

The Sturm–Tarski Theorem

Wenda Li

May 14, 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  $\neq 0 \vee$  poly q x  $\neq 0$  if coprime p q  
  for x :: 'a :: field  
   $\langle$ proof $\rangle$ 
```

```
lemma smult-cancel:  
  fixes p::'a::idom poly  
  assumes c $\neq 0$  and smult: smult c p = smult c q  
  shows p=q  
   $\langle$ proof $\rangle$ 
```

```
lemma dvd-monic:  
  fixes p q:: 'a :: idom poly  
  assumes monic:lead-coeff p=1 and p dvd (smult c q) and c $\neq 0$   
  shows p dvd q  $\langle$ proof $\rangle$ 
```

lemma *poly-power-n-eq*:

fixes $x::'a :: idom$

assumes $n \neq 0$

shows $poly\ ([:-a,1:]^{\wedge}n)\ x=0 \longleftrightarrow (x=a)$ *<proof>*

lemma *poly-power-n-odd*:

fixes $x\ a:: real$

assumes $odd\ n$

shows $poly\ ([:-a,1:]^{\wedge}n)\ x>0 \longleftrightarrow (x>a)$ *<proof>*

lemma *gcd-coprime-poly*:

fixes $p\ q::'a::\{factorial-ring-gcd,semiring-gcd-mult-normalize\}$ *poly*

assumes $nz: p \neq 0 \vee q \neq 0$ **and** $p': p = p' * gcd\ p\ q$ **and**

$q': q = q' * gcd\ p\ q$

shows *coprime* $p'\ q'$

<proof>

lemma *poly-mod*:

poly $(p\ mod\ q)\ x = poly\ p\ x$ **if** *poly* $q\ x = 0$

<proof>

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

<proof>

lemma *map-poly-eq-iff*:

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

shows *map-poly* $f\ x = map-poly\ f\ y \longleftrightarrow x=y$

<proof>

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

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

<proof>

lemma *pseudo-mod-mod*:

assumes $g \neq 0$

shows *smult* $(lead-coeff\ g^{\wedge}(Suc\ (degree\ f) - degree\ g))\ (f\ mod\ g) = pseudo-mod\ f\ g$

<proof>

lemma *poly-pseudo-mod*:

assumes *poly* $q\ x=0$ $q \neq 0$

shows *poly* $(pseudo-mod\ p\ q)\ x = (lead-coeff\ q^{\wedge}(Suc\ (degree\ p) - degree\ q)) * poly\ p\ x$

<proof>

lemma *degree-less-timesD*:

fixes $q::'a::idom$ *poly*

assumes $q*g=r$ **and** $deg:r=0 \vee degree\ g>degree\ r$ **and** $g \neq 0$

shows $q=0 \wedge r=0$

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

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

<proof>

lemma *eventually-at-left*:

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

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

<proof>

lemma *eventually-neg*:

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

shows $\neg \text{eventually } (\lambda x. \neg P x) F$

<proof>

lemma *poly-tendsto[simp]*:

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

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

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

<proof>

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>

lemma *not-eq-pos-or-neg-iff-2*:

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

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

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

<proof>

lemma *next-non-root-interval*:

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

assumes $p \neq 0$

obtains *ub* **where** $ub > lb$ **and** $(\forall z. lb < z \wedge z \leq ub \longrightarrow \text{poly } p \ z \neq 0)$
(*proof*)

lemma *last-non-root-interval*:

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

assumes $p \neq 0$

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

(*proof*)

2.2 Sign

definition *sign*:: $'a::\{\text{zero}, \text{linorder}\} \Rightarrow \text{int}$ **where**

$\text{sign } x \equiv (\text{if } x > 0 \text{ then } 1 \text{ else if } x = 0 \text{ then } 0 \text{ else } -1)$

lemma *sign-simps*[*simp*]:

$x > 0 \Longrightarrow \text{sign } x = 1$

$x = 0 \Longrightarrow \text{sign } x = 0$

$x < 0 \Longrightarrow \text{sign } x = -1$

(*proof*)

lemma *sign-cases* [*case-names neg zero pos*]:

$(\text{sign } x = -1 \Longrightarrow P) \Longrightarrow (\text{sign } x = 0 \Longrightarrow P) \Longrightarrow (\text{sign } x = 1 \Longrightarrow P) \Longrightarrow P$

(*proof*)

lemma *sign-times*:

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

shows $\text{sign } (x * y) = \text{sign } x * \text{sign } y$

(*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**

$sgn-neg-inf\ p \equiv$ if even (degree p) then sgn (lead-coeff p) else $-sgn$ (lead-coeff p)

lemma *sgn-inf-sym*:

fixes $p::real\ poly$

shows $sgn-pos-inf\ (pcompose\ p\ [:0,-1:]) = sgn-neg-inf\ p$ (**is** $?L=?R$)

$\langle proof \rangle$

lemma *poly-pinfty-gt-lc*:

fixes $p::real\ poly$

assumes $lead-coeff\ p > 0$

shows $\exists\ n. \forall\ x \geq n. poly\ p\ x \geq lead-coeff\ p$ $\langle proof \rangle$

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

fixes $p::real\ poly$

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

$\langle proof \rangle$

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

fixes $p::real\ poly$

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

$\langle proof \rangle$

lemma *root-ub*:

fixes $p::real\ poly$

assumes $p \neq 0$

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

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

$\langle proof \rangle$

lemma *root-lb*:

fixes $p::real\ poly$

assumes $p \neq 0$

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

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

$\langle proof \rangle$

2.4 Variation and cross

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

$variation\ x\ y = (if\ x*y \geq 0\ then\ 0\ else\ if\ x < y\ then\ 1\ else\ -1)$

definition *cross* $:: real\ poly \Rightarrow real \Rightarrow real \Rightarrow int$ **where**

$cross\ p\ a\ b = variation\ (poly\ p\ a)\ (poly\ p\ b)$

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

$\langle proof \rangle$

lemma *variation-comm*: $variation\ x\ y = -\ variation\ y\ x$ $\langle proof \rangle$

lemma *cross-0[simp]*: *cross 0 a b=0* \langle *proof* \rangle

lemma *variation-cases*:

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

\langle *proof* \rangle

lemma *variation-congr*:

assumes $\text{sgn } x = \text{sgn } x' \ \text{sgn } y = \text{sgn } y'$
shows $\text{variation } x \ y = \text{variation } x' \ y'$ \langle *proof* \rangle

lemma *variation-mult-pos*:

assumes $c > 0$
shows $\text{variation } (c*x) \ y = \text{variation } x \ y$ **and** $\text{variation } x \ (c*y) = \text{variation } x \ y$
 \langle *proof* \rangle

lemma *variation-mult-neg-1*:

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

lemma *variation-mult-neg-2*:

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

lemma *cross-no-root*:

assumes $a < b$ **and** *no-root*: $\forall x. a < x \wedge x < b \implies \text{poly } p \ x \neq 0$
shows $\text{cross } p \ a \ b = 0$
 \langle *proof* \rangle

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 $\text{sign-r-pos } p \ x = (\text{if } \text{poly } p \ x = 0 \text{ then } \text{sign-r-pos } (pderiv \ p) \ x \text{ else } \text{poly } p \ x > 0)$
 \langle *proof* \rangle

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 $\text{sign-r-pos } (p + q) \ x = \text{sign-r-pos } q \ x$
<proof>

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

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

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

2.7 Jump

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

where

jump-poly *q p x* \equiv (if $p \neq 0 \wedge q \neq 0 \wedge \text{odd}((\text{order } x \ p) - (\text{order } x \ q))$ 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* $= 0$

<proof>

lemma *jump-poly0[simp]*:

jump-poly 0 p x = 0

jump-poly q 0 x = 0

<proof>

