

A Formalisation of Sturm's Theorem

Manuel Eberl

May 6, 2022

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	3
1.1	Analysis	3
1.2	Polynomials	3
1.2.1	General simplification lemmas	3
1.2.2	Divisibility of polynomials	4
1.2.3	Sign changes of a polynomial	7
1.2.4	Limits of polynomials	8
1.2.5	Signs of polynomials for sufficiently large values	16
1.2.6	Positivity of polynomials	17
2	Proof of Sturm’s Theorem	22
2.1	Sign changes of polynomial sequences	22
2.2	Definition of Sturm sequences locale	23
2.3	Auxiliary lemmas about roots and sign changes	25
2.4	Constructing Sturm sequences	37
2.5	The canonical Sturm sequence	37
2.5.1	Canonical squarefree Sturm sequence	43
2.5.2	Optimisation for multiple roots	44
2.6	Root-counting functions	50
3	The “sturm” proof method	56
3.1	Preliminary lemmas	56
3.2	Reification	65
3.3	Setup for the “sturm” method	68
4	Example usage of the “sturm” method	68

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 (*rule-connected-local-const*)

```
show connected { $a..b$ }  $a \in \{a..b\} \ b \in \{a..b\}$  using  $\langle a \leq b \rangle$  by (auto intro: connected-Icc)
```

```
show  $\forall aa \in \{a..b\}. \text{eventually } (\lambda b. f \ aa = f \ b) \text{ (at } aa \text{ within } \{a..b\})$ 
```

proof

```
fix  $x$  assume  $x \in \{a..b\}$ 
```

```
with assms(2)[rule-format, of x]
```

```
show eventually  $(\lambda b. f \ x = f \ b) \text{ (at } x \text{ within } \{a..b\})$ 
```

```
by (auto simp: eventually-at-filter elim: eventually-mono)
```

qed

qed

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 –

```
from assms obtain  $r$  where  $p = q * r$  unfolding dvd-def by blast
```

```
hence  $q * \text{pderiv } p - p * \text{pderiv } q = (q * q) * \text{pderiv } r$ 
```

```
by (simp add: algebra-simps pderiv-mult)
```

```
thus  $q * q \ \text{dvd} \ (q * \text{pderiv } p - p * \text{pderiv } q)$  by simp
```

```
note  $0 = \text{pderiv-mult}[of \ q \ p \ \text{div} \ q]$ 
```

```
have  $1: q * (p \ \text{div} \ q) = p$ 
```

```
by (metis assms(1) assms(2) dvd-def nonzero-mult-div-cancel-left)
```

```
have  $f1: \text{pderiv } (p \ \text{div} \ q) * (q * q) \ \text{div} \ (q * q) = \text{pderiv } (p \ \text{div} \ q)$ 
```

```
by simp
```

```
have  $f2: \text{pderiv } p = q * \text{pderiv } (p \ \text{div} \ q) + p \ \text{div} \ q * \text{pderiv } q$ 
```

```
by (metis 0 1)
```

```
have  $p * \text{pderiv } q = \text{pderiv } q * (q * (p \ \text{div} \ q))$ 
```

```
by (metis 1 mult commute)
```

```
then have  $p * \text{pderiv } q = q * (p \ \text{div} \ q * \text{pderiv } q)$ 
```

```
by fastforce
```

```
then have  $q * \text{pderiv } p - p * \text{pderiv } q = q * (q * \text{pderiv } (p \ \text{div} \ q))$ 
```

using *f2* **by** (*metis add-diff-cancel-right' distrib-left*)
then show $pderiv (p \text{ div } q) = (q * pderiv p - p * pderiv q) \text{ div } (q * q)$
using *f1* **by** (*metis mult.commute mult.left-commute*)
qed

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 *clarify*

assume $poly p x = 0 \text{ poly } q x = 0$
then have $[-x, 1:] \text{ dvd } p \text{ } [-x, 1:] \text{ dvd } q$
by (*simp-all add: poly-eq-0-iff-dvd*)
with that have *is-unit* $[-x, 1:]$
by (*rule coprime-common-divisor*)
then show *False*
by (*auto simp add: is-unit-pCons-iff*)

qed

lemma *poly-div*:

assumes $poly q x \neq 0$ **and** $(q::'a :: field \text{ poly}) \text{ dvd } p$
shows $poly (p \text{ div } q) x = poly p x / poly q x$

proof –

from *assms* **have** $[simp]: q \neq 0$ **by** *force*
have $poly q x * poly (p \text{ div } q) x = poly (q * (p \text{ div } q)) x$ **by** *simp*
also have $q * (p \text{ div } q) = p$
using *assms* **by** (*simp add: div-mult-swap*)
finally show $poly (p \text{ div } q) x = poly p x / poly q x$
using *assms* **by** (*simp add: field-simps*)

qed

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

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

proof –

obtain $r \ s$ **where** *bezout-coefficients* $p (pderiv p) = (r, s)$
by (*auto simp add: prod-eq-iff*)
then have $r * p + s * pderiv p = gcd p (pderiv p)$
by (*rule bezout-coefficients*)
then have $rs: d = r * p + s * pderiv p$
by (*simp add: d-def*)
define t **where** $t = p \text{ div } d$
define p' **where** $[simp]: p' = pderiv p$
define d' **where** $[simp]: d' = pderiv d$

define u **where** $u = p' \text{ div } d$
have $A: p = t * d$ **and** $B: p' = u * d$
by (*simp-all add: t-def u-def d-def algebra-simps*)
from *poly-squarefree-decomp*[*OF assms(1) A B[unfolded p'-def] rs*]
show $\bigwedge x. \text{poly}(p \text{ div } d) x = 0 \iff \text{poly } p x = 0$ **by** (*auto simp: t-def*)

from rs **have** $C: s*t*d' = d * (1 - r*t - s*pderiv t)$
by (*simp add: A B algebra-simps pderiv-mult*)
from *assms* **have** [*simp*]: $p \neq 0 \ d \neq 0 \ t \neq 0$
by (*force, force, subst (asm) A, force*)

have $\bigwedge x. [x \text{ dvd } t; x \text{ dvd } (pderiv t)] \implies x \text{ dvd } 1$
proof –

fix x **assume** $x \text{ dvd } t \ x \text{ dvd } (pderiv t)$
then obtain $v \ w$ **where** vw :
 $t = x*v \ pderiv t = x*w$ **unfolding** *dvd-def* **by** *blast*
define $x' \ v'$ **where** [*simp*]: $x' = pderiv x$ **and** [*simp*]: $v' = pderiv v$
from vw **have** $x*v' + v*x' = x*w$ **by** (*simp add: pderiv-mult*)
hence $v*x' = x*(w - v')$ **by** (*simp add: algebra-simps*)
hence $x \text{ dvd } v*pderiv x$ **by** *simp*
then obtain y **where** $y: v*x' = x*y$ **unfolding** *dvd-def* **by** *force*
from $\langle t \neq 0 \rangle$ **and** vw **have** $x \neq 0$ **by** *simp*

have *x-pow-n-dvd-d*: $\bigwedge n. x^n \text{ dvd } d$
proof–

fix n **show** $x^n \text{ dvd } d$
proof (*induction n, simp, rename-tac n, case-tac n*)
fix n **assume** $n = (0::nat)$
from vw **and** C **have** $d = x*(d*r*v + d*s*w + s*v*d')$
by (*simp add: algebra-simps*)
with $\langle n = 0 \rangle$ **show** $x^{Suc\ n} \text{ dvd } d$ **by** (*force intro: dvdI*)
next
fix $n \ n'$ **assume** *IH*: $x^n \text{ dvd } d$ **and** $n = Suc\ n'$
hence [*simp*]: $Suc\ n' = n \ x * x^{n'} = x^n$ **by** *simp-all*
define $c :: 'a \text{ poly}$ **where** $c = [:of-nat\ n:]$
from *pderiv-power-Suc*[*of x n'*]
have [*simp*]: $pderiv (x^n) = c*x^{n'} * x'$ **unfolding** *c-def*
by *simp*

from *IH* **obtain** z **where** $d = x^n * z$ **unfolding** *dvd-def* **by** *blast*

define z' **where** [*simp*]: $z' = pderiv z$
from $d \ \langle d \neq 0 \rangle$ **have** $x^n \neq 0 \ z \neq 0$ **by** *force+*
from $C \ d$ **have** $x^n * z = z*r*v*x^{Suc\ n} + z*s*c*x^n*(v*x') +$
 $s*v*z'*x^{Suc\ n} + s*z*(v*x')*x^n + s*z*v'*x^{Suc\ n}$
by (*simp add: algebra-simps vw pderiv-mult*)
also have $\dots = x^n * x * (z*r*v + z*s*c*y + s*v*z' + s*z*y + s*z*v')$
by (*simp only: y, simp add: algebra-simps*)
finally have $z = x*(z*r*v + z*s*c*y + s*v*z' + s*z*y + s*z*v')$
using $\langle x^n \neq 0 \rangle$ **by** *force*

