

The Sturm–Tarski Theorem

Wenda Li

February 6, 2026

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 *poly-power-n-eq*:

fixes $x::'a :: idom$

assumes $n \neq 0$

shows $poly\ ([:-a, 1:] \hat{=} n)\ x = 0 \iff (x = a) \langle proof \rangle$

lemma *poly-power-n-odd*:

fixes $x\ a:: real$

assumes *odd n*

shows $poly\ ([:-a, 1:] \hat{=} n)\ x > 0 \iff (x > a) \langle proof \rangle$

lemma *pseudo-divmod-0[simp]*: $pseudo-divmod\ f\ 0 = (0, f)$

$\langle proof \rangle$

lemma *map-poly-eq-iff*:

assumes $f\ 0 = 0$ *inj f*

shows $map-poly\ f\ x = map-poly\ f\ y \iff x = y$

<proof>

lemma *pseudo-mod-0[simp]*:

shows *pseudo-mod p 0 = p pseudo-mod 0 q = 0*

<proof>

lemma *pseudo-mod-mod*:

assumes *g ≠ 0*

shows *smult (lead-coeff g ^ (Suc (degree f) - degree g)) (f mod g) = pseudo-mod f g*

<proof>

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>

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>

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::{archimedean-field, linorder-topology}*

shows *eventually P (at-right x) ⟷ (∃ b > x. ∀ y > x. y < b ⟶ P y)*

<proof>

lemma *eventually-at-left*:

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

shows *eventually P (at-left x) ⟷ (∃ b < x. ∀ y > b. y < x ⟶ P y)*

<proof>

lemma *eventually-neg*:

assumes *F ≠ bot and eve:eventually (λx. P x) F*

shows *¬ eventually (λx. ¬ P x) F*

<proof>

lemma *poly-tendsto[simp]*:
 (*poly p* \longrightarrow *poly p x*) (*at (x::real)*)
 (*poly p* \longrightarrow *poly p x*) (*at-left (x::real)*)
 (*poly p* \longrightarrow *poly p x*) (*at-right (x::real)*)
 \langle *proof* \rangle

lemma *not-eq-pos-or-neg-iff-1*:
fixes *p::real poly*
shows $(\forall z. lb < z \wedge z \leq ub \longrightarrow poly\ p\ z \neq 0) \longleftrightarrow$
 $(\forall z. lb < z \wedge z \leq ub \longrightarrow poly\ p\ z > 0) \vee (\forall z. lb < z \wedge z \leq ub \longrightarrow poly\ p\ z < 0)$ (**is** $?Q \longleftrightarrow ?P$)
 \langle *proof* \rangle

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** $?Q \longleftrightarrow ?P$)
 \langle *proof* \rangle

lemma *next-non-root-interval*:
fixes *p::real poly*
assumes $p \neq 0$
obtains *ub where* $ub > lb$ **and** $(\forall z. lb < z \wedge z \leq ub \longrightarrow poly\ p\ z \neq 0)$
 \langle *proof* \rangle

lemma *last-non-root-interval*:
fixes *p::real poly*
assumes $p \neq 0$
obtains *lb where* $lb < ub$ **and** $(\forall z. lb \leq z \wedge z < ub \longrightarrow poly\ p\ z \neq 0)$
 \langle *proof* \rangle

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]*:
 $x > 0 \implies sign\ x = 1$
 $x = 0 \implies sign\ x = 0$
 $x < 0 \implies sign\ x = -1$
 \langle *proof* \rangle

lemma *sign-cases [case-names neg zero pos]*:
 $(sign\ x = -1 \implies P) \implies (sign\ x = 0 \implies P) \implies (sign\ x = 1 \implies P) \implies P$
 \langle *proof* \rangle

lemma *sign-times*:
fixes *x::'a::linordered-ring-strict*
shows $sign\ (x * y) = sign\ x * sign\ y$

<proof>

lemma *sign-power*:

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

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

<proof>

lemma *sgn-sign-eq*: $\text{sgn} = \text{sign}$

<proof>

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

<proof>

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

<proof>

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>

lemma *poly-pinfy-gt-lc*:

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

assumes $\text{lead-coeff } p > 0$

shows $\exists n. \forall x \geq n. \text{poly } p x \geq \text{lead-coeff } p$ *<proof>*

lemma *poly-sgn-eventually-at-top*:

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

shows *eventually* $(\lambda x. \text{sgn } (\text{poly } p x) = \text{sgn-pos-inf } p)$ *at-top*

<proof>

lemma *poly-sgn-eventually-at-bot*:

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

shows *eventually* $(\lambda x. \text{sgn } (\text{poly } p x) = \text{sgn-neg-inf } p)$ *at-bot*

<proof>

lemma *root-ub*:

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

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⟩

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⟩

2.4 Variation and cross

definition *variation* :: *real* \Rightarrow *real* \Rightarrow *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* :: *real poly* \Rightarrow *real* \Rightarrow *real* \Rightarrow *int* **where**

cross $p \ a \ b = \text{variation} (\text{poly } p \ a) (\text{poly } p \ b)$

lemma *variation-0[simp]*: *variation* $0 \ y = 0$ *variation* $x \ 0 = 0$

⟨proof⟩

lemma *variation-comm*: *variation* $x \ y = - \text{variation } y \ x$ ⟨proof⟩

lemma *cross-0[simp]*: *cross* $0 \ a \ b = 0$ ⟨proof⟩

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⟩

lemma *variation-congr*:

assumes $\text{sgn } x = \text{sgn } x' \ \text{sgn } y = \text{sgn } y'$

shows *variation* $x \ y = \text{variation } x' \ y'$ ⟨proof⟩

lemma *variation-mult-pos*:

assumes $c > 0$

shows *variation* $(c * x) \ y = \text{variation } x \ y$ **and** *variation* $x \ (c * y) = \text{variation } x \ y$
 ⟨proof⟩

lemma *variation-mult-neg-1*:

assumes $c < 0$

shows *variation* $(c * x) \ y = \text{variation } x \ y + (\text{if } y = 0 \text{ then } 0 \text{ else sign } x)$

⟨proof⟩

lemma *variation-mult-neg-2*:

assumes $c < 0$

shows *variation* $x \ (c * y) = \text{variation } x \ y + (\text{if } x = 0 \text{ then } 0 \text{ else } - \text{sign } y)$

<proof>

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>

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

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

lemma *sign-r-pos-rec*:

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

assumes $p \neq 0$

shows $\text{sign-r-pos } p \ x = (\text{if } \text{poly } p \ x = 0 \text{ then } \text{sign-r-pos } (p \text{deriv } p) \ x \text{ else } \text{poly } p \ x > 0)$

<proof>

lemma *sign-r-pos-0[simp]*: $\neg \text{sign-r-pos } 0 \ (x::\text{real})$

<proof>

lemma *sign-r-pos-minus*:

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

assumes $p \neq 0$

shows $\text{sign-r-pos } p \ x = (\neg \text{sign-r-pos } (-p) \ x)$

<proof>

lemma *sign-r-pos-smult*:

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

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

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

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

<proof>

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>

lemma *sign-r-pos-add*:

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

assumes $\text{poly } p \ x=0 \ \text{poly } q \ x \neq 0$
shows $\text{sign-r-pos } (p+q) \ x = \text{sign-r-pos } q \ x$
 $\langle \text{proof} \rangle$

lemma sign-r-pos-mod :
fixes $p \ q :: \text{real poly}$
assumes $\text{poly } p \ x=0 \ \text{poly } q \ x \neq 0$
shows $\text{sign-r-pos } (q \bmod p) \ x = \text{sign-r-pos } q \ x$
 $\langle \text{proof} \rangle$

lemma sign-r-pos-pderiv :
fixes $p :: \text{real poly}$
assumes $\text{poly } p \ x=0 \ p \neq 0$
shows $\text{sign-r-pos } (pderiv \ p * p) \ x$
 $\langle \text{proof} \rangle$

lemma sign-r-pos-power :
fixes $p :: \text{real poly}$ **and** $a :: \text{real}$
shows $\text{sign-r-pos } ([: -a, 1:]^{\wedge} n) \ a$
 $\langle \text{proof} \rangle$

2.7 Jump

definition $\text{jump-poly} :: \text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{int}$
where

