The Sturm-Tarski Theorem

Wenda Li

April 20, 2020

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].

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*⟩

lemma *smult-cancel*:
fixes *p*::*'a*::*idom poly*
assumes *c*≠*0* **and** *smult*: *smult c p* = *smult c q*
shows *p*=*q*
⟨*proof*⟩

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* ⟨*proof*⟩

lemma *poly-power-n-eq*:
fixes *x*::*'a* :: *idom*
assumes *n*≠*0*

shows *poly* ([:−*a*,*1*:] ^ *n*) *x*=*0* ⟷ (*x*=*a*) ⟨*proof*⟩

lemma *poly-power-n-odd*:
fixes *x a*:: *real*
assumes *odd n*
shows *poly* ([:−*a*,*1*:] ^ *n*) *x*>*0* ⟷ (*x*>*a*) ⟨*proof*⟩

lemma *gcd-coprime-poly*:
fixes *p q*::'*a*::{*factorial-ring-gcd*,*semiring-gcd-mult-normalize*} *poly*
assumes *nz*: *p* ≠ *0* ∨ *q* ≠ *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*⟩

end

1 Sturm-Tarski Theorem

theory *Sturm-Tarski*
imports *Complex-Main PolyMisc HOL−Computational-Algebra.Field-as-Ring*
begin

2 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* (λ*x*. *P x*) *F*
shows ¬ *eventually* (λ*x*. ¬ *P x*) *F*
⟨*proof*⟩

lemma *poly-tendsto*[*simp*]:
(*poly p* ⟶ *poly p x*) (*at* (*x*::*real*))
(*poly p* ⟶ *poly p x*) (*at-left* (*x*::*real*))
(*poly p* ⟶ *poly p x*) (*at-right* (*x*::*real*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

3 Bound of polynomials

definition *sgn-pos-inf* :: (*'a* ::*linordered-idom*) *poly* ⇒ *'a* **where**
sgn-pos-inf p ≡ *sgn* (*lead-coeff p*)
definition *sgn-neg-inf* :: (*'a* ::*linordered-idom*) *poly* ⇒ *'a* **where**
sgn-neg-inf p ≡ *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*)
⟨*proof*⟩

lemma *poly-pinfty-gt-lc*:
fixes *p*:: *real poly*
assumes *lead-coeff p* > *0*
shows ∃ *n*. ∀ *x* ≥ *n*. *poly p x* ≥ *lead-coeff p* ⟨*proof*⟩

lemma *poly-sgn-eventually-at-top*:
fixes *p*::*real poly*
shows *eventually* (λ*x*. *sgn* (*poly p x*) = *sgn-pos-inf p*) *at-top*
⟨*proof*⟩

lemma *poly-sgn-eventually-at-bot*:
fixes *p*::*real poly*
shows *eventually* (λ*x*. *sgn* (*poly p x*) = *sgn-neg-inf p*) *at-bot*
⟨*proof*⟩

lemma *root-ub*:
fixes *p*:: *real poly*
assumes *p*≠*0*
obtains *ub* **where** ∀ *x*. *poly p x*=*0* ⟶ *x*<*ub*
and ∀ *x*≥*ub*. *sgn* (*poly p x*) = *sgn-pos-inf p*
⟨*proof*⟩

lemma *root-lb*:
fixes *p*:: *real poly*
assumes *p*≠*0*
obtains *lb* **where** ∀ *x*. *poly p x*=*0* ⟶ *x*>*lb*
and ∀ *x*≤*lb*. *sgn* (*poly p x*) = *sgn-neg-inf p*
⟨*proof*⟩

4 Sign

definition *sign*:: *'a*::{*zero*,*linorder*} ⇒ *int* **where**
sign x≡(*if x*>*0 then 1 else if x*=*0 then 0 else* −*1*)

lemma *sign-simps*[*simp*]:
x>*0* ⟹ *sign x*=*1*
x=*0* ⟹ *sign x*=*0*
x<*0* ⟹ *sign x*=−*1*
⟨*proof*⟩

lemma *sign-cases* [*case-names neg zero pos*]:
(*sign x* = −*1* ⟹ *P*) ⟹ (*sign x* = *0* ⟹ *P*) ⟹ (*sign x* =*1* ⟹ *P*) ⟹ *P*
⟨*proof*⟩

lemma *sign-times*:
fixes *x*::*'a*::*linordered-ring-strict*
shows *sign* (*x*∗*y*) = *sign x* ∗ *sign y*
⟨*proof*⟩

lemma *sign-power*:
fixes *x*::*'a*::*linordered-idom*
shows *sign* (*x*^*n*) = (*if n*=*0 then 1 else if even n then* |*sign x*| *else sign x*)
⟨*proof*⟩

lemma *sgn-sign-eq*:
fixes *x*::*'a*::{*linordered-idom*}
shows *sgn x* = *of-int* (*sign x*)
⟨*proof*⟩