hence $x \text{ dvd } z$ **by** (*metis dvd-triv-left*)
with d **show** $x \wedge \text{Suc } n \text{ dvd } d$ **by** *simp*
qed
qed

have $\text{degree } x = 0$
proof (*cases degree x, simp*)
case (*Suc n*)
hence $x \neq 0$ **by** *auto*
with *Suc* **have** $\text{degree } (x \wedge (\text{Suc } (\text{degree } d))) > \text{degree } d$
by (*subst degree-power-eq, simp-all*)
moreover from $x\text{-pow-}n\text{-dvd-}d$ [*of Suc (degree d)*] **and** $\langle d \neq 0 \rangle$
have $\text{degree } (x \wedge \text{Suc } (\text{degree } d)) \leq \text{degree } d$
by (*simp add: dvd-imp-degree-le*)
ultimately show *?thesis* **by** *simp*
qed
then obtain c **where** [*simp*]: $x = [:c:]$ **by** (*cases x, simp split: if-split-asm*)
moreover from $\langle x \neq 0 \rangle$ **have** $c \neq 0$ **by** *simp*
ultimately show $x \text{ dvd } 1$ **using** *dvdI* [*of 1 x [:inverse c:]*]
by *simp*
qed

then show *coprime t* (*pderiv t*)
by (*rule coprimeI*)
qed

lemma *normalize-field*:
 $\text{normalize } (x :: 'a :: \{\text{field}, \text{normalization-semidom}\}) = (\text{if } x = 0 \text{ then } 0 \text{ else } 1)$
by (*auto simp: is-unit-normalize dvd-field-iff*)

lemma *normalize-field-eq-1* [*simp*]:
 $x \neq 0 \implies \text{normalize } (x :: 'a :: \{\text{field}, \text{normalization-semidom}\}) = 1$
by (*simp add: normalize-field*)

lemma *unit-factor-field* [*simp*]:
 $\text{unit-factor } (x :: 'a :: \{\text{field}, \text{normalization-semidom}\}) = x$
by (*cases x = 0*) (*auto simp: is-unit-unit-factor dvd-field-iff*)

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–
have $?A \wedge (\forall x. ?B \ x)$
proof (*cases pderiv p = 0*)
case *False*

```

    from poly-div-gcd-squarefree-aux[OF this] show ?thesis
      unfolding d-def by auto
  next
  case True
  then obtain c where [simp]: p = [:c:] using pderiv-iszero by blast
  from assms(1) have c ≠ 0 by simp
  from True have d = smult (inverse c) p
    by (simp add: d-def normalize-poly-def map-poly-pCons field-simps)
  with ⟨p ≠ 0⟩ ⟨c ≠ 0⟩ have p div d = [:c:]
    by (simp add: pCons-one)
  with ⟨c ≠ 0⟩ show ?thesis
    by (simp add: normalize-const-poly is-unit-triv)
qed
thus ?A and  $\bigwedge x. ?B x$  by simp-all
qed

```

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 (cases $\text{poly } p \ a = 0 \vee \text{poly } p \ b = 0$)

case True

thus ?thesis using assms(1)

by (elim disjE, rule-tac exI[of - a], simp,
rule-tac exI[of - b], simp)

next

case False

hence [simp]: $\text{poly } p \ a \neq 0 \ \text{poly } p \ b \neq 0$ by simp-all

show ?thesis

proof (cases $\text{poly } p \ a < 0$)

case True

hence $\text{sgn}(\text{poly } p \ a) = -1$ by simp

with assms True have $\text{poly } p \ b > 0$

by (auto simp: sgn-real-def split: if-split-asm)

from poly-IVT-pos[OF ⟨a < b⟩ True this] guess x ..

thus ?thesis by (intro exI[of - x], simp)

next

case False

hence $\text{poly } p \ a > 0$ by (simp add: not-less less-eq-real-def)

hence $\text{sgn}(\text{poly } p \ a) = 1$ by simp

with assms False have $\text{poly } p \ b < 0$

by (auto simp: sgn-real-def not-less
less-eq-real-def split: if-split-asm)

from poly-IVT-neg[OF ⟨a < b⟩ ⟨poly p a > 0⟩ this] guess x ..

thus ?thesis by (intro exI[of - x], simp)

qed

qed

lemma *poly-different-sign-imp-root'*:
assumes $\text{sgn}(\text{poly } p \ a) \neq \text{sgn}(\text{poly } p \ (b::\text{real}))$
shows $\exists x. \text{poly } p \ x = 0$
using *assms* **by** (*cases* $a < b$, *auto* *dest!*: *poly-different-sign-imp-root*
simp: *less-eq-real-def not-less*)

lemma *no-roots-inbetween-imp-same-sign*:
assumes $a < b \ \forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq (0::\text{real})$
shows $\text{sgn}(\text{poly } p \ a) = \text{sgn}(\text{poly } p \ b)$
using *poly-different-sign-imp-root* *assms* **by** *auto*

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 –
{
fix $\varepsilon :: \text{real}$ **assume** $\varepsilon > 0$
have *fin*: *finite* $\{x. |x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0\}$
using *poly-roots-finite*[*OF* *assms*] **by** *simp*
with $\langle \varepsilon > 0 \rangle$ **have** $\exists \delta > 0. \delta \leq \varepsilon \wedge (\forall x. |x - x_0| < \delta \wedge x \neq x_0 \longrightarrow \text{poly } p \ x \neq 0)$
proof (*induction* *card* $\{x. |x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0\}$
arbitrary: ε *rule*: *less-induct*)
case (*less* ε)
let $?A = \{x. |x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0\}$
show *?case*
proof (*cases* *card* $?A$)
case 0
hence $?A = \{\}$ **using** *less* **by** *auto*
thus *?thesis* **using** *less(2)* **by** (*rule-tac* *exI*[*of* - ε], *auto*)
next
case (*Suc* -)
with *less(3)* **have** $\{x. |x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0\} \neq \{\}$ **by** *force*
then **obtain** x **where** *x-props*: $|x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0$ **by** *blast*
define ε' **where** $\varepsilon' = |x - x_0| / 2$
have $\varepsilon' > 0 \ \varepsilon' < \varepsilon$ **unfolding** ε' -*def* **using** *x-props* **by** *simp-all*
from *x-props*(1,2) **and** $\langle \varepsilon > 0 \rangle$
have $x \notin \{x'. |x' - x_0| < \varepsilon' \wedge x' \neq x_0 \wedge \text{poly } p \ x' = 0\}$ (*is* - \notin $?B$)
by (*auto* *simp*: ε' -*def*)
moreover **from** *x-props*
have $x \in \{x. |x - x_0| < \varepsilon \wedge x \neq x_0 \wedge \text{poly } p \ x = 0\}$ **by** *blast*
ultimately **have** $?B \subset ?A$ **by** *auto*
hence *card* $?B < \text{card } ?A$ *finite* $?B$
by (*rule* *psubset-card-mono*[*OF* *less(3)*],
blast *intro*: *finite-subset*[*OF* - *less(3)*])
from *less(1)*[*OF* *this(1)* $\langle \varepsilon' > 0 \rangle$ *this(2)*]


```

      show ?thesis using ‹ $\varepsilon' < \varepsilon$ › by force
    qed
  qed
}
from this[of 1]
  show ?thesis by (auto simp: eventually-at dist-real-def)
qed

```

lemma *poly-neighbourhood-same-sign*:

```

  assumes poly p (x0 :: real) ≠ 0
  shows eventually (λx. sgn (poly p x) = sgn (poly p x0)) (at x0)
proof -
  have cont: isCont (λx. sgn (poly p x)) x0
    by (rule isCont-sgn, rule poly-isCont, rule assms)
  then have eventually (λx. |sgn (poly p x) - sgn (poly p x0)| < 1) (at x0)
    by (auto simp: isCont-def tendsto-iff dist-real-def)
  then show ?thesis
    by (rule eventually-mono) (simp add: sgn-real-def split: if-split-asm)
qed

```

lemma *poly-lhopital*:

```

  assumes poly p (x::real) = 0 poly q x = 0 q ≠ 0
  assumes (λx. poly (pderiv p) x / poly (pderiv q) x) -x→ y
  shows (λx. poly p x / poly q x) -x→ y
using assms
proof (rule-tac lhopital)
  have isCont (poly p) x isCont (poly q) x by simp-all
  with assms(1,2) show poly p -x→ 0 poly q -x→ 0
    by (simp-all add: isCont-def)
  from ‹q ≠ 0› and ‹poly q x = 0› have pderiv q ≠ 0
    by (auto dest: pderiv-iszero)
  from poly-neighbourhood-without-roots[OF this]
  show eventually (λx. poly (pderiv q) x ≠ 0) (at x) .
qed (auto intro: poly-DERIV poly-neighbourhood-without-roots)

