A Formalisation of Sturm's Theorem

Manuel Eberl

March 19, 2025

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

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 $a \leq b$ $\forall x::real.\ a \leq x \wedge x \leq b \longrightarrow eventually\ (\lambda\xi.\ f\ \xi = f\ x)\ (at\ x)$
  shows $f\ a = f\ b$
⟨*proof*⟩

1.2 Polynomials

1.2.1 General simplification lemmas

lemma *pderiv-div*:
  assumes [*simp*]: $q\ dvd\ p$ $q \neq 0$
  shows $pderiv\ (p\ div\ q) = (q * pderiv\ p - p * pderiv\ q)\ div\ (q * q)$
    $q*q\ dvd\ (q * pderiv\ p - p * pderiv\ q)$
⟨*proof*⟩

1.2.2 Divisibility of polynomials

Two polynomials that are coprime have no common roots.

lemma *coprime-imp-no-common-roots*:
  $\neg\ (poly\ p\ x = 0 \wedge poly\ q\ x = 0)$ **if** *coprime p q*
    for $x :: 'a :: field$
⟨*proof*⟩

lemma *poly-div*:
  assumes $poly\ q\ x \neq 0$ **and** $(q::'a :: field\ poly)\ dvd\ p$
  shows $poly\ (p\ div\ q)\ x = poly\ p\ x\ /\ poly\ q\ x$
⟨*proof*⟩

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

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

lemma *normalize-field-eq-1* [*simp*]:
$x \neq 0 \Longrightarrow normalize\ (x :: 'a :: \{field,normalization\text{-}semidom\}) = 1$
⟨*proof*⟩

lemma *unit-factor-field* [*simp*]:
$unit\text{-}factor\ (x :: 'a :: \{field,normalization\text{-}semidom\}) = x$
⟨*proof*⟩

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

lemma *poly-div-gcd-squarefree*:
assumes $(p :: ('a::\{field\text{-}char\text{-}0,field\text{-}gcd\})\ poly) \neq 0$
defines $d \equiv gcd\ p\ (pderiv\ p)$
shows $coprime\ (p\ div\ d)\ (pderiv\ (p\ div\ d))$ (**is** *?A*) **and**
$\bigwedge x.\ poly\ (p\ div\ d)\ x = 0 \longleftrightarrow poly\ p\ x = 0$ (**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** $sgn\ (poly\ p\ a) \neq sgn\ (poly\ p\ (b::real))$
shows $\exists x.\ a \leq x \wedge x \leq b \wedge poly\ p\ x = 0$
⟨*proof*⟩

lemma *poly-different-sign-imp-root'*:
assumes $sgn\ (poly\ p\ a) \neq sgn\ (poly\ p\ (b::real))$
shows $\exists x.\ poly\ p\ x = 0$
⟨*proof*⟩

lemma *no-roots-inbetween-imp-same-sign*:
assumes $a < b\ \forall x.\ a \leq x \wedge x \leq b \longrightarrow poly\ p\ x \neq (0::real)$
shows $sgn\ (poly\ p\ a) = sgn\ (poly\ p\ b)$
⟨*proof*⟩

1.2.4 Limits of polynomials

lemma *poly-neighbourhood-without-roots*:
assumes $(p :: real\ poly) \neq 0$
shows $eventually\ (\lambda x.\ poly\ p\ x \neq 0)\ (at\ x_0)$
⟨*proof*⟩

lemma *poly-neighbourhood-same-sign*:
assumes $poly\ p\ (x_0 :: real) \neq 0$
shows $eventually\ (\lambda x.\ sgn\ (poly\ p\ x) = sgn\ (poly\ p\ x_0))\ (at\ x_0)$
⟨*proof*⟩

lemma *poly-lhopital*:
  assumes $poly\ p\ (x::real) = 0\ poly\ q\ x = 0\ q \neq 0$
  assumes $(\lambda x.\ poly\ (pderiv\ p)\ x\ /\ poly\ (pderiv\ q)\ x)\ -x\to\ y$
  shows $(\lambda x.\ poly\ p\ x\ /\ poly\ q\ x)\ -x\to\ y$
⟨*proof*⟩

lemma *poly-roots-bounds*:
  assumes $p \neq 0$
  obtains $l\ u$
  where $l \leq (u :: real)$
    and $poly\ p\ l \neq 0$
    and $poly\ p\ u \neq 0$
    and $\{x.\ x > l \wedge x \leq u \wedge poly\ p\ x = 0\ \} = \{x.\ poly\ p\ x = 0\}$
    and $\bigwedge x.\ x \leq l \Longrightarrow sgn\ (poly\ p\ x) = sgn\ (poly\ p\ l)$
    and $\bigwedge x.\ x \geq u \Longrightarrow sgn\ (poly\ p\ x) = sgn\ (poly\ p\ u)$
⟨*proof*⟩

definition *poly-inf* :: ($'a$::*real-normed-vector*) *poly* ⇒ $'a$ **where**
  poly-inf $p \equiv sgn\ (coeff\ p\ (degree\ p))$
definition *poly-neg-inf* :: ($'a$::*real-normed-vector*) *poly* ⇒ $'a$ **where**
  poly-neg-inf $p \equiv$ *if even* $(degree\ p)$ *then* $sgn\ (coeff\ p\ (degree\ p))$
    else $-sgn\ (coeff\ p\ (degree\ p))$
lemma *poly-inf-0-iff*[*simp*]:
  poly-inf $p = 0 \longleftrightarrow p = 0$ *poly-neg-inf* $p = 0 \longleftrightarrow p = 0$
  ⟨*proof*⟩