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>

lemma *jump-poly-mult*:

fixes *p q p'::real poly*

assumes $p' \neq 0$

shows *jump-poly (p'*q) (p'*p) x* = *jump-poly q p x*

<proof>

lemma *jump-poly-1-mult*:

fixes *p1 p2::real poly*

assumes *poly p1 x* $\neq 0 \vee$ *poly p2 x* $\neq 0$

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>

lemma *jump-poly-mod*:

fixes *p q::real poly*

shows *jump-poly q p x* = *jump-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-poly q p x* = *jump-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-poly (pderiv p * q) p x* = *sign (poly q x)*

<proof>

2.8 Cauchy index

definition *cindex-poly*:: $real \Rightarrow real \Rightarrow real \text{ poly} \Rightarrow real \text{ poly} \Rightarrow int$

where

cindex-poly $a \ b \ q \ p \equiv (\sum_{x \in \{x. \text{poly } p \ x = 0 \wedge a < x \wedge x < b\}}. \text{jump-poly } q \ p \ x)$

lemma *cindex-poly-0[simp]*: $cindex-poly \ a \ b \ 0 \ p = 0 \ cindex-poly \ a \ b \ q \ 0 = 0$

<proof>

lemma *cindex-poly-cross*:

fixes $p::real \text{ poly}$ **and** $a \ b::real$

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

shows $cindex-poly \ a \ b \ 1 \ p = \text{cross } p \ a \ b$

<proof>

lemma *cindex-poly-mult*:

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

assumes $p' \neq 0$

shows $cindex-poly \ a \ b \ (p' * q) \ (p' * p) = cindex-poly \ a \ b \ q \ p$

<proof>

lemma *cindex-poly-smult-1*:

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

shows $cindex-poly \ a \ b \ (\text{smult } c \ q) \ p = (\text{sign } c) * cindex-poly \ a \ b \ q \ p$

<proof>

lemma *cindex-poly-mod*:

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

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

<proof>

lemma *cindex-poly-inverse-add*:

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

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>

lemma *cindex-poly-inverse-add-cross*:

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

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

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

<proof>

lemma *cindex-poly-rec*:

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

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

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

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

<proof>

lemma *cindex-poly-congr*:

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

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

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

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

<proof>

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

<proof>

lemma *cindex-poly-taq*:

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

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

<proof>

2.9 Signed remainder sequence

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

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

<proof>

termination

<proof>

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

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

<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*:: $(\text{'a} :: \text{linordered-idom})\ \text{list} \Rightarrow \text{int}$ **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:*

changes xs = changes (map ((of-int::=>'c::{ring-1,linordered-idom}) o sign) xs)

<proof>

definition *changes-poly-at::('a ::linordered-idom) poly list => 'a => int where*

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

definition *changes-poly-pos-inf:: ('a ::linordered-idom) poly list => int where*

changes-poly-pos-inf ps = changes (map sgn-pos-inf ps)

definition *changes-poly-neg-inf:: ('a ::linordered-idom) poly list => 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