```

lemma *poly-roots-bounds*:

```

  assumes p ≠ 0
  obtains l u
  where l ≤ (u :: real)
    and poly p l ≠ 0
    and poly p u ≠ 0
    and {x. x > l ∧ x ≤ u ∧ poly p x = 0} = {x. poly p x = 0}
    and ∧x. x ≤ l ⇒ sgn (poly p x) = sgn (poly p l)
    and ∧x. x ≥ u ⇒ sgn (poly p x) = sgn (poly p u)
proof
  from assms have finite {x. poly p x = 0} (is finite ?roots)
    using poly-roots-finite by fast

```

let $?roots' = insert\ 0\ ?roots$
define l **where** $l = Min\ ?roots' - 1$
define u **where** $u = Max\ ?roots' + 1$
from $\langle finite\ ?roots \rangle$ **have** $A: finite\ ?roots'$ **by** *auto*
from $Min-le[OF\ this,\ of\ 0]$ **and** $Max-ge[OF\ this,\ of\ 0]$
show $l \leq u$ **by** *(simp add: l-def u-def)*
from $Min-le[OF\ A]$ **have** $l-props: \bigwedge x. x \leq l \implies poly\ p\ x \neq 0$
by *(fastforce simp: l-def)*
from $Max-ge[OF\ A]$ **have** $u-props: \bigwedge x. x \geq u \implies poly\ p\ x \neq 0$
by *(fastforce simp: u-def)*
from $l-props\ u-props$ **show** $[simp]: poly\ p\ l \neq 0\ poly\ p\ u \neq 0$ **by** *auto*
from $l-props$ **have** $\bigwedge x. poly\ p\ x = 0 \implies x > l$ **by** *(metis not-le)*
moreover from $u-props$ **have** $\bigwedge x. poly\ p\ x = 0 \implies x \leq u$ **by** *(metis linear)*
ultimately show $\{x. x > l \wedge x \leq u \wedge poly\ p\ x = 0\} = ?roots$ **by** *auto*
{
 fix x **assume** $A: x < l\ sgn\ (poly\ p\ x) \neq sgn\ (poly\ p\ l)$
 with $poly-IVT-pos[OF\ A(1),\ of\ p]\ poly-IVT-neg[OF\ A(1),\ of\ p]\ A(2)$
 have *False* **by** *(auto split: if-split-asm)*
 simp: sgn-real-def l-props not-less less-eq-real-def
}
thus $\bigwedge x. x \leq l \implies sgn\ (poly\ p\ x) = sgn\ (poly\ p\ l)$
by *(case-tac x = l, auto simp: less-eq-real-def)*
{
 fix x **assume** $A: x > u\ sgn\ (poly\ p\ x) \neq sgn\ (poly\ p\ u)$
 with $u-props\ poly-IVT-neg[OF\ A(1),\ of\ p]\ poly-IVT-pos[OF\ A(1),\ of\ p]\ A(2)$
 have *False* **by** *(auto split: if-split-asm)*
 simp: sgn-real-def l-props not-less less-eq-real-def
}
thus $\bigwedge x. x \geq u \implies sgn\ (poly\ p\ x) = sgn\ (poly\ p\ u)$
by *(case-tac x = u, auto simp: less-eq-real-def)*
qed

definition $poly-inf :: ('a::real-normed-vector)\ poly \Rightarrow 'a$ **where**
 $poly-inf\ p \equiv sgn\ (coeff\ p\ (degree\ p))$

definition $poly-neg-inf :: ('a::real-normed-vector)\ poly \Rightarrow 'a$ **where**
 $poly-neg-inf\ p \equiv if\ even\ (degree\ p)\ then\ sgn\ (coeff\ p\ (degree\ p))$
 else $-sgn\ (coeff\ p\ (degree\ p))$

lemma $poly-inf-0-iff[simp]:$
 $poly-inf\ p = 0 \iff p = 0\ poly-neg-inf\ p = 0 \iff p = 0$
by *(auto simp: poly-inf-def poly-neg-inf-def sgn-zero-iff)*

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
unfolding poly-inf-def poly-neg-inf-def
by ((cases p = 0  $\vee$  q = 0, auto simp: sgn-zero-iff
      degree-mult-eq[of p q] coeff-mult-degree-sum Real-Vector-Spaces.sgn-mult)[]) +

```

```

lemma poly-neg-0-at-infinity:
  assumes (p :: real poly)  $\neq$  0
  shows eventually ( $\lambda x$ . poly p x  $\neq$  0) at-infinity
proof -
  from poly-roots-bounds[OF assms] guess l u .
  note lu-props = this
  define b where b = max (-l) u
  show ?thesis
proof (subst eventually-at-infinity, rule exI[of - b], clarsimp)
  fix x assume A:  $|x| \geq b$  and B: poly p x = 0
  show False
proof (cases x  $\geq$  0)
  case True
    with A have x  $\geq$  u unfolding b-def by simp
    with lu-props(3, 6) show False by (metis sgn-zero-iff B)
  next
  case False
    with A have x  $\leq$  l unfolding b-def by simp
    with lu-props(2, 5) show False by (metis sgn-zero-iff B)
  qed
qed
qed

```