lemma *poly-inf-mult*[*simp*]:
  fixes p :: ($'a$::*real-normed-field*) *poly*
  shows *poly-inf* $(p*q)$ = *poly-inf* p ∗ *poly-inf* q
    poly-neg-inf $(p*q)$ = *poly-neg-inf* p ∗ *poly-neg-inf* q
⟨*proof*⟩

lemma *poly-neq-0-at-infinity*:
  assumes $(p :: real\ poly) \neq 0$
  shows *eventually* $(\lambda x.\ poly\ p\ x \neq 0)$ *at-infinity*
⟨*proof*⟩

lemma *poly-limit-aux*:
  fixes $p :: real\ poly$
  defines $n \equiv degree\ p$
  shows $((\lambda x.\ poly\ p\ x\ /\ x\ \widehat{}\ n) \longrightarrow coeff\ p\ n)$ *at-infinity*
⟨*proof*⟩

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

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

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

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

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

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

1.2.5 Signs of polynomials for sufficiently large values

lemma *polys-inf-sign-thresholds*:
  assumes *finite* (*ps* :: *real poly set*)
  obtains $l\ u$
  where $l \leq u$
    and $\bigwedge p.\ [\![p \in ps;\ p \neq 0]\!] \Longrightarrow$
        $\{x.\ l < x \wedge x \leq u \wedge poly\ p\ x = 0\} = \{x.\ poly\ p\ x = 0\}$
    and $\bigwedge p\ x.\ [\![p \in ps;\ x \geq u]\!] \Longrightarrow sgn\ (poly\ p\ x) = poly\text{-}inf\ p$
    and $\bigwedge p\ x.\ [\![p \in ps;\ x \leq l]\!] \Longrightarrow sgn\ (poly\ p\ x) = poly\text{-}neg\text{-}inf\ p$
⟨*proof*⟩

1.2.6 Positivity of polynomials

lemma *poly-pos*:
$(\forall x::real.\ poly\ p\ x > 0) \longleftrightarrow poly\text{-}inf\ p = 1 \land (\forall x.\ poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-greater*:
$(\forall x::real.\ x > a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$poly\text{-}inf\ p = 1 \land (\forall x.\ x > a \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-geq*:
$(\forall x::real.\ x \geq a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$poly\text{-}inf\ p = 1 \land (\forall x.\ x \geq a \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-less*:
$(\forall x::real.\ x < a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$poly\text{-}neg\text{-}inf\ p = 1 \land (\forall x.\ x < a \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-leq*:
$(\forall x::real.\ x \leq a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$poly\text{-}neg\text{-}inf\ p = 1 \land (\forall x.\ x \leq a \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-between-less-less*:
$(\forall x::real.\ a < x \land x < b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(a \geq b \lor poly\ p\ ((a+b)/2) > 0) \land (\forall x.\ a < x \land x < b \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-between-less-leq*:
$(\forall x::real.\ a < x \land x \leq b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(a \geq b \lor poly\ p\ b > 0) \land (\forall x.\ a < x \land x \leq b \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-between-leq-less*:
$(\forall x::real.\ a \leq x \land x < b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(a \geq b \lor poly\ p\ a > 0) \land (\forall x.\ a \leq x \land x < b \longrightarrow poly\ p\ x \neq 0)$
⟨*proof*⟩

lemma *poly-pos-between-leq-leq*:
$(\forall x::real.\ a \leq x \land x \leq b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(a > b \lor poly\ p\ a > 0) \land (\forall x.\ a \leq x \land x \leq b \longrightarrow poly\ p\ x \neq 0)$
⟨*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* $(\lambda p.$ *sgn* (*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 $p_1, \ldots, p_i, \ldots, p_n$ at x is the same as the sum of the sign changes of the sequence $p_1, \ldots, p_i$ and $p_i, \ldots, p_n$ as long as $p_i(x) \neq 0$.

lemma *sign-changes-distrib*:
poly p x $\neq 0 \Longrightarrow$
sign-changes $(ps_1$ @ $[p]$ @ $ps_2)\ x =$
sign-changes $(ps_1$ @ $[p])\ x +$ *sign-changes* $([p]$ @ $ps_2)\ x$
⟨*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 $\forall\, i <$ *length ps. sgn* (*poly* (*ps*!*i*) *x*) = *sgn* (*poly* (*ps'*!*i*) *y*)
shows *sign-changes ps x* = *sign-changes ps' y*
⟨*proof*⟩

lemma *sign-changes-cong'*:
assumes $\forall\, p \in$ *set ps. sgn* (*poly p x*) = *sgn* (*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** *sgn* (*poly r x*) = − *sgn* (*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**
sign-changes-inf ps =
  length (*remdups-adj* (*filter* ($\lambda x.\ x \neq 0$) (*map poly-inf ps*))) − *1*

definition *sign-changes-neg-inf* **where**
sign-changes-neg-inf ps =
  length (*remdups-adj* (*filter* ($\lambda x.\ x \neq 0$) (*map 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* :: (*real poly*) *list*
  assumes *last-ps-sgn-const*[*simp*]:
    $\bigwedge x\ y.$ *sgn* (*poly* (*last ps*) *x*) = *sgn* (*poly* (*last ps*) *y*)
  assumes *ps-not-Nil*[*simp*]: *ps* ≠ []
  assumes *signs*: $\bigwedge i\ x.$ ⟦*i* < *length ps* − *2*; *poly* (*ps* ! (*i*+*1*)) *x* = *0*⟧
    ⟹ (*poly* (*ps* ! (*i*+*2*)) *x*) ∗ (*poly* (*ps* ! *i*) *x*) < *0*

