟩ mult-eq-0-iff order-mult
  by (metis add-diff-cancel-left)
moreover have p' * q ≠ 0 ↔ q ≠ 0
  by (metis ⟨p' ≠ 0⟩ mult-eq-0-iff)
ultimately show jump-poly (p' * q) (p' * p) x = jump-poly q p x unfolding
jump-poly-def by auto
qed

lemma jump-poly-1-mult:
  fixes p1 p2 :: real poly
  assumes poly p1 x ≠ 0 ∨ poly p2 x ≠ 0
  shows jump-poly 1 (p1 * p2) x = sign (poly p2 x) * jump-poly 1 p1 x
    + sign (poly p1 x) * jump-poly 1 p2 x (is ?L = ?R)
proof (cases p1 = 0 ∨ p2 = 0)
  case True
  then show ?thesis by auto
next
  case False
  then have p1 ≠ 0 p2 ≠ 0 p1 * p2 ≠ 0 by auto
  have ?thesis when poly p1 x ≠ 0
  proof -
    have [simp]: order x p1 = 0 using that order-root by blast
    define simpL where simpL ≡ (if p2 ≠ 0 ∧ odd (order x p2) then if (poly p1
x > 0)
    ↔ sign-r-pos p2 x then 1 :: int else -1 else 0)
    have ?L = simpL
      unfolding simpL-def jump-poly-def
      using order-mult[OF ⟨p1 * p2 ≠ 0⟩]
      sign-r-pos-mult[OF ⟨p1 ≠ 0⟩ ⟨p2 ≠ 0⟩] sign-r-pos-rec[OF ⟨p1 ≠ 0⟩] ⟨poly p1
x ≠ 0⟩
      by auto
    moreover have poly p1 x > 0 ⇒ simpL = ?R
      unfolding simpL-def jump-poly-def using jump-poly-not-root[OF ⟨poly p1
x ≠ 0⟩]
      by auto
    moreover have poly p1 x < 0 ⇒ simpL = ?R
      unfolding simpL-def jump-poly-def using jump-poly-not-root[OF ⟨poly p1
x ≠ 0⟩]
      by auto
    ultimately show ?L = ?R using ⟨poly p1 x ≠ 0⟩ by (metis linorder-neqE-linordered-idom)
  qed
  moreover have ?thesis when poly p2 x ≠ 0
  proof -

```

```

have [simp]: order x p2 = 0 using that order-root by blast
define simpL where simpL ≡ (if p1 ≠ 0 ∧ odd (order x p1) then if (poly p2
x > 0)
  ←→ sign-r-pos p1 x then 1::int else -1 else 0)
have ?L = simpL
  unfolding simpL-def jump-poly-def
  using order-mult[OF ⟨p1 * p2 ≠ 0⟩]
      sign-r-pos-mult[OF ⟨p1 ≠ 0⟩ ⟨p2 ≠ 0⟩] sign-r-pos-rec[OF ⟨p2 ≠ 0⟩] ⟨poly p2
x ≠ 0⟩
  by auto
moreover have poly p2 x > 0 ⇒ simpL = ?R
  unfolding simpL-def jump-poly-def using jump-poly-not-root[OF ⟨poly p2
x ≠ 0⟩]
  by auto
moreover have poly p2 x < 0 ⇒ simpL = ?R
  unfolding simpL-def jump-poly-def using jump-poly-not-root[OF ⟨poly p2
x ≠ 0⟩]
  by auto
ultimately show ?L = ?R using ⟨poly p2 x ≠ 0⟩ by (metis linorder-neqE linordered-idom)
qed
ultimately show ?thesis using assms by auto
qed

```

lemma jump-poly-mod:

```

fixes p q::real poly
shows jump-poly q p x = jump-poly (q mod p) p x
proof (cases q=0 ∨ p=0)
  case True
  thus ?thesis by fastforce
next
  case False
  then have p ≠ 0 q ≠ 0 by auto
  define n where n ≡ min (order x q) (order x p)
  obtain q' where q' = q[: -x, 1:] ^ n * q'
    using n-def power-le-dvd[OF order-1[of x q], of n]
    by (metis dvdE min.cobounded2 min.commute)
  obtain p' where p' = p[: -x, 1:] ^ n * p'
    using n-def power-le-dvd[OF order-1[of x p], of n]
    by (metis dvdE min.cobounded2)
  have q' ≠ 0 and p' ≠ 0 using q' p' ⟨p ≠ 0⟩ ⟨q ≠ 0⟩ by auto
  have order x q' = 0 ∨ order x p' = 0
  proof (rule ccontr)
    assume ¬ (order x q' = 0 ∨ order x p' = 0)
    hence order x q' > 0 and order x p' > 0 by auto
    hence order x q > n and order x p > n unfolding q' p'
    using order-mult[OF ⟨q' ≠ 0⟩, [unfolded q'], of x] order-mult[OF ⟨p' ≠ 0⟩, [unfolded
p'], of x]
    order-power-n-n[of x n]
    by auto

```

```

    thus False using n-def by auto
  qed
  have cond:  $q' \neq 0 \wedge \text{odd}(\text{order } x \ p' - \text{order } x \ q')$ 
    =  $(q' \bmod p' \neq 0 \wedge \text{odd}(\text{order } x \ p' - \text{order } x \ (q' \bmod p')))$ 
  proof (cases order x p'=0)
    case True
      thus ?thesis by (metis  $\langle q' \neq 0 \rangle$  even-zero zero-diff)
    next
      case False
        hence order x q'=0 using  $\langle \text{order } x \ q'=0 \vee \text{order } x \ p'=0 \rangle$  by auto
        hence  $\neg [:-x, 1:] \text{ dvd } q'$ 
          by (metis  $\langle q' \neq 0 \rangle$  order-root poly-eq-0-iff-dvd)
        moreover have  $[:-x, 1:] \text{ dvd } p'$  using False
          by (metis order-root poly-eq-0-iff-dvd)
        ultimately have  $\neg [:-x, 1:] \text{ dvd } (q' \bmod p')$ 
          by (metis dvd-mod-iff)
        hence order x (q' mod p') = 0 and  $q' \bmod p' \neq 0$ 
          apply (metis order-root poly-eq-0-iff-dvd)
          by (metis  $\langle \neg [:-x, 1:] \text{ dvd } q' \bmod p' \rangle$  dvd-0-right)
        thus ?thesis using  $\langle \text{order } x \ q'=0 \rangle$  by auto
      qed
    moreover have  $q' \bmod p' \neq 0 \implies \text{poly } p' \ x = 0$ 
       $\implies \text{sign-r-pos } (q' * p') \ x = \text{sign-r-pos } (q' \bmod p' * p') \ x$ 
    proof -
      assume  $q' \bmod p' \neq 0 \text{ poly } p' \ x = 0$ 
      hence poly q' x ≠ 0 using  $\langle \text{order } x \ q'=0 \vee \text{order } x \ p'=0 \rangle$ 
        by (metis  $\langle p' \neq 0 \rangle \langle q' \neq 0 \rangle$  order-root)
      hence  $\text{sign-r-pos } q' \ x = \text{sign-r-pos } (q' \bmod p') \ x$ 
        using sign-r-pos-mod[OF  $\langle \text{poly } p' \ x = 0 \rangle$ ] by auto
      thus ?thesis
        unfolding sign-r-pos-mult[OF  $\langle q' \neq 0 \rangle \langle p' \neq 0 \rangle$ ] sign-r-pos-mult[OF  $\langle q' \bmod$ 
 $p' \neq 0 \rangle \langle p' \neq 0 \rangle$ ]
        by auto
      qed
    moreover have  $q' \bmod p' = 0 \vee \text{poly } p' \ x \neq 0 \implies \text{jump-poly } q' \ p' \ x = \text{jump-poly}$ 
       $(q' \bmod p') \ p' \ x$ 
    proof -
      assume assm:  $q' \bmod p' = 0 \vee \text{poly } p' \ x \neq 0$ 
      have  $q' \bmod p' = 0 \implies$  ?thesis unfolding jump-poly-def
        using cond by auto
      moreover have poly p' x ≠ 0
         $\implies \neg \text{odd}(\text{order } x \ p' - \text{order } x \ q') \wedge \neg \text{odd}(\text{order } x \ p' - \text{order } x \ (q' \bmod$ 
 $p'))$ 
        by (metis even-zero order-root zero-diff)
      hence  $\text{poly } p' \ x \neq 0 \implies$  ?thesis unfolding jump-poly-def by auto
      ultimately show ?thesis using assm by auto
    qed
    ultimately have  $\text{jump-poly } q' \ p' \ x = \text{jump-poly } (q' \bmod p') \ p' \ x$  unfolding
      jump-poly-def by force

```



**thus** *?thesis* **using**  $p' q'$  *jump-poly-mult* **by** *auto*  
**qed**

**lemma** *jump-poly-coprime*:

**fixes**  $p q$ :: *real poly*  
**assumes** *poly p x=0 coprime p q*  
**shows** *jump-poly q p x = jump-poly 1 (q\*p) x*  
**proof** (*cases p=0  $\vee$  q=0*)  
**case** *True*  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**then have**  $p \neq 0 \ q \neq 0$  **by** *auto*  
**then have** *poly p x $\neq$ 0  $\vee$  poly q x $\neq$ 0* **using** *coprime-poly-0[OF  $\langle$ coprime p  $\rangle$ ]*  
**by** *auto*  
**then have** *poly q x $\neq$ 0* **using**  $\langle$ poly p x=0 $\rangle$  **by** *auto*  
**then have** *order x q=0* **using** *order-root* **by** *blast*  
**then have** *order x p - order x q = order x (q \* p)*  
**using**  $\langle$ p $\neq$ 0 $\rangle$   $\langle$ q $\neq$ 0 $\rangle$  *order-mult [of q p x]* **by** *auto*  
**then show** *?thesis unfolding jump-poly-def* **using**  $\langle$ q $\neq$ 0 $\rangle$  **by** *auto*  
**qed**

**lemma** *jump-poly-sgn*:

**fixes**  $p q$ :: *real poly*  
**assumes**  $p \neq 0$  *poly p x=0*  
**shows** *jump-poly (pderiv p \* q) p x = sign (poly q x)*  
**proof** (*cases q=0*)  
**case** *True*  
**thus** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**have**  $pderiv p \neq 0$  **using**  $\langle$ p $\neq$ 0 $\rangle$   $\langle$ poly p x=0 $\rangle$   
**by** (*metis mult-poly-0-left sign-r-pos-0 sign-r-pos-pderiv*)  
**have** *elim-p-order: order x p - order x (pderiv p \* q)=1 - order x q*  
**proof** -  
**have** *order x p - order x (pderiv p \* q) = order x p - order x (pderiv p) -*  
*order x q*  
**using** *order-mult  $\langle$ pderiv p $\neq$ 0 $\rangle$  False* **by** (*metis diff-diff-left mult-eq-0-iff*)  
**moreover have** *order x p - order x (pderiv p) = 1*  
**using** *order-pderiv[OF  $\langle$ pderiv p $\neq$ 0 $\rangle$ , of x]  $\langle$ poly p x=0 $\rangle$  order-root[of p x]*  
 $\langle$ p $\neq$ 0 $\rangle$  **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**  
**have** *elim-p-sign-r-pos:sign-r-pos (pderiv p \* q \* p) x=sign-r-pos q x*  
**proof** -  
**have** *sign-r-pos (pderiv p \* q \* p) x = (sign-r-pos (pderiv p \* p) x  $\longleftrightarrow$  sign-r-pos*  
*q x)*  
**by** (*metis  $\langle$ q  $\neq$  0 $\rangle$   $\langle$ pderiv p  $\neq$  0 $\rangle$  assms(1) no-zero-divisors sign-r-pos-mult*)  
**thus** *?thesis using sign-r-pos-pderiv[OF  $\langle$ poly p x=0 $\rangle$   $\langle$ p $\neq$ 0 $\rangle$ ]* **by** *auto*

**qed**  
**define** *simpleL* **where** *simpleL*  $\equiv$  if *pderiv p \* q*  $\neq 0 \wedge$  odd (*1 - order x q*) then  
 if *sign-r-pos q x* then *1::int* else *- 1* else *0*  
**have** *jump-poly (pderiv p \* q) p x = simpleL*  
**unfolding** *simpleL-def jump-poly-def* **by** (*subst elim-p-order, subst elim-p-sign-r-pos, simp*)  
**moreover have** *poly q x = 0  $\implies$  simpleL = sign (poly q x)*  
**proof** –  
**assume** *poly q x = 0*  
**hence** *1 - order x q = 0* **using**  $\langle q \neq 0 \rangle$  **by** (*metis less-one not-gr0 order-root zero-less-diff*)  
**hence** *simpleL = 0* **unfolding** *simpleL-def* **by** *auto*  
**moreover have** *sign (poly q x) = 0* **using**  $\langle poly\ q\ x = 0 \rangle$  **by** *auto*  
**ultimately show** *?thesis* **by** *auto*  
**qed**  
**moreover have** *poly q x  $\neq 0 \implies$  simpleL = sign (poly q x)*  
**proof** –  
**assume** *poly q x  $\neq 0$*   
**hence** *odd (1 - order x q)* **by** (*simp add: order-root*)  
**moreover have** *pderiv p \* q  $\neq 0$*  **by** (*metis False  $\langle pderiv\ p \neq 0 \rangle$  no-zero-divisors*)  
**moreover have** *sign-r-pos q x = (poly q x > 0)*  
**using** *sign-r-pos-rec[OF False]  $\langle poly\ q\ x \neq 0 \rangle$*  **by** *auto*  
**ultimately have** *simpleL = (if poly q x > 0 then 1 else - 1)* **unfolding** *simpleL-def* **by** *auto*  
**thus** *?thesis* **using**  $\langle poly\ q\ x \neq 0 \rangle$  **by** *auto*  
**qed**  
**ultimately show** *?thesis* **by** *force*  
**qed**

## 2.8 Cauchy index

**definition** *cindex-poly:: real  $\Rightarrow$  real  $\Rightarrow$  real poly  $\Rightarrow$  real poly  $\Rightarrow$  int  
**where**  
*cindex-poly a b q p*  $\equiv$  ( $\sum x \in \{x. poly\ p\ x = 0 \wedge a < x \wedge x < b\}. jump-poly\ q\ p\ x$ )  
**lemma** *cindex-poly-0[simp]: cindex-poly a b 0 p = 0 cindex-poly a b q 0 = 0*  
**unfolding** *cindex-poly-def* **by** *auto**

**lemma** *cindex-poly-cross:*

**fixes** *p::real poly* **and** *a b::real*  
**assumes** *a < b poly p a  $\neq 0$  poly p b  $\neq 0$*   
**shows** *cindex-poly a b 1 p = cross p a b*  
**using**  $\langle poly\ p\ a \neq 0 \rangle \langle poly\ p\ b \neq 0 \rangle$   
**proof** (*cases  $\{x. poly\ p\ x = 0 \wedge a < x \wedge x < b\} \neq \{\}$ , induct degree p arbitrary:p rule:nat-less-induct*)  
**case 1**  
**then have** *p  $\neq 0$*  **by** *force*  
**define** *roots* **where** *roots*  $\equiv \{x. poly\ p\ x = 0 \wedge a < x \wedge x < b\}$   
**have** *finite roots* **unfolding** *roots-def* **using** *poly-roots-finite[OF  $\langle p \neq 0 \rangle$ ]* **by** *auto*  
**define** *max-r* **where** *max-r*  $\equiv$  *Max roots*

hence  $\text{poly } p \text{ max-r}=0$  and  $a < \text{max-r}$  and  $\text{max-r} < b$   
 using  $\text{Max-in}[OF \langle \text{finite roots} \rangle] 1.\text{prems}$  **unfolding**  $\text{roots-def}$  **by**  $\text{auto}$   
**define**  $\text{max-rp}$  **where**  $\text{max-rp} \equiv [-\text{max-r}, 1:]^{\wedge} \text{order max-r } p$   
**then obtain**  $p'$  **where**  $p' : p = p' * \text{max-rp}$  and  $\text{not-dvd} : \neg [-\text{max-r}, 1:] \text{ dvd } p'$   
**by**  $(\text{metis } \langle p \neq 0 \rangle \text{ mult.commute order-decomp})$   
 hence  $p' \neq 0$  and  $\text{max-rp} \neq 0$  and  $\text{poly } p' a \neq 0$  and  $\text{poly } p' b \neq 0$   
 and  $\text{poly } \text{max-rp } a \neq 0$  and  $\text{poly } \text{max-rp } b \neq 0$   
**using**  $\langle p \neq 0 \rangle \langle \text{poly } p a \neq 0 \rangle \langle \text{poly } p b \neq 0 \rangle$  **by**  $\text{auto}$   
**define**  $\text{max-r-sign}$  **where**  $\text{max-r-sign} \equiv \text{if odd}(\text{order max-r } p) \text{ then } -1 \text{ else } 1 :: \text{int}$   
**define**  $\text{roots}'$  **where**  $\text{roots}' \equiv \{x. a < x \wedge x < b \wedge \text{poly } p' x = 0\}$   
**have**  $(\sum x \in \text{roots}. \text{jump-poly } 1 p x) = (\sum x \in \text{roots}'. \text{jump-poly } 1 p x) + \text{jump-poly } 1 p \text{ max-r}$   
**proof** –  
**have**  $\text{roots} = \text{roots}' \cup \{x. a < x \wedge x < b \wedge \text{poly } \text{max-rp } x = 0\}$   
**unfolding**  $\text{roots-def}$   $\text{roots}'\text{-def}$   $p'$  **by**  $\text{auto}$   
**moreover have**  $\{x. a < x \wedge x < b \wedge \text{poly } \text{max-rp } x = 0\} = \{\text{max-r}\}$   
**unfolding**  $\text{max-rp-def}$  **using**  $\langle \text{poly } p \text{ max-r} = 0 \rangle$   
**by**  $(\text{auto simp add: } \langle a < \text{max-r} \rangle \langle \text{max-r} < b \rangle, \text{metis } 1.\text{prems}(1) \text{ neq0-conv order-root})$   
**moreover hence**  $\text{roots}' \cap \{x. a < x \wedge x < b \wedge \text{poly } \text{max-rp } x = 0\} = \{\}$   
**unfolding**  $\text{roots}'\text{-def}$  **using**  $\langle \neg [-\text{max-r}, 1:] \text{ dvd } p' \rangle$   
**by**  $(\text{metis } (\text{mono-tags}) \text{ Int-insert-right-if0 inf-bot-right mem-Collect-eq poly-eq-0-iff-dvd})$   
**moreover have**  $\text{finite roots}'$   
**using**  $p' \langle p \neq 0 \rangle$  **by**  $(\text{metis } \langle \text{finite roots} \rangle \text{ calculation}(1) \text{ calculation}(2) \text{ finite-Un})$   
**ultimately show**  $?thesis$  **using**  $\text{sum.union-disjoint}$  **by**  $\text{auto}$   
**qed**  
**moreover have**  $(\sum x \in \text{roots}'. \text{jump-poly } 1 p x) = \text{max-r-sign} * \text{cross } p' a b$   
**proof** –  
**have**  $(\sum x \in \text{roots}'. \text{jump-poly } 1 p x) = (\sum x \in \text{roots}'. \text{max-r-sign} * \text{jump-poly } 1 p' x)$   
**proof**  $(\text{rule } \text{sum.cong}, \text{rule } \text{refl})$   
**fix**  $x$  **assume**  $x \in \text{roots}'$   
**hence**  $x \neq \text{max-r}$  **using**  $\text{not-dvd}$  **unfolding**  $\text{roots}'\text{-def}$   
**by**  $(\text{metis } (\text{mono-tags}, \text{lifting}) \text{ mem-Collect-eq poly-eq-0-iff-dvd})$   
**hence**  $\text{poly } \text{max-rp } x \neq 0$  **using**  $\text{poly-power-n-eq}$  **unfolding**  $\text{max-rp-def}$  **by**  $\text{auto}$   
**hence**  $\text{order } x \text{ max-rp} = 0$  **by**  $(\text{metis } \text{order-root})$   
**moreover have**  $\text{jump-poly } 1 \text{ max-rp } x = 0$   
**using**  $\langle \text{poly } \text{max-rp } x \neq 0 \rangle$  **by**  $(\text{metis } \text{jump-poly-not-root})$   
**moreover have**  $x \in \text{roots}$   
**using**  $\langle x \in \text{roots}' \rangle$  **unfolding**  $\text{roots-def}$   $\text{roots}'\text{-def}$   $p'$  **by**  $\text{auto}$   
**hence**  $x < \text{max-r}$   
**using**  $\text{Max-ge}[OF \langle \text{finite roots} \rangle, \text{of } x] \langle x \neq \text{max-r} \rangle$  **by**  $(\text{fold } \text{max-r-def}, \text{auto})$   
**hence**  $\text{sign } (\text{poly } \text{max-rp } x) = \text{max-r-sign}$   
**using**  $\langle \text{poly } \text{max-rp } x \neq 0 \rangle$  **unfolding**  $\text{max-r-sign-def}$   $\text{max-rp-def}$   $\text{sign-def}$   
**by**  $(\text{subst } \text{poly-power}, \text{simp add: } \text{linorder-class.not-less zero-less-power-eq})$   
**ultimately show**  $\text{jump-poly } 1 p x = \text{max-r-sign} * \text{jump-poly } 1 p' x$   
**using**  $\text{jump-poly-1-mult}[\text{of } p' x \text{ max-rp}]$  **unfolding**  $p'$   
**by**  $(\text{simp add: } \langle \text{poly } \text{max-rp } x \neq 0 \rangle)$