```

lemma poly-limit-aux:
  fixes p :: real poly
  defines n  $\equiv$  degree p
  shows (( $\lambda x$ . poly p x / xn)  $\longrightarrow$  coeff p n) at-infinity
proof (subst filterlim-cong, rule refl, rule refl)
  show eventually ( $\lambda x$ . poly p x / xn = ( $\sum_{i \leq n}$  coeff p i / x(n-i)))
    at-infinity
proof (rule eventually-mono)
  show eventually ( $\lambda x$ ::real. x  $\neq$  0) at-infinity
    by (simp add: eventually-at-infinity, rule exI[of - 1], auto)
  fix x :: real assume [simp]: x  $\neq$  0
  show poly p x / xn = ( $\sum_{i \leq n}$  coeff p i / x(n-i))
    by (simp add: n-def sum-divide-distrib power-diff poly-altdef)
qed

```

let $?a = \lambda i. \text{if } i = n \text{ then coeff } p \ n \text{ else } 0$
have $\forall i \in \{..n\}. ((\lambda x. \text{coeff } p \ i / x^{(n-i)}) \longrightarrow ?a \ i) \text{ at-infinity}$
proof
fix i **assume** $i \in \{..n\}$
hence $i \leq n$ **by** *simp*
show $((\lambda x. \text{coeff } p \ i / x^{(n-i)}) \longrightarrow ?a \ i) \text{ at-infinity}$
proof (*cases* $i = n$)
case *True*
thus $?thesis$ **by** (*intro* *tendstoI*, *subst* *eventually-at-infinity*,
intro *exI*[*of - 1*], *simp* *add: dist-real-def*)
next
case *False*
hence $n - i > 0$ **using** $\langle i \leq n \rangle$ **by** *simp*
from *tendsto-inverse-0* **and** *divide-real-def*[*of 1*]
have $((\lambda x. 1 / x :: \text{real}) \longrightarrow 0) \text{ at-infinity}$ **by** *simp*
from *tendsto-power*[*OF this*, *of* $n - i$]
have $((\lambda x::\text{real}. 1 / x^{(n-i)}) \longrightarrow 0) \text{ at-infinity}$
using $\langle n - i > 0 \rangle$ **by** (*simp* *add: power-0-left power-one-over*)
from *tendsto-mult-right-zero*[*OF this*, *of* *coeff* $p \ i$]
have $((\lambda x. \text{coeff } p \ i / x^{(n-i)}) \longrightarrow 0) \text{ at-infinity}$
by (*simp* *add: field-simps*)
thus $?thesis$ **using** *False* **by** *simp*
qed
qed
hence $((\lambda x. \sum_{i \leq n}. \text{coeff } p \ i / x^{(n-i)}) \longrightarrow (\sum_{i \leq n}. ?a \ i)) \text{ at-infinity}$
by (*force* *intro!*: *tendsto-sum*)
also **have** $(\sum_{i \leq n}. ?a \ i) = \text{coeff } p \ n$ **by** (*subst* *sum.delta*, *simp-all*)
finally **show** $((\lambda x. \sum_{i \leq n}. \text{coeff } p \ i / x^{(n-i)}) \longrightarrow \text{coeff } p \ n) \text{ at-infinity}$.
qed

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

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

assumes $\text{degree } p \geq 1$ *coeff* $p \ (\text{degree } p) > 0$

shows $LIM \ x \ \text{at-top}. \ \text{poly } p \ x \ :=> \ \text{at-top}$

proof –

let $?n = \text{degree } p$

define $f \ g$ **where** $f \ x = \text{poly } p \ x / x^{?n}$ **and** $g \ x = x^{?n}$ **for** $x :: \text{real}$

from *poly-limit-aux* **have** $(f \longrightarrow \text{coeff } p \ (\text{degree } p)) \text{ at-top}$

using *tendsto-mono at-top-le-at-infinity* **unfolding** *f-def* **by** *blast*

moreover **from** *assms*

have $LIM \ x \ \text{at-top}. \ g \ x \ :=> \ \text{at-top}$

by (*auto* *simp* *add: g-def* *intro!*: *filterlim-pow-at-top filterlim-ident*)

ultimately **have** $LIM \ x \ \text{at-top}. \ f \ x * g \ x \ :=> \ \text{at-top}$

using *filterlim-tendsto-pos-mult-at-top* *assms* **by** *simp*

also **have** *eventually* $(\lambda x. f \ x * g \ x = \text{poly } p \ x) \text{ at-top}$

unfolding *f-def* *g-def*

by (subst eventually-at-top-linorder, rule exI[of - 1],
 simp add: poly-altdef field-simps sum-distrib-left power-diff)
 note filterlim-cong[OF refl refl this]
 finally show ?thesis .
 qed

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 –
 from poly-at-top-at-top[of -p] and assms
 have $\text{LIM } x \text{ at-top. } -\text{poly } p \ x \ :> \text{ at-top}$ by simp
 thus ?thesis by (simp add: filterlim-uminus-at-bot)
 qed

lemma poly-lim-inf:
 eventually $(\lambda x::\text{real. } \text{sgn } (\text{poly } p \ x) = \text{poly-inf } p)$ at-top
proof (cases degree $p \geq 1$)
 case False
 hence $\text{degree } p = 0$ by simp
 then obtain c where $p = [c]$ by (cases p, auto split: if-split-asm)
 thus ?thesis
 by (simp add: eventually-at-top-linorder poly-inf-def)
 next
 case True
 note deg = this
 let ?lc = coeff p (degree p)
 from True have ?lc $\neq 0$ by force
 show ?thesis
proof (cases ?lc > 0)
 case True
 from poly-at-top-at-top[OF deg this]
 obtain x_0 where $\bigwedge x. x \geq x_0 \implies \text{poly } p \ x \geq 1$
 by (fastforce simp: filterlim-at-top
 eventually-at-top-linorder less-eq-real-def)
 hence $\bigwedge x. x \geq x_0 \implies \text{sgn } (\text{poly } p \ x) = 1$ by force
 thus ?thesis by (simp only: eventually-at-top-linorder poly-inf-def,
 intro exI[of - x_0], simp add: True)
 next
 case False
 hence ?lc < 0 using $\langle ?lc \neq 0 \rangle$ by linarith
 from poly-at-bot-at-top[OF deg this]
 obtain x_0 where $\bigwedge x. x \geq x_0 \implies \text{poly } p \ x \leq -1$
 by (fastforce simp: filterlim-at-bot
 eventually-at-top-linorder less-eq-real-def)
 hence $\bigwedge x. x \geq x_0 \implies \text{sgn } (\text{poly } p \ x) = -1$ by force
 thus ?thesis by (simp only: eventually-at-top-linorder poly-inf-def,
 intro exI[of - x_0], simp add: $\langle ?lc < 0 \rangle$)

qed
qed

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 $\text{LIM } x \text{ at-bot. } \text{poly } p \ x :=> (\text{if even } (\text{degree } p) \text{ then at-top else at-bot})$

proof –

let $?n = \text{degree } p$

define $f \ g$ **where** $f \ x = \text{poly } p \ x / x \wedge ?n$ **and** $g \ x = x \wedge ?n$ **for** $x :: \text{real}$

from *poly-limit-aux* **have** $(f \longrightarrow \text{coeff } p (\text{degree } p)) \text{ at-bot}$

using *tendsto-mono at-bot-le-at-infinity* **by** (*force simp: f-def [abs-def]*)

moreover from *assms*

have $\text{LIM } x \text{ at-bot. } g \ x :=> (\text{if even } (\text{degree } p) \text{ then at-top else at-bot})$

by (*auto simp add: g-def split: if-split-asm intro: filterlim-pow-at-bot-even filterlim-pow-at-bot-odd filterlim-ident*)

ultimately have $\text{LIM } x \text{ at-bot. } f \ x * g \ x :=>$

(if even ?n then at-top else at-bot)

by (*auto simp: assms intro: filterlim-tendsto-pos-mult-at-top filterlim-tendsto-pos-mult-at-bot*)

also have *eventually* $(\lambda x. f \ x * g \ x = \text{poly } p \ x) \text{ at-bot}$

unfolding *f-def g-def*

by (*subst eventually-at-bot-linorder, rule exI[of - -1],*

simp add: poly-altdef field-simps sum-distrib-left power-diff)

note *filterlim-cong[OF refl refl this]*

finally show *?thesis* .

qed

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 $\text{LIM } x \text{ at-bot. } \text{poly } p \ x :=> (\text{if even } (\text{degree } p) \text{ then at-bot else at-top})$

proof –

from *poly-at-top-or-bot-at-bot[of -p]* **and** *assms*

have $\text{LIM } x \text{ at-bot. } -\text{poly } p \ x :=>$

(if even (degree p) then at-top else at-bot) **by** *simp*

thus *?thesis* **by** (*auto simp: filterlim-uminus-at-bot*)

qed

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 (*cases degree p ≥ 1*)

case *False*

hence $\text{degree } p = 0$ **by** *simp*

then obtain c **where** $p = [c:]$ **by** (*cases p, auto split: if-split-asm*)

thus *?thesis*

by (*simp add: eventually-at-bot-linorder poly-neg-inf-def*)

```

next
case True
  note deg = this
  let ?lc = coeff p (degree p)
  from True have ?lc ≠ 0 by force
  show ?thesis
  proof (cases ?lc > 0)
    case True
      note lc-pos = this
      show ?thesis
      proof (cases even (degree p))
        case True
          from poly-at-top-or-bot-at-bot[OF deg lc-pos] and True
          obtain x₀ where  $\bigwedge x. x \leq x_0 \implies \text{poly } p \ x \geq 1$ 
            by (fastforce simp add: filterlim-at-top filterlim-at-bot
              eventually-at-bot-linorder less-eq-real-def)
          hence  $\bigwedge x. x \leq x_0 \implies \text{sgn } (\text{poly } p \ x) = 1$  by force
          thus ?thesis
            by (simp add: True eventually-at-bot-linorder poly-neg-inf-def,
              intro exI[of - x₀], simp add: lc-pos)
        case False
          from poly-at-top-or-bot-at-bot[OF deg lc-pos] and False
          obtain x₀ where  $\bigwedge x. x \leq x_0 \implies \text{poly } p \ x \leq -1$ 
            by (fastforce simp add: filterlim-at-bot
              eventually-at-bot-linorder less-eq-real-def)
          hence  $\bigwedge x. x \leq x_0 \implies \text{sgn } (\text{poly } p \ x) = -1$  by force
          thus ?thesis
            by (simp add: False eventually-at-bot-linorder poly-neg-inf-def,
              intro exI[of - x₀], simp add: lc-pos)
      qed
    next
      case False
        hence lc-neg: ?lc < 0 using ‹?lc ≠ 0› by linarith
        show ?thesis
        proof (cases even (degree p))
          case True
            with poly-at-bot-or-top-at-bot[OF deg lc-neg]
            obtain x₀ where  $\bigwedge x. x \leq x_0 \implies \text{poly } p \ x \leq -1$ 
              by (fastforce simp: filterlim-at-bot
                eventually-at-bot-linorder less-eq-real-def)
            hence  $\bigwedge x. x \leq x_0 \implies \text{sgn } (\text{poly } p \ x) = -1$  by force
            thus ?thesis
              by (simp only: True eventually-at-bot-linorder poly-neg-inf-def,
                intro exI[of - x₀], simp add: lc-neg)
          case False
            with poly-at-bot-or-top-at-bot[OF deg lc-neg]
            obtain x₀ where  $\bigwedge x. x \leq x_0 \implies \text{poly } p \ x \geq 1$ 

```

by (fastforce simp: filterlim-at-top
 eventually-at-bot-linorder less-eq-real-def)
 hence $\bigwedge x. x \leq x_0 \implies \text{sgn}(\text{poly } p \ x) = 1$ by force
 thus ?thesis
 by (simp only: False eventually-at-bot-linorder poly-neg-inf-def,
 intro exI[of - x₀], simp add: lc-neg)
 qed
 qed
 qed

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 $\bigwedge p. \llbracket p \in ps; p \neq 0 \rrbracket \implies$

$\{x. l < x \wedge x \leq u \wedge \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0\}$

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 goal-cases

case prems: 1

have $\exists l \ u. l \leq u \wedge (\forall p \ x. p \in ps \wedge x \geq u \longrightarrow \text{sgn}(\text{poly } p \ x) = \text{poly-inf } p) \wedge$
 $(\forall p \ x. p \in ps \wedge x \leq l \longrightarrow \text{sgn}(\text{poly } p \ x) = \text{poly-neg-inf } p)$

(is $\exists l \ u. ?P \ ps \ l \ u$)

proof (induction rule: finite-subset-induct[OF assms(1), where A = UNIV])

case 1

show ?case by simp

next

case 2

show ?case by (intro exI[of - 42], simp)

next

case prems: (3 p ps)

from prems(4) obtain l u where lu-props: ?P ps l u by blast

from poly-lim-inf obtain u'

where u'-props: $\forall x \geq u'. \text{sgn}(\text{poly } p \ x) = \text{poly-inf } p$

by (force simp add: eventually-at-top-linorder)

from poly-lim-neg-inf obtain l'

where l'-props: $\forall x \leq l'. \text{sgn}(\text{poly } p \ x) = \text{poly-neg-inf } p$

by (force simp add: eventually-at-bot-linorder)

show ?case

by (rule exI[of - min l l'], rule exI[of - max u u'],
 insert lu-props l'-props u'-props, auto)

qed

then obtain l u where lu-props: $l \leq u$

$\bigwedge p \ x. p \in ps \implies u \leq x \implies \text{sgn}(\text{poly } p \ x) = \text{poly-inf } p$

$\bigwedge p \ x. p \in ps \implies x \leq l \implies \text{sgn}(\text{poly } p \ x) = \text{poly-neg-inf } p$ by blast

moreover {

fix p x assume A: $p \in ps \ p \neq 0 \ \text{poly } p \ x = 0$