Now we define a Sturm sequence $p_1, \ldots, p_n$ 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* :: *real poly*

assumes *hd-ps-p*[*simp*]: *hd ps* = *p*
assumes *length-ps-ge-2*[*simp*]: *length ps* ≥ *2*
assumes *deriv*: $\bigwedge x_0$. *poly p* x_0 = *0* ⟹
eventually (λ*x*. *sgn* (*poly* (*p* ∗ *ps*!*1*) *x*) =
(*if x* > x_0 *then 1 else* −*1*)) (*at* x_0)
assumes *p-squarefree*: ⋀*x*. ¬(*poly p x* = *0* ∧ *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* ≠ []
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 ⋀*i x*. ⟦*i* < *length ps* − *2*; *poly* (*ps* ! (*i*+*1*)) *x* = *0*⟧
⟹ *sgn* (*poly* (*ps* ! (*i*+*2*)) *x*) = − *sgn* (*poly* (*ps* ! *i*) *x*)
shows ∀ *j*≤*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*]] |
split-sign-changes [*p*,*q*] *x* = [[*p*,*q*]] |
split-sign-changes (*p*#*q*#*r*#*ps*) *x* =
(*if poly p x* ≠ *0* ∧ *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′ ∈ *set* (*split-sign-changes ps x*) ⟹ *set ps*′ ⊆ *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*:
$[\![\bigwedge p\ x.\ P\ [p]\ x;\ \bigwedge p\ q\ x.\ \textit{quasi-sturm-seq}\ [p,q] \Longrightarrow P\ [p,q]\ x;$
$\bigwedge p\ q\ r\ ps\ x.\ \textit{quasi-sturm-seq}\ (p\#q\#r\#ps) \Longrightarrow$
$[\![\textit{poly}\ p\ x \neq 0 \Longrightarrow \textit{poly}\ q\ x = 0 \Longrightarrow P\ (r\#ps)\ x;$
$\textit{poly}\ q\ x \neq 0 \Longrightarrow P\ (q\#r\#ps)\ x;$
$\textit{poly}\ p\ x = 0 \Longrightarrow P\ (q\#r\#ps)\ x]\!]$
$\Longrightarrow P\ (p\#q\#r\#ps)\ x]\!] \Longrightarrow P\ ps\ x$
⟨*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 $\textit{poly}\ (\textit{hd}\ ps)\ x_0 \neq 0$
defines $\textit{sign-changes}' \equiv \lambda ps\ x.$
$\sum ps'{\leftarrow}\textit{split-sign-changes}\ ps\ x.\ \textit{sign-changes}\ ps'\ x$
shows $\textit{sign-changes}'\ ps\ x_0 = \textit{sign-changes}\ ps\ x_0$
⟨*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 $\textit{poly}\ (\textit{hd}\ ps)\ x_0 \neq 0$
defines $\textit{sign-changes}' \equiv \lambda x_0\ ps\ x.$
$\sum ps'{\leftarrow}\textit{split-sign-changes}\ ps\ x_0.\ \textit{sign-changes}\ ps'\ x$
shows $\textit{eventually}\ (\lambda x.\ \textit{sign-changes}'\ x_0\ ps\ x = \textit{sign-changes}\ ps\ x)\ (\textit{at}\ x_0)$
⟨*proof*⟩

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

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

shows *eventually* ($\lambda x.$ *sign-changes ps* x =
sign-changes ps x_0 + (*if* $x{<}x_0$ *then 1 else 0*)) (*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 $f : \mathbb{R} \to \mathbb{N}$ 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 $\forall x$::*real*. $a < x \wedge x \leq b \longrightarrow$ *eventually* ($\lambda\xi$. $f\ \xi = (f\ x$::*nat*)) (*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 *sign-changes ps a* − *sign-changes ps b* =
card $\{x.\ x > a \wedge x \leq b \wedge$ *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 *sign-changes ps a* − *sign-changes-inf ps* =
card $\{x.\ x > a \wedge$ *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 *sign-changes-neg-inf ps* − *sign-changes ps a* =
card $\{x.\ x \leq a \wedge$ *poly* $p\ x = 0\}$
⟨*proof*⟩

lemma (**in** *sturm-seq*) *count-roots*:
assumes $p \neq 0$
shows *sign-changes-neg-inf ps* − *sign-changes-inf ps* =
card $\{x.$ *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* :: *real poly*) *q* =
  (*if degree q* = *0 then* [*p*,*q*] *else p* # *sturm-aux q* (−(*p mod q*)))
  ⟨*proof*⟩
termination ⟨*proof*⟩
definition *sturm* **where** *sturm p* = *sturm-aux p* (*pderiv p*)