```

qed
also have ... = max-r-sign * ( $\sum_{x \in \text{roots}'}$ . jump-poly 1 p' x)
  by (simp add: sum-distrib-left)
also have ... = max-r-sign * cross p' a b
proof (cases roots'={})
  case True
  hence cross p' a b=0 unfolding roots'-def using cross-no-root[OF <a<b>]
by auto
  thus ?thesis using True by simp
next
  case False
  moreover have degree max-rp $\neq$ 0
  unfolding max-rp-def degree-linear-power
  by (metis 1.premis(1) <poly p max-r = 0> order-root)
  hence degree p' < degree p unfolding p' degree-mult-eq[OF <p' $\neq$ 0>
<max-rp $\neq$ 0>]
  by auto
  ultimately have cindex-poly a b 1 p' = cross p' a b
  unfolding roots'-def
  using 1.hyps[rule-format,of degree p' p'] <p' $\neq$ 0> <poly p' a $\neq$ 0> <poly p'
b $\neq$ 0>
  by auto
  moreover have cindex-poly a b 1 p' = sum (jump-poly 1 p') roots'
  unfolding cindex-poly-def roots'-def
  apply simp
  by (metis (no-types, lifting) )
  ultimately show ?thesis by auto
qed
finally show ?thesis .
qed
moreover have max-r-sign * cross p' a b + jump-poly 1 p max-r = cross p a
b (is ?L=?R)
proof (cases odd (order max-r p))
  case True
  have poly max-rp a < 0
  using poly-power-n-odd[OF True,of max-r a] <poly max-rp a $\neq$ 0> <max-r>a>

  unfolding max-rp-def by linarith
  moreover have poly max-rp b > 0
  using poly-power-n-odd[OF True,of max-r b] <max-r<b>
  unfolding max-rp-def by linarith
  ultimately have ?R=cross p' a b + sign (poly p' a)
  unfolding p' cross-def poly-mult
  using variation-mult-neg-1[of poly max-rp a, simplified mult.commute]
  variation-mult-pos(2)[of poly max-rp b, simplified mult.commute] <poly p'
b $\neq$ 0>
  by auto
  moreover have ?L=- cross p' a b + sign (poly p' b)
proof -

```

**have**  $\text{sign-r-pos } p' \text{ max-r} = (\text{poly } p' \text{ max-r} > 0)$   
**using**  $\text{sign-r-pos-rec}[OF \langle p' \neq 0 \rangle]$  **not-dvd** **by**  $(\text{metis poly-eq-0-iff-dvd})$   
**moreover have**  $(\text{poly } p' \text{ max-r} > 0) = (\text{poly } p' b > 0)$   
**proof**  $(\text{rule ccontr})$   
**assume**  $(0 < \text{poly } p' \text{ max-r}) \neq (0 < \text{poly } p' b)$   
**hence**  $\text{poly } p' \text{ max-r} * \text{poly } p' b < 0$   
**using**  $\langle \text{poly } p' b \neq 0 \rangle$  **not-dvd** $[\text{folded poly-eq-0-iff-dvd}]$   
**by**  $(\text{metis } (\text{poly-guards-query}) \text{ linorder-neqE-linordered-idom mult-less-0-iff})$   
**then obtain**  $r$  **where**  $r > \text{max-r}$  **and**  $r < b$  **and**  $\text{poly } p' r = 0$   
**using**  $\text{poly-IVT}[OF \langle \text{max-r} < b \rangle]$  **by**  $\text{auto}$   
**hence**  $r \in \text{roots}$  **unfolding**  $\text{roots-def } p'$  **using**  $\langle \text{max-r} > a \rangle$  **by**  $\text{auto}$   
**thus**  $\text{False}$  **using**  $\langle r > \text{max-r} \rangle$   $\text{Max-ge}[OF \langle \text{finite roots} \rangle, \text{of } r]$  **unfolding**  
 $\text{max-r-def}$  **by**  $\text{auto}$   
**qed**  
**moreover have**  $\text{sign-r-pos max-rp max-r}$   
**using**  $\text{sign-r-pos-power}$  **unfolding**  $\text{max-rp-def}$  **by**  $\text{auto}$   
**ultimately show**  $?thesis$   
**using**  $\text{True } \langle \text{poly } p' b \neq 0 \rangle \langle \text{max-rp} \neq 0 \rangle \langle p' \neq 0 \rangle$   $\text{sign-r-pos-mult}[OF \langle p' \neq 0 \rangle$   
 $\langle \text{max-rp} \neq 0 \rangle]$   
**unfolding**  $\text{max-r-sign-def } p' \text{ jump-poly-def}$   
**by**  $\text{simp}$   
**qed**  
**moreover have**  $\text{variation } (\text{poly } p' a) (\text{poly } p' b) + \text{sign } (\text{poly } p' a)$   
 $= - \text{variation } (\text{poly } p' a) (\text{poly } p' b) + \text{sign } (\text{poly } p' b)$  **unfolding**  $\text{cross-def}$   
**by**  $(\text{cases } \text{poly } p' b \text{ rule:linorder-cases}[\text{of } 0], (\text{cases } \text{poly } p' a \text{ rule:linorder-cases}[\text{of}$   
 $0],$   
 $\text{auto } \text{simp } \text{add:variation-cases } \langle \text{poly } p' a \neq 0 \rangle \langle \text{poly } p' b \neq 0 \rangle)$   
**ultimately show**  $?thesis$  **unfolding**  $\text{cross-def}$  **by**  $\text{auto}$   
**next**  
**case**  $\text{False}$   
**hence**  $\text{poly max-rp } a > 0$  **and**  $\text{poly max-rp } b > 0$   
**unfolding**  $\text{max-rp-def poly-power}$   
**using**  $\langle \text{poly max-rp } a \neq 0 \rangle \langle \text{poly max-rp } b \neq 0 \rangle$   $1.\text{prems}(1-2)$   $\langle \text{poly } p \text{ max-r}$   
 $= 0 \rangle$   
**apply**  $(\text{unfold zero-less-power-eq})$   
**by**  $\text{auto}$   
**moreover have**  $\text{poly max-rp } b > 0$   
**unfolding**  $\text{max-rp-def poly-power}$   
**using**  $\langle \text{poly max-rp } b \neq 0 \rangle$   $\text{False max-rp-def poly-power}$   
 $\text{zero-le-even-power}[\text{of order max-r } p \text{ } b - \text{max-r}]$   
**by**  $(\text{auto } \text{simp } \text{add: le-less})$   
**ultimately have**  $?R = \text{cross } p' a b$   
**apply**  $(\text{simp only: } p' \text{ mult commute cross-def})$  **using**  $\text{variation-mult-pos}$   
**by**  $\text{auto}$   
**thus**  $?thesis$  **unfolding**  $\text{max-r-sign-def jump-poly-def}$  **using**  $\text{False}$  **by**  $\text{auto}$   
**qed**  
**ultimately have**  $\text{sum } (\text{jump-poly } 1 \text{ } p) \text{ roots} = \text{cross } p a b$  **by**  $\text{auto}$   
**then show**  $?case$  **unfolding**  $\text{roots-def cindex-poly-def}$  **by**  $\text{simp}$   
**next**

**case** *False*  
**hence** *cross p a b=0* **using** *cross-no-root[OF ‹a<b›]* **by** *auto*  
**thus** *?thesis* **using** *False unfolding cindex-poly-def* **by** (*metis sum.empty*)  
**qed**

**lemma** *cindex-poly-mult*:  
**fixes** *p q p'::real poly*  
**assumes** *p' ≠ 0*  
**shows** *cindex-poly a b (p' \* q) (p' \* p) = cindex-poly a b q p*  
**proof** (*cases p=0*)  
**case** *True*  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**show** *?thesis* **unfolding** *cindex-poly-def*  
**apply** (*rule sum.mono-neutral-cong-right*)  
**subgoal** **using** *‹p≠0› ‹p'≠0›* **by** (*simp add: poly-roots-finite*)  
**subgoal** **by** *auto*  
**subgoal** **using** *jump-poly-mult jump-poly-not-root assms* **by** *fastforce*  
**subgoal** **for** *x* **using** *jump-poly-mult[OF ‹p'≠0›]* **by** *auto*  
**done**  
**qed**

**lemma** *cindex-poly-smult-1*:  
**fixes** *p q::real poly and c::real*  
**shows** *cindex-poly a b (smult c q) p = (sign c) \* cindex-poly a b q p*  
**unfolding** *cindex-poly-def*  
**using** *sum-distrib-left[THEN sym, of sign c λx. jump-poly q p x*  
*{x. poly p x = (0::real) ∧ a < x ∧ x < b}] jump-poly-smult-1*  
**by** *auto*

**lemma** *cindex-poly-mod*:  
**fixes** *p q::real poly*  
**shows** *cindex-poly a b q p = cindex-poly a b (q mod p) p*  
**unfolding** *cindex-poly-def* **using** *jump-poly-mod* **by** *auto*

**lemma** *cindex-poly-inverse-add*:  
**fixes** *p q::real poly*  
**assumes** *coprime p q*  
**shows** *cindex-poly a b q p + cindex-poly a b p q = cindex-poly a b 1 (q\*p)*  
*(is ?L=?R)*  
**proof** (*cases p=0 ∨ q=0*)  
**case** *True*  
**then show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**then have** *p≠0 q≠0* **by** *auto*  
**define** *A* **where** *A ≡ {x. poly p x = 0 ∧ a < x ∧ x < b}*  
**define** *B* **where** *B ≡ {x. poly q x = 0 ∧ a < x ∧ x < b}*

**have**  $?L = \text{sum } (\lambda x. \text{jump-poly } 1 (q * p) x) A + \text{sum } (\lambda x. \text{jump-poly } 1 (q * p) x) B$   
**proof** –  
**have**  $\text{cindex-poly } a b q p = \text{sum } (\lambda x. \text{jump-poly } 1 (q * p) x) A$  **unfolding**  $A\text{-def}$   
 $\text{cindex-poly-def}$   
**using**  $\text{jump-poly-coprime}[OF - \langle \text{coprime } p q \rangle]$  **by**  $\text{auto}$   
**moreover** **have**  $\text{coprime } q p$  **using**  $\langle \text{coprime } p q \rangle$   
**by**  $(\text{simp add: ac-simps})$   
**hence**  $\text{cindex-poly } a b p q = \text{sum } (\lambda x. \text{jump-poly } 1 (q * p) x) B$  **unfolding**  $B\text{-def}$   
 $\text{cindex-poly-def}$   
**using**  $\text{jump-poly-coprime} [of q - p]$  **by**  $(\text{auto simp add: ac-simps})$   
**ultimately show**  $?thesis$  **by**  $\text{auto}$   
**qed**  
**moreover** **have**  $A \cup B = \{x. \text{poly } (q * p) x = 0 \wedge a < x \wedge x < b\}$  **unfolding**  $\text{poly-mult}$   
 $A\text{-def } B\text{-def}$  **by**  $\text{auto}$   
**moreover** **have**  $A \cap B = \{\}$   
**proof**  $(\text{rule ccontr})$   
**assume**  $A \cap B \neq \{\}$   
**then obtain**  $x$  **where**  $x \in A$  **and**  $x \in B$  **by**  $\text{auto}$   
**hence**  $\text{poly } p x = 0$  **and**  $\text{poly } q x = 0$  **unfolding**  $A\text{-def } B\text{-def}$  **by**  $\text{auto}$   
**hence**  $\text{gcd } p q \neq 1$  **by**  $(\text{metis poly-1 poly-eq-0-iff-dvd gcd-greatest zero-neq-one})$   
**thus**  $\text{False}$  **using**  $\langle \text{coprime } p q \rangle$  **by**  $\text{auto}$   
**qed**  
**moreover** **have**  $\text{finite } A$  **and**  $\text{finite } B$   
**unfolding**  $A\text{-def } B\text{-def}$  **using**  $\text{poly-roots-finite} \langle p \neq 0 \rangle \langle q \neq 0 \rangle$  **by**  $\text{fast+}$   
**ultimately** **have**  $\text{cindex-poly } a b q p + \text{cindex-poly } a b p q$   
 $= \text{sum } (\text{jump-poly } 1 (q * p)) \{x. \text{poly } (q * p) x = 0 \wedge a < x \wedge x < b\}$   
**using**  $\text{sum.union-disjoint}$  **by**  $\text{metis}$   
**then show**  $?thesis$  **unfolding**  $\text{cindex-poly-def}$  **by**  $\text{auto}$   
**qed**

**lemma**  $\text{cindex-poly-inverse-add-cross}$ :  
**fixes**  $p q :: \text{real poly}$   
**assumes**  $a < b$   $\text{poly } (p * q) a \neq 0$   $\text{poly } (p * q) b \neq 0$   
**shows**  $\text{cindex-poly } a b q p + \text{cindex-poly } a b p q = \text{cross } (p * q) a b$  **(is**  $?L = ?R$ **)**  
**proof** –  
**have**  $p \neq 0$  **and**  $q \neq 0$  **using**  $\langle \text{poly } (p * q) a \neq 0 \rangle$  **by**  $\text{auto}$   
**define**  $g$  **where**  $g \equiv \text{gcd } p q$   
**obtain**  $p' q'$  **where**  $p' : p = p' * g$  **and**  $q' : q = q' * g$   
**using**  $\text{gcd-dvd1 gcd-dvd2 dvd-def}[of gcd p q, \text{simplified mult.commute}]$   $g\text{-def}$  **by**  
 $\text{metis}$   
**hence**  $\text{coprime } p' q'$  **using**  $\text{gcd-coprime} \langle p \neq 0 \rangle$  **unfolding**  $g\text{-def}$  **by**  $\text{auto}$   
**have**  $p' \neq 0$   $q' \neq 0$   $g \neq 0$  **using**  $p' q' \langle p \neq 0 \rangle \langle q \neq 0 \rangle$  **by**  $\text{auto}$   
**have**  $?L = \text{cindex-poly } a b q' p' + \text{cindex-poly } a b p' q'$   
**apply**  $(\text{simp only: } p' q' \text{ mult.commute})$   
**using**  $\text{cindex-poly-mult}[OF \langle q \neq 0 \rangle]$   $\text{cindex-poly-mult}[OF \langle g \neq 0 \rangle]$   
**by**  $\text{auto}$   
**also** **have**  $\dots = \text{cindex-poly } a b 1 (q' * p')$   
**using**  $\text{cindex-poly-inverse-add}[OF \langle \text{coprime } p' q' \rangle, of a b]$  .