5 Variation and cross

definition *variation* :: *real* ⇒ *real* ⇒ *int* **where**
variation x y=(*if x*∗*y*≥*0 then 0 else if x*<*y then 1 else* −*1*)

definition *cross* :: *real poly* ⇒ *real* ⇒ *real* ⇒ *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*
⟨*proof*⟩

lemma *variation-comm*: *variation x y*= − *variation y x* ⟨*proof*⟩

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

lemma *variation-cases*:
⟦*x*>*0*;*y*>*0*⟧⟹*variation x y* = *0*
⟦*x*>*0*;*y*<*0*⟧⟹*variation x y* = −*1*
⟦*x*<*0*;*y*>*0*⟧⟹*variation x y* = *1*
⟦*x*<*0*;*y*<*0*⟧⟹*variation x y* = *0*
⟨*proof*⟩

lemma *variation-congr*:
assumes *sgn x*=*sgn x*′ *sgn y*=*sgn y*′
shows *variation x y*=*variation x*′ *y*′ ⟨*proof*⟩

lemma *variation-mult-pos*:
assumes *c*>*0*
shows *variation* (*c*∗*x*) *y* =*variation x y* **and** *variation x* (*c*∗*y*) =*variation x y*
⟨*proof*⟩

lemma *variation-mult-neg-1*:
assumes *c*<*0*
shows *variation* (*c*∗*x*) *y* =*variation x y* + (*if y*=*0 then 0 else sign x*)
⟨*proof*⟩

lemma *variation-mult-neg-2*:
assumes *c*<*0*
shows *variation x* (*c*∗*y*) = *variation x y* + (*if x*=*0 then 0 else* − *sign y*)
⟨*proof*⟩

lemma *cross-no-root*:
assumes *a*<*b* **and** *no-root*:∀ *x*. *a*<*x*∧*x*<*b* ⟶ *poly p x*≠*0*
shows *cross p a b*=*0*
⟨*proof*⟩

6 Tarski query

definition *taq* :: *'a*::*linordered-idom set* ⇒ *'a poly* ⇒ *int* **where**

taq s q ≡ $\sum x{\in}s.$ *sign* (*poly q x*)

7 Sign at the right

definition *sign-r-pos* :: *real poly* ⇒ *real* ⇒ *bool*
where
sign-r-pos p x≡ (*eventually* (λ*x*. *poly p x*>*0*) (*at-right x*))

lemma *sign-r-pos-rec*:
fixes *p*:: *real poly*
assumes *p*≠*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*⟩

lemma *sign-r-pos-0*[*simp*]:¬ *sign-r-pos 0* (*x*::*real*)
⟨*proof*⟩

lemma *sign-r-pos-minus*:
fixes *p*:: *real poly*
assumes *p*≠*0*
shows *sign-r-pos p x* = (¬ *sign-r-pos* (−*p*) *x*)
⟨*proof*⟩

lemma *sign-r-pos-smult*:
fixes *p* :: *real poly*
assumes *c*≠*0 p*≠*0*
shows *sign-r-pos* (*smult c p*) *x*= (*if c*>*0 then sign-r-pos p x else* ¬ *sign-r-pos p*
x)
(**is** *?L*=*?R*)
⟨*proof*⟩

lemma *sign-r-pos-mult*:
fixes *p q* :: *real poly*
assumes *p*≠*0 q*≠*0*
shows *sign-r-pos* (*p*∗*q*) *x*= (*sign-r-pos p x* ⟷ *sign-r-pos q x*)
⟨*proof*⟩

lemma *sign-r-pos-add*:
fixes *p q* :: *real poly*
assumes *poly p x*=*0 poly q x*≠*0*
shows *sign-r-pos* (*p*+*q*) *x*=*sign-r-pos q x*
⟨*proof*⟩

lemma *sign-r-pos-mod*:
fixes *p q* :: *real poly*
assumes *poly p x*=*0 poly q x*≠*0*
shows *sign-r-pos* (*q mod p*) *x*=*sign-r-pos q x*
⟨*proof*⟩

lemma *sign-r-pos-pderiv*:
fixes *p*:: *real poly*
assumes *poly p x=0 p≠0*
shows *sign-r-pos* (*pderiv p* ∗ *p*) *x*
⟨*proof*⟩

lemma *sign-r-pos-power*:
fixes *p*:: *real poly* **and** *a*::*real*
shows *sign-r-pos* ([:−*a*,*1*:]^*n*) *a*
⟨*proof*⟩