Next, we show some simple facts about this construction:

lemma *sturm-0*[*simp*]: *sturm 0* = [*0*,*0*]
  ⟨*proof*⟩

lemma [*simp*]: *sturm-aux p q* = [] ⟷ *False*
  ⟨*proof*⟩

lemma *sturm-neq-Nil*[*simp*]: *sturm p* ≠ [] ⟨*proof*⟩

lemma [*simp*]: *hd* (*sturm p*) = *p*
  ⟨*proof*⟩

lemma [*simp*]: *p* ∈ *set* (*sturm p*)
  ⟨*proof*⟩

lemma [*simp*]: *length* (*sturm p*) ≥ *2*
⟨*proof*⟩

lemma [*simp*]: *degree* (*last* (*sturm p*)) = *0*
⟨*proof*⟩

lemma [*simp*]: *sturm-aux p q* ! *0* = *p*
  ⟨*proof*⟩
lemma [*simp*]: *sturm-aux p q* ! *Suc 0* = *q*
  ⟨*proof*⟩

lemma [*simp*]: *sturm p* ! *0* = *p*

⟨*proof*⟩
lemma [*simp*]: *sturm p* ! *Suc 0* = *pderiv p*
⟨*proof*⟩

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

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*: $r \in set\ (sturm\text{-}aux\ p\ q) \Longrightarrow gcd\ p\ q\ dvd\ r$
⟨*proof*⟩

lemma *sturm-gcd*: $r \in set\ (sturm\ p) \Longrightarrow gcd\ p\ (pderiv\ p)\ dvd\ r$
⟨*proof*⟩

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 $\forall j \leq i+1.\ poly\ (sturm\ p\ !\ j)\ x = 0$
⟨*proof*⟩

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 $\neg rsquarefree\ p$
⟨*proof*⟩

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* (λx. *sgn* (*poly* ($p*q$) x) = (*if* $x > x_0$ *then 1 else* -1)) (*at* x_0)
⟨*proof*⟩

lemma *sturm-firsttwo-signs*:
fixes *ps* :: *real poly list*
assumes *squarefree*: *rsquarefree p*
assumes *p-0*: *poly p* (x_0::*real*) = *0*
shows *eventually* (λx. *sgn* (*poly* (p * *sturm p* ! *1*) x) =
(*if* $x > x_0$ *then 1 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*: $i <$ *length* (*sturm* (p :: *real poly*)) $-$ *2*
assumes *q-0*: *poly* (*sturm p* ! (i+*1*)) x = *0* (**is** *poly ?q x* = *0*)
shows *poly* (*sturm p* ! (i+*2*)) x * *poly* (*sturm p* ! i) x < *0*
(**is** *poly ?p x* * *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* (*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**
sturm-squarefree p = *sturm* (p *div* (*gcd p* (*pderiv p*)))

lemma *sturm-squarefree-not-Nil*[*simp*]: *sturm-squarefree p* ≠ []
⟨*proof*⟩

lemma *sturm-seq-sturm-squarefree*:
assumes [*simp*]: $p \neq 0$
defines [*simp*]: $p' \equiv p$ *div gcd p* (*pderiv p*)
shows *sturm-seq* (*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 = (*let d* = *gcd p* (*pderiv p*)
  in map (λ*p′*. *p′ div d*) (*sturm p*))

This construction also has all the desired properties:

lemma *sturm-squarefree′-adjacent-root-propagate-left*:
  assumes *p* ≠ *0*
  assumes *i* < *length* (*sturm-squarefree′* (*p* :: *real poly*)) − *1*
  assumes *poly* (*sturm-squarefree′ p* ! *i*) *x* = *0*
    and *poly* (*sturm-squarefree′ p* ! (*i* + *1*)) *x* = *0*
  shows ∀ *j*≤*i*+*1*. *poly* (*sturm-squarefree′ p* ! *j*) *x* = *0*
⟨*proof*⟩

lemma *sturm-squarefree′-adjacent-roots*:
  assumes *p* ≠ *0*
    i < *length* (*sturm-squarefree′* (*p* :: *real poly*)) − *1*
    poly (*sturm-squarefree′ p* ! *i*) *x* = *0*
    poly (*sturm-squarefree′ p* ! (*i* + *1*)) *x* = *0*
  shows *False*
⟨*proof*⟩

lemma *sturm-squarefree′-signs*:
  assumes *p* ≠ *0*
  assumes *i-in-range*: *i* < *length* (*sturm-squarefree′* (*p* :: *real poly*)) − *2*
  assumes *q-0*: *poly* (*sturm-squarefree′ p* ! (*i*+*1*)) *x* = *0* (**is** *poly ?q x* = *0*)
  shows *poly* (*sturm-squarefree′ p* ! (*i*+*2*)) *x* ∗
    poly (*sturm-squarefree′ p* ! *i*) *x* < *0*
      (**is** *poly ?r x* ∗ *poly ?p x* < *0*)
⟨*proof*⟩

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

lemma *sturm-seq-sturm-squarefree′*:
  assumes (*p* :: *real poly*) ≠ *0*
  defines *d* ≡ *gcd p* (*pderiv p*)
  shows *sturm-seq* (*sturm-squarefree′ p*) (*p div d*)
    (**is** *sturm-seq ?ps′ ?p′*)
⟨*proof*⟩

