

# A Formalisation of Sturm's Theorem

Manuel Eberl

May 26, 2024

## Abstract

*Sturm sequences* are a method for computing the number of real roots of a real polynomial inside a given interval efficiently. In this project, this fact and a number of methods to construct Sturm sequences efficiently have been formalised with the interactive theorem prover Isabelle/HOL. Building upon this, an Isabelle/HOL proof method was then implemented to prove statements about the number of roots of a real polynomial and related properties.

# Contents

<b>1</b>	<b>Miscellaneous</b>	<b>3</b>
1.1	Analysis . . . . .	3
1.2	Polynomials . . . . .	3
1.2.1	General simplification lemmas . . . . .	3
1.2.2	Divisibility of polynomials . . . . .	3
1.2.3	Sign changes of a polynomial . . . . .	4
1.2.4	Limits of polynomials . . . . .	4
1.2.5	Signs of polynomials for sufficiently large values . . . . .	6
1.2.6	Positivity of polynomials . . . . .	7
<b>2</b>	<b>Proof of Sturm’s Theorem</b>	<b>8</b>
2.1	Sign changes of polynomial sequences . . . . .	8
2.2	Definition of Sturm sequences locale . . . . .	9
2.3	Auxiliary lemmas about roots and sign changes . . . . .	10
2.4	Constructing Sturm sequences . . . . .	13
2.5	The canonical Sturm sequence . . . . .	13
2.5.1	Canonical squarefree Sturm sequence . . . . .	15
2.5.2	Optimisation for multiple roots . . . . .	16
2.6	Root-counting functions . . . . .	17
<b>3</b>	<b>The “sturm” proof method</b>	<b>19</b>
3.1	Preliminary lemmas . . . . .	19
3.2	Reification . . . . .	23
3.3	Setup for the “sturm” method . . . . .	25
<b>4</b>	<b>Example usage of the “sturm” method</b>	<b>26</b>

# 1 Miscellaneous

```
theory Misc-Polynomial
imports HOL-Computational-Algebra.Polynomial HOL-Computational-Algebra.Polynomial-Factorial
Pure-ex.Guess
begin
```

## 1.1 Analysis

```
lemma fun-eq-in-ivl:
  assumes  $a \leq b \ \forall x::\text{real}. a \leq x \wedge x \leq b \longrightarrow \text{eventually } (\lambda \xi. f \ \xi = f \ x) \text{ (at } x)$ 
  shows  $f \ a = f \ b$ 
<proof>
```

## 1.2 Polynomials

### 1.2.1 General simplification lemmas

```
lemma pderiv-div:
  assumes [simp]:  $q \ \text{dvd} \ p \ \ q \neq 0$ 
  shows  $\text{pderiv } (p \ \text{div} \ q) = (q * \text{pderiv } p - p * \text{pderiv } q) \ \text{div} \ (q * q)$ 
 $q * q \ \text{dvd} \ (q * \text{pderiv } p - p * \text{pderiv } q)$ 
<proof>
```

### 1.2.2 Divisibility of polynomials

Two polynomials that are coprime have no common roots.

```
lemma coprime-imp-no-common-roots:
   $\neg (\text{poly } p \ x = 0 \wedge \text{poly } q \ x = 0) \ \text{if } \text{coprime } p \ q$ 
  for  $x :: 'a :: \text{field}$ 
<proof>
```

```
lemma poly-div:
  assumes  $\text{poly } q \ x \neq 0$  and  $(q::'a :: \text{field poly}) \ \text{dvd} \ p$ 
  shows  $\text{poly } (p \ \text{div} \ q) \ x = \text{poly } p \ x / \text{poly } q \ x$ 
<proof>
```

```
lemma poly-div-gcd-squarefree-aux:
  assumes  $\text{pderiv } (p::('a::\{\text{field-char-0,field-gcd}\}) \ \text{poly}) \neq 0$ 
  defines  $d \equiv \text{gcd } p \ (\text{pderiv } p)$ 
  shows  $\text{coprime } (p \ \text{div} \ d) \ (\text{pderiv } (p \ \text{div} \ d))$  and
 $\bigwedge x. \text{poly } (p \ \text{div} \ d) \ x = 0 \longleftrightarrow \text{poly } p \ x = 0$ 
<proof>
```

```
lemma normalize-field:
   $\text{normalize } (x :: 'a :: \{\text{field,normalization-semidom}\}) = (\text{if } x = 0 \text{ then } 0 \text{ else } 1)$ 
<proof>
```

**lemma** *normalize-field-eq-1* [*simp*]:

$x \neq 0 \implies \text{normalize } (x :: 'a :: \{\text{field}, \text{normalization-semidom}\}) = 1$   
(*proof*)

**lemma** *unit-factor-field* [*simp*]:

$\text{unit-factor } (x :: 'a :: \{\text{field}, \text{normalization-semidom}\}) = x$   
(*proof*)

Dividing a polynomial by its gcd with its derivative yields a squarefree polynomial with the same roots.

**lemma** *poly-div-gcd-squarefree*:

**assumes**  $(p :: ('a :: \{\text{field-char-0}, \text{field-gcd}\}) \text{poly}) \neq 0$

**defines**  $d \equiv \text{gcd } p \ (\text{pderiv } p)$

**shows** *coprime*  $(p \ \text{div} \ d) \ (\text{pderiv } (p \ \text{div} \ d))$  (**is** ?A) **and**

$\bigwedge x. \text{poly } (p \ \text{div} \ d) \ x = 0 \iff \text{poly } p \ x = 0$  (**is**  $\bigwedge x. ?B \ x$ )

(*proof*)

### 1.2.3 Sign changes of a polynomial

If a polynomial has different signs at two points, it has a root inbetween.

**lemma** *poly-different-sign-imp-root*:

**assumes**  $a < b$  **and**  $\text{sgn } (\text{poly } p \ a) \neq \text{sgn } (\text{poly } p \ (b :: \text{real}))$

**shows**  $\exists x. a \leq x \wedge x \leq b \wedge \text{poly } p \ x = 0$

(*proof*)

**lemma** *poly-different-sign-imp-root'*:

**assumes**  $\text{sgn } (\text{poly } p \ a) \neq \text{sgn } (\text{poly } p \ (b :: \text{real}))$

**shows**  $\exists x. \text{poly } p \ x = 0$

(*proof*)

**lemma** *no-roots-inbetween-imp-same-sign*:

**assumes**  $a < b \ \forall x. a \leq x \wedge x \leq b \implies \text{poly } p \ x \neq (0 :: \text{real})$

**shows**  $\text{sgn } (\text{poly } p \ a) = \text{sgn } (\text{poly } p \ b)$

(*proof*)

### 1.2.4 Limits of polynomials

**lemma** *poly-neighbourhood-without-roots*:

**assumes**  $(p :: \text{real poly}) \neq 0$

**shows** *eventually*  $(\lambda x. \text{poly } p \ x \neq 0)$  (at  $x_0$ )

(*proof*)

**lemma** *poly-neighbourhood-same-sign*:

**assumes**  $\text{poly } p \ (x_0 :: \text{real}) \neq 0$

**shows** *eventually*  $(\lambda x. \text{sgn } (\text{poly } p \ x) = \text{sgn } (\text{poly } p \ x_0))$  (at  $x_0$ )

(*proof*)

**lemma** *poly-lhopital*:

**assumes**  $\text{poly } p (x::\text{real}) = 0$   $\text{poly } q x = 0$   $q \neq 0$

**assumes**  $(\lambda x. \text{poly } (\text{pderiv } p) x / \text{poly } (\text{pderiv } q) x) -x \rightarrow y$

**shows**  $(\lambda x. \text{poly } p x / \text{poly } q x) -x \rightarrow y$

*<proof>*

**lemma** *poly-roots-bounds*:

**assumes**  $p \neq 0$

**obtains**  $l u$

**where**  $l \leq (u :: \text{real})$

**and**  $\text{poly } p l \neq 0$

**and**  $\text{poly } p u \neq 0$

**and**  $\{x. x > l \wedge x \leq u \wedge \text{poly } p x = 0\} = \{x. \text{poly } p x = 0\}$

**and**  $\bigwedge x. x \leq l \implies \text{sgn } (\text{poly } p x) = \text{sgn } (\text{poly } p l)$

**and**  $\bigwedge x. x \geq u \implies \text{sgn } (\text{poly } p x) = \text{sgn } (\text{poly } p u)$

*<proof>*

**definition** *poly-inf* ::  $('a::\text{real-normed-vector}) \text{poly} \Rightarrow 'a$  **where**

$\text{poly-inf } p \equiv \text{sgn } (\text{coeff } p (\text{degree } p))$

**definition** *poly-neg-inf* ::  $('a::\text{real-normed-vector}) \text{poly} \Rightarrow 'a$  **where**

$\text{poly-neg-inf } p \equiv \text{if even } (\text{degree } p) \text{ then } \text{sgn } (\text{coeff } p (\text{degree } p))$