```

also have ... = cross (p' * q') a b
  using cindex-poly-cross[OF ⟨a<b⟩, of q'*p'] ⟨p'≠0⟩ ⟨q'≠0⟩
    ⟨poly (p * q) a ≠0⟩ ⟨poly (p * q) b ≠0⟩
  unfolding p' q'
  apply (subst (2) mult.commute)
  by auto
also have ... = ?R
proof -
  have poly (p * q) a = poly (g*g) a * poly (p' * q') a
    and poly (p * q) b = poly (g*g) b * poly (p' * q') b
  unfolding p' q' by auto
  moreover have poly g a ≠0 using ⟨poly (p * q) a ≠0⟩
  unfolding p' by auto
  hence poly (g*g) a >0
  by (metis (poly-guards-query) not-real-square-gt-zero poly-mult)
  moreover have poly g b ≠0 using ⟨poly (p * q) b ≠0⟩
  unfolding p' by auto
  hence poly (g*g) b >0 by (metis (poly-guards-query) not-real-square-gt-zero poly-mult)
  ultimately show ?thesis
  unfolding cross-def using variation-mult-pos by auto
qed
finally show ?L = ?R .
qed

```

```

lemma cindex-poly-rec:
  fixes p q::real poly
  assumes a < b poly (p * q) a ≠0 poly (p * q) b ≠0
  shows cindex-poly a b q p = cross (p * q) a b + cindex-poly a b (- (p mod q))
  q (is ?L=?R)
proof -
  have q≠0 using ⟨poly (p * q) a ≠0⟩ by auto
  note cindex-poly-inverse-add-cross[OF assms]
  moreover have - cindex-poly a b p q = cindex-poly a b (- (p mod q)) q
    using cindex-poly-mod cindex-poly-smult-1[of a b -1]
    by auto
  ultimately show ?thesis by auto
qed

```

```

lemma cindex-poly-congr:
  fixes p q:: real poly
  assumes a<a' a'<b' b'<b
  assumes ∀ x. ((a<x∧x≤a') ∨ (b'≤x ∧ x<b)) → poly p x ≠0
  shows cindex-poly a b q p = cindex-poly a' b' q p
proof (cases p=0)
  case True
  then show ?thesis by auto
next
  case False

```



```

show ?thesis unfolding cindex-poly-def
  apply (rule sum.mono-neutral-right)
  subgoal using poly-roots-finite[OF ‹p≠0›] by auto
  subgoal using assms by auto
  subgoal using assms(4) by fastforce
  done
qed

lemma greaterThanLessThan-unfold:{a<..b} = {x. a<x ∧ x<b}
  by fastforce

lemma cindex-poly-taq:
  fixes p q::real poly
  shows taq {x. poly p x = 0 ∧ a < x ∧ x < b} q=cindex-poly a b (pderiv p * q) p
proof (cases p=0)
  case True
  define S where S={x. poly p x = 0 ∧ a < x ∧ x < b}
  have ?thesis when a≥b
  proof –
    have S = {} using that unfolding S-def by auto
    then show ?thesis using True unfolding taq-def by (fold S-def,simp)
  qed
  moreover have ?thesis when a<b
  proof –
    have infinite {x. a<x ∧ x<b} using infinite-Ioo[OF ‹a<bunfolding greaterThanLessThan-unfold by simp
    then have infinite S unfolding S-def using True by auto
    then show ?thesis using True unfolding taq-def by (fold S-def,simp)
  qed
  ultimately show ?thesis by fastforce
next
  case False
  show ?thesis
    unfolding cindex-poly-def taq-def
    by (rule sum.cong,auto simp add:jump-poly-sgn[OF ‹p≠0›])
qed

```

## 2.9 Signed remainder sequence