This construction is obviously more expensive to compute than the one that *first* divides p by $\gcd(p, p')$ 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-squarefree 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 $length\ (remdups\text{-}adj\ (filter\ (\lambda x.\ x \neq 0)\ (map\ ((*)\ d \circ f)\ xs))) =$
        $length\ (remdups\text{-}adj\ (filter\ (\lambda x.\ x \neq 0)\ (map\ f\ xs)))$
⟨*proof*⟩

lemma *sturm-sturm-squarefree′-same-sign-changes*:
  fixes $p :: real\ poly$
  defines $ps \equiv sturm\ p$ **and** $ps' \equiv sturm\text{-}squarefree'\ p$
  shows $poly\ p\ x \neq 0 \lor poly\ (pderiv\ p)\ x \neq 0 \Longrightarrow$
            $sign\text{-}changes\ ps'\ x = sign\text{-}changes\ ps\ x$
        $p \neq 0 \Longrightarrow sign\text{-}changes\text{-}inf\ ps' = sign\text{-}changes\text{-}inf\ ps$
        $p \neq 0 \Longrightarrow sign\text{-}changes\text{-}neg\text{-}inf\ ps' = sign\text{-}changes\text{-}neg\text{-}inf\ ps$
⟨*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**
$count\text{-}roots\text{-}between\ p\ a\ b = (if\ a \leq b \land p \neq 0\ then$
  $(let\ ps = sturm\text{-}squarefree\ p$
    $in\ sign\text{-}changes\ ps\ a - sign\text{-}changes\ ps\ b)\ else\ 0)$

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

definition *count-roots-above* **where**
$count\text{-}roots\text{-}above\ p\ a = (if\ (p::real\ poly) = 0\ then\ 0\ else$
  $(let\ ps = sturm\text{-}squarefree\ p$
    $in\ sign\text{-}changes\ ps\ a - sign\text{-}changes\text{-}inf\ ps))$

definition *count-roots-below* **where**
$count\text{-}roots\text{-}below\ p\ a = (if\ (p::real\ poly) = 0\ then\ 0\ else$
  $(let\ ps = sturm\text{-}squarefree\ p$
    $in\ sign\text{-}changes\text{-}neg\text{-}inf\ ps - sign\text{-}changes\ ps\ a))$

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

lemma *count-roots-correct*:
  fixes *p* :: *real poly*
  shows *count-roots p* = *card* {*x*. *poly p x* = *0*} (**is** - = *card ?S*)
⟨*proof*⟩

lemma *count-roots-above-correct*:
  fixes *p* :: *real poly*
  shows *count-roots-above p a* = *card* {*x*. *x* > *a* ∧ *poly p x* = *0*}
    (**is** - = *card ?S*)
⟨*proof*⟩

lemma *count-roots-below-correct*:
  fixes *p* :: *real poly*
  shows *count-roots-below p a* = *card* {*x*. *x* ≤ *a* ∧ *poly p x* = *0*}
    (**is** - = *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 = pderiv p
    in if a > b ∨ p = 0 then 0
    else if (poly p a ≠ 0 ∨ poly q a ≠ 0) ∧ (poly p b ≠ 0 ∨ poly q b ≠ 0)
         then (let ps = sturm p
               in sign-changes ps a − sign-changes ps b)
         else (let ps = sturm-squarefree p
               in sign-changes ps a − sign-changes ps b))
```
⟨*proof*⟩

lemma *count-roots-code*[*code*]:
```
count-roots (p::real poly) =
  (if p = 0 then 0
   else let ps = sturm p
        in sign-changes-neg-inf ps − sign-changes-inf ps)
```
⟨*proof*⟩

lemma *count-roots-above-code*[*code*]:
```
count-roots-above 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 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*} =

(*count-roots-between p a b* +
(*if* $p \neq 0 \wedge a < b \wedge$ *poly p a* = *0 then 1 else 0*) −
(*if* $p \neq 0 \wedge a < b \wedge$ *poly p b* = *0 then 1 else 0*))
⟨*proof*⟩

lemma *poly-card-roots*:
card {*x*::*real*. *poly p x* = *0*} = *count-roots p*
⟨*proof*⟩

lemma *poly-no-roots*:
(∀ *x*. *poly p x* ≠ *0*) ⟷ (*p* ≠ *0* ∧ *count-roots p* = *0*)
⟨*proof*⟩

lemma *poly-pos*:
(∀ *x*. *poly p x* > *0*) ⟷ (
p ≠ *0* ∧ *poly-inf p* = *1* ∧ *count-roots p* = *0*)
⟨*proof*⟩

lemma *poly-card-roots-greater*:
card {*x*::*real*. *x* > *a* ∧ *poly p x* = *0*} = *count-roots-above p a*
⟨*proof*⟩

lemma *poly-card-roots-leq*:
card {*x*::*real*. *x* ≤ *a* ∧ *poly p x* = *0*} = *count-roots-below p a*
⟨*proof*⟩

lemma *poly-card-roots-geq*:
card {*x*::*real*. *x* ≥ *a* ∧ *poly p x* = *0*} = (
count-roots-above p a + (*if poly p a* = *0* ∧ *p* ≠ *0 then 1 else 0*))
⟨*proof*⟩

lemma *poly-card-roots-less*:
card {*x*::*real*. *x* < *a* ∧ *poly p x* = *0*} =
(*count-roots-below p a* − (*if poly p a* = *0* ∧ *p* ≠ *0 then 1 else 0*))
⟨*proof*⟩

lemma *poly-no-roots-less-leq*:
(∀ *x*. *a* < *x* ∧ *x* ≤ *b* ⟶ *poly p x* ≠ *0*) ⟷
((*a* ≥ *b* ∨ (*p* ≠ *0* ∧ *count-roots-between p a b* = *0*)))
⟨*proof*⟩

lemma *poly-pos-between-less-leq*:
(∀ *x*. *a* < *x* ∧ *x* ≤ *b* ⟶ *poly p x* > *0*) ⟷
((*a* ≥ *b* ∨ (*p* ≠ *0* ∧ *poly p b* > *0* ∧ *count-roots-between p a b* = *0*)))
⟨*proof*⟩