$\text{else } -\text{sgn } (\text{coeff } p (\text{degree } p))$

**lemma** *poly-inf-0-iff*[simp]:

$\text{poly-inf } p = 0 \iff p = 0$   $\text{poly-neg-inf } p = 0 \iff p = 0$

*<proof>*

**lemma** *poly-inf-mult*[simp]:

**fixes**  $p :: ('a::\text{real-normed-field}) \text{poly}$

**shows**  $\text{poly-inf } (p*q) = \text{poly-inf } p * \text{poly-inf } q$

$\text{poly-neg-inf } (p*q) = \text{poly-neg-inf } p * \text{poly-neg-inf } q$

*<proof>*

**lemma** *poly-neq-0-at-infinity*:

**assumes**  $(p :: \text{real poly}) \neq 0$

**shows** *eventually*  $(\lambda x. \text{poly } p x \neq 0)$  *at-infinity*

*<proof>*

**lemma** *poly-limit-aux*:

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

**defines**  $n \equiv \text{degree } p$

**shows**  $((\lambda x. \text{poly } p x / x^n) \longrightarrow \text{coeff } p n)$  *at-infinity*

*<proof>*

**lemma** *poly-at-top-at-top*:

**fixes**  $p :: \text{real poly}$   
**assumes**  $\text{degree } p \geq 1 \text{ coeff } p (\text{degree } p) > 0$   
**shows**  $\text{LIM } x \text{ at-top. } \text{poly } p \ x \text{ :> at-top}$   
(proof)

**lemma** *poly-at-bot-at-top*:

**fixes**  $p :: \text{real poly}$   
**assumes**  $\text{degree } p \geq 1 \text{ coeff } p (\text{degree } p) < 0$   
**shows**  $\text{LIM } x \text{ at-top. } \text{poly } p \ x \text{ :> at-bot}$   
(proof)

**lemma** *poly-lim-inf*:

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

**lemma** *poly-at-top-or-bot-at-bot*:

**fixes**  $p :: \text{real poly}$   
**assumes**  $\text{degree } p \geq 1 \text{ coeff } p (\text{degree } p) > 0$   
**shows**  $\text{LIM } x \text{ at-bot. } \text{poly } p \ x \text{ :> (if even (degree } p) \text{ then at-top else at-bot)}$   
(proof)

**lemma** *poly-at-bot-or-top-at-bot*:

**fixes**  $p :: \text{real poly}$   
**assumes**  $\text{degree } p \geq 1 \text{ coeff } p (\text{degree } p) < 0$   
**shows**  $\text{LIM } x \text{ at-bot. } \text{poly } p \ x \text{ :> (if even (degree } p) \text{ then at-bot else at-top)}$   
(proof)

**lemma** *poly-lim-neg-inf*:

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

## 1.2.5 Signs of polynomials for sufficiently large values

**lemma** *polys-inf-sign-thresholds*:

**assumes** *finite* ( $ps :: \text{real poly set}$ )  
**obtains**  $l \ u$   
**where**  $l \leq u$   
**and**  $\bigwedge p. \llbracket p \in ps; p \neq 0 \rrbracket \implies$   
 $\{x. l < x \wedge x \leq u \wedge \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0\}$   
**and**  $\bigwedge p \ x. \llbracket p \in ps; x \geq u \rrbracket \implies \text{sgn } (\text{poly } p \ x) = \text{poly-inf } p$   
**and**  $\bigwedge p \ x. \llbracket p \in ps; x \leq l \rrbracket \implies \text{sgn } (\text{poly } p \ x) = \text{poly-neg-inf } p$   
(proof)

## 1.2.6 Positivity of polynomials

**lemma** *poly-pos*:

$$(\forall x::\text{real}. \text{poly } p \ x > 0) \longleftrightarrow \text{poly-inf } p = 1 \wedge (\forall x. \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-greater*:

$$(\forall x::\text{real}. x > a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$\text{poly-inf } p = 1 \wedge (\forall x. x > a \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-geq*:

$$(\forall x::\text{real}. x \geq a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$\text{poly-inf } p = 1 \wedge (\forall x. x \geq a \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-less*:

$$(\forall x::\text{real}. x < a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$\text{poly-neg-inf } p = 1 \wedge (\forall x. x < a \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-leq*:

$$(\forall x::\text{real}. x \leq a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$\text{poly-neg-inf } p = 1 \wedge (\forall x. x \leq a \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-between-less-less*:

$$(\forall x::\text{real}. a < x \wedge x < b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$(a \geq b \vee \text{poly } p \ ((a+b)/2) > 0) \wedge (\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-between-less-leq*:

$$(\forall x::\text{real}. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$(a \geq b \vee \text{poly } p \ b > 0) \wedge (\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-between-leq-less*:

$$(\forall x::\text{real}. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$(a \geq b \vee \text{poly } p \ a > 0) \wedge (\forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**lemma** *poly-pos-between-leq-leq*:

$$(\forall x::\text{real}. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$
$$(a > b \vee \text{poly } p \ a > 0) \wedge (\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0)$$

*<proof>*

**end**

## 2 Proof of Sturm's Theorem

```
theory Sturm-Theorem
  imports HOL-Computational-Algebra.Polynomial
         Lib/Sturm-Library HOL-Computational-Algebra.Field-as-Ring
begin
```

### 2.1 Sign changes of polynomial sequences

For a given sequence of polynomials, this function computes the number of sign changes of the sequence of polynomials evaluated at a given position  $x$ . A sign change is a change from a negative value to a positive one or vice versa; zeros in the sequence are ignored.

**definition** *sign-changes where*

```
sign-changes ps (x::real) =
  length (remdups-adj (filter ( $\lambda x. x \neq 0$ ) (map ( $\lambda p. \text{sgn} (\text{poly } p \ x)$ ) ps))) - 1
```

The number of sign changes of a sequence distributes over a list in the sense that the number of sign changes of a sequence  $p_1, \dots, p_i, \dots, p_n$  at  $x$  is the same as the sum of the sign changes of the sequence  $p_1, \dots, p_i$  and  $p_i, \dots, p_n$  as long as  $p_i(x) \neq 0$ .

**lemma** *sign-changes-distrib:*

```
poly p x  $\neq 0 \implies$ 
  sign-changes (ps1 @ [p] @ ps2) x =
  sign-changes (ps1 @ [p]) x + sign-changes ([p] @ ps2) x
<proof>
```

The following two congruences state that the number of sign changes is the same if all the involved signs are the same.

**lemma** *sign-changes-cong:*

```
assumes length ps = length ps'
assumes  $\forall i < \text{length } ps. \text{sgn} (\text{poly } (ps!i) \ x) = \text{sgn} (\text{poly } (ps'!i) \ y)$ 
shows sign-changes ps x = sign-changes ps' y
<proof>
```

**lemma** *sign-changes-cong':*

```
assumes  $\forall p \in \text{set } ps. \text{sgn} (\text{poly } p \ x) = \text{sgn} (\text{poly } p \ y)$ 
shows sign-changes ps x = sign-changes ps y
<proof>
```

For a sequence of polynomials of length 3, if the first and the third polynomial have opposite and nonzero sign at some  $x$ , the number of sign changes is always 1, irrespective of the sign of the second polynomial.