```

function smods:: real poly ⇒ real poly ⇒ (real poly) list where
  smods p q = (if p=0 then [] else Cons p (smods q (-(p mod q))))
by auto
termination
  apply (relation measure (λ(p,q).if p=0 then 0 else if q=0 then 1 else 2+degree
q))
  apply simp-all
  apply (metis degree-mod-less)
done

```

**lemma** *smods-nil-eq*:  $\text{smods } p \ q = [] \iff (p=0)$  **by** *auto*  
**lemma** *smods-singleton*:  $[x] = \text{smods } p \ q \implies (p \neq 0 \wedge q=0 \wedge x=p)$   
**by** (*metis list.discI list.inject smods.elims*)

**lemma** *smods-0*[*simp*]:  
 $\text{smods } 0 \ q = []$   
 $\text{smods } p \ 0 = (\text{if } p=0 \text{ then } [] \text{ else } [p])$   
**by** *auto*

**lemma** *no-0-in-smods*:  $0 \notin \text{set } (\text{smods } p \ q)$   
**apply** (*induct smods p q arbitrary:p q*)  
**by** (*simp,metis list.inject neq-Nil-conv set-ConsD smods.elims*)

**fun** *changes*:: ('a :: *linordered-idom*) *list*  $\Rightarrow$  *int* **where**  
 $\text{changes } [] = 0$   
 $\text{changes } [-] = 0$  |  
 $\text{changes } (x1 \# x2 \# xs) = (\text{if } x1 * x2 < 0 \text{ then } 1 + \text{changes } (x2 \# xs)$   
 $\quad \text{else if } x2 = 0 \text{ then } \text{changes } (x1 \# xs)$   
 $\quad \text{else } \text{changes } (x2 \# xs))$

**lemma** *changes-map-sgn-eq*:  
 $\text{changes } xs = \text{changes } (\text{map } \text{sgn } xs)$   
**proof** (*induct xs rule:changes.induct*)  
**case** ( $\exists x1 \ x2 \ xs$ )  
**moreover** **have**  $x1 * x2 < 0 \iff \text{sgn } x1 * \text{sgn } x2 < 0$   
**by** (*unfold mult-less-0-iff sgn-less sgn-greater, simp*)  
**moreover** **have**  $x2 = 0 \iff \text{sgn } x2 = 0$  **by** (*rule sgn-0-0[symmetric]*)  
**ultimately** **show** *?case* **by** *auto*  
**qed** *simp-all*

**lemma** *changes-map-sign-eq*:  
 $\text{changes } xs = \text{changes } (\text{map } \text{sign } xs)$   
**proof** (*induct xs rule:changes.induct*)  
**case** ( $\exists x1 \ x2 \ xs$ )  
**moreover** **have**  $x1 * x2 < 0 \iff \text{sign } x1 * \text{sign } x2 < 0$   
**by** (*simp add: mult-less-0-iff sign-def*)  
**moreover** **have**  $x2 = 0 \iff \text{sign } x2 = 0$  **by** (*simp add: sign-def*)  
**ultimately** **show** *?case* **by** *auto*  
**qed** *simp-all*

**lemma** *changes-map-sign-of-int-eq*:  
 $\text{changes } xs = \text{changes } (\text{map } ((\text{of-int}::\Rightarrow 'c::\{\text{ring-1, linordered-idom}\}) \circ \text{sign}) \ xs)$   
**proof** (*induct xs rule:changes.induct*)  
**case** ( $\exists x1 \ x2 \ xs$ )  
**moreover** **have**  $x1 * x2 < 0 \iff$   
 $((\text{of-int}::\Rightarrow 'c::\{\text{ring-1, linordered-idom}\}) \circ \text{sign}) \ x1$   
 $* ((\text{of-int}::\Rightarrow 'c::\{\text{ring-1, linordered-idom}\}) \circ \text{sign}) \ x2 < 0$   
**by** (*simp add: mult-less-0-iff sign-def*)  
**moreover** **have**  $x2 = 0 \iff (\text{of-int } \circ \text{sign}) \ x2 = 0$  **by** (*simp add: sign-def*)

**ultimately show** *?case by auto*  
**qed** *simp-all*

**definition** *changes-poly-at::('a ::linordered-idom) poly list  $\Rightarrow$  'a  $\Rightarrow$  int where*  
*changes-poly-at ps a = changes (map ( $\lambda p$ . poly p a) ps)*

**definition** *changes-poly-pos-inf:: ('a ::linordered-idom) poly list  $\Rightarrow$  int where*  
*changes-poly-pos-inf ps = changes (map sgn-pos-inf ps)*

**definition** *changes-poly-neg-inf:: ('a ::linordered-idom) poly list  $\Rightarrow$  int where*  
*changes-poly-neg-inf ps = changes (map sgn-neg-inf ps)*

**lemma** *changes-poly-at-0[simp]:*  
*changes-poly-at [] a = 0*  
*changes-poly-at [p] a = 0*

**unfolding** *changes-poly-at-def by auto*

**definition** *changes-itv-smods:: real  $\Rightarrow$  real  $\Rightarrow$  real poly  $\Rightarrow$  real poly  $\Rightarrow$  int where*  
*changes-itv-smods a b p q = (let ps = smods p q in changes-poly-at ps a - changes-poly-at ps b)*

**definition** *changes-gt-smods:: real  $\Rightarrow$  real poly  $\Rightarrow$  real poly  $\Rightarrow$  int where*  
*changes-gt-smods a p q = (let ps = smods p q in changes-poly-at ps a - changes-poly-pos-inf ps)*

**definition** *changes-le-smods:: real  $\Rightarrow$  real poly  $\Rightarrow$  real poly  $\Rightarrow$  int where*  
*changes-le-smods b p q = (let ps = smods p q in changes-poly-neg-inf ps - changes-poly-at ps b)*

**definition** *changes-R-smods:: real poly  $\Rightarrow$  real poly  $\Rightarrow$  int where*  
*changes-R-smods p q = (let ps = smods p q in changes-poly-neg-inf ps - changes-poly-pos-inf ps)*

**lemma** *changes-R-smods-0[simp]:*  
*changes-R-smods 0 q = 0*  
*changes-R-smods p 0 = 0*

**unfolding** *changes-R-smods-def changes-poly-neg-inf-def changes-poly-pos-inf-def*  
**by** *auto*

**lemma** *changes-itv-smods-0[simp]:*  
*changes-itv-smods a b 0 q = 0*  
*changes-itv-smods a b p 0 = 0*

**unfolding** *changes-itv-smods-def*  
**by** *auto*

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

**assumes** *a < b poly (p\*q) a  $\neq$  0 poly (p\*q) b  $\neq$  0*

**shows** *changes-itv-smods a b p q = cross (p\*q) a b + changes-itv-smods a b q*  
*(-(p mod q))*

**proof** (*cases*  $p=0 \vee q=0 \vee p \bmod q = 0$ )  
**case** *True*  
**moreover have**  $p=0 \vee q=0 \implies ?thesis$   
**unfolding** *changes-itv-smods-def changes-poly-at-def* **by** (*erule HOL.disjE,auto*)  
**moreover have**  $p \bmod q = 0 \implies ?thesis$   
**unfolding** *changes-itv-smods-def changes-poly-at-def cross-def*  
**apply** (*insert assms(2,3)*)  
**apply** (*subst (asm) (1 2) neq-iff*)  
**by** (*auto simp add: variation-cases*)  
**ultimately show** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**hence**  $p \neq 0 \ q \neq 0 \ p \bmod q \neq 0$  **by** *auto*  
**then obtain** *ps* **where**  $ps:smods \ p \ q = p \# q \# -(p \bmod q) \# ps \ smods \ q \ (- (p \bmod q)) = q \# -(p \bmod q) \# ps$   
**by** *auto*  
**define** *changes-diff* **where**  $changes-diff \equiv \lambda x. \ changes-poly-at \ (p \# q \# -(p \bmod q) \# ps) \ x$   
–  $changes-poly-at \ (q \# -(p \bmod q) \# ps) \ x$   
**have**  $\bigwedge x. \ poly \ p \ x * poly \ q \ x < 0 \implies changes-diff \ x = 1$   
**unfolding** *changes-diff-def changes-poly-at-def* **by** *auto*  
**moreover have**  $\bigwedge x. \ poly \ p \ x * poly \ q \ x > 0 \implies changes-diff \ x = 0$   
**unfolding** *changes-diff-def changes-poly-at-def* **by** *auto*  
**ultimately have**  $changes-diff \ a - changes-diff \ b = cross \ (p * q) \ a \ b$   
**unfolding** *cross-def*  
**apply** (*cases rule:neqE[OF <poly (p\*q) a ≠ 0>]*)  
**by** (*cases rule:neqE[OF <poly (p\*q) b ≠ 0>],auto simp add:variation-cases*) +  
**thus** *?thesis* **unfolding** *changes-itv-smods-def changes-diff-def changes-poly-at-def*  
  
**using** *ps* **by** *auto*  
**qed**

**lemma** *changes-smods-congr*:  
**fixes**  $p \ q :: \text{real } poly$   
**assumes**  $a \neq a' \ poly \ p \ a \neq 0$   
**assumes**  $\forall p \in set \ (smods \ p \ q). \ \forall x. \ ((a < x \wedge x \leq a') \vee (a' \leq x \wedge x < a)) \longrightarrow poly \ p \ x \neq 0$   
**shows**  $changes-poly-at \ (smods \ p \ q) \ a = changes-poly-at \ (smods \ p \ q) \ a'$   
**using** *assms(2-3)*  
**proof** (*induct smods p q arbitrary:p q rule:length-induct*)  
**case** *1*  
**have**  $p \neq 0$  **using** *<poly p a ≠ 0>* **by** *auto*  
**define** *r1* **where**  $r1 \equiv - \ (p \bmod q)$   
**have**  $a - a' - rel: \forall pp \in set \ (smods \ p \ q). \ poly \ pp \ a * poly \ pp \ a' \geq 0$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\forall pp \in set \ (smods \ p \ q). \ 0 \leq poly \ pp \ a * poly \ pp \ a')$   
**then obtain** *pp* **where**  $pp:pp \in set \ (smods \ p \ q) \ poly \ pp \ a * poly \ pp \ a' < 0$   
**using** *<p ≠ 0>* **by** (*metis less-eq-real-def linorder-neqE-linordered-idom*)  
**hence**  $a < a' \implies False$  **using** *1.premis(2) poly-IVT[of a a' pp]* **by** *auto*

**moreover have**  $a' < a \implies \text{False}$   
**using**  $pp[\text{unfolded mult.commute}[of\ poly\ pp\ a]]\ 1.premis(2)\ poly-IVT[of\ a'\ a\ pp]$  **by** *auto*  
**ultimately show** *False* **using**  $\langle a \neq a' \rangle$  **by** *force*  
**qed**  
**have**  $q=0 \implies ?case$  **by** *auto*  
**moreover have**  $\llbracket q \neq 0; poly\ q\ a=0 \rrbracket \implies ?case$   
**proof** –  
**assume**  $q \neq 0\ poly\ q\ a=0$   
**define**  $r2$  **where**  $r2 \equiv - (q \bmod r1)$   
**have** –  $poly\ r1\ a = poly\ p\ a$   
**by** (*metis*  $\langle poly\ q\ a = 0 \rangle\ add.inverse-inverse\ add.left-neutral\ div-mult-mod-eq$   
 $mult-zero-right\ poly-add\ poly-minus\ poly-mult\ r1-def$ )  
**hence**  $r1 \neq 0$  **and**  $poly\ r1\ a \neq 0$  **and**  $poly\ p\ a * poly\ r1\ a < 0$  **using**  $\langle poly\ p\ a \neq 0 \rangle$   
**apply** *auto*  
**using** *mult-less-0-iff* **by** *fastforce*  
**then obtain**  $ps$  **where**  $ps: smods\ p\ q = p \# q \# r1 \# ps\ smods\ r1\ r2 = r1 \# ps$   
**by** (*metis*  $\langle p \neq 0 \rangle\ \langle q \neq 0 \rangle\ r1-def\ r2-def\ smods.simps$ )  
**hence**  $length\ (smods\ r1\ r2) < length\ (smods\ p\ q)$  **by** *auto*  
**moreover have**  $(\forall p \in set\ (smods\ r1\ r2). \forall x. a < x \wedge x \leq a' \vee a' \leq x \wedge x < a \longrightarrow poly\ p\ x \neq 0)$   
**using**  $1.premis(2)$  **unfolding**  $ps$  **by** *auto*  
**ultimately have**  $changes-poly-at\ (smods\ r1\ r2)\ a = changes-poly-at\ (smods\ r1\ r2)\ a'$   
**using**  $1.hyps\ \langle r1 \neq 0 \rangle\ \langle poly\ r1\ a \neq 0 \rangle$  **by** *metis*  
**moreover have**  $changes-poly-at\ (smods\ p\ q)\ a = 1 + changes-poly-at\ (smods\ r1\ r2)\ a$   
**unfolding**  $ps\ changes-poly-at-def$  **using**  $\langle poly\ q\ a=0 \rangle\ \langle poly\ p\ a * poly\ r1\ a < 0 \rangle$   
**by** *auto*  
**moreover have**  $changes-poly-at\ (smods\ p\ q)\ a' = 1 + changes-poly-at\ (smods\ r1\ r2)\ a'$   
**proof** –  
**have**  $poly\ p\ a * poly\ p\ a' \geq 0$  **and**  $poly\ r1\ a * poly\ r1\ a' \geq 0$   
**using** *a-a'-rel* **unfolding**  $ps$  **by** *auto*  
**moreover have**  $poly\ p\ a' \neq 0$  **and**  $poly\ q\ a' \neq 0$  **and**  $poly\ r1\ a' \neq 0$   
**using**  $1.premis(2)[\text{unfolded}\ ps]\ \langle a \neq a' \rangle$  **by** *auto*  
**ultimately show** *?thesis* **using**  $\langle poly\ p\ a * poly\ r1\ a < 0 \rangle$  **unfolding**  $ps\ changes-poly-at-def$   
**by** (*auto simp add: zero-le-mult-iff, auto simp add: mult-less-0-iff*)  
**qed**  
**ultimately show** *?thesis* **by** *simp*  
**qed**  
**moreover have**  $\llbracket q \neq 0; poly\ q\ a \neq 0 \rrbracket \implies ?case$   
**proof** –  
**assume**  $q \neq 0\ poly\ q\ a \neq 0$   
**then obtain**  $ps$  **where**  $ps: smods\ p\ q = p \# q \# ps\ smods\ q\ r1 = q \# ps$   
**by** (*metis*  $\langle p \neq 0 \rangle\ r1-def\ smods.simps$ )  
**hence**  $length\ (smods\ q\ r1) < length\ (smods\ p\ q)$  **by** *auto*

**moreover have**  $(\forall p \in \text{set } (\text{smods } q \ r1). \forall x. a < x \wedge x \leq a' \vee a' \leq x \wedge x < a \longrightarrow \text{poly } p \ x \neq 0)$   
**using**  $1.\text{prems}(2)$  **unfolding**  $ps$  **by**  $\text{auto}$   
**ultimately have**  $\text{changes-poly-at } (\text{smods } q \ r1) \ a = \text{changes-poly-at } (\text{smods } q \ r1) \ a'$   
**using**  $1.\text{hyps } \langle q \neq 0 \rangle \langle \text{poly } q \ a \neq 0 \rangle$  **by**  $\text{metis}$   
**moreover have**  $\text{poly } p \ a' \neq 0$  **and**  $\text{poly } q \ a' \neq 0$   
**using**  $1.\text{prems}(2)[\text{unfolded } ps]$   $\langle a \neq a' \rangle$  **by**  $\text{auto}$   
**moreover have**  $\text{poly } p \ a * \text{poly } p \ a' \geq 0$  **and**  $\text{poly } q \ a * \text{poly } q \ a' \geq 0$   
**using**  $a\text{-}a'\text{-rel}$  **unfolding**  $ps$  **by**  $\text{auto}$   
**ultimately show**  $?thesis$  **unfolding**  $ps$   $\text{changes-poly-at-def}$  **using**  $\langle \text{poly } q \ a \neq 0 \rangle \langle \text{poly } p \ a \neq 0 \rangle$   
**by**  $(\text{auto simp add: zero-le-mult-iff, auto simp add: mult-less-0-iff})$   
**qed**  
**ultimately show**  $?case$  **by**  $\text{blast}$   
**qed**

**lemma**  $\text{changes-itv-smods-congr}$ :

**fixes**  $p \ q :: \text{real poly}$   
**assumes**  $a < a' \ a' < b' \ b' < b \ \text{poly } p \ a \neq 0 \ \text{poly } p \ b \neq 0$   
**assumes**  $\text{no-root: } \forall p \in \text{set } (\text{smods } p \ q). \forall x. ((a < x \wedge x \leq a') \vee (b' \leq x \wedge x < b)) \longrightarrow \text{poly } p \ x \neq 0$   
**shows**  $\text{changes-itv-smods } a \ b \ p \ q = \text{changes-itv-smods } a' \ b' \ p \ q$   
**proof** –  
**have**  $\text{changes-poly-at } (\text{smods } p \ q) \ a = \text{changes-poly-at } (\text{smods } p \ q) \ a'$   
**apply**  $(\text{rule } \text{changes-smods-congr}[OF \ \text{order.strict-implies-not-eq}[OF \ \langle a < a' \rangle] \langle \text{poly } p \ a \neq 0 \rangle])$   
**by**  $(\text{metis } \text{assms}(1) \ \text{less-eq-real-def} \ \text{less-irrefl} \ \text{less-trans} \ \text{no-root})$   
**moreover have**  $\text{changes-poly-at } (\text{smods } p \ q) \ b = \text{changes-poly-at } (\text{smods } p \ q) \ b'$   
**apply**  $(\text{rule } \text{changes-smods-congr}[OF \ \text{order.strict-implies-not-eq}[OF \ \langle b' < b \rangle, \ \text{symmetric}] \langle \text{poly } p \ b \neq 0 \rangle])$   
**by**  $(\text{metis } \text{assms}(3) \ \text{less-eq-real-def} \ \text{less-trans} \ \text{no-root})$   
**ultimately show**  $?thesis$  **unfolding**  $\text{changes-itv-smods-def}$   $\text{Let-def}$  **by**  $\text{auto}$   
**qed**

**lemma**  $\text{cindex-poly-changes-itv-mods}$ :

**assumes**  $a < b \ \text{poly } p \ a \neq 0 \ \text{poly } p \ b \neq 0$   
**shows**  $\text{cindex-poly } a \ b \ q \ p = \text{changes-itv-smods } a \ b \ p \ q$  **using**  $\text{assms}$   
**proof**  $(\text{induct } \text{smods } p \ q \ \text{arbitrary: } p \ q \ a \ b)$   
**case**  $\text{Nil}$   
**hence**  $p = 0$  **by**  $(\text{metis } \text{smods-nil-eq})$   
**thus**  $?case$  **using**  $\langle \text{poly } p \ a \neq 0 \rangle$  **by**  $\text{simp}$   
**next**  
**case**  $(\text{Cons } x1 \ xs)$   
**have**  $p \neq 0$  **using**  $\langle \text{poly } p \ a \neq 0 \rangle$  **by**  $\text{auto}$   
**obtain**  $a' \ b'$  **where**  $a < a' \ a' < b' \ b' < b$   
**and**  $\text{no-root: } \forall p \in \text{set } (\text{smods } p \ q). \forall x. ((a < x \wedge x \leq a') \vee (b' \leq x \wedge x < b)) \longrightarrow \text{poly } p \ x \neq 0$   
**proof**  $(\text{induct } \text{smods } p \ q \ \text{arbitrary: } p \ q \ \text{thesis})$

**case Nil**  
**define**  $a' b'$  **where**  $a' \equiv 2/3 * a + 1/3 * b$  **and**  $b' \equiv 1/3 * a + 2/3 * b$   
**have**  $a < a'$  **and**  $a' < b'$  **and**  $b' < b$  **unfolding**  $a'$ -def  $b'$ -def **using**  $\langle a < b \rangle$  **by**  
*auto*  
**moreover have**  $\forall p \in \text{set } (smods \ p \ q). \forall x. a < x \wedge x \leq a' \vee b' \leq x \wedge x < b$   
 $\longrightarrow poly \ p \ x \neq 0$   
**unfolding**  $\langle [] = smods \ p \ q \rangle$  [symmetric] **by** *auto*  
**ultimately show** ?case **using** Nil **by** *auto*  
**next**  
**case** (Cons  $x1 \ xs$ )  
**define**  $r$  **where**  $r \equiv - (p \ \text{mod} \ q)$   
**then have**  $smods \ p \ q = p \ \# \ xs$  **and**  $smods \ q \ r = xs$  **and**  $p \neq 0$   
**using**  $\langle x1 \ \# \ xs = smods \ p \ q \rangle$   
**by** (*auto simp del: smods.simps simp add: smods.simps [of p q] split: if-splits*)  
**obtain**  $a1 \ b1$  **where**  
 $a < a1 \ a1 < b1 \ b1 < b$  **and**  
 $a1\text{-}b1\text{-no-root} : \forall p \in \text{set } xs. \forall x. a < x \wedge x \leq a1 \vee b1 \leq x \wedge x < b \longrightarrow poly$   