lemma *poly-no-roots-leq-leq*:
$(\forall x.\ a \le x \wedge x \le b \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$((a > b \vee (p \ne 0 \wedge poly\ p\ a \ne 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b = 0)))$
⟨*proof*⟩

lemma *poly-pos-between-leq-leq*:
$(\forall x.\ a \le x \wedge x \le b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$((a > b \vee (p \ne 0 \wedge poly\ p\ a > 0 \wedge$
$count\text{-}roots\text{-}between\ p\ a\ b = 0)))$
⟨*proof*⟩

lemma *poly-no-roots-less-less*:
$(\forall x.\ a < x \wedge x < b \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$((a \ge b \vee p \ne 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b =$
$(if\ poly\ p\ b = 0\ then\ 1\ else\ 0)))$
⟨*proof*⟩

lemma *poly-pos-between-less-less*:
$(\forall x.\ a < x \wedge x < b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$((a \ge b \vee (p \ne 0 \wedge poly\ p\ ((a+b)/2) > 0 \wedge$
$count\text{-}roots\text{-}between\ p\ a\ b = (if\ poly\ p\ b = 0\ then\ 1\ else\ 0))))$
⟨*proof*⟩

lemma *poly-no-roots-leq-less*:
$(\forall x.\ a \le x \wedge x < b \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$((a \ge b \vee p \ne 0 \wedge poly\ p\ a \ne 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b =$
$(if\ a < b \wedge poly\ p\ b = 0\ then\ 1\ else\ 0)))$
⟨*proof*⟩

lemma *poly-pos-between-leq-less*:
$(\forall x.\ a \le x \wedge x < b \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$((a \ge b \vee (p \ne 0 \wedge poly\ p\ a > 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b =$
$(if\ a < b \wedge poly\ p\ b = 0\ then\ 1\ else\ 0))))$
⟨*proof*⟩

lemma *poly-no-roots-greater*:
$(\forall x.\ x > a \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$((p \ne 0 \wedge count\text{-}roots\text{-}above\ p\ a = 0))$
⟨*proof*⟩

lemma *poly-pos-greater*:
$(\forall x.\ x > a \longrightarrow poly\ p\ x > 0) \longleftrightarrow ($
$p \ne 0 \wedge poly\text{-}inf\ p = 1 \wedge count\text{-}roots\text{-}above\ p\ a = 0)$
⟨*proof*⟩

lemma *poly-no-roots-leq*:

$(\forall x.\ x \le a \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$(\ (p \ne 0 \wedge count\text{-}roots\text{-}below\ p\ a = 0))$
⟨*proof*⟩

lemma *poly-pos-leq*:
$(\forall x.\ x \le a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(\ p \ne 0 \wedge poly\text{-}neg\text{-}inf\ p = 1 \wedge count\text{-}roots\text{-}below\ p\ a = 0)$
⟨*proof*⟩

lemma *poly-no-roots-geq*:
$(\forall x.\ x \ge a \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$(\ (p \ne 0 \wedge poly\ p\ a \ne 0 \wedge count\text{-}roots\text{-}above\ p\ a = 0))$
⟨*proof*⟩

lemma *poly-pos-geq*:
$(\forall x.\ x \ge a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(p \ne 0 \wedge poly\text{-}inf\ p = 1 \wedge poly\ p\ a \ne 0 \wedge count\text{-}roots\text{-}above\ p\ a = 0)$
⟨*proof*⟩

lemma *poly-no-roots-less*:
$(\forall x.\ x < a \longrightarrow poly\ p\ x \ne 0) \longleftrightarrow$
$((p \ne 0 \wedge count\text{-}roots\text{-}below\ p\ a = (if\ poly\ p\ a = 0\ then\ 1\ else\ 0)))$
⟨*proof*⟩

lemma *poly-pos-less*:
$(\forall x.\ x < a \longrightarrow poly\ p\ x > 0) \longleftrightarrow$
$(p \ne 0 \wedge poly\text{-}neg\text{-}inf\ p = 1 \wedge count\text{-}roots\text{-}below\ p\ a =$
$(if\ poly\ p\ a = 0\ then\ 1\ else\ 0))$
⟨*proof*⟩

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-TAG* $x \equiv x$

lemma *sturm-id-PR-prio0*:
$\{x::real.\ P\ x\} = \{x::real.\ (PR\text{-}TAG\ P)\ x\}$
$(\forall x::real.\ f\ x < g\ x) = (\forall x::real.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ P\ x) = (\forall x::real.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$\langle proof \rangle$

lemma *sturm-id-PR-prio1*:
$\{x::real.\ x < a \wedge P\ x\} = \{x::real.\ x < a \wedge (PR\text{-}TAG\ P)\ x\}$
$\{x::real.\ x \le a \wedge P\ x\} = \{x::real.\ x \le a \wedge (PR\text{-}TAG\ P)\ x\}$
$\{x::real.\ x \ge b \wedge P\ x\} = \{x::real.\ x \ge b \wedge (PR\text{-}TAG\ P)\ x\}$
$\{x::real.\ x > b \wedge P\ x\} = \{x::real.\ x > b \wedge (PR\text{-}TAG\ P)\ x\}$
$(\forall x::real < a.\ f\ x < g\ x) = (\forall x::real < a.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real \le a.\ f\ x < g\ x) = (\forall x::real \le a.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real > a.\ f\ x < g\ x) = (\forall x::real > a.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real \ge a.\ f\ x < g\ x) = (\forall x::real \ge a.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real < a.\ P\ x) = (\forall x::real < a.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real > a.\ P\ x) = (\forall x::real > a.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real \le a.\ P\ x) = (\forall x::real \le a.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real \ge a.\ P\ x) = (\forall x::real \ge a.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$\langle proof \rangle$