```

from  $A$  have  $l < x < u$ 
  by (auto simp: not-le[symmetric] dest: lu-props(2,3))
}
note  $A = \text{this}$ 
have  $\bigwedge p. p \in ps \implies p \neq 0 \implies$ 
   $\{x. l < x \wedge x \leq u \wedge \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0\}$ 
  by (auto dest: A)

from prems[OF lu-props(1) this lu-props(2,3)] show thesis .
qed

```

1.2.6 Positivity of polynomials

lemma *poly-pos*:

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

proof (*intro iffI conjI*)

assume $A: \forall x::\text{real}. \text{poly } p \ x > 0$

have $\bigwedge x. \text{poly } p \ (x::\text{real}) > 0 \implies \text{poly } p \ x \neq 0$ **by** *simp*

with A **show** $\forall x::\text{real}. \text{poly } p \ x \neq 0$ **by** *simp*

from *poly-lim-inf* **obtain** x **where** $\text{sgn } (\text{poly } p \ x) = \text{poly-inf } p$

by (*auto simp: eventually-at-top-linorder*)

with A **show** $\text{poly-inf } p = 1$

by (*simp add: sgn-real-def split: if-split-asm*)

next

assume $\text{poly-inf } p = 1 \wedge (\forall x. \text{poly } p \ x \neq 0)$

hence $A: \text{poly-inf } p = 1$ **and** $B: (\forall x. \text{poly } p \ x \neq 0)$ **by** *simp-all*

from *poly-lim-inf* **obtain** x **where** $C: \text{sgn } (\text{poly } p \ x) = \text{poly-inf } p$

by (*auto simp: eventually-at-top-linorder*)

show $\forall x. \text{poly } p \ x > 0$

proof (*rule ccontr*)

assume $\neg(\forall x. \text{poly } p \ x > 0)$

then obtain x' **where** $\text{poly } p \ x' \leq 0$ **by** (*auto simp: not-less*)

with A **and** C **have** $\text{sgn } (\text{poly } p \ x') \neq \text{sgn } (\text{poly } p \ x)$

by (*auto simp: sgn-real-def split: if-split-asm*)

from *poly-different-sign-imp-root'[OF this]* **and** B

show *False* **by** *blast*

qed

qed

lemma *poly-pos-greater*:

$(\forall x::\text{real}. x > a \implies \text{poly } p \ x > 0) \iff$

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

proof (*intro iffI conjI*)

assume $A: \forall x::\text{real}. x > a \implies \text{poly } p \ x > 0$

have $\bigwedge x. \text{poly } p \ (x::\text{real}) > 0 \implies \text{poly } p \ x \neq 0$ **by** *simp*

with A **show** $\forall x::\text{real}. x > a \implies \text{poly } p \ x \neq 0$ **by** *auto*

from *poly-lim-inf* **obtain** x_0 **where**

$\forall x \geq x_0. \text{sgn}(\text{poly } p \ x) = \text{poly-inf } p$
by (*auto simp: eventually-at-top-linorder*)
hence $\text{poly-inf } p = \text{sgn}(\text{poly } p \ (\max x_0 \ (a + 1)))$ **by** *simp*
also from A **have** $\dots = 1$ **by** *force*
finally show $\text{poly-inf } p = 1$.

next
assume $\text{poly-inf } p = 1 \wedge (\forall x. x > a \longrightarrow \text{poly } p \ x \neq 0)$
hence A : $\text{poly-inf } p = 1$ **and**
 B : $(\forall x. x > a \longrightarrow \text{poly } p \ x \neq 0)$ **by** *simp-all*
from *poly-lim-inf* **obtain** x_0 **where**
 C : $\forall x \geq x_0. \text{sgn}(\text{poly } p \ x) = \text{poly-inf } p$
by (*auto simp: eventually-at-top-linorder*)
hence $\text{sgn}(\text{poly } p \ (\max x_0 \ (a+1))) = \text{poly-inf } p$ **by** *simp*
with A **have** D : $\text{sgn}(\text{poly } p \ (\max x_0 \ (a+1))) = 1$ **by** *simp*
show $\forall x. x > a \longrightarrow \text{poly } p \ x > 0$
proof (*rule ccontr*)
assume $\neg(\forall x. x > a \longrightarrow \text{poly } p \ x > 0)$
then obtain x' **where** $x' > a$ $\text{poly } p \ x' \leq 0$ **by** (*auto simp: not-less*)
with A **and** D **have** E : $\text{sgn}(\text{poly } p \ x') \neq \text{sgn}(\text{poly } p \ (\max x_0 \ (a+1)))$
by (*auto simp: sgn-real-def split: if-split-asm*)
show *False*
apply (*cases x' max x_0 (a+1) rule: linorder-cases*)
using $B \ E \ \langle x' > a \rangle$
apply (*force dest!: poly-different-sign-imp-root[of - - p]*)
done

qed
qed

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 (*intro iffI conjI*)
assume A : $\forall x::\text{real}. x \geq a \longrightarrow \text{poly } p \ x > 0$
hence $\forall x::\text{real}. x > a \longrightarrow \text{poly } p \ x > 0$ **by** *simp*
also note *poly-pos-greater*
finally have $\text{poly-inf } p = 1 \ (\forall x > a. \text{poly } p \ x \neq 0)$ **by** *simp-all*
moreover from A **have** $\text{poly } p \ a > 0$ **by** *simp*
ultimately show $\text{poly-inf } p = 1 \ \forall x \geq a. \text{poly } p \ x \neq 0$
by (*auto simp: less-eq-real-def*)

next
assume $\text{poly-inf } p = 1 \wedge (\forall x. x \geq a \longrightarrow \text{poly } p \ x \neq 0)$
hence A : $\text{poly-inf } p = 1$ **and**
 B : $\text{poly } p \ a \neq 0$ **and** C : $\forall x > a. \text{poly } p \ x \neq 0$ **by** *simp-all*
from A **and** C **and** *poly-pos-greater* **have** $\forall x > a. \text{poly } p \ x > 0$ **by** *simp*
moreover with $B \ C$ *poly-IVT-pos*[*of a a+1 p*] **have** $\text{poly } p \ a > 0$ **by** *force*
ultimately show $\forall x \geq a. \text{poly } p \ x > 0$ **by** (*auto simp: less-eq-real-def*)

qed

lemma *poly-pos-less*:

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

proof (*intro iffI conjI*)

assume $A: \forall x::real. x < a \longrightarrow poly\ p\ x > 0$

have $\bigwedge x. poly\ p\ (x::real) > 0 \implies poly\ p\ x \neq 0$ **by** *simp*

with A **show** $\forall x::real. x < a \longrightarrow poly\ p\ x \neq 0$ **by** *auto*

from *poly-lim-neg-inf* **obtain** x_0 **where**

$\forall x \leq x_0. sgn\ (poly\ p\ x) = poly\text{-neg-inf}\ p$

by (*auto simp: eventually-at-bot-linorder*)

hence $poly\text{-neg-inf}\ p = sgn\ (poly\ p\ (min\ x_0\ (a - 1)))$ **by** *simp*

also from A **have** $\dots = 1$ **by** *force*

finally show $poly\text{-neg-inf}\ p = 1$.

next

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

hence $A: poly\text{-neg-inf}\ p = 1$ **and**

$B: (\forall x. x < a \longrightarrow poly\ p\ x \neq 0)$ **by** *simp-all*

from *poly-lim-neg-inf* **obtain** x_0 **where**

$C: \forall x \leq x_0. sgn\ (poly\ p\ x) = poly\text{-neg-inf}\ p$

by (*auto simp: eventually-at-bot-linorder*)

hence $sgn\ (poly\ p\ (min\ x_0\ (a - 1))) = poly\text{-neg-inf}\ p$ **by** *simp*

with A **have** $D: sgn\ (poly\ p\ (min\ x_0\ (a - 1))) = 1$ **by** *simp*

show $\forall x. x < a \longrightarrow poly\ p\ x > 0$

proof (*rule ccontr*)

assume $\neg(\forall x. x < a \longrightarrow poly\ p\ x > 0)$

then obtain x' **where** $x' < a$ $poly\ p\ x' \leq 0$ **by** (*auto simp: not-less*)

with A **and** D **have** $E: sgn\ (poly\ p\ x') \neq sgn\ (poly\ p\ (min\ x_0\ (a - 1)))$

by (*auto simp: sgn-real-def split: if-split-asm*)

show *False*

apply (*cases x' min x_0 (a - 1) rule: linorder-cases*)

using $B\ E\ \langle x' < a \rangle$

apply (*auto dest!: poly-different-sign-imp-root[of - - p]*)
done

qed

qed

lemma *poly-pos-leg*:

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

proof (*intro iffI conjI*)

assume $A: \forall x::real. x \leq a \longrightarrow poly\ p\ x > 0$

hence $\forall x::real. x < a \longrightarrow poly\ p\ x > 0$ **by** *simp*

also note *poly-pos-less*