**lemma** *sign-changes-sturm-triple:*

```
assumes poly p x  $\neq 0$  and  $\text{sgn} (\text{poly } r \ x) = - \text{sgn} (\text{poly } p \ x)$ 
shows sign-changes [p,q,r] x = 1
<proof>
```

Finally, we define two additional functions that count the sign changes “at infinity”.

**definition** *sign-changes-inf* **where**

$$\text{sign-changes-inf } ps = \text{length } (\text{remdups-adj } (\text{filter } (\lambda x. x \neq 0) (\text{map } \text{poly-inf } ps))) - 1$$

**definition** *sign-changes-neg-inf* **where**

$$\text{sign-changes-neg-inf } ps = \text{length } (\text{remdups-adj } (\text{filter } (\lambda x. x \neq 0) (\text{map } \text{poly-neg-inf } ps))) - 1$$

## 2.2 Definition of Sturm sequences locale

We first define the notion of a “Quasi-Sturm sequence”, which is a weakening of a Sturm sequence that captures the properties that are fulfilled by a nonempty suffix of a Sturm sequence:

- The sequence is nonempty.
- The last polynomial does not change its sign.
- If the middle one of three adjacent polynomials has a root at  $x$ , the other two have opposite and nonzero signs at  $x$ .

**locale** *quasi-sturm-seq* =

**fixes**  $ps :: (\text{real poly}) \text{ list}$

**assumes**  $\text{last-ps-sgn-const}[simp]$ :

$$\bigwedge x y. \text{sgn } (\text{poly } (\text{last } ps) x) = \text{sgn } (\text{poly } (\text{last } ps) y)$$

**assumes**  $\text{ps-not-Nil}[simp]$ :  $ps \neq []$

**assumes**  $\text{signs}$ :  $\bigwedge i x. \llbracket i < \text{length } ps - 2; \text{poly } (ps ! (i+1)) x = 0 \rrbracket$   
 $\implies (\text{poly } (ps ! (i+2)) x) * (\text{poly } (ps ! i) x) < 0$

Now we define a Sturm sequence  $p_1, \dots, p_n$  of a polynomial  $p$  in the following way:

- The sequence contains at least two elements.
- $p$  is the first polynomial, i. e.  $p_1 = p$ .
- At any root  $x$  of  $p$ ,  $p_2$  and  $p$  have opposite sign left of  $x$  and the same sign right of  $x$  in some neighbourhood around  $x$ .
- The first two polynomials in the sequence have no common roots.
- If the middle one of three adjacent polynomials has a root at  $x$ , the other two have opposite and nonzero signs at  $x$ .

**locale** *sturm-seq* = *quasi-sturm-seq* +

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

```

assumes hd-ps-p[simp]:  $hd\ ps = p$ 
assumes length-ps-ge-2[simp]:  $length\ ps \geq 2$ 
assumes deriv:  $\bigwedge x_0. poly\ p\ x_0 = 0 \implies$ 
    eventually  $(\lambda x. sgn\ (poly\ (p * ps!1)\ x) =$ 
     $(if\ x > x_0\ then\ 1\ else\ -1))\ (at\ x_0)$ 
assumes p-squarefree:  $\bigwedge x. \neg(poly\ p\ x = 0 \wedge poly\ (ps!1)\ x = 0)$ 
begin

```

Any Sturm sequence is obviously a Quasi-Sturm sequence.

```

lemma quasi-sturm-seq: quasi-sturm-seq ps <proof><proof><proof><proof>end
<proof>

```

Any suffix of a Quasi-Sturm sequence is again a Quasi-Sturm sequence.

```

lemma quasi-sturm-seq-Cons:
  assumes quasi-sturm-seq  $(p\#\!ps)$  and  $ps \neq []$ 
  shows quasi-sturm-seq ps
<proof>

```

## 2.3 Auxiliary lemmas about roots and sign changes

```

lemma sturm-adjacent-root-aux:
  assumes  $i < length\ (ps :: real\ poly\ list) - 1$ 
  assumes  $poly\ (ps\ !\ i)\ x = 0$  and  $poly\ (ps\ !\ (i + 1))\ x = 0$ 
  assumes  $\bigwedge i\ x. \llbracket i < length\ ps - 2; poly\ (ps\ !\ (i+1))\ x = 0 \rrbracket$ 
     $\implies sgn\ (poly\ (ps\ !\ (i+2))\ x) = -\ sgn\ (poly\ (ps\ !\ i)\ x)$ 
  shows  $\forall j \leq i+1. poly\ (ps\ !\ j)\ x = 0$ 
<proof>

```

This function splits the sign list of a Sturm sequence at a position  $x$  that is not a root of  $p$  into a list of sublists such that the number of sign changes within every sublist is constant in the neighbourhood of  $x$ , thus proving that the total number is also constant.

```

fun split-sign-changes where
  split-sign-changes  $[p]$   $(x :: real) = [[p]] \mid$ 
  split-sign-changes  $[p,q]$   $x = [[p,q]] \mid$ 
  split-sign-changes  $(p\#\!q\#\!r\#\!ps)$   $x =$ 
     $(if\ poly\ p\ x \neq 0 \wedge poly\ q\ x = 0\ then$ 
     $[p,q,r] \# split-sign-changes\ (r\#\!ps)\ x$ 
    else
     $[p,q] \# split-sign-changes\ (q\#\!r\#\!ps)\ x)$ 

```

```

lemma (in quasi-sturm-seq) split-sign-changes-subset $[dest]$ :
   $ps' \in set\ (split-sign-changes\ ps\ x) \implies set\ ps' \subseteq set\ ps$ 
<proof>

```

A custom induction rule for *split-sign-changes* that uses the fact that all the intermediate parameters in calls of *split-sign-changes* are quasi-Sturm sequences.

**lemma** (in *quasi-sturm-seq*) *split-sign-changes-induct*:  

$$\llbracket \bigwedge p x. P [p] x; \bigwedge p q x. \text{quasi-sturm-seq} [p, q] \implies P [p, q] x; \bigwedge p q r ps x. \text{quasi-sturm-seq} (p \# q \# r \# ps) \implies \llbracket \text{poly } p x \neq 0 \implies \text{poly } q x = 0 \implies P (r \# ps) x; \text{poly } q x \neq 0 \implies P (q \# r \# ps) x; \text{poly } p x = 0 \implies P (q \# r \# ps) x \rrbracket \implies P (p \# q \# r \# ps) x \rrbracket \implies P ps x$$

*<proof>*

The total number of sign changes in the split list is the same as the number of sign changes in the original list.

**lemma** (in *quasi-sturm-seq*) *split-sign-changes-correct*:  
**assumes**  $\text{poly } (\text{hd } ps) x_0 \neq 0$   
**defines**  $\text{sign-changes}' \equiv \lambda ps x. \sum ps' \leftarrow \text{split-sign-changes } ps x. \text{sign-changes } ps' x$   
**shows**  $\text{sign-changes}' ps x_0 = \text{sign-changes } ps x_0$

*<proof>*

We now prove that if  $p(x) \neq 0$ , the number of sign changes of a Sturm sequence of  $p$  at  $x$  is constant in a neighbourhood of  $x$ .

**lemma** (in *quasi-sturm-seq*) *split-sign-changes-correct-nbh*:  
**assumes**  $\text{poly } (\text{hd } ps) x_0 \neq 0$   
**defines**  $\text{sign-changes}' \equiv \lambda x_0 ps x. \sum ps' \leftarrow \text{split-sign-changes } ps x_0. \text{sign-changes } ps' x$   
**shows** *eventually*  $(\lambda x. \text{sign-changes}' x_0 ps x = \text{sign-changes } ps x)$  (at  $x_0$ )

*<proof>*

**lemma** (in *quasi-sturm-seq*) *hd-nonzero-imp-sign-changes-const-aux*:  
**assumes**  $\text{poly } (\text{hd } ps) x_0 \neq 0$  **and**  $ps' \in \text{set } (\text{split-sign-changes } ps x_0)$   
**shows** *eventually*  $(\lambda x. \text{sign-changes } ps' x = \text{sign-changes } ps' x_0)$  (at  $x_0$ )

*<proof>*

**lemma** (in *quasi-sturm-seq*) *hd-nonzero-imp-sign-changes-const*:  
**assumes**  $\text{poly } (\text{hd } ps) x_0 \neq 0$   
**shows** *eventually*  $(\lambda x. \text{sign-changes } ps x = \text{sign-changes } ps x_0)$  (at  $x_0$ )

*<proof>*

**lemma** (in *sturm-seq*) *p-nonzero-imp-sign-changes-const*:  
 $\text{poly } p x_0 \neq 0 \implies$   
*eventually*  $(\lambda x. \text{sign-changes } ps x = \text{sign-changes } ps x_0)$  (at  $x_0$ )

*<proof>*

If  $x$  is a root of  $p$  and  $p$  is not the zero polynomial, the number of sign changes of a Sturm chain of  $p$  decreases by 1 at  $x$ .