lemma *sturm-id-PR-prio2*:
$\{x::real.\ x > a \wedge x \le b \wedge P\ x\} =$
$\{x::real.\ x > a \wedge x \le b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x \ge a \wedge x \le b \wedge P\ x\} =$
$\{x::real.\ x \ge a \wedge x \le b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x \ge a \wedge x < b \wedge P\ x\} =$
$\{x::real.\ x \ge a \wedge x < b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x > a \wedge x < b \wedge P\ x\} =$
$\{x::real.\ x > a \wedge x < b \wedge PR\text{-}TAG\ P\ x\}$
$(\forall x::real.\ a < x \wedge x \le b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a < x \wedge x \le b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a \le x \wedge x \le b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a \le x \wedge x \le b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a \le x \wedge x < b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a \le x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a < x \wedge x \le b \longrightarrow P\ x) =$
$(\forall x::real.\ a < x \wedge x \le b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real.\ a \le x \wedge x \le b \longrightarrow P\ x) =$
$(\forall x::real.\ a \le x \wedge x \le b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$

$(\forall\, x{::}real.\ a \le x \wedge x < b \longrightarrow P\ x) =$
$\quad (\forall\, x{::}real.\ a \le x \wedge x < b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall\, x{::}real.\ a < x \wedge x < b \longrightarrow P\ x) =$
$\quad (\forall\, x{::}real.\ a < x \wedge x < b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
⟨*proof*⟩

lemma *PR-TAG-intro-prio0*:
fixes $P :: real \Rightarrow bool$ **and** $f :: real \Rightarrow real$
shows
$PR\text{-}TAG\ P = P' \Longrightarrow PR\text{-}TAG\ (\lambda x.\ \neg(\neg P\ x)) = P'$
$[\![PR\text{-}TAG\ P = (\lambda x.\ poly\ p\ x = 0);\ PR\text{-}TAG\ Q = (\lambda x.\ poly\ q\ x = 0)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ P\ x \wedge Q\ x) = (\lambda x.\ poly\ (gcd\ p\ q)\ x = 0)$ **and**
$[\![PR\text{-}TAG\ P = (\lambda x.\ poly\ p\ x = 0);\ PR\text{-}TAG\ Q = (\lambda x.\ poly\ q\ x = 0)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ P\ x \vee Q\ x) = (\lambda x.\ poly\ (p{*}q)\ x = 0)$ **and**

$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x = g\ x) = (\lambda x.\ poly\ (p{-}q)\ x = 0)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x \neq g\ x) = (\lambda x.\ poly\ (p{-}q)\ x \neq 0)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x < g\ x) = (\lambda x.\ poly\ (q{-}p)\ x > 0)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x \le g\ x) = (\lambda x.\ poly\ (q{-}p)\ x \ge 0)$

$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ -f\ x) = (\lambda x.\ poly\ (-p)\ x)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x + g\ x) = (\lambda x.\ poly\ (p{+}q)\ x)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x - g\ x) = (\lambda x.\ poly\ (p{-}q)\ x)$
$[\![PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x);\ PR\text{-}TAG\ g = (\lambda x.\ poly\ q\ x)]\!]$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x * g\ x) = (\lambda x.\ poly\ (p{*}q)\ x)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ (f\,x)\hat{}n) = (\lambda x.\ poly\ (p\hat{}n)\ x)$
$PR\text{-}TAG\ (\lambda x.\ poly\ p\ x :: real) = (\lambda x.\ poly\ p\ x)$
$PR\text{-}TAG\ (\lambda x.\ x{::}real) = (\lambda x.\ poly\ [{:}0,1{:}]\ x)$
$PR\text{-}TAG\ (\lambda x.\ a{::}real) = (\lambda x.\ poly\ [{:}a{:}]\ x)$
⟨*proof*⟩

lemma *PR-TAG-intro-prio1*:
fixes $f :: real \Rightarrow real$
shows
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x = 0) = (\lambda x.\ poly\ p\ x = 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x \neq 0) = (\lambda x.\ poly\ p\ x \neq 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ 0 = f\,x) = (\lambda x.\ poly\ p\ x = 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ 0 \neq f\,x) = (\lambda x.\ poly\ p\ x \neq 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x \ge 0) = (\lambda x.\ poly\ p\ x \ge 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\,x > 0) = (\lambda x.\ poly\ p\ x > 0)$

$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x \le 0) = (\lambda x.\ poly\ (-p)\ x \ge 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < 0) = (\lambda x.\ poly\ (-p)\ x > 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow$
$\quad PR\text{-}TAG\ (\lambda x.\ 0 \le f\ x) = (\lambda x.\ poly\ (-p)\ x \le 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x) \Longrightarrow$
$\quad PR\text{-}TAG\ (\lambda x.\ 0 < f\ x) = (\lambda x.\ poly\ (-p)\ x < 0)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ a * f\ x) = (\lambda x.\ poly\ (smult\ a\ p)\ x)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x * a) = (\lambda x.\ poly\ (smult\ a\ p)\ x)$
$PR\text{-}TAG\ f = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x\ /\ a) = (\lambda x.\ poly\ (smult\ (inverse\ a)\ p)\ x)$
$PR\text{-}TAG\ (\lambda x.\ x\hat{}n :: real) = (\lambda x.\ poly\ (monom\ 1\ n)\ x)$
⟨*proof*⟩