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

lemma $\text{jump-poly-not-root}$: $\text{poly } p \ x \neq 0 \Longrightarrow \text{jump-poly } q \ p \ x = 0$
 $\langle \text{proof} \rangle$

lemma jump-poly0 [simp]:
 $\text{jump-poly } 0 \ p \ x = 0$
 $\text{jump-poly } q \ 0 \ x = 0$
 $\langle \text{proof} \rangle$

lemma jump-poly-smult-1 :
fixes $p \ q :: \text{real poly}$ **and** $c :: \text{real}$
shows $\text{jump-poly } (\text{smult } c \ q) \ p \ x = \text{sign } c * \text{jump-poly } q \ p \ x$ (**is** $?L=?R$)
 $\langle \text{proof} \rangle$

lemma jump-poly-mult :
fixes $p \ q \ p' :: \text{real poly}$
assumes $p' \neq 0$
shows $\text{jump-poly } (p' * q) \ (p' * p) \ x = \text{jump-poly } q \ p \ x$
 $\langle \text{proof} \rangle$

lemma jump-poly-1-mult :

fixes $p1\ p2::real\ poly$
assumes $poly\ p1\ x \neq 0 \vee poly\ p2\ x \neq 0$
shows $jump\text{-}poly\ 1\ (p1 * p2)\ x = sign\ (poly\ p2\ x) * jump\text{-}poly\ 1\ p1\ x$
 $+ sign\ (poly\ p1\ x) * jump\text{-}poly\ 1\ p2\ x$ (**is** ?L=?R)
 <proof>

lemma *jump-poly-mod*:
fixes $p\ q::real\ poly$
shows $jump\text{-}poly\ q\ p\ x = jump\text{-}poly\ (q\ mod\ p)\ p\ x$
 <proof>

lemma *jump-poly-coprime*:
fixes $p\ q::real\ poly$
assumes $poly\ p\ x = 0\ coprime\ p\ q$
shows $jump\text{-}poly\ q\ p\ x = jump\text{-}poly\ 1\ (q * p)\ x$
 <proof>

lemma *jump-poly-sgn*:
fixes $p\ q::real\ poly$
assumes $p \neq 0\ poly\ p\ x = 0$
shows $jump\text{-}poly\ (pderiv\ p * q)\ p\ x = sign\ (poly\ q\ x)$
 <proof>

2.8 Cauchy index

definition *cindex-poly*:: $real \Rightarrow real \Rightarrow real\ poly \Rightarrow real\ poly \Rightarrow int$
where
 $cindex\text{-}poly\ a\ b\ q\ p \equiv (\sum x \in \{x. poly\ p\ x = 0 \wedge a < x \wedge x < b\}. jump\text{-}poly\ q\ p\ x)$

lemma *cindex-poly-0[simp]*: $cindex\text{-}poly\ a\ b\ 0\ p = 0\ cindex\text{-}poly\ a\ b\ q\ 0 = 0$
 <proof>

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 $cindex\text{-}poly\ a\ b\ 1\ p = cross\ p\ a\ b$
 <proof>

lemma *cindex-poly-mult*:
fixes $p\ q\ p'::real\ poly$
assumes $p' \neq 0$
shows $cindex\text{-}poly\ a\ b\ (p' * q)\ (p' * p) = cindex\text{-}poly\ a\ b\ q\ p$
 <proof>

lemma *cindex-poly-smult-1*:
fixes $p\ q::real\ poly$ **and** $c::real$
shows $cindex\text{-}poly\ a\ b\ (smult\ c\ q)\ p = (sign\ c) * cindex\text{-}poly\ a\ b\ q\ p$
 <proof>

lemma *cindex-poly-mod*:

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

shows $\text{cindex-poly } a \ b \ q \ p = \text{cindex-poly } a \ b \ (q \ \text{mod } p) \ p$

$\langle \text{proof} \rangle$

lemma *cindex-poly-inverse-add*:

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

assumes *coprime* $p \ q$

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

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

$\langle \text{proof} \rangle$

lemma *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)

$\langle \text{proof} \rangle$

lemma *cindex-poly-rec*:

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

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

shows $\text{cindex-poly } a \ b \ q \ p = \text{cross } (p * q) \ a \ b + \text{cindex-poly } a \ b \ (- (p \ \text{mod } q))$
 q (**is** ?L=?R)

$\langle \text{proof} \rangle$

lemma *cindex-poly-congr*:

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

assumes $a < a' \ b' < b$

assumes $\forall x. ((a < x \wedge x \leq a') \vee (b' \leq x \wedge x < b)) \longrightarrow \text{poly } p \ x \neq 0$

shows $\text{cindex-poly } a \ b \ q \ p = \text{cindex-poly } a' \ b' \ q \ p$

$\langle \text{proof} \rangle$

lemma *greaterThanLessThan-unfold*: $\{a < .. < b\} = \{x. a < x \wedge x < b\}$

$\langle \text{proof} \rangle$

lemma *cindex-poly-taq*:

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

shows $\text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < b\} \ q = \text{cindex-poly } a \ b \ (\text{pderiv } p * q) \ p$
 $\langle \text{proof} \rangle$

2.9 Signed remainder sequence

function *smods*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow (\text{real poly}) \ \text{list}$ **where**

$\text{smods } p \ q = (\text{if } p = 0 \ \text{then } [] \ \text{else } \text{Cons } p \ (\text{smods } q \ (- (p \ \text{mod } q)))$)

$\langle \text{proof} \rangle$

termination

$\langle \text{proof} \rangle$

lemma *smods-nil-eq*: $\text{smods } p \ q = [] \longleftrightarrow (p = 0)$ $\langle \text{proof} \rangle$

lemma *smods-singleton*: $[x] = \text{smods } p \ q \implies (p \neq 0 \wedge q = 0 \wedge x = p)$
 ⟨proof⟩

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

lemma *no-0-in-smods*: $0 \notin \text{set } (\text{smods } p \ q)$
 ⟨proof⟩

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

lemma *changes-map-sgn-eq*:
 $\text{changes } xs = \text{changes } (\text{map } \text{sgn } xs)$
 ⟨proof⟩

lemma *changes-map-sign-eq*:
 $\text{changes } xs = \text{changes } (\text{map } \text{sign } xs)$
 ⟨proof⟩

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⟩

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

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

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

lemma *changes-poly-at-0*[simp]:
 $\text{changes-poly-at } [] \ a = 0$
 $\text{changes-poly-at } [p] \ a = 0$
 ⟨proof⟩

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

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
 ⟨*proof*⟩

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

lemma *changes-itv-smods-rec*:
assumes *a < b poly (p*q) a ≠ 0 poly (p*q) b ≠ 0*
shows *changes-itv-smods a b p q* = *cross (p*q) a b* + *changes-itv-smods a b q*
 (-(*p mod q*))
 ⟨*proof*⟩

lemma *changes-smods-congr*:
fixes *p q*:: *real poly*
assumes *a ≠ a' poly p a ≠ 0*
assumes $\forall p \in \text{set } (\text{smods } p \ q). \forall x. ((a < x \wedge x \leq a') \vee (a' \leq x \wedge x < a)) \longrightarrow \text{poly } p \ x \neq 0$
shows *changes-poly-at (smods p q) a* = *changes-poly-at (smods p q) a'*
 ⟨*proof*⟩

lemma *changes-itv-smods-congr*:
fixes *p q*:: *real poly*
assumes *a < a' a' < b' b' < b poly p a ≠ 0 poly p b ≠ 0*
assumes *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 *changes-itv-smods a b p q* = *changes-itv-smods a' b' p q*
 ⟨*proof*⟩

lemma *cindex-poly-changes-itv-mods*:
assumes *a < b poly p a ≠ 0 poly p b ≠ 0*
shows *cindex-poly a b q p* = *changes-itv-smods a b p q* ⟨*proof*⟩

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** $ub > a$

<proof>

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** $lb < b$

<proof>

theorem *sturm-tarski-interval*:

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

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>

theorem *sturm-tarski-above*:

assumes $\text{poly } p \ a \neq 0$

shows $\text{taq } \{x. \text{poly } p \ x = 0 \wedge a < x\} \ q = \text{changes-gt-smods } a \ p \ (\text{pderiv } p * \ q)$

<proof>

theorem *sturm-tarski-below*:

assumes $\text{poly } p \ b \neq 0$

shows $\text{taq } \{x. \text{poly } p \ x = 0 \wedge x < b\} \ q = \text{changes-le-smods } b \ p \ (\text{pderiv } p * \ q)$

<proof>

theorem *sturm-tarski-R*:

shows $\text{taq } \{x. \text{poly } p \ x = 0\} \ q = \text{changes-R-smods } p \ (\text{pderiv } p * \ q)$

<proof>

theorem *sturm-interval*:

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

shows $\text{card } \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < b\} = \text{changes-itv-smods } a \ b \ p \ (\text{pderiv } p)$

<proof>

theorem *sturm-above*:

assumes $\text{poly } p \ a \neq 0$

shows $\text{card } \{x. \text{poly } p \ x = 0 \wedge a < x\} = \text{changes-gt-smods } a \ p \ (\text{pderiv } p)$