finally have $poly\text{-neg-inf}\ p = 1$ ($\forall x < a. poly\ p\ x \neq 0$) **by** *simp-all*

moreover from A **have** $poly\ p\ a > 0$ **by** *simp*

ultimately show $poly\text{-neg-inf}\ p = 1 \wedge \forall x \leq a. poly\ p\ x \neq 0$

by (*auto simp: less-eq-real-def*)

next

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

hence A : *poly-neg-inf* $p = 1$ and

B : *poly* $p \ a \neq 0$ and C : $\forall x < a. \text{poly } p \ x \neq 0$ by *simp-all*

from A and C and *poly-pos-less* have $\forall x < a. \text{poly } p \ x > 0$ by *simp*

moreover with $B \ C$ *poly-IVT-neg*[of $a - 1 \ a \ p$] have *poly* $p \ a > 0$ by *force*
ultimately show $\forall x \leq a. \text{poly } p \ x > 0$ by (*auto simp: less-eq-real-def*)

qed

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 (*intro iffI conjI*)

assume A : $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x > 0$

have $\bigwedge x. \text{poly } p \ (x :: \text{real}) > 0 \implies \text{poly } p \ x \neq 0$ by *simp*

with A show $\forall x :: \text{real}. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$ by *auto*

from A show $a \geq b \vee \text{poly } p \ ((a+b)/2) > 0$ by (*cases* $a < b$, *auto*)

next

assume $(b \leq a \vee 0 < \text{poly } p \ ((a+b)/2)) \wedge (\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0)$

hence A : $b \leq a \vee 0 < \text{poly } p \ ((a+b)/2)$ and

B : $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$ by *simp-all*

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

proof (*cases* $a \geq b$, *simp*, *clarify*, *rule-tac ccontr*,

simp only: not-le not-less)

fix x assume $a < b \ a < x \ x < b \ \text{poly } p \ x \leq 0$

with B have *poly* $p \ x < 0$ by (*simp add: less-eq-real-def*)

moreover from A and $\langle a < b \rangle$ have *poly* $p \ ((a+b)/2) > 0$ by *simp*

ultimately have *sgn* (*poly* $p \ x$) \neq *sgn* (*poly* $p \ ((a+b)/2)$) by *simp*

thus *False* using B

apply (*cases* $x \ (a+b)/2$ *rule: linorder-cases*)

apply (*drule poly-different-sign-imp-root*[of $- \ - \ p$], *assumption*,
insert $\langle a < b \rangle \ \langle a < x \rangle \ \langle x < b \rangle$, *force*) \square

apply *simp*

apply (*drule poly-different-sign-imp-root*[of $- \ - \ p$], *simp*,
insert $\langle a < b \rangle \ \langle a < x \rangle \ \langle x < b \rangle$, *force*)

done

qed

qed

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 (*intro iffI conjI*)

assume A : $\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0$

have $\bigwedge x. \text{poly } p \ (x :: \text{real}) > 0 \implies \text{poly } p \ x \neq 0$ by *simp*

with A show $\forall x :: \text{real}. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0$ by *auto*

from A show $a \geq b \vee \text{poly } p \ b > 0$ by (*cases* $a < b$, *auto*)

next

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

hence A : $b \leq a \vee 0 < \text{poly } p \ b$ and B : $\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0$
by *simp-all*

show $\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0$
proof (*cases* $a \geq b$, *simp*, *clarify*, *rule-tac ccontr*,
simp only: not-le not-less)
fix x **assume** $a < b$ $a < x$ $x \leq b$ $\text{poly } p \ x \leq 0$
with B **have** $\text{poly } p \ x < 0$ **by** (*simp add: less-eq-real-def*)
moreover from A **and** $\langle a < b \rangle$ **have** $\text{poly } p \ b > 0$ **by** *simp*
ultimately have $x < b$ **using** $\langle x \leq b \rangle$ **by** (*auto simp: less-eq-real-def*)
from $\langle \text{poly } p \ x < 0 \rangle$ **and** $\langle \text{poly } p \ b > 0 \rangle$
have $\text{sgn} (\text{poly } p \ x) \neq \text{sgn} (\text{poly } p \ b)$ **by** *simp*
from *poly-different-sign-imp-root*[*OF* $\langle x < b \rangle$ *this*] **and** B **and** $\langle x > a \rangle$
show *False* **by** *auto*
qed
qed

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 (*intro iffI conjI*)

assume $A: \forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x > 0$
have $\bigwedge x. \text{poly } p \ (x::\text{real}) > 0 \implies \text{poly } p \ x \neq 0$ **by** *simp*
with A **show** $\forall x::\text{real}. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$ **by** *auto*
from A **show** $a \geq b \vee \text{poly } p \ a > 0$ **by** (*cases* $a < b$, *auto*)

next

assume $(b \leq a \vee 0 < \text{poly } p \ a) \wedge (\forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0)$
hence $A: b \leq a \vee 0 < \text{poly } p \ a$ **and** $B: \forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$
by *simp-all*

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

proof (*cases* $a \geq b$, *simp*, *clarify*, *rule-tac ccontr*,
simp only: not-le not-less)

fix x **assume** $a < b$ $a \leq x$ $x < b$ $\text{poly } p \ x \leq 0$
with B **have** $\text{poly } p \ x < 0$ **by** (*simp add: less-eq-real-def*)
moreover from A **and** $\langle a < b \rangle$ **have** $\text{poly } p \ a > 0$ **by** *simp*
ultimately have $x > a$ **using** $\langle x \geq a \rangle$ **by** (*auto simp: less-eq-real-def*)
from $\langle \text{poly } p \ x < 0 \rangle$ **and** $\langle \text{poly } p \ a > 0 \rangle$
have $\text{sgn} (\text{poly } p \ a) \neq \text{sgn} (\text{poly } p \ x)$ **by** *simp*
from *poly-different-sign-imp-root*[*OF* $\langle x > a \rangle$ *this*] **and** B **and** $\langle x < b \rangle$
show *False* **by** *auto*

qed

qed

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 (*intro iffI conjI*)

assume $A: \forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0$
have $\bigwedge x. \text{poly } p \ (x::\text{real}) > 0 \implies \text{poly } p \ x \neq 0$ **by** *simp*
with A **show** $\forall x::\text{real}. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0$ **by** *auto*
from A **show** $a > b \vee \text{poly } p \ a > 0$ **by** (*cases* $a \leq b$, *auto*)

next

```

assume  $(b < a \vee 0 < \text{poly } p \ a) \wedge (\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0)$ 
hence  $A: b < a \vee 0 < \text{poly } p \ a$  and  $B: \forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0$ 
  by simp-all
show  $\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0$ 
proof (cases  $a > b$ , simp, clarify, rule-tac ccontr,
  simp only: not-le not-less)
  fix  $x$  assume  $a \leq b$   $a \leq x$   $x \leq b$   $\text{poly } p \ x \leq 0$ 
  with  $B$  have  $\text{poly } p \ x < 0$  by (simp add: less-eq-real-def)
  moreover from  $A$  and  $\langle a \leq b \rangle$  have  $\text{poly } p \ a > 0$  by simp
  ultimately have  $x > a$  using  $\langle x \geq a \rangle$  by (auto simp: less-eq-real-def)
  from  $\langle \text{poly } p \ x < 0 \rangle$  and  $\langle \text{poly } p \ a > 0 \rangle$ 
    have  $\text{sgn} (\text{poly } p \ a) \neq \text{sgn} (\text{poly } p \ x)$  by simp
  from poly-different-sign-imp-root[OF  $\langle x > a \rangle$  this] and  $B$  and  $\langle x \leq b \rangle$ 
    show False by auto
qed
qed