lemma *PR-TAG-intro-prio2*:
$PR\text{-}TAG\ (\lambda x.\ 1\ /\ b) = (\lambda x.\ inverse\ b)$
$PR\text{-}TAG\ (\lambda x.\ a\ /\ b) = (\lambda x.\ a\ /\ b)$
$PR\text{-}TAG\ (\lambda x.\ a\ /\ b * x\hat{}n :: real) = (\lambda x.\ poly\ (monom\ (a/b)\ n)\ x)$
$PR\text{-}TAG\ (\lambda x.\ x\hat{}n * a\ /\ b :: real) = (\lambda x.\ poly\ (monom\ (a/b)\ n)\ x)$
$PR\text{-}TAG\ (\lambda x.\ a * x\hat{}n :: real) = (\lambda x.\ poly\ (monom\ a\ n)\ x)$
$PR\text{-}TAG\ (\lambda x.\ x\hat{}n * a :: real) = (\lambda x.\ poly\ (monom\ a\ n)\ x)$
$PR\text{-}TAG\ (\lambda x.\ x\hat{}n\ /\ a :: real) = (\lambda x.\ poly\ (monom\ (inverse\ a)\ n)\ x)$

$PR\text{-}TAG\ (\lambda x.\ f\ x\hat{}(Suc\ (Suc\ 0)) :: real) = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x * f\ x :: real) = (\lambda x.\ poly\ p\ x)$
$PR\text{-}TAG\ (\lambda x.\ (f\ x)\hat{}Suc\ n :: real) = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ (f\ x)\hat{}n * f\ x :: real) = (\lambda x.\ poly\ p\ x)$
$PR\text{-}TAG\ (\lambda x.\ (f\ x)\hat{}Suc\ n :: real) = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x * (f\ x)\hat{}n :: real) = (\lambda x.\ poly\ p\ x)$
$PR\text{-}TAG\ (\lambda x.\ (f\ x)\hat{}(m+n) :: real) = (\lambda x.\ poly\ p\ x)$
$\quad \Longrightarrow PR\text{-}TAG\ (\lambda x.\ (f\ x)\hat{}m * (f\ x)\hat{}n :: real) = (\lambda x.\ poly\ p\ x)$
⟨*proof*⟩

lemma *sturm-meta-spec*: $(\bigwedge x::real.\ P\ x) \Longrightarrow P\ x$ ⟨*proof*⟩
lemma *sturm-imp-conv*:
$(a < x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$
$(a \le x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a \le x \wedge x < b \longrightarrow c)$
$(a < x \longrightarrow x \le b \longrightarrow c) \longleftrightarrow (a < x \wedge x \le b \longrightarrow c)$
$(a \le x \longrightarrow x \le b \longrightarrow c) \longleftrightarrow (a \le x \wedge x \le b \longrightarrow c)$
$(x < b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$
$(x < b \longrightarrow a \le x \longrightarrow c) \longleftrightarrow (a \le x \wedge x < b \longrightarrow c)$
$(x \le b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x \le b \longrightarrow c)$
$(x \le b \longrightarrow a \le x \longrightarrow c) \longleftrightarrow (a \le x \wedge x \le b \longrightarrow c)$
⟨*proof*⟩

3.3 Setup for the "sturm" method

⟨*ML*⟩

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 b proven by the *sturm* method.

lemma
$\forall x::real.\ x\hat{}2 + 1 \neq 0$
⟨*proof*⟩

lemma
  fixes $x :: real$
  shows $x\hat{}2 + 1 \neq 0$ ⟨*proof*⟩

lemma $(x::real) > 1 \Longrightarrow x\hat{}3 > 1$ ⟨*proof*⟩

lemma $\forall x::real.\ x*x \neq -1$ ⟨*proof*⟩

schematic-goal *A*:
$card\ \{x::real.\ -0.010831 < x \wedge x < 0.010831 \wedge$
  $1/120*x\hat{}5 + 1/24*x\hat{}4 + 1/6*x\hat{}3 - 49/16777216*x\hat{}2 - 17/2097152*x = 0\}$
  $= ?n$
  ⟨*proof*⟩

lemma $card\ \{x::real.\ x\hat{}3 + x = 2*x\hat{}2 \wedge x\hat{}3 - 6*x\hat{}2 + 11*x = 6\} = 1$
⟨*proof*⟩

schematic-goal $card\ \{x::real.\ x\hat{}3 + x = 2*x\hat{}2 \vee x\hat{}3 - 6*x\hat{}2 + 11*x = 6\} = ?n$ ⟨*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 $(x::real)*x+1 \neq 0 \wedge (x\hat{~}2+1)*(x\hat{~}2+2) \neq 0$ ⟨*proof*⟩

3 examples related to continued fraction approximants to exp: LCP

lemma fixes *x::real*
shows $-7.29347719 \leq x \Longrightarrow 0 < x\hat{~}5 + 30*x\hat{~}4 + 420*x\hat{~}3 + 3360*x\hat{~}2 + 15120*x + 30240$
⟨*proof*⟩

lemma fixes *x::real*
shows $0 < x\hat{~}6 + 42*x\hat{~}5 + 840*x\hat{~}4 + 10080*x\hat{~}3 + 75600*x^2 + 332640*x + 665280$
⟨*proof*⟩

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

end