 $p \ x \neq 0$   
**using** Cons(1) [OF  $\langle smods \ q \ r = xs \rangle$ ] [symmetric]  $\langle smods \ q \ r = xs \rangle$  **by** *auto*  
**obtain**  $a2 \ b2$  **where**  
 $a < a2$  **and**  $a2 : \forall x. a < x \wedge x \leq a2 \longrightarrow poly \ p \ x \neq 0$   
 $b2 < b$  **and**  $b2 : \forall x. b2 \leq x \wedge x < b \longrightarrow poly \ p \ x \neq 0$   
**using** next-non-root-interval [OF  $\langle p \neq 0 \rangle$ ] last-non-root-interval [OF  $\langle p \neq 0 \rangle$ ]  
**by** (*metis less-numeral-extra (3)*)  
**define**  $a' b'$  **where**  $a' \equiv$  if  $b2 > a$  then  $\text{Min}\{a1, b2, a2\}$  else  $\text{min } a1 \ a2$   
**and**  $b' \equiv$  if  $a2 < b$  then  $\text{Max}\{b1, a2, b2\}$  else  $\text{max } b1 \ b2$   
**have**  $a < a' \ a' < b' \ b' < b$  **unfolding**  $a'$ -def  $b'$ -def  
**using**  $\langle a < a1 \rangle \langle a1 < b1 \rangle \langle b1 < b \rangle \langle a < a2 \rangle \langle b2 < b \rangle \langle a < b \rangle$  **by** *auto*  
**moreover have**  $\forall p \in \text{set } xs. \forall x. a < x \wedge x \leq a' \vee b' \leq x \wedge x < b \longrightarrow poly \ p$   
 $x \neq 0$   
**using**  $a1\text{-}b1\text{-no-root}$  **unfolding**  $a'$ -def  $b'$ -def **by** *auto*  
**moreover have**  $\forall x. a < x \wedge x \leq a' \vee b' \leq x \wedge x < b \longrightarrow poly \ p \ x \neq 0$   
**using**  $a2 \ b2$  **unfolding**  $a'$ -def  $b'$ -def **by** *auto*  
**ultimately show** ?case **using** Cons(3) [unfolded  $\langle smods \ p \ q = p \ \# \ xs \rangle$ ] **by** *auto*  
**qed**  
**have**  $q = 0 \implies ?case$  **by** *simp*  
**moreover have**  $q \neq 0 \implies ?case$   
**proof** –  
**assume**  $q \neq 0$   
**define**  $r$  **where**  $r \equiv - (p \ \text{mod} \ q)$   
**obtain**  $ps$  **where**  $ps : smods \ p \ q = p \ \# \ q \ \# \ ps$   $smods \ q \ r = q \ \# \ ps$  **and**  $xs = q \ \# \ ps$   
**unfolding**  $r$ -def **using**  $\langle q \neq 0 \rangle \langle p \neq 0 \rangle \langle x1 \ \# \ xs = smods \ p \ q \rangle$   
**by** (*metis list.inject smods.simps*)  
**have**  $poly \ p \ a' \neq 0 \ poly \ p \ b' \neq 0 \ poly \ q \ a' \neq 0 \ poly \ q \ b' \neq 0$   
**using** no-root [unfolded  $ps$ ]  $\langle a' > a \rangle \langle b' < b \rangle$  **by** *auto*  
**moreover hence**  
 $changes\text{-}itv\text{-smods } a' \ b' \ p \ q = cross \ (p * q) \ a' \ b' + changes\text{-}itv\text{-smods } a' \ b'$   
 $q \ r$   
 $cindex\text{-}poly \ a' \ b' \ q \ p = cross \ (p * q) \ a' \ b' + cindex\text{-}poly \ a' \ b' \ r \ q$

**using** *changes-itv-smods-rec*[*OF*  $\langle a' < b' \rangle$ , of  $p$  *q*, folded *r-def*]  
*cindex-poly-rec*[*OF*  $\langle a' < b' \rangle$ , of  $p$  *q*, folded *r-def*] **by** *auto*  
**moreover have** *changes-itv-smods*  $a' b' q r = \text{cindex-poly } a' b' r q$   
**using** *Cons.hyps(1)*[of  $q r a' b'$ ]  $\langle a' < b' \rangle \langle q \neq 0 \rangle \langle xs = q \# ps \rangle ps(2)$   
 $\langle \text{poly } q a' \neq 0 \rangle \langle \text{poly } q b' \neq 0 \rangle$  **by** *simp*  
**ultimately have** *changes-itv-smods*  $a' b' p q = \text{cindex-poly } a' b' q p$  **by** *auto*  
**thus** *?thesis*  
**using**  
*changes-itv-smods-congr*[*OF*  $\langle a < a' \rangle \langle a' < b' \rangle \langle b' < b \rangle$  *Cons(4,5)*, of  $q$ ]  
*no-root cindex-poly-congr*[*OF*  $\langle a < a' \rangle \langle a' < b' \rangle \langle b' < b \rangle$ ]  $ps$   
**by** (*metis insert-iff list.set(2)*)  
**qed**  
**ultimately show** *?case* **by** *metis*  
**qed**

**lemma** *root-list-ub*:

**fixes**  $ps:: (\text{real poly}) \text{ list}$  **and**  $a:: \text{real}$   
**assumes**  $0 \notin \text{set } ps$   
**obtains**  $ub$  **where**  $\forall p \in \text{set } ps. \forall x. \text{poly } p x = 0 \longrightarrow x < ub$   
**and**  $\forall x \geq ub. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p x) = \text{sgn-pos-inf } p$  **and**  $ub > a$   
**using** *assms*  
**proof** (*induct ps arbitrary:thesis*)  
**case** *Nil*  
**show** *?case* **using** *Nil(1)*[of  $a+1$ ] **by** *auto*  
**next**  
**case** (*Cons p ps*)  
**hence**  $p \neq 0$  **and**  $0 \notin \text{set } ps$  **by** *auto*  
**then obtain**  $ub1$  **where**  $ub1: \forall p \in \text{set } ps. \forall x. \text{poly } p x = 0 \longrightarrow x < ub1$  **and**  
 $ub1\text{-sgn}: \forall x \geq ub1. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p x) = \text{sgn-pos-inf } p$  **and**  $ub1 > a$   
**using** *Cons.hyps* **by** *auto*  
**obtain**  $ub2$  **where**  $ub2: \forall x. \text{poly } p x = 0 \longrightarrow x < ub2$   
**and**  $ub2\text{-sgn}: \forall x \geq ub2. \text{sgn } (\text{poly } p x) = \text{sgn-pos-inf } p$   
**using** *root-ub*[*OF*  $\langle p \neq 0 \rangle$ ] **by** *auto*  
**define**  $ub$  **where**  $ub \equiv \max ub1 ub2$   
**have**  $\forall p \in \text{set } (p \# ps). \forall x. \text{poly } p x = 0 \longrightarrow x < ub$  **using**  $ub1 ub2$  *ub-def* **by**  
*force*  
**moreover have**  $\forall x \geq ub. \forall p \in \text{set } (p \# ps). \text{sgn } (\text{poly } p x) = \text{sgn-pos-inf } p$   
**using**  $ub1\text{-sgn } ub2\text{-sgn } ub\text{-def}$  **by** *auto*  
**ultimately show** *?case* **using** *Cons(2)*[of  $ub$ ]  $\langle ub1 > a \rangle$  *ub-def* **by** *auto*  
**qed**

**lemma** *root-list-lb*:

**fixes**  $ps:: (\text{real poly}) \text{ list}$  **and**  $b:: \text{real}$   
**assumes**  $0 \notin \text{set } ps$   
**obtains**  $lb$  **where**  $\forall p \in \text{set } ps. \forall x. \text{poly } p x = 0 \longrightarrow x > lb$   
**and**  $\forall x \leq lb. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p x) = \text{sgn-neg-inf } p$  **and**  $lb < b$   
**using** *assms*  
**proof** (*induct ps arbitrary:thesis*)  
**case** *Nil*



**show** *?case* **using** *Nil(1)[of b - 1]* **by** *auto*  
**next**  
**case** (*Cons p ps*)  
**hence**  $p \neq 0$  **and**  $0 \notin \text{set } ps$  **by** *auto*  
**then obtain** *lb1* **where**  $lb1: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb1$  **and**  
 $lb1\text{-sgn}: \forall x \leq lb1. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$  **and**  $lb1 < b$   
**using** *Cons.hyps* **by** *auto*  
**obtain** *lb2* **where**  $lb2: \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb2$   
**and**  $lb2\text{-sgn}: \forall x \leq lb2. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$   
**using** *root-lb[OF <p≠0>]* **by** *auto*  
**define** *lb* **where**  $lb \equiv \min \ lb1 \ lb2$   
**have**  $\forall p \in \text{set } (p \ \# \ ps). \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb$  **using** *lb1 lb2 lb-def* **by** *force*  
**moreover have**  $\forall x \leq lb. \forall p \in \text{set } (p \ \# \ ps). \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$   
**using** *lb1-sgn lb2-sgn lb-def* **by** *auto*  
**ultimately show** *?case* **using** *Cons(2)[of lb] <lb1 < b>* *lb-def* **by** *auto*  
**qed**

**theorem** *sturm-tarski-interval*:

**assumes**  $a < b$  *poly p a ≠ 0 poly p b ≠ 0*  
**shows**  $\text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < b\}$   $q = \text{changes-itv-smods } a \ b \ p \ (\text{pderiv } p \ * \ q)$   
**proof** –  
**have**  $p \neq 0$  **using** *<poly p a ≠ 0>* **by** *auto*  
**thus** *?thesis* **using** *cindex-poly-taq cindex-poly-changes-itv-smods[OF assms]* **by**  
*auto*  
**qed**

**theorem** *sturm-tarski-above*:

**assumes** *poly p a ≠ 0*  
**shows**  $\text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x\}$   $q = \text{changes-gt-smods } a \ p \ (\text{pderiv } p \ * \ q)$   
**proof** –  
**define** *ps* **where**  $ps \equiv \text{smods } p \ (\text{pderiv } p \ * \ q)$   
**have**  $p \neq 0$  **and**  $p \in \text{set } ps$  **using** *<poly p a ≠ 0>* *ps-def* **by** *auto*  
**obtain** *ub* **where**  $ub: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x < ub$   
**and**  $ub\text{-sgn}: \forall x \geq ub. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-pos-inf } p$   
**and**  $ub > a$   
**using** *root-list-ub[OF no-0-in-smods, of p pderiv p \* q, folded ps-def]*  
**by** *auto*  
**have**  $\text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x\}$   $q = \text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < ub\}$   $q$   
**unfolding** *taq-def* **by** (*rule sum.cong, insert ub <p ∈ set ps>, auto*)  
**moreover have**  $\text{changes-gt-smods } a \ p \ (\text{pderiv } p \ * \ q) = \text{changes-itv-smods } a \ ub \ p$   
 $(\text{pderiv } p \ * \ q)$   
**proof** –  
**have**  $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \ ub)) \ ps = \text{map } \text{sgn-pos-inf } ps$   
**using** *ub-sgn[THEN spec, of ub, simplified]*  
**by** (*metis (mono-tags, lifting) comp-def list.map-cong0*)  
**hence**  $\text{changes-poly-at } ps \ ub = \text{changes-poly-pos-inf } ps$   
**unfolding** *changes-poly-pos-inf-def changes-poly-at-def*  
**by** (*subst changes-map-sgn-eq, metis map-map*)

**thus** *?thesis* **unfolding** *changes-gt-smods-def changes-itv-smods-def ps-def*  
**by** *metis*  
**qed**  
**moreover** **have** *poly p ub ≠ 0* **using** *ub ⟨p ∈ set ps⟩* **by** *auto*  
**ultimately show** *?thesis* **using** *sturm-tarski-interval[OF ⟨ub > a⟩ assms]* **by** *auto*  
**qed**

**theorem** *sturm-tarski-below*:

**assumes** *poly p b ≠ 0*  
**shows** *taq {x. poly p x = 0 ∧ x < b} q = changes-le-smods b p (pderiv p \* q)*  
**proof** –  
**define** *ps* **where** *ps ≡ smods p (pderiv p \* q)*  
**have** *p ≠ 0* **and** *p ∈ set ps* **using** *⟨poly p b ≠ 0⟩ ps-def* **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 < b*  
**using** *root-list-lb[OF no-0-in-smods, of p pderiv p \* q, folded ps-def]*  
**by** *auto*  
**have** *taq {x. poly p x = 0 ∧ x < b} q = taq {x. poly p x = 0 ∧ lb < x ∧ x < b} q*  
**unfolding** *taq-def* **by** *(rule sum.cong, insert lb ⟨p ∈ set ps⟩, auto)*  
**moreover** **have** *changes-le-smods b p (pderiv p \* q) = changes-itv-smods lb b p*  
*(pderiv p \* q)*  
**proof** –  
**have** *map (sgn ∘ (λp. poly p lb)) ps = map sgn-neg-inf ps*  
**using** *lb-sgn[THEN spec, of lb, simplified]*  
**by** *(metis (mono-tags, lifting) comp-def list.map-cong0)*  
**hence** *changes-poly-at ps lb = changes-poly-neg-inf ps*  
**unfolding** *changes-poly-neg-inf-def changes-poly-at-def*  
**by** *(subst changes-map-sgn-eq, metis map-map)*  
**thus** *?thesis* **unfolding** *changes-le-smods-def changes-itv-smods-def ps-def*  
**by** *metis*  
**qed**  
**moreover** **have** *poly p lb ≠ 0* **using** *lb ⟨p ∈ set ps⟩* **by** *auto*  
**ultimately show** *?thesis* **using** *sturm-tarski-interval[OF ⟨lb < b⟩ - assms]* **by**  
*auto*  
**qed**

**theorem** *sturm-tarski-R*:

**shows** *taq {x. poly p x = 0} q = changes-R-smods p (pderiv p \* q)*  
**proof** *(cases p = 0)*  
**case** *True*  
**then show** *?thesis*  
**unfolding** *taq-def* **using** *infinite-UNIV-char-0* **by** *(auto intro!: sum.infinite)*  
**next**  
**case** *False*  
**define** *ps* **where** *ps ≡ smods p (pderiv 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 pderiv p \* q, folded ps-def*]  
**by** *auto*  
**obtain**  $ub$  **where**  $ub: \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x < ub$   
**and**  $ub\text{-sgn}: \forall x \geq ub. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-pos-inf } p$   
**and**  $ub > 0$   
**using** *root-list-ub*[*OF no-0-in-smods, of p pderiv p \* q, folded ps-def*]  
**by** *auto*  
**have**  $\text{taq } \{x. \text{poly } p \ x = 0\} \ q = \text{taq } \{x. \text{poly } p \ x = 0 \wedge lb < x \wedge x < ub\} \ q$   
**unfolding** *taq-def* **by** (*rule sum.cong, insert lb ub <p ∈ set ps>, auto*)  
**moreover** **have**  $\text{changes-R-smods } p \ (\text{pderiv } p * q) = \text{changes-itv-smods } lb \ ub \ p$   
(*pderiv p \* q*)  
**proof** –  
**have**  $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \ lb)) \ ps = \text{map } \text{sgn-neg-inf } ps$   
**and**  $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \ ub)) \ ps = \text{map } \text{sgn-pos-inf } ps$   
**using** *lb-sgn*[*THEN spec, of lb, simplified*] *ub-sgn*[*THEN spec, of ub, simplified*]  
**by** (*metis (mono-tags, lifting) comp-def list.map-cong0*) +  
**hence**  $\text{changes-poly-at } ps \ lb = \text{changes-poly-neg-inf } ps$   
 $\wedge \text{changes-poly-at } ps \ ub = \text{changes-poly-pos-inf } ps$   
**unfolding** *changes-poly-neg-inf-def changes-poly-at-def changes-poly-pos-inf-def*  
**by** (*subst (1 3) changes-map-sgn-eq, metis map-map*)  
**thus** *?thesis* **unfolding** *changes-R-smods-def changes-itv-smods-def ps-def*  
**by** *metis*  
**qed**  
**moreover** **have**  $\text{poly } p \ lb \neq 0$  **and**  $\text{poly } p \ ub \neq 0$  **using**  $lb \ ub \ \langle p \in \text{set } ps \rangle$  **by** *auto*  
**moreover** **have**  $lb < ub$  **using**  $\langle lb < 0 \rangle \ \langle 0 < ub \rangle$  **by** *auto*  
**ultimately show** *?thesis* **using** *sturm-tarski-interval* **by** *auto*  
**qed**

**theorem** *sturm-interval*:

**assumes**  $a < b$   $\text{poly } p \ a \neq 0$   $\text{poly } p \ b \neq 0$   
**shows**  $\text{card } \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < b\} = \text{changes-itv-smods } a \ b \ p \ (\text{pderiv } p)$   
**using** *sturm-tarski-interval*[*OF assms, unfolded taq-def, of 1*] **by** *force*

**theorem** *sturm-above*:

**assumes**  $\text{poly } p \ a \neq 0$   
**shows**  $\text{card } \{x. \text{poly } p \ x = 0 \wedge a < x\} = \text{changes-gt-smods } a \ p \ (\text{pderiv } p)$   
**using** *sturm-tarski-above*[*OF assms, unfolded taq-def, of 1*] **by** *force*

**theorem** *sturm-below*:

**assumes**  $\text{poly } p \ b \neq 0$   
**shows**  $\text{card } \{x. \text{poly } p \ x = 0 \wedge x < b\} = \text{changes-le-smods } b \ p \ (\text{pderiv } p)$   
**using** *sturm-tarski-below*[*OF assms, unfolded taq-def, of 1*] **by** *force*

**theorem** *sturm-R*:

**shows**  $\text{card } \{x. \text{poly } p \ x = 0\} = \text{changes-R-smods } p \ (\text{pderiv } p)$   
**using** *sturm-tarski-R*[*of - 1, unfolded taq-def*] **by** *force*

end

### 3 An implementation for calculating pseudo remainder sequences

**theory** *Pseudo-Remainder-Sequence*

**imports** *Sturm-Tarski*

*HOL-Computational-Algebra.Computational-Algebra*

*Polynomial-Interpolation.Ring-Hom-Poly*

**begin**

#### 3.1 Misc

**function** *smods* :: 'a::idom poly  $\Rightarrow$  'a poly  $\Rightarrow$  ('a poly) list **where**

*smods* p q = (if p=0 then [] else

let

*m*=(if even(degree p+1-degree q) then -1 else -lead-coeff q)

in

Cons p (*smods* q (*smult* m (pseudo-mod p q))))

**by** *auto*

**termination**

**apply** (relation measure ( $\lambda(p,q).$ if p=0 then 0 else if q=0 then 1 else 2+degree q))

**by** (*simp-all* add: degree-pseudo-mod-less)

**declare** *smods.simps*[*simp del*]

**lemma** *smods-0*[*simp*]:

*smods* 0 q = []

*smods* p 0 = (if p=0 then [] else [p])

**by** (*auto simp:smods.simps*)

**lemma** *smods-nil-eq*:*smods* p q = []  $\longleftrightarrow$  (p=0)

**by** (*metis list.distinct(1) smods.elims*)

**lemma** *changes-poly-at-alternative*:

*changes-poly-at* ps a = *changes* (map ( $\lambda p.$  sign(poly p a)) ps)

*changes-poly-at* ps a = *changes* (map ( $\lambda p.$  sgn(poly p a)) ps)

**unfolding** *changes-poly-at-def*

**subgoal by** (*subst changes-map-sign-eq*) (*auto simp add:comp-def*)

**subgoal by** (*subst changes-map-sgn-eq*) (*auto simp add:comp-def*)

**done**

**lemma** *smods-smult-length*:

**assumes** a $\neq$ 0 b $\neq$ 0

**shows** length (*smods* p q) = length (*smods* (*smult* a p) (*smult* b q)) **using** *assms*