**lemma** (in *sturm-seq*) *p-zero*:  
**assumes**  $\text{poly } p x_0 = 0$   $p \neq 0$

**shows** *eventually* ( $\lambda x. \text{sign-changes } ps \ x =$   
 $\text{sign-changes } ps \ x_0 + (\text{if } x < x_0 \text{ then } 1 \text{ else } 0)) \text{ (at } x_0)$   
 <proof>

With these two results, we can now show that if  $p$  is nonzero, the number of roots in an interval of the form  $(a; b]$  is the difference of the sign changes of a Sturm sequence of  $p$  at  $a$  and  $b$ .

First, however, we prove the following auxiliary lemma that shows that if a function  $f : \mathbb{R} \rightarrow \mathbb{N}$  is locally constant at any  $x \in (a; b]$ , it is constant across the entire interval  $(a; b]$ :

**lemma** *count-roots-between-aux*:

**assumes**  $a \leq b$

**assumes**  $\forall x::\text{real}. a < x \wedge x \leq b \longrightarrow \text{eventually } (\lambda \xi. f \ \xi = (f \ x::\text{nat})) \text{ (at } x)$

**shows**  $\forall x. a < x \wedge x \leq b \longrightarrow f \ x = f \ b$

<proof>

Now we can prove the actual root-counting theorem:

**theorem** (*in sturm-seq*) *count-roots-between*:

**assumes** [*simp*]:  $p \neq 0 \ a \leq b$

**shows**  $\text{sign-changes } ps \ a - \text{sign-changes } ps \ b =$   
 $\text{card } \{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\}$

<proof>

By applying this result to a sufficiently large upper bound, we can effectively count the number of roots “between  $a$  and infinity”, i. e. the roots greater than  $a$ :

**lemma** (*in sturm-seq*) *count-roots-above*:

**assumes**  $p \neq 0$

**shows**  $\text{sign-changes } ps \ a - \text{sign-changes-inf } ps =$   
 $\text{card } \{x. x > a \wedge \text{poly } p \ x = 0\}$

<proof>

The same works analogously for the number of roots below  $a$  and the total number of roots.

**lemma** (*in sturm-seq*) *count-roots-below*:

**assumes**  $p \neq 0$

**shows**  $\text{sign-changes-neg-inf } ps - \text{sign-changes } ps \ a =$   
 $\text{card } \{x. x \leq a \wedge \text{poly } p \ x = 0\}$

<proof>

**lemma** (*in sturm-seq*) *count-roots*:

**assumes**  $p \neq 0$

**shows**  $\text{sign-changes-neg-inf } ps - \text{sign-changes-inf } ps =$   
 $\text{card } \{x. \text{poly } p \ x = 0\}$

<proof>

## 2.4 Constructing Sturm sequences

## 2.5 The canonical Sturm sequence

In this subsection, we will present the canonical Sturm sequence construction for a polynomial  $p$  without multiple roots that is very similar to the Euclidean algorithm:

$$p_i = \begin{cases} p & \text{for } i = 1 \\ p' & \text{for } i = 2 \\ -p_{i-2} \bmod p_{i-1} & \text{otherwise} \end{cases}$$

We break off the sequence at the first constant polynomial.

*<proof>*

**function** *sturm-aux* **where**

*sturm-aux* ( $p :: \text{real poly}$ )  $q =$

(if degree  $q = 0$  then  $[p, q]$  else  $p \# \text{sturm-aux } q \ (-(p \bmod q))$ )

*<proof>*

**termination** *<proof>*

**definition** *sturm* **where**  $\text{sturm } p = \text{sturm-aux } p \ (\text{pderiv } p)$

Next, we show some simple facts about this construction:

**lemma** *sturm-0[simp]*:  $\text{sturm } 0 = [0, 0]$

*<proof>*

**lemma** *[simp]*:  $\text{sturm-aux } p \ q = [] \longleftrightarrow \text{False}$

*<proof>*

**lemma** *sturm-neq-Nil[simp]*:  $\text{sturm } p \neq []$  *<proof>*

**lemma** *[simp]*:  $\text{hd } (\text{sturm } p) = p$

*<proof>*

**lemma** *[simp]*:  $p \in \text{set } (\text{sturm } p)$

*<proof>*

**lemma** *[simp]*:  $\text{length } (\text{sturm } p) \geq 2$

*<proof>*

**lemma** *[simp]*:  $\text{degree } (\text{last } (\text{sturm } p)) = 0$

*<proof>*

**lemma** *[simp]*:  $\text{sturm-aux } p \ q \ ! \ 0 = p$

*<proof>*

**lemma** *[simp]*:  $\text{sturm-aux } p \ q \ ! \ \text{Suc } 0 = q$

*<proof>*

**lemma** *[simp]*:  $\text{sturm } p \ ! \ 0 = p$

$\langle proof \rangle$   
**lemma** [simp]:  $sturm\ p\ !\ Suc\ 0 = pderiv\ p$   
 $\langle proof \rangle$

**lemma** *sturm-indices*:  
**assumes**  $i < length\ (sturm\ p) - 2$   
**shows**  $sturm\ p!(i+2) = -(sturm\ p!i\ mod\ sturm\ p!(i+1))$   
 $\langle proof \rangle$

If the Sturm sequence construction is applied to polynomials  $p$  and  $q$ , the greatest common divisor of  $p$  and  $q$  a divisor of every element in the sequence. This is obvious from the similarity to Euclid's algorithm for computing the GCD.

**lemma** *sturm-aux-gcd*:  $r \in set\ (sturm\ aux\ p\ q) \implies gcd\ p\ q\ dvd\ r$   
 $\langle proof \rangle$

**lemma** *sturm-gcd*:  $r \in set\ (sturm\ p) \implies gcd\ p\ (pderiv\ p)\ dvd\ r$   
 $\langle proof \rangle$

If two adjacent polynomials in the result of the canonical Sturm chain construction both have a root at some  $x$ , this  $x$  is a root of all polynomials in the sequence.

**lemma** *sturm-adjacent-root-propagate-left*:  
**assumes**  $i < length\ (sturm\ (p :: real\ poly)) - 1$   
**assumes**  $poly\ (sturm\ p\ !\ i)\ x = 0$   
**and**  $poly\ (sturm\ p\ !\ (i + 1))\ x = 0$   
**shows**  $\forall j \leq i+1. poly\ (sturm\ p\ !\ j)\ x = 0$   
 $\langle proof \rangle$

Consequently, if this is the case in the canonical Sturm chain of  $p$ ,  $p$  must have multiple roots.

**lemma** *sturm-adjacent-root-not-squarefree*:  
**assumes**  $i < length\ (sturm\ (p :: real\ poly)) - 1$   
 $poly\ (sturm\ p\ !\ i)\ x = 0\ poly\ (sturm\ p\ !\ (i + 1))\ x = 0$   
**shows**  $\neg rsquarefree\ p$   
 $\langle proof \rangle$

Since the second element of the sequence is chosen to be the derivative of  $p$ ,  $p_1$  and  $p_2$  fulfil the property demanded by the definition of a Sturm sequence that they locally have opposite sign left of a root  $x$  of  $p$  and the same sign to the right of  $x$ .