<proof>

definition *changes-itv-smods:: real => real => real poly => real poly => 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 => real poly => real poly => 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 => real poly => real poly => 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 => real poly => 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*

$(-(p \text{ mod } q))$
 $\langle \text{proof} \rangle$

lemma *changes-smods-congr*:

fixes $p\ q:: \text{real poly}$
assumes $a \neq a' \text{ poly } p \ a \neq 0$
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 $\text{changes-poly-at } (\text{smods } p\ q)\ a = \text{changes-poly-at } (\text{smods } p\ q)\ a'$
 $\langle \text{proof} \rangle$

lemma *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 $\text{changes-itv-smods } a\ b\ p\ q = \text{changes-itv-smods } a'\ b'\ p\ q$
 $\langle \text{proof} \rangle$

lemma *cindex-poly-changes-itv-mods*:

assumes $a < b \ \text{poly } p \ a \neq 0 \ \text{poly } p \ b \neq 0$
shows $\text{cindex-poly } a\ b\ q\ p = \text{changes-itv-smods } a\ b\ p\ q \ \langle \text{proof} \rangle$

lemma *root-list-ub*:

fixes $ps:: (\text{real poly}) \text{ list and } a:: \text{real}$
assumes $0 \notin \text{set } ps$
obtains $ub \ \text{where } \forall p \in \text{set } ps. \forall x. \text{poly } p\ x = 0 \longrightarrow x < ub$
and $\forall x \geq ub. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p\ x) = \text{sgn-pos-inf } p \ \text{and } ub > a$
 $\langle \text{proof} \rangle$

lemma *root-list-lb*:

fixes $ps:: (\text{real poly}) \text{ list and } b:: \text{real}$
assumes $0 \notin \text{set } ps$
obtains $lb \ \text{where } \forall p \in \text{set } ps. \forall x. \text{poly } p\ x = 0 \longrightarrow x > lb$
and $\forall x \leq lb. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p\ x) = \text{sgn-neg-inf } p \ \text{and } lb < b$
 $\langle \text{proof} \rangle$

theorem *sturm-tarski-interval*:

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

theorem *sturm-tarski-above*:

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

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*⟩

theorem *sturm-tarski-R:*
shows *taq {x. poly p x=0} q = changes-R-smods p (pderiv p * q)*
 ⟨*proof*⟩

theorem *sturm-interval:*
assumes *a < b poly p a ≠ 0 poly p b ≠ 0*
shows *card {x. poly p x = 0 ∧ a < x ∧ x < b} = changes-itv-smods a b p (pderiv p)*
 ⟨*proof*⟩

theorem *sturm-above:*
assumes *poly p a ≠ 0*
shows *card {x. poly p x = 0 ∧ a < x} = changes-gt-smods a p (pderiv p)*
 ⟨*proof*⟩

theorem *sturm-below:*
assumes *poly p b ≠ 0*
shows *card {x. poly p x = 0 ∧ x < b} = changes-le-smods b p (pderiv p)*
 ⟨*proof*⟩

theorem *sturm-R:*
shows *card {x. poly p x=0} = changes-R-smods p (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 ⇒ 'a poly ⇒ ('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 (smods q (smult m (pseudo-mod p q)))
⟨proof⟩
termination
⟨proof⟩

declare smods.simps[simp del]

lemma smods-0[simp]:
  smods 0 q = []
  smods p 0 = (if p=0 then [] else [p])
⟨proof⟩

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

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)
⟨proof⟩

lemma smods-smult-length:
  assumes  $a \neq 0$   $b \neq 0$ 
  shows length (smods p q) = length (smods (smult a p) (smult b q)) ⟨proof⟩

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

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

lemma changes-poly-at-smods-smult:
  assumes  $m > 0$ 
  shows changes-poly-at (smods p (smult m q)) x = changes-poly-at (smods p q) x
⟨proof⟩

lemma smods-smods-sgn-map-eq:
  fixes p q::real poly and x::real
  defines  $f \equiv \lambda p$ . sgn (poly p x)
  shows map f (smods p q) = map f (smods p q)