<proof>

theorem *sturm-below*:

assumes $\text{poly } p \ b \neq 0$

shows $\text{card } \{x. \text{poly } p \ x = 0 \wedge x < b\} = \text{changes-le-smods } b \ p \ (\text{pderiv } p)$

<proof>

```

theorem sturm-R:
  shows  $\text{card } \{x. \text{poly } p \ x=0\} = \text{changes-R-smods } p \ (\text{pderiv } p)$ 
  <proof>

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 spmods :: 'a::idom poly  $\Rightarrow$  'a poly  $\Rightarrow$  ('a poly) list where
  spmods 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 (spmods q (smult m (pseudo-mod p q))))
  <proof>
termination
  <proof>

```

```

declare spmods.simps[simp del]

```

```

lemma spmods-0[simp]:
  spmods 0 q = []
  spmods p 0 = (if p=0 then [] else [p])
  <proof>

```

```

lemma spmods-nil-eq:spmods p q = []  $\longleftrightarrow$  (p=0)
  <proof>

```

```

lemma changes-poly-at-alternative:
  changes-poly-at ps a = changes (map ( $\lambda p. \text{sign}(\text{poly } p \ a)$ ) ps)
  changes-poly-at ps a = changes (map ( $\lambda p. \text{sgn}(\text{poly } p \ a)$ ) ps)
  <proof>

```

```

lemma smods-smult-length:
  assumes a $\neq$ 0 b $\neq$ 0
  shows  $\text{length } (\text{smods } p \ q) = \text{length } (\text{smods } (\text{smult } a \ p) \ (\text{smult } b \ q))$  <proof>

```

lemma *smods-smult-nth*[*rule-format*]:
fixes $p q::\text{real poly}$
assumes $a \neq 0 \ b \neq 0$
defines $xs \equiv \text{smods } p \ q$ **and** $ys \equiv \text{smods } (\text{smult } a \ p) \ (\text{smult } b \ q)$
shows $\forall n < \text{length } xs. ys!n = (\text{if even } n \text{ then } \text{smult } a \ (xs!n) \ \text{else } \text{smult } b \ (xs!n))$
 $\langle \text{proof} \rangle$

lemma *smods-smult-sgn-map-eq*:
fixes $x::\text{real}$
assumes $m > 0$
defines $f \equiv \lambda p. \text{sgn}(\text{poly } p \ x)$
shows $\text{map } f \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } f \ (\text{smods } p \ q)$
 $\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q)$
 $\text{map } \text{sgn-neg-inf} \ (\text{smods } p \ (\text{smult } m \ q)) = \text{map } \text{sgn-neg-inf} \ (\text{smods } p \ q)$
 $\langle \text{proof} \rangle$

lemma *changes-poly-at-smods-smult*:
assumes $m > 0$
shows $\text{changes-poly-at} \ (\text{smods } p \ (\text{smult } m \ q)) \ x = \text{changes-poly-at} \ (\text{smods } p \ q) \ x$
 $\langle \text{proof} \rangle$

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 $\text{map } f \ (\text{smods } p \ q) = \text{map } f \ (\text{spmods } p \ q)$
 $\text{map } \text{sgn-pos-inf} \ (\text{smods } p \ q) = \text{map } \text{sgn-pos-inf} \ (\text{spmods } p \ q)$
 $\text{map } \text{sgn-neg-inf} \ (\text{smods } p \ q) = \text{map } \text{sgn-neg-inf} \ (\text{spmods } p \ q)$
 $\langle \text{proof} \rangle$

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

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

lemma *of-rat-inj*[*simp*]: inj of-rat
 $\langle \text{proof} \rangle$

lemma (**in** *ring-char-0*) *of-int-inj*[*simp*]: inj of-int
 $\langle \text{proof} \rangle$

lemma *int-of-rat-id*: $\text{int-of-rat } o \ \text{of-int} = \text{id}$
 $\langle \text{proof} \rangle$

lemma *int-of-rat-0*[*simp*]: $\text{int-of-rat } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *int-of-rat-inv*: $r \in \mathbb{Z} \Longrightarrow \text{of-int} \ (\text{int-of-rat } r) = r$

<proof>

lemma *int-of-rat-0-iff*: $x \in \mathbf{Z} \implies \text{int-of-rat } x = 0 \iff x = 0$