8 Jump

definition *jump-poly* :: *real poly* ⇒ *real poly* ⇒*real* ⇒ *int*
where
jump-poly q p x≡ (*if p≠0* ∧ *q≠0* ∧ *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≠0*⟹ *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'≠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≠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*⟩

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≠0 poly p x=0*
shows *jump-poly* (*pderiv p ∗ q*) *p x* = *sign* (*poly q x*)
⟨*proof*⟩

9 Cauchy index

definition *cindex-poly*:: *real* ⇒ *real* ⇒ *real poly* ⇒ *real poly* ⇒ *int*
where
cindex-poly a b q p≡ (∑ *x*∈{*x*. *poly p x=0* ∧ *a*< *x*∧ *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*
⟨*proof*⟩

lemma *cindex-poly-cross*:
fixes *p*::*real poly* **and** *a b*::*real*
assumes *a*<*b poly p a≠0 poly p b≠0*
shows *cindex-poly a b 1 p* = *cross p a b*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *cindex-poly-mod*:
fixes *p q*::*real poly*
shows *cindex-poly a b q p* = *cindex-poly a b* (*q mod p*) *p*
⟨*proof*⟩

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*⟩

lemma *cindex-poly-inverse-add-cross*:
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* + *cindex-poly a b p q* = *cross* (*p* ∗ *q*) *a b* (**is** *?L=?R*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *greaterThanLessThan-unfold*:{*a*<..<*b*} = {*x*. *a*<*x* ∧ *x*<*b*}
⟨*proof*⟩

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*⟩

10 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*))))
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *smods-nil-eq*:*smods p q* = [] ⟷ (*p*=*0*) ⟨*proof*⟩
lemma *smods-singleton*:[*x*] = *smods p q* ⟹ (*p*≠*0* ∧ *q*=*0* ∧ *x*=*p*)
⟨*proof*⟩

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

lemma *no-0-in-smods*: *0∉set* (*smods p q*)
⟨*proof*⟩

fun *changes*:: (*'a* ::*linordered-idom*) *list* ⇒ *int* **where**
changes [] = *0*|
changes [-] = *0* |
changes (*x1*#*x2*#*xs*) = (*if x1*∗*x2*<*0 then 1*+*changes* (*x2*#*xs*)
else if x2=*0 then changes* (*x1*#*xs*)
else changes (*x2*#*xs*))

lemma *changes-map-sgn-eq*:
changes xs = *changes* (*map sgn xs*)
⟨*proof*⟩

definition *changes-poly-at*::(*'a* ::*linordered-idom*) *poly list* ⇒ *'a* ⇒ *int* **where**
changes-poly-at ps a= *changes* (*map* (λ*p*. *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 mod q*))
⟨*proof*⟩

lemma *changes-smods-congr*:
fixes *p q*:: *real poly*
assumes *a*≠*a*′ *poly p a*≠*0*
assumes ∀ *p*∈*set* (*smods p q*). ∀ *x*. ((*a*<*x*∧*x*≤*a*′) ∨ (*a*′≤*x* ∧ *x*<*a*)) ⟶ *poly p x* ≠*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*:∀ *p*∈*set* (*smods p q*). ∀ *x*. ((*a*<*x*∧*x*≤*a*′) ∨ (*b*′≤*x* ∧ *x*<*b*)) ⟶ *poly p x* ≠*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*:: (*real poly*) *list* **and** *a*::*real*
assumes *0*∉*set ps*
obtains *ub* **where** ∀ *p*∈*set ps*. ∀ *x*. *poly p x*=*0* ⟶ *x*<*ub*
and ∀ *x*≥*ub*. ∀ *p*∈*set ps*. *sgn* (*poly p x*) = *sgn-pos-inf p* **and** *ub*>*a*
⟨*proof*⟩

lemma *root-list-lb*:
fixes *ps*:: (*real poly*) *list* **and** *b*::*real*
assumes *0*∉*set ps*
obtains *lb* **where** ∀ *p*∈*set ps*. ∀ *x*. *poly p x*=*0* ⟶ *x*>*lb*
and ∀ *x*≤*lb*. ∀ *p*∈*set ps*. *sgn* (*poly p x*) = *sgn-neg-inf p* **and** *lb*<*b*
⟨*proof*⟩

theorem *sturm-tarski-interval*:
assumes *a*<*b poly p a*≠*0 poly p b*≠*0*
shows *taq* {*x*. *poly p x*=*0* ∧ *a*<*x* ∧ *x*<*b*} *q* = *changes-itv-smods a b p* (*pderiv p* ∗ *q*)

⟨*proof*⟩

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

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

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.