```

$map\ sgn\ pos\ inf\ (smods\ p\ q) = map\ sgn\ pos\ inf\ (spmods\ p\ q)$
 $map\ sgn\ neg\ inf\ (smods\ p\ q) = map\ sgn\ neg\ inf\ (spmods\ p\ q)$
 ⟨proof⟩

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*
 ⟨proof⟩

lemma (**in** *ring-char-0*) *of-int-inj[simp]*: *inj of-int*
 ⟨proof⟩

lemma *int-of-rat-id*: $int\ of\ rat\ o\ of\ int = id$
 ⟨proof⟩

lemma *int-of-rat-0[simp]*: $int\ of\ rat\ 0 = 0$
 ⟨proof⟩

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

lemma *int-of-rat-0-iff:x $\in\mathbb{Z}$* \Longrightarrow $int\ of\ rat\ x = 0 \iff x = 0$
 ⟨proof⟩

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

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

lemma *de-lcm-pCons:de-lcm (pCons a p)* = $lcm\ (snd\ (quotient\ of\ a))\ (de\ lcm\ p)$
 ⟨proof⟩

lemma *de-lcm-0[simp]*: $de\ lcm\ 0 = 1$ ⟨proof⟩

lemma *de-lcm-pos[simp]*: $de\ lcm\ p > 0$
 ⟨proof⟩

lemma *de-lcm-ints*:
fixes *x::rat*
shows $x \in set\ (coeffs\ p) \Longrightarrow rat\ of\ int\ (de\ lcm\ p) * x \in \mathbb{Z}$
 ⟨proof⟩

definition *clear-de::rat poly* \Rightarrow *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⟩

lemma *clear-de-0[simp]:clear-de 0 = 0*
 ⟨proof⟩

lemma [code abstract]: *coeffs* (*clear-de p*) =
 (let *lcm* = de-lcm *p* in map (λx . int-of-rat (of-int *lcm* * *x*)) (coeffs *p*))
 ⟨proof⟩

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: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 ⇒ 'b ⇒ 'a poly ⇒ 'a poly ⇒ 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 ⇒ 'a poly ⇒ 'a poly ⇒ 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 ⇒ 'a poly ⇒ 'a poly ⇒ 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 ⇒ 'a poly ⇒ 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 R₁ 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**

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

definition $TaQ\text{-gt-rats}:: \text{rat} \Rightarrow \text{rat poly} \Rightarrow \text{rat poly} \Rightarrow \text{int}$ **where**

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

definition $TaQ\text{-le-rats}:: \text{rat} \Rightarrow \text{rat poly} \Rightarrow \text{rat poly} \Rightarrow \text{int}$ **where**

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

lemma taq-smult-pos :

assumes $a > 0$

shows $\text{taq } s \ (\text{smult } a \ p) = \text{taq } s \ p$

$\langle \text{proof} \rangle$

lemma $\text{taq-proots-R-code}[code]$:

$TaQ\text{-R-rats } p \ q = (\text{let}$
 $\ \ ip = \text{clear-de } p;$
 $\ \ iq = \text{clear-de } q$
 $\ \ \text{in ri-changes-R-spmods } ip \ (\text{pderiv } ip * iq))$

$\langle \text{proof} \rangle$

lemma $\text{taq-proots-itv-code}[code]$:

$TaQ\text{-itv-rats } a \ b \ p \ q = (\text{if } a \geq b \ \text{then}$
 $\ \ 0$
 $\ \ \text{else if poly } p \ a \neq 0 \wedge \text{poly } p \ b \neq 0 \ \text{then}$
 $\ \ \ (\text{let}$
 $\ \ \ \ ip = \text{clear-de } p;$
 $\ \ \ \ iq = \text{clear-de } q$
 $\ \ \ \ \text{in ri-changes-itv-spmods } a \ b \ ip \ (\text{pderiv } ip * iq))$
 $\ \ \ \text{else}$
 $\ \ \ \ \text{Code.abort } (\text{STR } \text{"Roots at border yet to be supported"})$
 $\ \ \ \ \ (\lambda-. \text{TaQ-itv-rats } a \ b \ p \ q)$

\rangle
 $\langle \text{proof} \rangle$

lemma $\text{taq-proots-gt-code}[code]$:

$TaQ\text{-gt-rats } a \ p \ q = (\text{if poly } p \ a \neq 0 \ \text{then}$
 $\ \ (\text{let}$
 $\ \ \ ip = \text{clear-de } p;$
 $\ \ \ iq = \text{clear-de } q$
 $\ \ \ \text{in ri-changes-gt-spmods } a \ ip \ (\text{pderiv } ip * iq))$
 $\ \ \text{else}$
 $\ \ \ \text{Code.abort } (\text{STR } \text{"Roots at border yet to be supported"})$
 $\ \ \ \ (\lambda-. \text{TaQ-gt-rats } a \ p \ q)$

\rangle
 $\langle \text{proof} \rangle$

```

lemma taq-roots-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.