<proof>

lemma [*code*]: *int-of-rat* $r = (\text{let } (a,b) = \text{quotient-of } r \text{ in}$
if $b=1$ then a else $\text{Code.abort } (\text{STR } \text{"Failed to convert rat to int"})$
($\lambda-. \text{int-of-rat } r$)

<proof>

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*: $\text{de-lcm } (p\text{Cons } a \ p) = \text{lcm } (\text{snd } (\text{quotient-of } a)) (\text{de-lcm } p)$

<proof>

lemma *de-lcm-0[simp]*: $\text{de-lcm } 0 = 1$ *<proof>*

lemma *de-lcm-pos[simp]*: $\text{de-lcm } p > 0$

<proof>

lemma *de-lcm-ints*:

fixes $x::\text{rat}$

shows $x \in \text{set } (\text{coeffs } p) \implies \text{rat-of-int } (\text{de-lcm } p) * x \in \mathbf{Z}$

<proof>

definition *clear-de*::*rat poly* \Rightarrow *int poly* **where**

$\text{clear-de } p = (\text{SOME } q. (\text{map-poly } \text{of-int } q) = \text{smult } (\text{of-int } (\text{de-lcm } p)) \ p)$

lemma *clear-de:of-int-poly*: $\text{clear-de } p = \text{smult } (\text{of-int } (\text{de-lcm } p)) \ p$

<proof>

lemma *clear-de-0[simp]*: $\text{clear-de } 0 = 0$

<proof>

lemma [*code abstract*]: $\text{coeffs } (\text{clear-de } p) =$

($\text{let } \text{lcm} = \text{de-lcm } p \text{ in } \text{map } (\lambda x. \text{int-of-rat } (\text{of-int } \text{lcm} * x)) (\text{coeffs } p)$)

<proof>

3.3 Sign variations for pseudo-remainder sequences

locale *order-hom* =

fixes $\text{hom} :: 'a :: \text{ord} \Rightarrow 'b :: \text{ord}$

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* $\text{hom} + \text{inj-idom-hom } \text{hom}$

for $\text{hom} :: 'a :: \text{linordered-idom} \Rightarrow 'b :: \text{linordered-idom}$

begin

lemma *sgn-sign:sgn (hom x) = of-int (sign x)*

<proof>

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 = R2 (hom x)*
begin

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

poly (map-poly R1 p) (R2 x) = R2 (poly (map-poly hom p) x)

<proof>

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

changes-hpoly-at ps a = changes (map (λp . eval-poly hom p a) ps)

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

<proof>

definition *changes-itv-spmods:: 'b \Rightarrow 'b \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 (R2 a) (R2 b) (map-poly R1 p) (map-poly R1 q)

and *changes-R-spmods p q = changes-R-smods (map-poly R1 p) (map-poly R1 q)*

and *changes-gt-spmods a p q = changes-gt-smods (R2 a) (map-poly R1 p) (map-poly R1 q)*

and *changes-le-spmods b p q = changes-le-smods (R2 b) (map-poly R1 p) (map-poly*

R_1 q)
 ⟨proof⟩

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-spmods = *rat-int.changes-itv-spmods* **and**

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

ri-changes-le-spmods = *rat-int.changes-le-spmods* **and**

ri-changes-R-spmods = *rat-int.changes-R-spmods*

⟨proof⟩

definition *TaQ-R-rats::rat poly ⇒ rat poly ⇒ int* **where**

TaQ-R-rats p q = *taq* { x . *poly* (*map-poly real-of-rat* p) x = ($0::real$)}
 (*map-poly real-of-rat* q)

definition *TaQ-itv-rats::rat ⇒ rat ⇒ rat poly ⇒ rat poly ⇒ int* **where**

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

definition *TaQ-gt-rats::rat ⇒ rat poly ⇒ rat poly ⇒ int* **where**

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

definition *TaQ-le-rats::rat ⇒ rat poly ⇒ rat poly ⇒ int* **where**

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

lemma *taq-smult-pos*:

assumes $a > 0$

shows *taq* s (*smult* a p) = *taq* s p

⟨proof⟩

lemma *taq-proots-R-code*[*code*]:

TaQ-R-rats p q = (*let*

ip = *clear-de* p ;

iq = *clear-de* q

in ri-changes-R-spmods ip (*pderiv ip* * *iq*)

⟨proof⟩

```

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-spmods 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⟩

```

```

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

```

```

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⟩

```

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.