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 $(x::\text{real}) =$

$\text{length} (\text{remdups-adj} (\text{filter} (\lambda x. x \neq 0) (\text{map} (\lambda p. \text{sgn} (\text{poly } p \ x)) \ ps))) - 1$

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:*

$\text{poly } p \ x \neq 0 \implies$

$\text{sign-changes} (ps_1 @ [p] @ ps_2) \ x =$

$\text{sign-changes} (ps_1 @ [p]) \ x + \text{sign-changes} ([p] @ ps_2) \ x$

by (*simp add: sign-changes-def sgn-zero-iff, subst remdups-adj-append, simp*)

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 $\text{length } ps = \text{length } ps'$
assumes $\forall i < \text{length } ps. \text{sgn } (\text{poly } (ps!i) x) = \text{sgn } (\text{poly } (ps'!i) y)$
shows $\text{sign-changes } ps x = \text{sign-changes } ps' y$
proof –
from *assms*(2) **have** $A: \text{map } (\lambda p. \text{sgn } (\text{poly } p x)) ps = \text{map } (\lambda p. \text{sgn } (\text{poly } p y)) ps'$
proof (*induction rule: list-induct2[OF assms(1)]*)
case 1
then show ?*case* **by** *simp*
next
case (2 *p ps p' ps'*)
from 2(3)
have $\forall i < \text{length } ps. \text{sgn } (\text{poly } (ps ! i) x) = \text{sgn } (\text{poly } (ps' ! i) y)$ **by** *auto*
from 2(2)[*OF this*] 2(3) **show** ?*case* **by** *auto*
qed
show ?*thesis* **unfolding** *sign-changes-def* **by** (*simp add: A*)
qed

lemma *sign-changes-cong'*:
assumes $\forall p \in \text{set } ps. \text{sgn } (\text{poly } p x) = \text{sgn } (\text{poly } p y)$
shows $\text{sign-changes } ps x = \text{sign-changes } ps y$
using *assms* **by** (*intro sign-changes-cong, simp-all*)

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 $\text{poly } p x \neq 0$ **and** $\text{sgn } (\text{poly } r x) = - \text{sgn } (\text{poly } p x)$
shows $\text{sign-changes } [p, q, r] x = 1$
unfolding *sign-changes-def* **by** (*insert assms, auto simp: sgn-real-def*)

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 *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 *ps-not-Nil*[simp]: $ps \neq []$
assumes *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]: $\text{hd } ps = p$
assumes *length-ps-ge-2*[simp]: $\text{length } ps \geq 2$
assumes *deriv*: $\bigwedge x_0. \text{poly } p x_0 = 0 \implies$
eventually $(\lambda x. \text{sgn} (\text{poly} (p * ps!1) x) =$
 $(\text{if } x > x_0 \text{ then } 1 \text{ else } -1)) (\text{at } x_0)$
assumes *p-squarefree*: $\bigwedge x. \neg(\text{poly } p x = 0 \wedge \text{poly} (ps!1) x = 0)$
begin

Any Sturm sequence is obviously a Quasi-Sturm sequence.

lemma *quasi-sturm-seq*: *quasi-sturm-seq* $ps \dots$
end

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 (unfold-locales)
  show  $ps \neq []$  by fact
next
  from assms(1) interpret quasi-sturm-seq p#ps .
  fix  $x y$ 
  from last-ps-sgn-const and  $\langle ps \neq [] \rangle$ 
    show  $sgn (poly (last ps) x) = sgn (poly (last ps) y)$  by simp-all
next
  from assms(1) interpret quasi-sturm-seq p#ps .
  fix  $i x$ 
  assume  $i < length ps - 2$  and  $poly (ps ! (i+1)) x = 0$ 
  with signs[of i+1]
    show  $poly (ps ! (i+2)) x * poly (ps ! i) x < 0$  by simp
qed

```

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  $\bigwedge i x. [i < length ps - 2; poly (ps ! (i+1)) x = 0]$ 
     $\implies sgn (poly (ps ! (i+2)) x) = - sgn (poly (ps ! i) x)$ 
  shows  $\forall j \leq i+1. poly (ps ! j) x = 0$ 
using assms
proof (induction i)
  case 0 thus ?case by (clarsimp, rename-tac j, case-tac j, simp-all)
next
  case (Suc i)
    from Suc.premis(1,2)
      have  $sgn (poly (ps ! (i + 2)) x) = - sgn (poly (ps ! i) x)$ 
      by (intro assms(4), simp-all)
    with Suc.premis(3) have  $poly (ps ! i) x = 0$  by (simp add: sgn-zero-iff)
    with Suc.premis have  $\forall j \leq i+1. poly (ps ! j) x = 0$ 
      by (intro Suc.IH, simp-all)
    with Suc.premis(3) show ?case
      by (clarsimp, rename-tac j, case-tac j = Suc (Suc i), simp-all)
qed

```

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  $poly p x \neq 0 \wedge poly q x = 0$  then
      [p,q,r] # split-sign-changes (r#ps) x
    else

```

$[p,q] \# \text{split-sign-changes } (q\#r\#ps) x$

lemma (in *quasi-sturm-seq*) *split-sign-changes-subset*[*dest*]:
 $ps' \in \text{set } (\text{split-sign-changes } ps x) \implies \text{set } ps' \subseteq \text{set } ps$
apply (*insert ps-not-Nil*)
apply (*induction ps x rule: split-sign-changes.induct*)
apply (*simp, simp, rename-tac p q r ps x,*
case-tac poly p x $\neq 0 \wedge$ poly q x = 0, auto)
done

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 *goal-cases*
case *prems: 1*
have *quasi-sturm-seq ps ..*
with *prems show ?thesis*
proof (*induction ps x rule: split-sign-changes.induct*)
case ($\exists p q r ps x$)
show *?case*
proof (*rule 3(5)[OF 3(6)]*)
assume *A: poly p x $\neq 0$ poly q x = 0*
from $3(6)$ **have** *quasi-sturm-seq (r#ps)*
by (*force dest: quasi-sturm-seq-Cons*)
with $3 A$ **show** $P (r \# ps) x$ **by** *blast*
next
assume *A: poly q x $\neq 0$*
from $3(6)$ **have** *quasi-sturm-seq (q#r#ps)*
by (*force dest: quasi-sturm-seq-Cons*)
with $3 A$ **show** $P (q \# r \# ps) x$ **by** *blast*
next
assume *A: poly p x = 0*
from $3(6)$ **have** *quasi-sturm-seq (q#r#ps)*
by (*force dest: quasi-sturm-seq-Cons*)
with $3 A$ **show** $P (q \# r \# ps) x$ **by** *blast*
qed
qed *simp-all*
qed

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 poly (hd ps) x0 ≠ 0
defines sign-changes' ≡ λps x.
    ∑ ps' ← split-sign-changes ps x. sign-changes ps' x
shows sign-changes' ps x0 = sign-changes ps x0
using assms(1)
proof (induction x0 rule: split-sign-changes-induct)
case (∃ p q r ps x0)
  hence poly p x0 ≠ 0 by simp
  note IH = ∃(2,3,4)
  show ?case
  proof (cases poly q x0 = 0)
    case True
      from ∃ interpret quasi-sturm-seq p#q#r#ps by simp
      from signs[of 0] and True have
        sgn-r-x0: poly r x0 * poly p x0 < 0 by simp
      with ∃ have poly r x0 ≠ 0 by force
      from sign-changes-distrib[OF this, of [p,q] ps]
      have sign-changes (p#q#r#ps) x0 =
        sign-changes ([p, q, r]) x0 + sign-changes (r # ps) x0 by simp
      also have sign-changes (r#ps) x0 = sign-changes' (r#ps) x0
        using ⟨poly q x0 = 0⟩ ⟨poly p x0 ≠ 0⟩ ∃(5) ⟨poly r x0 ≠ 0⟩
        by (intro IH(1)[symmetric], simp-all)
      finally show ?thesis unfolding sign-changes'-def
        using True ⟨poly p x0 ≠ 0⟩ by simp
    next
      case False
        from sign-changes-distrib[OF this, of [p] r#ps]
        have sign-changes (p#q#r#ps) x0 =
          sign-changes ([p,q]) x0 + sign-changes (q#r#ps) x0 by simp
        also have sign-changes (q#r#ps) x0 = sign-changes' (q#r#ps) x0
          using ⟨poly q x0 ≠ 0⟩ ⟨poly p x0 ≠ 0⟩ ∃(5)
          by (intro IH(2)[symmetric], simp-all)
        finally show ?thesis unfolding sign-changes'-def
          using False by simp
  qed
qed (simp-all add: sign-changes-def sign-changes'-def)