**lemma** *sturm-firsttwo-signs-aux*:  
**assumes**  $(p :: real\ poly) \neq 0\ q \neq 0$   
**assumes**  $q\ pderiv$ :  
 $eventually\ (\lambda x. sgn\ (poly\ q\ x) = sgn\ (poly\ (pderiv\ p)\ x))\ (at\ x_0)$   
**assumes**  $p-0$ :  $poly\ p\ (x_0 :: real) = 0$

**shows** eventually  $(\lambda x. \text{sgn} (\text{poly} (p * q) x) = (\text{if } x > x_0 \text{ then } 1 \text{ else } -1))$  (at  $x_0$ )  
 <proof>

**lemma** *sturm-firsttwo-signs*:

**fixes**  $ps :: \text{real poly list}$

**assumes** *squarefree*:  $rsquarefree\ p$

**assumes** *p-0*:  $\text{poly } p (x_0 :: \text{real}) = 0$

**shows** eventually  $(\lambda x. \text{sgn} (\text{poly} (p * \text{sturm } p ! 1) x) =$   
 $(\text{if } x > x_0 \text{ then } 1 \text{ else } -1))$  (at  $x_0$ )

<proof>

The construction also obviously fulfils the property about three adjacent polynomials in the sequence.

**lemma** *sturm-signs*:

**assumes** *squarefree*:  $rsquarefree\ p$

**assumes** *i-in-range*:  $i < \text{length} (\text{sturm } (p :: \text{real poly})) - 2$

**assumes** *q-0*:  $\text{poly} (\text{sturm } p ! (i+1)) x = 0$  (**is**  $\text{poly } ?q x = 0$ )

**shows**  $\text{poly} (\text{sturm } p ! (i+2)) x * \text{poly} (\text{sturm } p ! i) x < 0$   
 (**is**  $\text{poly } ?p x * \text{poly } ?r x < 0$ )

<proof>

Finally, if  $p$  contains no multiple roots, *sturm*  $p$ , i.e. the canonical Sturm sequence for  $p$ , is a Sturm sequence and can be used to determine the number of roots of  $p$ .

**lemma** *sturm-seq-sturm[simp]*:

**assumes** *rsquarefree*  $p$

**shows** *sturm-seq*  $(\text{sturm } p) p$

<proof>

### 2.5.1 Canonical squarefree Sturm sequence

The previous construction does not work for polynomials with multiple roots, but we can simply “divide away” multiple roots by dividing  $p$  by the GCD of  $p$  and  $p'$ . The resulting polynomial has the same roots as  $p$ , but with multiplicity 1, allowing us to again use the canonical construction.

**definition** *sturm-squarefree* **where**

$\text{sturm-squarefree } p = \text{sturm } (p \text{ div } (\text{gcd } p (pderiv\ p)))$

**lemma** *sturm-squarefree-not-Nil[simp]*:  $\text{sturm-squarefree } p \neq []$

<proof>

**lemma** *sturm-seq-sturm-squarefree*:

**assumes** [*simp*]:  $p \neq 0$

**defines** [*simp*]:  $p' \equiv p \text{ div } \text{gcd } p (pderiv\ p)$

**shows** *sturm-seq*  $(\text{sturm-squarefree } p) p'$

<proof>

## 2.5.2 Optimisation for multiple roots

We can also define the following non-canonical Sturm sequence that is obtained by taking the canonical Sturm sequence of  $p$  (possibly with multiple roots) and then dividing the entire sequence by the GCD of  $p$  and its derivative.

**definition** *sturm-squarefree'* **where**

*sturm-squarefree'*  $p = (\text{let } d = \text{gcd } p \text{ (pderiv } p)$   
                   *in map*  $(\lambda p'. p' \text{ div } d) (\text{sturm } p)$ )

This construction also has all the desired properties:

**lemma** *sturm-squarefree'-adjacent-root-propagate-left*:

**assumes**  $p \neq 0$   
**assumes**  $i < \text{length } (\text{sturm-squarefree}' (p :: \text{real poly})) - 1$   
**assumes**  $\text{poly } (\text{sturm-squarefree}' p ! i) x = 0$   
**and**  $\text{poly } (\text{sturm-squarefree}' p ! (i + 1)) x = 0$   
**shows**  $\forall j \leq i+1. \text{poly } (\text{sturm-squarefree}' p ! j) x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *sturm-squarefree'-adjacent-roots*:

**assumes**  $p \neq 0$   
 $i < \text{length } (\text{sturm-squarefree}' (p :: \text{real poly})) - 1$   
 $\text{poly } (\text{sturm-squarefree}' p ! i) x = 0$   
 $\text{poly } (\text{sturm-squarefree}' p ! (i + 1)) x = 0$   
**shows** *False*  
 $\langle \text{proof} \rangle$

**lemma** *sturm-squarefree'-signs*:

**assumes**  $p \neq 0$   
**assumes** *i-in-range*:  $i < \text{length } (\text{sturm-squarefree}' (p :: \text{real poly})) - 2$   
**assumes** *q-0*:  $\text{poly } (\text{sturm-squarefree}' p ! (i+1)) x = 0$  (**is**  $\text{poly } ?q x = 0$ )  
**shows**  $\text{poly } (\text{sturm-squarefree}' p ! (i+2)) x *$   
 $\text{poly } (\text{sturm-squarefree}' p ! i) x < 0$   
**(is**  $\text{poly } ?r x * \text{poly } ?p x < 0$ )  
 $\langle \text{proof} \rangle$

This approach indeed also yields a valid squarefree Sturm sequence for the polynomial  $p/\text{gcd}(p, p')$ .

**lemma** *sturm-seq-sturm-squarefree'*:

**assumes**  $(p :: \text{real poly}) \neq 0$   
**defines**  $d \equiv \text{gcd } p \text{ (pderiv } p)$   
**shows**  $\text{sturm-seq } (\text{sturm-squarefree}' p) (p \text{ div } d)$   
**(is**  $\text{sturm-seq } ?ps' ?p'$ )  
 $\langle \text{proof} \rangle$

This construction is obviously more expensive to compute than the one that *first* divides  $p$  by  $\text{gcd}(p, p')$  and *then* applies the canonical construction. In this construction, we *first* compute the canonical Sturm sequence of  $p$  as

if it had no multiple roots and *then* divide by the GCD. However, it can be seen quite easily that unless  $x$  is a multiple root of  $p$ , i.e. as long as  $\gcd(P, P') \neq 0$ , the number of sign changes in a sequence of polynomials does not actually change when we divide the polynomials by  $\gcd(p, p')$ . Therefore we can use the canonical Sturm sequence even in the non-square-free case as long as the borders of the interval we are interested in are not multiple roots of the polynomial.

**lemma** *sign-changes-mult-aux*:

**assumes**  $d \neq (0::real)$

**shows**  $length\ (remdups\ adj\ (filter\ (\lambda x. x \neq 0)\ (map\ ((*)\ d \circ f)\ xs))) =$   
 $length\ (remdups\ adj\ (filter\ (\lambda x. x \neq 0)\ (map\ f\ xs)))$

*<proof>*

**lemma** *sturm-sturm-squarefree'-same-sign-changes*:

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

**defines**  $ps \equiv sturm\ p$  **and**  $ps' \equiv sturm\ squarefree'\ p$

**shows**  $poly\ p\ x \neq 0 \vee poly\ (pderiv\ p)\ x \neq 0 \implies$

$sign\ changes\ ps'\ x = sign\ changes\ ps\ x$

$p \neq 0 \implies sign\ changes\ inf\ ps' = sign\ changes\ inf\ ps$

$p \neq 0 \implies sign\ changes\ neg\ inf\ ps' = sign\ changes\ neg\ inf\ ps$

*<proof>*

## 2.6 Root-counting functions

With all these results, we can now define functions that count roots in bounded and unbounded intervals:

**definition** *count-roots-between* **where**

*count-roots-between*  $p\ a\ b = (if\ a \leq b \wedge p \neq 0\ then$

$(let\ ps = sturm\ squarefree\ p$

$in\ sign\ changes\ ps\ a - sign\ changes\ ps\ b)$  *else* 0)