```

proof (induct smods p q arbitrary:p q a b )
  case Nil
  thus ?case by (simp split:if-splits)
next
  case (Cons x xs)
  hence p≠0 by auto
  define r where r≡- (p mod q)
  have smods q r = xs using Cons.hyps(2) ⟨p≠0⟩ unfolding r-def by auto
  hence length (smods q r) = length (smods (smult b q) (smult a r))
    using Cons.hyps(1)[of q r b a] Cons by auto
  moreover have smult a p≠0 using ⟨a≠0⟩ ⟨p≠0⟩ by auto
  moreover have -((smult a p) mod (smult b q)) = (smult a r)
    by (simp add: Cons.prem(2) mod-smult-left mod-smult-right r-def)
  ultimately show ?case
    unfolding r-def by auto
qed

lemma smods-smult-nth[rule-format]:
  fixes p q::real poly
  assumes a≠0 b≠0
  defines xs≡smods p q and ys≡smods (smult a p) (smult b q)
  shows ∀ n<length xs. ys!n = (if even n then smult a (xs!n) else smult b (xs!n))
using assms
proof (induct smods p q arbitrary:p q a b xs ys)
  case Nil
  thus ?case by (simp split:if-splits)
next
  case (Cons x xs)
  hence p≠0 by auto
  define r where r≡- (p mod q)
  have xs:xs=smods q r p#xs=smods p q using Cons.hyps(2) ⟨p≠0⟩ unfolding
r-def by auto
  define ys where ys≡smods (smult b q) (smult a r)
  have - ((smult a p) mod (smult b q)) = smult a r
    by (simp add: Cons.hyps(4) mod-smult-left mod-smult-right r-def)
  hence ys:smult a p # ys = smods (smult a p) (smult b q) using ⟨p≠0⟩ ⟨a≠0⟩
    unfolding ys-def r-def by auto
  have hyps:∧n. n<length xs ⇒ ys ! n = (if even n then smult b (xs ! n) else
smult a (xs ! n))
    using Cons.hyps(1)[of q r b a,folded xs ys-def] ⟨a≠0⟩ ⟨b≠0⟩ by auto
  thus ?case
    apply (fold xs ys)
    apply auto
    by (case-tac n,auto)+
qed

lemma smods-smult-sgn-map-eq:
  fixes x::real
  assumes m>0

```

```

defines  $f \equiv \lambda p. \text{sgn}(\text{poly } p \ x)$ 
shows  $\text{map } f \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } f \ (\text{smods } p \ q)$ 
 $\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q)$ 
 $\text{map } \text{sgn-neg-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-neg-inf} \ (\text{smods } p \ q)$ 
proof –
define  $xs \ ys$  where  $xs \equiv \text{smods } p \ q$  and  $ys \equiv \text{smods } p \ (\text{smult } m \ q)$ 
have  $m \neq 0$  using  $\langle m > 0 \rangle$  by simp
have  $\text{len-eq}:\text{length } xs = \text{length } ys$ 
using  $\text{smods-smult-length}[\text{of } 1 \ m] \ \langle m > 0 \rangle$  unfolding  $xs\text{-def } ys\text{-def}$  by auto
moreover have
 $(\text{map } f \ xs) ! i = (\text{map } f \ ys) ! i$ 
 $(\text{map } \text{sgn-pos-inf} \ xs) ! i = (\text{map } \text{sgn-pos-inf} \ ys) ! i$ 
 $(\text{map } \text{sgn-neg-inf} \ xs) ! i = (\text{map } \text{sgn-neg-inf} \ ys) ! i$ 
when  $i < \text{length } xs$  for  $i$ 
proof –
note  $\text{nth-eq} = \text{smods-smult-nth}[\text{OF one-neg-zero } \langle m \neq 0 \rangle, \text{of } - \ p \ q, \text{unfolded smult-1-left,}$ 
 $\text{folded } xs\text{-def } ys\text{-def, OF } \langle i < \text{length } xs \rangle ]$ 
then show  $\text{map } f \ xs ! i = \text{map } f \ ys ! i$ 
 $(\text{map } \text{sgn-pos-inf} \ xs) ! i = (\text{map } \text{sgn-pos-inf} \ ys) ! i$ 
 $(\text{map } \text{sgn-neg-inf} \ xs) ! i = (\text{map } \text{sgn-neg-inf} \ ys) ! i$ 
using that
unfolding  $f\text{-def}$  using  $\text{len-eq } \langle m > 0 \rangle$ 
by (auto simp add:sgn-mult sgn-pos-inf-def sgn-neg-inf-def lead-coeff-smult)
qed
ultimately show  $\text{map } f \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } f \ (\text{smods } p \ q)$ 
 $\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q)$ 
 $\text{map } \text{sgn-neg-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-neg-inf} \ (\text{smods } p \ q)$ 
apply (fold xs-def ys-def)
by (auto intro: nth-equalityI)
qed

```

**lemma** *changes-poly-at-smods-smult:*

**assumes**  $m > 0$

**shows**  $\text{changes-poly-at} \ (\text{smods } p \ (\text{smult } m \ q)) \ x = \text{changes-poly-at} \ (\text{smods } p \ q) \ x$

**using**  $\text{smods-smult-sgn-map-eq}[\text{OF } \langle m > 0 \rangle]$

**by** (*metis changes-poly-at-alternative(2)*)

**lemma** *spmods-smods-sgn-map-eq:*

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

**defines**  $f \equiv \lambda p. \text{sgn} \ (\text{poly } p \ x)$

**shows**  $\text{map } f \ (\text{smods } p \ q) = \text{map } f \ (\text{spmods } p \ q)$

$\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q) = \text{map } \text{sgn-pos-inf} \ (\text{spmods } p \ q)$

$\text{map } \text{sgn-neg-inf} \ (\text{smods } p \ q) = \text{map } \text{sgn-neg-inf} \ (\text{spmods } p \ q)$

**proof** (*induct spmods p q arbitrary:p q*)

**case** *Nil*

**hence**  $p = 0$  **using**  $\text{spmods-nil-eq}$  **by** *metis*

**thus**  $\text{map } f \ (\text{smods } p \ q) = \text{map } f \ (\text{spmods } p \ q)$

$\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q) = \text{map } \text{sgn-pos-inf} \ (\text{spmods } p \ q)$

```

      map sgn-neg-inf (smods p q) = map sgn-neg-inf (spmods p q)
    by auto
next
case (Cons p' xs)
hence p≠0 by auto
define r where r≡- (p mod q)
define exp where exp≡degree p + 1 - degree q
define m where m≡(if even exp then 1 else lead-coeff q)
  * (lead-coeff q ^ exp)
have xs1:p#xs=spmods p q
  by (metis (no-types) Cons.hyps(4) list.distinct(1) list.inject spmods.simps)
have xs2:xs=spmods q (smult m r) when q≠0
proof -
  define m' where m'≡if even exp then - 1 else - lead-coeff q
  have smult m' (pseudo-mod p q) = smult m r
    unfolding m-def m'-def r-def
    apply (subst pseudo-mod-mod[symmetric])
    using that exp-def by auto
  thus ?thesis using ⟨p≠0⟩ xs1 unfolding r-def
    by (simp add:spmods.simps[of p q,folded exp-def, folded m'-def] del:spmods.simps)
qed
define ys where ys≡smods q r
have ys:p#ys=smods p q using ⟨p≠0⟩ unfolding ys-def r-def by auto
have qm:q≠0 ⇒ m>0
  using ⟨p≠0⟩ unfolding m-def
  apply auto
  subgoal by (simp add: zero-less-power-eq)
  subgoal using zero-less-power-eq by fastforce
  done
show map f (smods p q) = map f (spmods p q)
proof (cases q≠0)
case True
  then have map f (spmods q (smult m r)) = map f (smods q r)
    using smods-smult-sgn-map-eq(1)[of m x q r,folded f-def] qm
    Cons.hyps(1)[OF xs2,folded f-def]
  by simp
  thus ?thesis
    apply (fold xs1 xs2[OF True] ys ys-def)
    by auto
next
case False
  thus ?thesis by auto
qed
show map sgn-pos-inf (smods p q) = map sgn-pos-inf (spmods p q)
proof (cases q≠0)
case True
  then have map sgn-pos-inf (spmods q (smult m r)) = map sgn-pos-inf (smods
q r)
    using Cons.hyps(2)[OF xs2,folded f-def] qm[OF True]

```

```

      smods-smult-sgn-map-eq(2)[of m q r, folded f-def] by auto
    thus ?thesis
    apply (fold xs1 xs2[OF True] ys ys-def)
    by (simp add:f-def)
  next
    case False
    thus ?thesis by auto
  qed
show map sgn-neg-inf (smods p q) = map sgn-neg-inf (spmods p q)
proof (cases q≠0)
  case True
  then have map sgn-neg-inf (spmods q (smult m r)) = map sgn-neg-inf (smods
q r)
    using Cons.hyps(3)[OF xs2, folded f-def] qm[OF True]
    smods-smult-sgn-map-eq(3)[of m q r, folded f-def] by auto
  thus ?thesis
  apply (fold xs1 xs2[OF True] ys ys-def)
  by (simp add:f-def)
next
  case False
  thus ?thesis by auto
qed
qed

```

### 3.2 Converting *rat poly* to *int poly* by clearing the denominators

**definition** *int-of-rat::rat*  $\Rightarrow$  *int* **where**  
*int-of-rat* = *inv of-int*

**lemma** *of-rat-inj*[*simp*]: *inj of-rat*  
**by** (*simp add: linorder-injI*)

**lemma** (**in** *ring-char-0*) *of-int-inj*[*simp*]: *inj of-int*  
**by** (*simp add: inj-on-def*)

**lemma** *int-of-rat-id*: *int-of-rat o of-int* = *id*  
**unfolding** *int-of-rat-def*  
**by** *auto*

**lemma** *int-of-rat-0*[*simp*]: *int-of-rat 0* = *0*  
**by** (*metis id-apply int-of-rat-id o-def of-int-0*)

**lemma** *int-of-rat-inv*:  $r \in \mathbb{Z} \Rightarrow$  *of-int* (*int-of-rat r*) = *r*  
**unfolding** *int-of-rat-def*  
**by** (*simp add: Ints-def f-inv-into-f*)

**lemma** *int-of-rat-0-iff*:  $x \in \mathbb{Z} \Rightarrow$  *int-of-rat x* = *0*  $\iff$  *x* = *0*  
**using** *int-of-rat-inv* **by** *force*



**lemma** [code]:int-of-rat  $r = (\text{let } (a,b) = \text{quotient-of } r \text{ in}$   
 $\text{if } b=1 \text{ then } a \text{ else Code.abort (STR "Failed to convert rat to int")}$   
 $(\lambda-. \text{int-of-rat } r))$   
**apply** (auto simp add:split-beta int-of-rat-def)  
**by** (metis Fract-of-int-quotient inv-f-eq of-int-inj of-int-rat quotient-of-div surjective-pairing)

**definition** de-lcm::rat poly  $\Rightarrow$  int **where**  
 $\text{de-lcm } p = \text{Lcm}(\text{set}(\text{map } (\lambda x. \text{snd } (\text{quotient-of } x)) (\text{coeffs } p)))$

**lemma** de-lcm-pCons:de-lcm (pCons a p) = lcm (snd (quotient-of a)) (de-lcm p)  
**unfolding** de-lcm-def  
**by** (cases a=0 $\wedge$ p=0,auto)

**lemma** de-lcm-0[simp]:de-lcm 0 = 1 **unfolding** de-lcm-def **by** auto

**lemma** de-lcm-pos[simp]:de-lcm p > 0  
**apply** (induct p)  
**apply** (auto simp add:de-lcm-pCons)  
**by** (metis lcm-pos-int less-numeral-extra(3) quotient-of-denom-pos')+

**lemma** de-lcm-ints:  
**fixes** x::rat  
**shows**  $x \in \text{set } (\text{coeffs } p) \implies \text{rat-of-int } (\text{de-lcm } p) * x \in \mathbf{Z}$   
**proof** (induct p)  
**case** 0  
**then show** ?case **by** auto  
**next**  
**case** (pCons a p)  
**define** a1 a2 **where**  $a1 \equiv \text{fst } (\text{quotient-of } a)$  **and**  $a2 \equiv \text{snd } (\text{quotient-of } a)$   
**have**  $a : a = (\text{rat-of-int } a1) / (\text{rat-of-int } a2)$  **and**  $a2 > 0$   
**using** quotient-of-denom-pos'[of a] **unfolding** a1-def a2-def  
**by** (auto simp add: quotient-of-div)  
**define** mp1 **where**  $mp1 \equiv a2 \text{ div gcd } (\text{de-lcm } p) a2$   
**define** mp2 **where**  $mp2 \equiv \text{de-lcm } p \text{ div gcd } a2 (\text{de-lcm } p)$   
**have** lcm-times1:lcm a2 (de-lcm p) = de-lcm p \* mp1  
**using** lcm-altdef-int[of de-lcm p a2,folded mp1-def]  $\langle a2 > 0 \rangle$   
**unfolding** mp1-def  
**apply** (subst div-mult-swap)  
**by** (auto simp add: abs-of-pos gcd.commute lcm-altdef-int mult.commute)  
**have** lcm-times2:lcm a2 (de-lcm p) = a2 \* mp2  
**using** lcm-altdef-int[of a2 de-lcm p,folded mp1-def]  $\langle a2 > 0 \rangle$   
**unfolding** mp2-def **by** (subst div-mult-swap, auto simp add:abs-of-pos)  
**show** ?case  
**proof** (cases x  $\in$  set (coeffs p))  
**case** True  
**show** ?thesis **using** pCons(2)[OF True]  
**by** (smt (verit) Ints-mult Ints-of-int a2-def de-lcm-pCons lcm-times1)