**definition** *count-roots* **where**

*count-roots*  $p = (if\ (p::real\ poly) = 0\ then\ 0\ else$

$(let\ ps = sturm\ squarefree\ p$

$in\ sign\ changes\ neg\ inf\ ps - sign\ changes\ inf\ ps))$

**definition** *count-roots-above* **where**

*count-roots-above*  $p\ a = (if\ (p::real\ poly) = 0\ then\ 0\ else$

$(let\ ps = sturm\ squarefree\ p$

$in\ sign\ changes\ ps\ a - sign\ changes\ inf\ ps))$

**definition** *count-roots-below* **where**

*count-roots-below*  $p\ a = (if\ (p::real\ poly) = 0\ then\ 0\ else$

$(let\ ps = sturm\ squarefree\ p$

$in\ sign\ changes\ neg\ inf\ ps - sign\ changes\ ps\ a))$

**lemma** *count-roots-between-correct*:

*count-roots-between*  $p$   $a$   $b = \text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p x = 0\}$   
*<proof>*

**lemma** *count-roots-correct*:

**fixes**  $p :: \text{real poly}$   
**shows** *count-roots*  $p = \text{card } \{x. \text{poly } p x = 0\}$  (**is**  $- = \text{card } ?S$ )  
*<proof>*

**lemma** *count-roots-above-correct*:

**fixes**  $p :: \text{real poly}$   
**shows** *count-roots-above*  $p$   $a = \text{card } \{x. x > a \wedge \text{poly } p x = 0\}$   
(**is**  $- = \text{card } ?S$ )  
*<proof>*

**lemma** *count-roots-below-correct*:

**fixes**  $p :: \text{real poly}$   
**shows** *count-roots-below*  $p$   $a = \text{card } \{x. x \leq a \wedge \text{poly } p x = 0\}$   
(**is**  $- = \text{card } ?S$ )  
*<proof>*

The optimisation explained above can be used to prove more efficient code equations that use the more efficient construction in the case that the interval borders are not multiple roots:

**lemma** *count-roots-between*[code]:

*count-roots-between*  $p$   $a$   $b =$   
  (*let*  $q = \text{pderiv } p$   
  *in if*  $a > b \vee p = 0$  *then*  $0$   
  *else if*  $(\text{poly } p a \neq 0 \vee \text{poly } q a \neq 0) \wedge (\text{poly } p b \neq 0 \vee \text{poly } q b \neq 0)$   
  *then* (*let*  $ps = \text{sturm } p$   
  *in*  $\text{sign-changes } ps a - \text{sign-changes } ps b$ )  
  *else* (*let*  $ps = \text{sturm-squarefree } p$   
  *in*  $\text{sign-changes } ps a - \text{sign-changes } ps b$ )  
*<proof>*

**lemma** *count-roots-code*[code]:

*count-roots*  $(p::\text{real poly}) =$   
  (*if*  $p = 0$  *then*  $0$   
  *else let*  $ps = \text{sturm } p$   
  *in*  $\text{sign-changes-neg-inf } ps - \text{sign-changes-inf } ps$ )  
*<proof>*

**lemma** *count-roots-above-code*[code]:

*count-roots-above*  $p$   $a =$   
  (*let*  $q = \text{pderiv } p$   
  *in if*  $p = 0$  *then*  $0$   
  *else if*  $\text{poly } p a \neq 0 \vee \text{poly } q a \neq 0$

```

    then (let ps = sturm p
          in sign-changes ps a - sign-changes-inf ps)
    else (let ps = sturm-squarefree p
          in sign-changes ps a - sign-changes-inf ps))
⟨proof⟩

```

**lemma** *count-roots-below-code*[code]:

```

count-roots-below p a =
  (let q = pderiv p
   in if p = 0 then 0
   else if poly p a ≠ 0 ∨ poly q a ≠ 0
        then (let ps = sturm p
              in sign-changes-neg-inf ps - sign-changes ps a)
        else (let ps = sturm-squarefree p
              in sign-changes-neg-inf ps - sign-changes ps a))
⟨proof⟩

```

**end**

### 3 The “sturm” proof method

```

theory Sturm-Method
imports Sturm-Theorem
begin

```

#### 3.1 Preliminary lemmas

In this subsection, we prove lemmas that reduce root counting and related statements to simple, computable expressions using the *count-roots* function family.

**lemma** *poly-card-roots-less-leq*:

```

card {x. a < x ∧ x ≤ b ∧ poly p x = 0} = count-roots-between p a b
⟨proof⟩

```

**lemma** *poly-card-roots-leq-leq*:

```

card {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0} =
  ( count-roots-between p a b +
    (if (a ≤ b ∧ poly p a = 0 ∧ p ≠ 0) ∨ (a = b ∧ p = 0) then 1 else 0))
⟨proof⟩

```

**lemma** *poly-card-roots-less-less*:

```

card {x. a < x ∧ x < b ∧ poly p x = 0} =
  ( count-roots-between p a b -
    (if poly p b = 0 ∧ a < b ∧ p ≠ 0 then 1 else 0))
⟨proof⟩

```

**lemma** *poly-card-roots-leq-less*:

```

card {x::real. a ≤ x ∧ x < b ∧ poly p x = 0} =

```

$(\text{count-roots-between } p \ a \ b +$   
 $(\text{if } p \neq 0 \wedge a < b \wedge \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0) -$   
 $(\text{if } p \neq 0 \wedge a < b \wedge \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0))$   
 <proof>

**lemma** *poly-card-roots*:

$\text{card } \{x::\text{real}. \text{poly } p \ x = 0\} = \text{count-roots } p$   
 <proof>

**lemma** *poly-no-roots*:

$(\forall x. \text{poly } p \ x \neq 0) \iff (p \neq 0 \wedge \text{count-roots } p = 0)$   
 <proof>

**lemma** *poly-pos*:

$(\forall x. \text{poly } p \ x > 0) \iff ($   
 $p \neq 0 \wedge \text{poly-inf } p = 1 \wedge \text{count-roots } p = 0)$   
 <proof>

**lemma** *poly-card-roots-greater*:

$\text{card } \{x::\text{real}. x > a \wedge \text{poly } p \ x = 0\} = \text{count-roots-above } p \ a$   
 <proof>

**lemma** *poly-card-roots-leq*:

$\text{card } \{x::\text{real}. x \leq a \wedge \text{poly } p \ x = 0\} = \text{count-roots-below } p \ a$   
 <proof>

**lemma** *poly-card-roots-geq*:

$\text{card } \{x::\text{real}. x \geq a \wedge \text{poly } p \ x = 0\} = ($   
 $\text{count-roots-above } p \ a + (\text{if } \text{poly } p \ a = 0 \wedge p \neq 0 \text{ then } 1 \text{ else } 0))$   
 <proof>

**lemma** *poly-card-roots-less*:

$\text{card } \{x::\text{real}. x < a \wedge \text{poly } p \ x = 0\} =$   
 $(\text{count-roots-below } p \ a - (\text{if } \text{poly } p \ a = 0 \wedge p \neq 0 \text{ then } 1 \text{ else } 0))$   
 <proof>

**lemma** *poly-no-roots-less-leq*:

$(\forall x. a < x \wedge x \leq b \implies \text{poly } p \ x \neq 0) \iff$   
 $((a \geq b \vee (p \neq 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$   
 <proof>

**lemma** *poly-pos-between-less-leq*:

$(\forall x. a < x \wedge x \leq b \implies \text{poly } p \ x > 0) \iff$   
 $((a \geq b \vee (p \neq 0 \wedge \text{poly } p \ b > 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$   
 <proof>

**lemma** *poly-no-roots-leq-leq*:

$$(\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$$

$$((a > b \vee (p \neq 0 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$$

*<proof>*

**lemma** *poly-pos-between-leq-leq*:

$$(\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$

$$((a > b \vee (p \neq 0 \wedge \text{poly } p \ a > 0 \wedge$$

$$\text{count-roots-between } p \ a \ b = 0)))$$

*<proof>*

**lemma** *poly-no-roots-less-less*:

$$(\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$$

$$((a \geq b \vee p \neq 0 \wedge \text{count-roots-between } p \ a \ b =$$

$$(\text{if } \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0)))$$

*<proof>*

**lemma** *poly-pos-between-less-less*:

$$(\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$

$$((a \geq b \vee (p \neq 0 \wedge \text{poly } p \ ((a+b)/2) > 0 \wedge$$

$$\text{count-roots-between } p \ a \ b = (\text{if } \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0))))$$