```

      mult.assoc mult.commute of-int-mult)
next
  case False
  then have x=a
  using pCons cCons-not-0-eq coeffs-pCons-eq-cCons insert-iff list.set(2) not-0-cCons-eq

  by fastforce
  show ?thesis unfolding ⟨x=a⟩ de-lcm-pCons
  apply (fold a2-def,unfold a)
  by (simp add: de-lcm-pCons lcm-times2 of-rat-divide)
qed
qed

definition clear-de::rat poly ⇒ int poly where
  clear-de p = (SOME q. (map-poly of-int q) = smult (of-int (de-lcm p)) p)

lemma clear-de:of-int-poly(clear-de p) = smult (of-int (de-lcm p)) p
proof –
  have ∃ q. (of-int-poly q) = smult (of-int (de-lcm p)) p
  proof (induct p)
  case 0
  show ?case by (metis map-poly-0 smult-0-right)
next
  case (pCons a p)
  then obtain q1::int poly where q1:of-int-poly q1 = smult (rat-of-int (de-lcm
p)) p
  by auto
  define a1 a2 where a1≡fst (quotient-of a) and a2≡snd (quotient-of a)
  have a:a=(rat-of-int a1)/ (rat-of-int a2) and a2>0
  using quotient-of-denom-pos' quotient-of-div
  unfolding a1-def a2-def by auto
  define mp1 where mp1≡a2 div gcd (de-lcm p) a2
  define mp2 where mp2≡de-lcm p div gcd a2 (de-lcm p)
  have lcm-times1:lcm a2 (de-lcm p) = de-lcm p * mp1
  using lcm-altdef-int[of de-lcm p a2,folded mp1-def] ⟨a2>0⟩
  unfolding mp1-def
  by (subst div-mult-swap, auto simp add: abs-of-pos gcd.commute lcm-altdef-int
mult.commute)
  have lcm-times2:lcm a2 (de-lcm p) = a2 * mp2
  using lcm-altdef-int[of a2 de-lcm p,folded mp1-def] ⟨a2>0⟩
  unfolding mp2-def by (subst div-mult-swap, auto simp add:abs-of-pos)
  define q2 where q2≡pCons (mp2 * a1) (smult mp1 q1)
  have of-int-poly q2 = smult (rat-of-int (de-lcm (pCons a p))) (pCons a p)
using ⟨a2>0⟩
  apply (simp add:de-lcm-pCons )
  apply (fold a2-def)
  apply (unfold a)
  apply (subst lcm-times2,subst lcm-times1)
  by (simp add: Polynomial.map-poly-pCons mult.commute of-int-hom.map-poly-hom-smult

```

```

q1 q2-def)
  then show ?case by auto
qed
then show ?thesis unfolding clear-de-def by (meson someI-ex)
qed

lemma clear-de-0[simp]:clear-de 0 = 0
  using clear-de[of 0] by auto

lemma [code abstract]: coeffs (clear-de p) =
  (let lcm = de-lcm p in map ( $\lambda x. \text{int-of-rat } (\text{of-int } \text{lcm} * x)$ ) (coeffs p))
proof -
  define mul where mul $\equiv$ rat-of-int (de-lcm p)
  have map-poly int-of-rat (of-int-poly q) = q for q
    apply (subst map-poly-map-poly)
    by (auto simp add:int-of-rat-id)
  then have clear-eq:clear-de p = map-poly int-of-rat (smult (of-int (de-lcm p)) p)
    using arg-cong[where f=map-poly int-of-rat,OF clear-de]
    by auto
  show ?thesis
  proof (cases p=0)
    case True
      then show ?thesis by auto
    next
    case False
      define g where g $\equiv$ ( $\lambda x. \text{int-of-rat } (\text{rat-of-int } (\text{de-lcm } p) * x)$ )
      have de-lcm p  $\neq$  0 using de-lcm-pos by (metis less-irrefl)
      moreover have last (coeffs p)  $\neq$  0
        by (simp add: False last-coeffs-eq-coeff-degree)
      have False when asm:last (map g (coeffs p)) = 0
        proof -
          have coeffs p  $\neq$  [] using False by auto
          hence g (last (coeffs p)) = 0 using asm last-map[of coeffs p g] by auto
          hence last (coeffs p) = 0
            unfolding g-def using  $\langle$ coeffs p  $\neq$  [] $\rangle$ ,  $\langle$ de-lcm p  $\neq$  0 $\rangle$ 
            apply (subst (asm) int-of-rat-0-iff)
            by (auto intro!: de-lcm-ints )
          thus False using  $\langle$ last (coeffs p)  $\neq$  0 $\rangle$  by simp
        qed
      ultimately show ?thesis
        apply (auto simp add: coeffs-smult clear-eq comp-def smult-conv-map-poly
          map-poly-map-poly coeffs-map-poly)
        apply (fold g-def)
        by (metis False Ring-Hom-Poly.coeffs-map-poly coeffs-eq-Nil last-coeffs-eq-coeff-degree

          last-map)
      qed
    qed
  qed

```

### 3.3 Sign variations for pseudo-remainder sequences

**locale** *order-hom* =

**fixes** *hom* :: 'a :: ord  $\Rightarrow$  'b :: ord  
**assumes** *hom-less*:  $x < y \iff \text{hom } x < \text{hom } y$   
**and** *hom-less-eq*:  $x \leq y \iff \text{hom } x \leq \text{hom } y$

**locale** *linordered-idom-hom* = *order-hom* *hom* + *inj-idom-hom* *hom*

**for** *hom* :: 'a :: *linordered-idom*  $\Rightarrow$  'b :: *linordered-idom*

**begin**

**lemma** *sgn-sign*:  $\text{sgn} (\text{hom } x) = \text{of-int} (\text{sign } x)$

**by** (*simp add: sign-def hom-less sgn-if*)

**end**

**locale** *hom-pseudo-smods* = *comm-semiring-hom* *hom*

+ *r1:linordered-idom-hom* *R1* + *r2:linordered-idom-hom* *R2*

**for** *hom*::'a::*linordered-idom*  $\Rightarrow$  'b::{*comm-semiring-1*,*linordered-idom*}

**and** *R1*::'a  $\Rightarrow$  *real*

**and** *R2*::'b  $\Rightarrow$  *real* +

**assumes** *R-hom*:*R1*  $x = R_2 (\text{hom } x)$

**begin**

**lemma** *map-poly-R-hom-commute*:

$\text{poly} (\text{map-poly } R_1 \ p) (R_2 \ x) = R_2 (\text{poly} (\text{map-poly } \text{hom } \ p) \ x)$

**apply** (*induct* *p*)

**using** *r2.hom-add* *r2.hom-mult* *R-hom* **by** *auto*

**definition** *changes-hpoly-at*::'a *poly list*  $\Rightarrow$  'b  $\Rightarrow$  *int* **where**

*changes-hpoly-at* *ps* *a* = *changes* (*map* ( $\lambda p. \text{eval-poly } \text{hom } p \ a$ ) *ps*)

**lemma** *changes-hpoly-at-Nil*[*simp*]: *changes-hpoly-at* [] *a* = 0

**unfolding** *changes-hpoly-at-def* **by** *simp*

**definition** *changes-itv-spmods*::'b  $\Rightarrow$  'a  $\Rightarrow$  'a *poly*  $\Rightarrow$  'a *poly*  $\Rightarrow$  *int* **where**

*changes-itv-spmods* *a* *b* *p* *q* = (*let* *ps* = *spmods* *p* *q* *in*

*changes-hpoly-at* *ps* *a* - *changes-hpoly-at* *ps* *b*)

**definition** *changes-gt-spmods*::'b  $\Rightarrow$  'a *poly*  $\Rightarrow$  'a *poly*  $\Rightarrow$  *int* **where**

*changes-gt-spmods* *a* *p* *q* = (*let* *ps* = *spmods* *p* *q* *in*

*changes-hpoly-at* *ps* *a* - *changes-poly-pos-inf* *ps*)

**definition** *changes-le-spmods*::'b  $\Rightarrow$  'a *poly*  $\Rightarrow$  'a *poly*  $\Rightarrow$  *int* **where**

*changes-le-spmods* *b* *p* *q* = (*let* *ps* = *spmods* *p* *q* *in*

*changes-poly-neg-inf* *ps* - *changes-hpoly-at* *ps* *b*)

**definition** *changes-R-spmods*::'a *poly*  $\Rightarrow$  'a *poly*  $\Rightarrow$  *int* **where**

*changes-R-spmods p q* = (let ps = *spmods p q* in *changes-poly-neg-inf ps*  
 – *changes-poly-pos-inf ps*)

**lemma** *changes-spmods-smods*:

**shows** *changes-itv-spmods a b p q*

= *changes-itv-smods (R<sub>2</sub> a) (R<sub>2</sub> b) (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q)*

**and** *changes-R-spmods p q* = *changes-R-smods (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q)*

**and** *changes-gt-spmods a p q* = *changes-gt-smods (R<sub>2</sub> a) (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q)*

**and** *changes-le-spmods b p q* = *changes-le-smods (R<sub>2</sub> b) (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q)*

**proof** –

**define** *pp qq* where *pp* = *map-poly R<sub>1</sub> p* **and** *qq* = *map-poly R<sub>1</sub> q*

**have** *spmods-eq:spmods (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q) = map (map-poly R<sub>1</sub>) (spmods p q)*

**proof** (*induct spmods p q arbitrary:p q*)

**case** *Nil*

**thus** ?*case* **by** (*metis list.simps(8) map-poly-0 spmods-nil-eq*)

**next**

**case** (*Cons p' xs*)

**hence** *p ≠ 0* **by** *auto*

**define** *m* where *m* ≡ (*if even (degree p + 1 – degree q) then – 1 else – lead-coeff q*)

**define** *r* where *r* ≡ *smult m (pseudo-mod p q)*

**have** *xs1:p#xs=spmods p q*

**by** (*metis (no-types) Cons.hyps(2) list.distinct(1) list.inject spmods.simps*)

**have** *xs2:xs=spmods q r* **using** *xs1 <p ≠ 0> r-def*

**by** (*auto simp add:spmods.simps[of p q,folded exp-def,folded m-def]*)

**define** *ys* where *ys* ≡ *spmods (map-poly R<sub>1</sub> q) (map-poly R<sub>1</sub> r)*

**have** *ys:(map-poly R<sub>1</sub> p)#ys=spmods (map-poly R<sub>1</sub> p) (map-poly R<sub>1</sub> q)*

**using** *<p ≠ 0> unfolding ys-def r-def*

**apply** (*subst (2) spmods.simps*)

**unfolding m-def** **by** (*auto simp:r1.pseudo-mod-hom hom-distrib*)

**show** ?*case* **using** *Cons.hyps(1)[OF xs2]*

**apply** (*fold xs1 xs2 ys ys-def*)

**by** *auto*

**qed**

**have** *changes-eq-at:changes-poly-at (smods pp qq) (R<sub>2</sub> x) = changes-hpoly-at (spmods p q) x*

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

**for** *x*

**proof** –

**define** *ff* where *ff* = (*λp. sgn (poly p (R<sub>2</sub> x))*)

**have** ?*L* = *changes (map ff (smods pp qq))*

**using** *changes-poly-at-alternative* **unfolding ff-def** **by** *blast*

**also have** ... = *changes (map ff (spmods pp qq))*

**unfolding ff-def** **using** *spmods-smods-sgn-map-eq* **by** *simp*

**also have** ... = *changes* (map ff (map (map-poly  $R_1$ ) (smods p q)))  
**unfolding** pp-def qq-def **using** smods-eq **by** simp  
**also have** ... = ?R  
**proof** –  
**have** ff  $\circ$  map-poly  $R_1$  = sign  $\circ$  ( $\lambda p$ . eval-poly hom p x)  
**unfolding** ff-def comp-def  
**by** (simp add: map-poly-R-hom-commute poly-map-poly-eval-poly r2.sgn-sign)  
**then show** ?thesis  
**unfolding** changes-hpoly-at-def  
**apply** (subst (2) changes-map-sign-of-int-eq)  
**by** (simp add: comp-def)  
**qed**  
**finally show** ?thesis .  
**qed**

**have** changes-eq-neg-inf:  
*changes-poly-neg-inf* (smods pp qq) = *changes-poly-neg-inf* (smods p q)  
(is ?L=?R)  
**proof** –  
**have** ?L = *changes* (map sgn-neg-inf (map (map-poly  $R_1$ ) (smods p q)))  
**unfolding** changes-poly-neg-inf-def smods-smods-*sgn-map-eq*  
**by** (simp add: smods-eq[folded pp-def qq-def])  
**also have** ... = *changes* (map (sgn-neg-inf  $\circ$  (map-poly  $R_1$ )) (smods p q))  
**using** map-map **by** simp  
**also have** ... = *changes* (map ((sign:: -  $\Rightarrow$  real)  $\circ$  sgn-neg-inf) (smods p q))  
**proof** –  
**have** (sgn-neg-inf  $\circ$  (map-poly  $R_1$ )) = of-int  $\circ$  sign  $\circ$  sgn-neg-inf  
**unfolding** sgn-neg-inf-def comp-def  
**by** (auto simp: r1.sgn-sign)  
**then show** ?thesis **by** (simp add: comp-def)  
**qed**  
**also have** ... = *changes* (map sgn-neg-inf (smods p q))  
**apply** (subst (2) changes-map-sign-of-int-eq)  
**by** (simp add: comp-def)  
**also have** ... = ?R  
**unfolding** changes-poly-neg-inf-def **by** simp  
**finally show** ?thesis .  
**qed**

**have** changes-eq-pos-inf:  
*changes-poly-pos-inf* (smods pp qq) = *changes-poly-pos-inf* (smods p q)  
(is ?L=?R)  
**proof** –  
**have** ?L = *changes* (map sgn-pos-inf (map (map-poly  $R_1$ ) (smods p q)))  
**unfolding** changes-poly-pos-inf-def smods-smods-*sgn-map-eq*  
**by** (simp add: smods-eq[folded pp-def qq-def])  
**also have** ... = *changes* (map (sgn-pos-inf  $\circ$  (map-poly  $R_1$ )) (smods p q))  
**using** map-map **by** simp  
**also have** ... = *changes* (map ((sign:: -  $\Rightarrow$  real)  $\circ$  sgn-pos-inf) (smods p q))