*<proof>*

**lemma** *poly-no-roots-leq-less*:

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

$$((a \geq b \vee p \neq 0 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-between } p \ a \ b =$$

$$(\text{if } a < b \wedge \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0)))$$

*<proof>*

**lemma** *poly-pos-between-leq-less*:

$$(\forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$$

$$((a \geq b \vee (p \neq 0 \wedge \text{poly } p \ a > 0 \wedge \text{count-roots-between } p \ a \ b =$$

$$(\text{if } a < b \wedge \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0))))$$

*<proof>*

**lemma** *poly-no-roots-greater*:

$$(\forall x. x > a \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$$

$$((p \neq 0 \wedge \text{count-roots-above } p \ a = 0))$$

*<proof>*

**lemma** *poly-pos-greater*:

$$(\forall x. x > a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow ($$

$$p \neq 0 \wedge \text{poly-inf } p = 1 \wedge \text{count-roots-above } p \ a = 0)$$

*<proof>*

**lemma** *poly-no-roots-leq*:

$(\forall x. x \leq a \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$   
 $(p \neq 0 \wedge \text{count-roots-below } p \ a = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-pos-leq*:

$(\forall x. x \leq a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$   
 $(p \neq 0 \wedge \text{poly-neg-inf } p = 1 \wedge \text{count-roots-below } p \ a = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-no-roots-geq*:

$(\forall x. x \geq a \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$   
 $(p \neq 0 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-above } p \ a = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-pos-geq*:

$(\forall x. x \geq a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$   
 $(p \neq 0 \wedge \text{poly-inf } p = 1 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-above } p \ a = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-no-roots-less*:

$(\forall x. x < a \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$   
 $((p \neq 0 \wedge \text{count-roots-below } p \ a = (\text{if } \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0)))$   
 $\langle \text{proof} \rangle$

**lemma** *poly-pos-less*:

$(\forall x. x < a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$   
 $(p \neq 0 \wedge \text{poly-neg-inf } p = 1 \wedge \text{count-roots-below } p \ a =$   
 $(\text{if } \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0))$   
 $\langle \text{proof} \rangle$

**lemmas** *sturm-card-substs = poly-card-roots poly-card-roots-less-leq*  
*poly-card-roots-leq-less poly-card-roots-less-less poly-card-roots-leq-leq*  
*poly-card-roots-less poly-card-roots-leq poly-card-roots-greater*  
*poly-card-roots-geq*

**lemmas** *sturm-prop-substs = poly-no-roots poly-no-roots-less-leq*  
*poly-no-roots-leq-leq poly-no-roots-less-less poly-no-roots-leq-less*  
*poly-no-roots-leq poly-no-roots-less poly-no-roots-geq*  
*poly-no-roots-greater*  
*poly-pos poly-pos-greater poly-pos-geq poly-pos-less poly-pos-leq*  
*poly-pos-between-leq-less poly-pos-between-less-leq*  
*poly-pos-between-leq-leq poly-pos-between-less-less*

### 3.2 Reification

This subsection defines a number of equations to automatically convert statements about roots of polynomials into a canonical form so that they can be proven using the above substitutions.

**definition**  $PR\text{-}TAG\ x \equiv x$

**lemma**  $sturm\text{-}id\text{-}PR\text{-}prio0$ :

$$\begin{aligned} \{x::real. P\ x\} &= \{x::real. (PR\text{-}TAG\ P)\ x\} \\ (\forall x::real. f\ x < g\ x) &= (\forall x::real. PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real. P\ x) &= (\forall x::real. \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ &\langle proof \rangle \end{aligned}$$

**lemma**  $sturm\text{-}id\text{-}PR\text{-}prio1$ :

$$\begin{aligned} \{x::real. x < a \wedge P\ x\} &= \{x::real. x < a \wedge (PR\text{-}TAG\ P)\ x\} \\ \{x::real. x \leq a \wedge P\ x\} &= \{x::real. x \leq a \wedge (PR\text{-}TAG\ P)\ x\} \\ \{x::real. x \geq b \wedge P\ x\} &= \{x::real. x \geq b \wedge (PR\text{-}TAG\ P)\ x\} \\ \{x::real. x > b \wedge P\ x\} &= \{x::real. x > b \wedge (PR\text{-}TAG\ P)\ x\} \\ (\forall x::real < a. f\ x < g\ x) &= (\forall x::real < a. PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real \leq a. f\ x < g\ x) &= (\forall x::real \leq a. PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real > a. f\ x < g\ x) &= (\forall x::real > a. PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real \geq a. f\ x < g\ x) &= (\forall x::real \geq a. PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real < a. P\ x) &= (\forall x::real < a. \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ (\forall x::real > a. P\ x) &= (\forall x::real > a. \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ (\forall x::real \leq a. P\ x) &= (\forall x::real \leq a. \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ (\forall x::real \geq a. P\ x) &= (\forall x::real \geq a. \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ &\langle proof \rangle \end{aligned}$$

**lemma**  $sturm\text{-}id\text{-}PR\text{-}prio2$ :

$$\begin{aligned} \{x::real. x > a \wedge x \leq b \wedge P\ x\} &= \\ &\{x::real. x > a \wedge x \leq b \wedge PR\text{-}TAG\ P\ x\} \\ \{x::real. x \geq a \wedge x \leq b \wedge P\ x\} &= \\ &\{x::real. x \geq a \wedge x \leq b \wedge PR\text{-}TAG\ P\ x\} \\ \{x::real. x \geq a \wedge x < b \wedge P\ x\} &= \\ &\{x::real. x \geq a \wedge x < b \wedge PR\text{-}TAG\ P\ x\} \\ \{x::real. x > a \wedge x < b \wedge P\ x\} &= \\ &\{x::real. x > a \wedge x < b \wedge PR\text{-}TAG\ P\ x\} \\ (\forall x::real. a < x \wedge x \leq b \longrightarrow f\ x < g\ x) &= \\ &(\forall x::real. a < x \wedge x \leq b \longrightarrow PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real. a \leq x \wedge x \leq b \longrightarrow f\ x < g\ x) &= \\ &(\forall x::real. a \leq x \wedge x \leq b \longrightarrow PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real. a < x \wedge x < b \longrightarrow f\ x < g\ x) &= \\ &(\forall x::real. a < x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real. a \leq x \wedge x < b \longrightarrow f\ x < g\ x) &= \\ &(\forall x::real. a \leq x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x. f\ x < g\ x)\ x) \\ (\forall x::real. a < x \wedge x \leq b \longrightarrow P\ x) &= \\ &(\forall x::real. a < x \wedge x \leq b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \\ (\forall x::real. a \leq x \wedge x \leq b \longrightarrow P\ x) &= \\ &(\forall x::real. a \leq x \wedge x \leq b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x. \neg P\ x))\ x) \end{aligned}$$

$(\forall x::real. a \leq x \wedge x < b \longrightarrow P x) =$   
 $(\forall x::real. a \leq x \wedge x < b \longrightarrow \neg(PR-TAG (\lambda x. \neg P x)) x)$   
 $(\forall x::real. a < x \wedge x < b \longrightarrow P x) =$   
 $(\forall x::real. a < x \wedge x < b \longrightarrow \neg(PR-TAG (\lambda x. \neg P x)) x)$   
 <proof>

**lemma** *PR-TAG-intro-prio0*:

**fixes**  $P :: real \Rightarrow bool$  **and**  $f :: real \Rightarrow real$

**shows**

$PR-TAG P = P' \Longrightarrow PR-TAG (\lambda x. \neg(\neg P x)) = P'$   
 $\llbracket PR-TAG P = (\lambda x. poly p x = 0); PR-TAG Q = (\lambda x. poly q x = 0) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. P x \wedge Q x) = (\lambda x. poly (gcd p q) x = 0)$  **and**  
 $\llbracket PR-TAG P = (\lambda x. poly p x = 0); PR-TAG Q = (\lambda x. poly q x = 0) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. P x \vee Q x) = (\lambda x. poly (p*q) x = 0)$  **and**

$\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x = g x) = (\lambda x. poly (p-q) x = 0)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x \neq g x) = (\lambda x. poly (p-q) x \neq 0)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x < g x) = (\lambda x. poly (q-p) x > 0)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x \leq g x) = (\lambda x. poly (q-p) x \geq 0)$

$PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. -f x) = (\lambda x. poly (-p) x)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x + g x) = (\lambda x. poly (p+q) x)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x - g x) = (\lambda x. poly (p-q) x)$   
 $\llbracket PR-TAG f = (\lambda x. poly p x); PR-TAG g = (\lambda x. poly q x) \rrbracket$   
 $\Longrightarrow PR-TAG (\lambda x. f x * g x) = (\lambda x. poly (p*q) x)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. (f x) \widehat{n}) = (\lambda x. poly (p \widehat{n}) x)$   
 $PR-TAG (\lambda x. poly p x :: real) = (\lambda x. poly p x)$   
 $PR-TAG (\lambda x. x::real) = (\lambda x. poly [0,1:] x)$   
 $PR-TAG (\lambda x. a::real) = (\lambda x. poly [:a:] x)$   
 <proof>

**lemma** *PR-TAG-intro-prio1*:

**fixes**  $f :: real \Rightarrow real$

**shows**

$PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. f x = 0) = (\lambda x. poly p x = 0)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. f x \neq 0) = (\lambda x. poly p x \neq 0)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. 0 = f x) = (\lambda x. poly p x = 0)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. 0 \neq f x) = (\lambda x. poly p x \neq 0)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. f x \geq 0) = (\lambda x. poly p x \geq 0)$   
 $PR-TAG f = (\lambda x. poly p x) \Longrightarrow PR-TAG (\lambda x. f x > 0) = (\lambda x. poly p x > 0)$

$PR\text{-TAG } f = (\lambda x. \text{poly } p \ x) \implies PR\text{-TAG } (\lambda x. f \ x \leq 0) = (\lambda x. \text{poly } (-p) \ x \geq 0)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x) \implies PR\text{-TAG } (\lambda x. f \ x < 0) = (\lambda x. \text{poly } (-p) \ x > 0)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x) \implies$   
 $\quad PR\text{-TAG } (\lambda x. 0 \leq f \ x) = (\lambda x. \text{poly } (-p) \ x \leq 0)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x) \implies$   
 $\quad PR\text{-TAG } (\lambda x. 0 < f \ x) = (\lambda x. \text{poly } (-p) \ x < 0)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. a * f \ x) = (\lambda x. \text{poly } (\text{smult } a \ p) \ x)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. f \ x * a) = (\lambda x. \text{poly } (\text{smult } a \ p) \ x)$   
 $PR\text{-TAG } f = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. f \ x / a) = (\lambda x. \text{poly } (\text{smult } (\text{inverse } a) \ p) \ x)$   
 $PR\text{-TAG } (\lambda x. x \hat{\ } n :: \text{real}) = (\lambda x. \text{poly } (\text{monom } 1 \ n) \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *PR-TAG-intro-prio2*:

$PR\text{-TAG } (\lambda x. 1 / b) = (\lambda x. \text{inverse } b)$   
 $PR\text{-TAG } (\lambda x. a / b) = (\lambda x. a / b)$   
 $PR\text{-TAG } (\lambda x. a / b * x \hat{\ } n :: \text{real}) = (\lambda x. \text{poly } (\text{monom } (a/b) \ n) \ x)$   
 $PR\text{-TAG } (\lambda x. x \hat{\ } n * a / b :: \text{real}) = (\lambda x. \text{poly } (\text{monom } (a/b) \ n) \ x)$   
 $PR\text{-TAG } (\lambda x. a * x \hat{\ } n :: \text{real}) = (\lambda x. \text{poly } (\text{monom } a \ n) \ x)$   
 $PR\text{-TAG } (\lambda x. x \hat{\ } n * a :: \text{real}) = (\lambda x. \text{poly } (\text{monom } a \ n) \ x)$   
 $PR\text{-TAG } (\lambda x. x \hat{\ } n / a :: \text{real}) = (\lambda x. \text{poly } (\text{monom } (\text{inverse } a) \ n) \ x)$

$PR\text{-TAG } (\lambda x. f \ x \hat{\ } (\text{Suc } (\text{Suc } 0)) :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. f \ x * f \ x :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $PR\text{-TAG } (\lambda x. (f \ x) \hat{\ } \text{Suc } n :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. (f \ x) \hat{\ } n * f \ x :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $PR\text{-TAG } (\lambda x. (f \ x) \hat{\ } \text{Suc } n :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. f \ x * (f \ x) \hat{\ } n :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $PR\text{-TAG } (\lambda x. (f \ x) \hat{\ } (m+n) :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $\implies PR\text{-TAG } (\lambda x. (f \ x) \hat{\ } m * (f \ x) \hat{\ } n :: \text{real}) = (\lambda x. \text{poly } p \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *sturm-meta-spec*:  $(\bigwedge x :: \text{real}. P \ x) \implies P \ x \langle \text{proof} \rangle$

**lemma** *sturm-imp-conv*:

$(a < x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$   
 $(a \leq x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a \leq x \wedge x < b \longrightarrow c)$   
 $(a < x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow (a < x \wedge x \leq b \longrightarrow c)$   
 $(a \leq x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow (a \leq x \wedge x \leq b \longrightarrow c)$   
 $(x < b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$   
 $(x < b \longrightarrow a \leq x \longrightarrow c) \longleftrightarrow (a \leq x \wedge x < b \longrightarrow c)$   
 $(x \leq b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x \leq b \longrightarrow c)$   
 $(x \leq b \longrightarrow a \leq x \longrightarrow c) \longleftrightarrow (a \leq x \wedge x \leq b \longrightarrow c)$   
 $\langle \text{proof} \rangle$

### 3.3 Setup for the “sturm” method

$\langle ML \rangle$

end

```
theory Sturm
imports Sturm-Method
begin
```

end

## 4 Example usage of the “sturm” method

```
theory Sturm-Ex
imports ../Sturm
begin
```

In this section, we give a variety of statements about real polynomials that can be proven by the *sturm* method.

```
lemma
   $\forall x::real. x^2 + 1 \neq 0$ 
  <proof>
```

```
lemma
  fixes x :: real
  shows  $x^2 + 1 \neq 0$  <proof>
```

```
lemma (x::real) > 1  $\implies x^3 > 1$  <proof>
```

```
lemma  $\forall x::real. x*x \neq -1$  <proof>
```

**schematic-goal A:**

```
card {x::real.  $-0.010831 < x \wedge x < 0.010831 \wedge$ 
   $1/120*x^5 + 1/24*x^4 + 1/6*x^3 - 49/16777216*x^2 - 17/2097152*x =$ 
  0}
  = ?n
  <proof>
```

```
lemma card {x::real.  $x^3 + x = 2*x^2 \wedge x^3 - 6*x^2 + 11*x = 6$ } = 1
  <proof>
```

```
schematic-goal card {x::real.  $x^3 + x = 2*x^2 \vee x^3 - 6*x^2 + 11*x = 6$ }
  = ?n <proof>
```

```
lemma
  card {x::real.  $-0.010831 < x \wedge x < 0.010831 \wedge$ 
  poly [0, -17/2097152, -49/16777216, 1/6, 1/24, 1/120:] x = 0} = 3
  <proof>
```

```
lemma  $\forall x::real. x*x \neq 0 \vee x*x - 1 \neq 2*x$  <proof>
```

**lemma**  $(x::real)*x+1 \neq 0 \wedge (x^2+1)*(x^2+2) \neq 0$  *(proof)*

3 examples related to continued fraction approximants to exp: LCP

**lemma fixes**  $x::real$

**shows**  $-7.29347719 \leq x \implies 0 < x^5 + 30*x^4 + 420*x^3 + 3360*x^2 + 15120*x + 30240$

*(proof)*

**lemma fixes**  $x::real$

**shows**  $0 < x^6 + 42*x^5 + 840*x^4 + 10080*x^3 + 75600*x^2 + 332640*x + 665280$

*(proof)*

**schematic-goal card**  $\{x::real. x^7 + 56*x^6 + 1512*x^5 + 25200*x^4 + 277200*x^3 + 1995840*x^2 + 8648640*x = -17297280\} = ?n$

*(proof)*

**end**