```

proof –
  have (sgn-pos-inf ∘ (map-poly  $R_1$ )) = of-int ∘ sign ∘ sgn-pos-inf
    unfolding sgn-pos-inf-def comp-def
    by (auto simp:r1.sgn-sign)
  then show ?thesis by (auto simp:comp-def)
qed
also have ... = changes (map sgn-pos-inf (smods  $p$   $q$ ))
  apply (subst (2) changes-map-sign-of-int-eq)
  by (simp add:comp-def)
also have ... = ? $R$ 
  unfolding changes-poly-pos-inf-def by simp
finally show ?thesis .
qed

show changes-itv-smods  $a$   $b$   $p$   $q$ 
  = changes-itv-smods ( $R_2$   $a$ ) ( $R_2$   $b$ ) (map-poly  $R_1$   $p$ ) (map-poly  $R_1$   $q$ )
  unfolding changes-itv-smods-def changes-itv-smods-def
  using changes-eq-at by (simp add: Let-def pp-def qq-def)
show changes-R-smods  $p$   $q$  = changes-R-smods (map-poly  $R_1$   $p$ ) (map-poly  $R_1$ 
 $q$ )
  unfolding changes-R-smods-def changes-R-smods-def Let-def
  using changes-eq-neg-inf changes-eq-pos-inf
  by (simp add: pp-def qq-def)
show changes-gt-smods  $a$   $p$   $q$  = changes-gt-smods
  ( $R_2$   $a$ ) (map-poly  $R_1$   $p$ ) (map-poly  $R_1$   $q$ )
  unfolding changes-gt-smods-def changes-gt-smods-def Let-def
  using changes-eq-at changes-eq-pos-inf
  by (simp add: pp-def qq-def)
show changes-le-smods  $b$   $p$   $q$  = changes-le-smods
  ( $R_2$   $b$ ) (map-poly  $R_1$   $p$ ) (map-poly  $R_1$   $q$ )
  unfolding changes-le-smods-def changes-le-smods-def Let-def
  using changes-eq-at changes-eq-neg-inf
  by (simp add: pp-def qq-def)
qed

end

end

```

## 4 TaQ for polynomials with rational coefficients

**theory** *Tarski-Query-Impl* **imports**

*Pseudo-Remainder-Sequence Sturm-Tarski*

**begin**

**global-interpretation** *rat-int:hom-pseudo-smods rat-of-int real-of-int real-of-rat*

**defines**

*ri-changes-itv-smods* = *rat-int.changes-itv-smods* **and**

*ri-changes-gt-smods* = *rat-int.changes-gt-smods* **and**

$ri\text{-changes-le-spmods} = rat\text{-int.changes-le-spmods}$  **and**  
 $ri\text{-changes-R-spmods} = rat\text{-int.changes-R-spmods}$   
**apply** *unfold-locales*  
**by** (*simp-all add: of-rat-less of-rat-less-eq*)

**definition**  $TaQ\text{-R-rats}::rat\ poly \Rightarrow rat\ poly \Rightarrow int$  **where**  
 $TaQ\text{-R-rats } p\ q = taq\ \{x.\ poly\ (map\text{-poly}\ real\text{-of-rat}\ p)\ x = (0::real)\}$   
 $(map\text{-poly}\ real\text{-of-rat}\ q)$

**definition**  $TaQ\text{-itv-rats}::rat \Rightarrow rat \Rightarrow rat\ poly \Rightarrow rat\ poly \Rightarrow int$  **where**  
 $TaQ\text{-itv-rats } a\ b\ p\ q = taq\ \{x.\ poly\ (map\text{-poly}\ real\text{-of-rat}\ p)\ x = (0::real)\}$   
 $\wedge\ of\text{-rat } a < x \wedge x < of\text{-rat } b\} (map\text{-poly}\ real\text{-of-rat}\ q)$

**definition**  $TaQ\text{-gt-rats}::rat \Rightarrow rat\ poly \Rightarrow rat\ poly \Rightarrow int$  **where**  
 $TaQ\text{-gt-rats } a\ p\ q = taq\ \{x.\ poly\ (map\text{-poly}\ real\text{-of-rat}\ p)\ x = (0::real)\}$   
 $\wedge\ of\text{-rat } a < x\} (map\text{-poly}\ real\text{-of-rat}\ q)$

**definition**  $TaQ\text{-le-rats}::rat \Rightarrow rat\ poly \Rightarrow rat\ poly \Rightarrow int$  **where**  
 $TaQ\text{-le-rats } b\ p\ q = taq\ \{x.\ poly\ (map\text{-poly}\ real\text{-of-rat}\ p)\ x = (0::real)\}$   
 $\wedge\ x < of\text{-rat } b\} (map\text{-poly}\ real\text{-of-rat}\ q)$

**lemma** *taq-smult-pos*:  
**assumes**  $a > 0$   
**shows**  $taq\ s\ (smult\ a\ p) = taq\ s\ p$   
**unfolding** *taq-def* **by** (*simp add: assms sign-times*)

**lemma** *taq-proots-R-code*[code]:  
 $TaQ\text{-R-rats } p\ q = (let$   
 $ip = clear\text{-de } p;$   
 $iq = clear\text{-de } q$   
 $in\ ri\text{-changes-R-spmods } ip\ (pderiv\ ip * iq))$

**proof** –

**define**  $ip\ iq$  **where**  $ip = clear\text{-de } p$  **and**  $iq = clear\text{-de } q$   
**define**  $dp\ dq$  **where**  $dp = rat\text{-of-int } (de\text{-lcm } p)$  **and**  $dq = rat\text{-of-int } (de\text{-lcm } q)$

**have**  $dp > 0\ dq > 0$   
**unfolding** *dp-def dq-def* **by** *simp-all*  
**have**  $ip:of\text{-int-poly } ip = smult\ dp\ p$  **and**  $iq:of\text{-int-poly } iq = smult\ dq\ q$   
**using** *clear-de* **unfolding** *ip-def iq-def dp-def dq-def* **by** *auto*

**have**  $TaQ\text{-R-rats } p\ q = taq\ \{x.\ poly\ (map\text{-poly}\ real\text{-of-rat}\ (of\text{-int-poly } ip))\ x =$   
 $0\}$   
 $(map\text{-poly}\ real\text{-of-rat}\ (of\text{-int-poly } iq))$   
**unfolding** *TaQ-R-rats-def ip iq* **using**  $\langle dp > 0 \rangle \langle dq > 0 \rangle$   
**by** (*simp add:of-rat-hom.map-poly-hom-smult taq-smult-pos*)  
**also have**  $\dots = taq\ \{x.\ poly\ (of\text{-int-poly } ip)\ x = (0::real)\}\ (of\text{-int-poly } iq)$   
**by** (*simp add:map-poly-map-poly comp-def*)  
**also have**  $\dots = changes\text{-R-smods } (of\text{-int-poly } ip)\ (pderiv\ (of\text{-int-poly } ip) * of\text{-int-poly } iq)$



```

    using sturm-tarski-R by simp
  also have ... = changes-R-smods (of-int-poly ip) (of-int-poly (pderiv ip * iq))
    by (simp add: of-int-hom.map-poly-pderiv of-int-poly-hom.hom-mult)
  also have ... = ri-changes-R-smods ip (pderiv ip * iq)
    using rat-int.changes-smods-smods by simp
  finally have TaQ-R-rats p q = ri-changes-R-smods ip (pderiv ip * iq) .
  then show ?thesis unfolding Let-def ip-def iq-def .
qed

lemma taq-proots-itv-code[code]:
  TaQ-itv-rats a b p q = (if a ≥ b then
    0
  else if poly p a ≠ 0 ∧ poly p b ≠ 0 then
    (let
      ip = clear-de p;
      iq = clear-de q
      in ri-changes-itv-smods a b ip (pderiv ip * iq))
    else
      Code.abort (STR "Roots at border yet to be supported")
      (λ-. TaQ-itv-rats a b p q)
  )
proof (cases a ≥ b ∨ poly p a = 0 ∨ poly p b = 0)
  case True
  moreover have ?thesis if a ≥ b
  proof -
    have {x. poly (map-poly of-rat p) x = 0 ∧ real-of-rat a < x ∧ x < real-of-rat
    b}
      = {}
      using that rat-int.r2.hom-less-eq by fastforce
    then have TaQ-itv-rats a b p q = taq {} (map-poly real-of-rat q)
      unfolding TaQ-itv-rats-def by metis
    also have ... = 0
      unfolding taq-def by simp
    finally show ?thesis using that by auto
  qed
  moreover have ?thesis if ¬ a ≥ b poly p a = 0 ∨ poly p b = 0
    using that by auto
  ultimately show ?thesis by auto
next
  case False

  define ip iq where ip = clear-de p and iq = clear-de q
  define dp dq where dp = rat-of-int (de-lcm p) and dq = rat-of-int (de-lcm q)
  define aa bb where aa = real-of-rat a and bb = real-of-rat b

  have dp > 0 dq > 0
    unfolding dp-def dq-def by simp-all
  have ip:of-int-poly ip = smult dp p and iq:of-int-poly iq = smult dq q
    using clear-de unfolding ip-def iq-def dp-def dq-def by auto

```

```

have  $TaQ$ -itv-rats  $a\ b\ p\ q = taq\ \{x.\ poly\ (map\text{-}poly\ real\text{-}of\text{-}rat\ (of\text{-}int\text{-}poly\ ip))\ x$ 
 $= 0$ 
   $\wedge\ aa < x \wedge x < bb\}$ 
   $(map\text{-}poly\ real\text{-}of\text{-}rat\ (of\text{-}int\text{-}poly\ iq))$ 
  unfolding  $TaQ$ -itv-rats-def  $ip\ iq\ aa$ -def  $bb$ -def using  $\langle dp > 0 \rangle\ \langle dq > 0 \rangle$ 
  by  $(simp\ add:of\text{-}rat\text{-}hom.map\text{-}poly\text{-}hom\text{-}smult\ taq\text{-}smult\text{-}pos)$ 
also have  $\dots = taq\ \{x.\ poly\ (of\text{-}int\text{-}poly\ ip)\ x = (0::real)$ 
   $\wedge\ aa < x \wedge x < bb\}$   $(of\text{-}int\text{-}poly\ iq)$ 
  by  $(simp\ add:map\text{-}poly\text{-}map\text{-}poly\ comp\text{-}def)$ 
also have  $\dots = changes\text{-}itv\text{-}smods\ aa\ bb\ (of\text{-}int\text{-}poly\ ip)$ 
   $(pderiv\ (of\text{-}int\text{-}poly\ ip) * of\text{-}int\text{-}poly\ iq)$ 
proof -
  have  $aa < bb\ poly\ (map\text{-}poly\ of\text{-}int\ ip)\ aa \neq 0$ 
   $poly\ (map\text{-}poly\ of\text{-}int\ ip)\ bb \neq 0$ 
  unfolding  $aa$ -def  $bb$ -def
  subgoal by  $(meson\ False\ not\text{-}less\ of\text{-}rat\text{-}less)$ 
  subgoal using  $False\ \langle 0 < dp \rangle\ ip\ rat\text{-}int.map\text{-}poly\text{-}R\text{-}hom\text{-}commute$  by force
  subgoal using  $False\ \langle 0 < dp \rangle\ ip\ rat\text{-}int.map\text{-}poly\text{-}R\text{-}hom\text{-}commute$  by force
  done
  from  $sturm\text{-}tarski\text{-}interval[OF\ this]$ 
  show  $?thesis$  by auto
qed
also have  $\dots = changes\text{-}itv\text{-}smods\ aa\ bb\ (of\text{-}int\text{-}poly\ ip)\ (of\text{-}int\text{-}poly\ (pderiv\ ip * iq))$ 
  by  $(simp\ add:of\text{-}int\text{-}hom.map\text{-}poly\text{-}pderiv\ of\text{-}int\text{-}poly\text{-}hom.hom\text{-}mult)$ 
also have  $\dots = ri\text{-}changes\text{-}itv\text{-}spmolds\ a\ b\ ip\ (pderiv\ ip * iq)$ 
  using  $rat\text{-}int.changes\text{-}spmolds\text{-}smolds$  unfolding  $aa$ -def  $bb$ -def by simp
finally have  $TaQ$ -itv-rats  $a\ b\ p\ q = ri\text{-}changes\text{-}itv\text{-}spmolds\ a\ b\ ip\ (pderiv\ ip * iq)$ 
then show  $?thesis$  unfolding  $Let$ -def  $ip$ -def  $iq$ -def using  $False$  by presburger
qed

lemma  $taq$ -roots-gt-code[code]:
   $TaQ$ -gt-rats  $a\ p\ q = ($ 
     $if\ poly\ p\ a \neq 0\ then$ 
     $(let$ 
       $ip = clear\text{-}de\ p;$ 
       $iq = clear\text{-}de\ q$ 
       $in\ ri\text{-}changes\text{-}gt\text{-}spmolds\ a\ ip\ (pderiv\ ip * iq))$ 
     $else$ 
       $Code.abort\ (STR\ "Roots\ at\ border\ yet\ to\ be\ supported")$ 
       $(\lambda\_.\ TaQ$ -gt-rats  $a\ p\ q)$ 
     $)$ 
  proof  $(cases\ poly\ p\ a = 0)$ 
  case True
  then show  $?thesis$  by auto
next
  case False

```

```

define ip iq where ip = clear-de p and iq = clear-de q
define dp dq where dp = rat-of-int (de-lcm p) and dq = rat-of-int (de-lcm q)
define aa where aa = real-of-rat a

have dp > 0 dq > 0
  unfolding dp-def dq-def by simp-all
have ip:of-int-poly ip = smult dp p and iq:of-int-poly iq = smult dq q
  using clear-de unfolding ip-def iq-def dp-def dq-def by auto

have TaQ-gt-rats a p q = taq {x. poly (map-poly real-of-rat (of-int-poly ip)) x =
0
  ∧ aa < x}
  (map-poly real-of-rat (of-int-poly iq))
  unfolding TaQ-gt-rats-def ip iq aa-def using ⟨dp > 0⟩ ⟨dq > 0⟩
  by (simp add:of-rat-hom.map-poly-hom-smult taq-smult-pos)
also have ... = taq {x. poly (of-int-poly ip) x = (0::real)
  ∧ aa < x } (of-int-poly iq)
  by (simp add:map-poly-map-poly comp-def)
also have ... = changes-gt-smods aa (of-int-poly ip)
  (pderiv (of-int-poly ip) * of-int-poly iq)
proof -
  have poly (map-poly of-int ip) aa ≠ 0
  unfolding aa-def using False ⟨0 < dp⟩ ip rat-int.map-poly-R-hom-commute
by force
  from sturm-tarski-above[OF this]
  show ?thesis by auto
qed
also have ... = changes-gt-smods aa (of-int-poly ip) (of-int-poly (pderiv ip * iq))
  by (simp add: of-int-hom.map-poly-pderiv of-int-poly-hom.hom-mult)
also have ... = ri-changes-gt-spmods a ip (pderiv ip * iq)
  using rat-int.changes-spmods-smods unfolding aa-def by simp
finally have TaQ-gt-rats a p q = ri-changes-gt-spmods a ip (pderiv ip * iq) .
then show ?thesis unfolding Let-def ip-def iq-def using False by presburger
qed

lemma taq-proots-le-code[code]:
  TaQ-le-rats b p q = (
  if poly p b ≠ 0 then
  (let
  ip = clear-de p;
  iq = clear-de q
  in ri-changes-le-spmods b ip (pderiv ip * iq))
  else
  Code.abort (STR "Roots at border yet to be supported")
  (λ-. TaQ-le-rats b p q)
  )
proof (cases poly p b = 0)
  case True

```

```

then show ?thesis by auto
next
case False

define ip iq where ip = clear-de p and iq = clear-de q
define dp dq where dp = rat-of-int (de-lcm p) and dq = rat-of-int (de-lcm q)
define bb where bb = real-of-rat b

have dp > 0 dq > 0
  unfolding dp-def dq-def by simp-all
have ip:of-int-poly ip = smult dp p and iq:of-int-poly iq = smult dq q
  using clear-de unfolding ip-def iq-def dp-def dq-def by auto

have TaQ-le-rats b p q = taq {x. poly (map-poly real-of-rat (of-int-poly ip)) x =
0
  ^ x < bb}
  (map-poly real-of-rat (of-int-poly iq))
  unfolding TaQ-le-rats-def ip iq bb-def using ‹dp > 0› ‹dq > 0›
  by (simp add:of-rat-hom.map-poly-hom-smult taq-smult-pos)
also have ... = taq {x. poly (of-int-poly ip) x = (0::real)
  ^ x < bb} (of-int-poly iq)
  by (simp add:map-poly-map-poly comp-def)
also have ... = changes-le-smods bb (of-int-poly ip)
  (pderiv (of-int-poly ip) * of-int-poly iq)
proof -
  have poly (map-poly of-int ip) bb ≠ 0
  unfolding bb-def using False ‹0 < dp› ip rat-int.map-poly-R-hom-commute
by force
  from sturm-tarski-below[OF this]
  show ?thesis by auto
qed
also have ... = changes-le-smods bb (of-int-poly ip) (of-int-poly (pderiv ip * iq))
  by (simp add: of-int-hom.map-poly-pderiv of-int-poly-hom.hom-mult)
also have ... = ri-changes-le-spmods b ip (pderiv ip * iq)
  using rat-int.changes-spmods-smods unfolding bb-def by simp
  finally have TaQ-le-rats b p q = ri-changes-le-spmods b ip (pderiv ip * iq) .
then show ?thesis unfolding Let-def ip-def iq-def using False by presburger
qed

end

```

## References

- [1] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

- [2] C. Cohen. *Formalized algebraic numbers: construction and first-order theory*. PhD thesis, École polytechnique, Nov 2012.
- [3] W. Li and L. C. Paulson. A modular, efficient formalisation of real algebraic numbers. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2016*, pages 66–75, New York, NY, USA, 2016. ACM.