```

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 poly (hd ps) x0 ≠ 0
defines sign-changes' ≡ λx0 ps x.
    ∑ ps' ← split-sign-changes ps x0. sign-changes ps' x
shows eventually (λx. sign-changes' x0 ps x = sign-changes ps x) (at x0)
proof (rule eventually-mono)
  show eventually (λx. ∀ p ∈ {p ∈ set ps. poly p x0 ≠ 0}. sgn (poly p x) = sgn (poly
    p x0)) (at x0)
  by (rule eventually-ball-finite, auto intro: poly-neighbourhood-same-sign)
next

```

```

fix x
show ( $\forall p \in \{p \in \text{set } ps. \text{poly } p \ x_0 \neq 0\}. \text{sgn } (\text{poly } p \ x) = \text{sgn } (\text{poly } p \ x_0) \implies$ 
   $\text{sign-changes}' \ x_0 \ ps \ x = \text{sign-changes } ps \ x$ )
proof -
  fix x assume nbh:  $\forall p \in \{p \in \text{set } ps. \text{poly } p \ x_0 \neq 0\}. \text{sgn } (\text{poly } p \ x) = \text{sgn } (\text{poly } p \ x_0)$ 
  thus  $\text{sign-changes}' \ x_0 \ ps \ x = \text{sign-changes } ps \ x$  using assms(1)
  proof (induction  $x_0$  rule: split-sign-changes-induct)
  case ( $\exists p \ q \ r \ ps \ x_0$ )
    hence  $\text{poly } p \ x_0 \neq 0$  by simp
    note  $IH = \exists(2,3,4)$ 
    show ?case
    proof (cases  $\text{poly } q \ x_0 = 0$ )
      case True
        from  $\exists$  interpret quasi-sturm-seq  $p\#q\#r\#ps$  by simp
        from signs[of 0] and True have
           $\text{sgn-r-x0: poly } r \ x_0 * \text{poly } p \ x_0 < 0$  by simp
        with  $\exists$  have  $\text{poly } r \ x_0 \neq 0$  by force
        with nbh  $\exists(5)$  have  $\text{poly } r \ x \neq 0$  by (auto simp: sgn-zero-iff)
        from sign-changes-distrib[OF this, of [p,q] ps]
          have  $\text{sign-changes } (p\#q\#r\#ps) \ x =$ 
             $\text{sign-changes } ([p, q, r]) \ x + \text{sign-changes } (r \# ps) \ x$  by simp
        also have  $\text{sign-changes } (r\#ps) \ x = \text{sign-changes}' \ x_0 \ (r\#ps) \ x$ 
          using  $\langle \text{poly } q \ x_0 = 0 \rangle$  nbh  $\langle \text{poly } p \ x_0 \neq 0 \rangle$   $\exists(5)$   $\langle \text{poly } r \ x_0 \neq 0 \rangle$ 
          by (intro IH(1)[symmetric], simp-all)
        finally show ?thesis unfolding sign-changes'-def
          using True  $\langle \text{poly } p \ x_0 \neq 0 \rangle$  by simp
      next
      case False
        with nbh  $\exists(5)$  have  $\text{poly } q \ x \neq 0$  by (auto simp: sgn-zero-iff)
        from sign-changes-distrib[OF this, of [p] r#ps]
          have  $\text{sign-changes } (p\#q\#r\#ps) \ x =$ 
             $\text{sign-changes } ([p,q]) \ x + \text{sign-changes } (q\#r\#ps) \ x$  by simp
        also have  $\text{sign-changes } (q\#r\#ps) \ x = \text{sign-changes}' \ x_0 \ (q\#r\#ps) \ x$ 
          using  $\langle \text{poly } q \ x_0 \neq 0 \rangle$  nbh  $\langle \text{poly } p \ x_0 \neq 0 \rangle$   $\exists(5)$ 
          by (intro IH(2)[symmetric], simp-all)
        finally show ?thesis unfolding sign-changes'-def
          using False by simp
    qed
  qed (simp-all add: sign-changes-def sign-changes'-def)
qed
qed

```

```

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  $x_0$ )
using assms

```

proof (*induction* x_0 *rule: split-sign-changes-induct*)
case (1 p x)
thus ?*case* **by** (*simp add: sign-changes-def*)
next
case (2 p q x_0)
hence [*simp*]: $ps' = [p, q]$ **by** *simp*
from 2 **have** *poly* p $x_0 \neq 0$ **by** *simp*
from 2(1) **interpret** *quasi-sturm-seq* [p, q] .
from *poly-neighbourhood-same-sign*[*OF* $\langle poly\ p\ x_0 \neq 0 \rangle$]
have *eventually* ($\lambda x. sgn\ (poly\ p\ x) = sgn\ (poly\ p\ x_0)$) (*at* x_0) .
moreover **from** *last-ps-sgn-const*
have *sgn-q*: $\bigwedge x. sgn\ (poly\ q\ x) = sgn\ (poly\ q\ x_0)$ **by** *simp*
ultimately **have** A : *eventually* ($\lambda x. \forall p \in set[p, q]. sgn\ (poly\ p\ x) =$
 $sgn\ (poly\ p\ x_0)$) (*at* x_0) **by** *simp*
thus ?*case* **by** (*force intro: eventually-mono*[*OF* A]
sign-changes-cong')
next
case (3 p q r ps'' x_0)
hence *p-not-0*: *poly* p $x_0 \neq 0$ **by** *simp*
note *sturm* = 3(1)
note *IH* = 3(2,3)
note *ps''-props* = 3(6)
show ?*case*
proof (*cases* *poly* q $x_0 = 0$)
case *True*
note *q-0* = *this*
from *sturm* **interpret** *quasi-sturm-seq* $p \# q \# r \# ps''$.
from *signs*[*of* 0] **and** *q-0*
have *signs'*: *poly* r $x_0 * poly\ p\ x_0 < 0$ **by** *simp*
with *p-not-0* **have** *r-not-0*: *poly* r $x_0 \neq 0$ **by** *force*
show ?*thesis*
proof (*cases* $ps' \in set\ (split-sign-changes\ (r \# ps'')\ x_0)$)
case *True*
show ?*thesis* **by** (*rule* *IH*(1), *fact*, *fact*, *simp add: r-not-0*, *fact*)
next
case *False*
with *ps''-props* *p-not-0* *q-0* **have** *ps'-props*: $ps' = [p, q, r]$ **by** *simp*
from *signs*[*of* 0] **and** *q-0*
have *sgn-r*: *poly* r $x_0 * poly\ p\ x_0 < 0$ **by** *simp*
from *p-not-0* *sgn-r*
have A : *eventually* ($\lambda x. sgn\ (poly\ p\ x) = sgn\ (poly\ p\ x_0) \wedge$
 $sgn\ (poly\ r\ x) = sgn\ (poly\ r\ x_0)$) (*at* x_0)
by (*intro eventually-conj poly-neighbourhood-same-sign*,
simp-all add: r-not-0)
show ?*thesis*
proof (*rule eventually-mono*[*OF* A], *clarify*,
subst ps'-props, *subst sign-changes-sturm-triple*)
fix x **assume** A : $sgn\ (poly\ p\ x) = sgn\ (poly\ p\ x_0)$
and B : $sgn\ (poly\ r\ x) = sgn\ (poly\ r\ x_0)$

```

have prod-neg:  $\bigwedge a (b::real). \llbracket a>0; b>0; a*b<0 \rrbracket \implies False$ 
              $\bigwedge a (b::real). \llbracket a<0; b<0; a*b<0 \rrbracket \implies False$ 
  by (drule mult-pos-pos, simp, simp,
      drule mult-neg-neg, simp, simp)
from A and  $\langle poly\ p\ x_0 \neq 0 \rangle$  show  $poly\ p\ x \neq 0$ 
  by (force simp: sgn-zero-iff)

with sgn-r p-not-0 r-not-0 A B
  have  $poly\ r\ x * poly\ p\ x < 0 \implies poly\ r\ x \neq 0$ 
  by (metis sgn-less sgn-mult, metis sgn-0-0)
with sgn-r show sgn-r':  $sgn (poly\ r\ x) = -sgn (poly\ p\ x)$ 
  apply (simp add: sgn-real-def not-le not-less
            split: if-split-asm, intro conjI impI)
  using prod-neg[of poly r x poly p x] apply force+
  done

show  $1 = sign-changes\ ps'\ x_0$ 
  by (subst ps'-props, subst sign-changes-sturm-triple,
      fact, metis A B sgn-r', simp)
qed
qed
next
case False
  note q-not-0 = this
  show ?thesis
  proof (cases  $ps' \in set (split-sign-changes (q \# r \# ps'') x_0)$ )
    case True
      show ?thesis by (rule IH(2), fact, simp add: q-not-0, fact)
  next
  case False
    with ps''-props and q-not-0 have  $ps' = [p, q]$  by simp
    hence [simp]:  $\forall p \in set\ ps'. poly\ p\ x_0 \neq 0$ 
      using q-not-0 p-not-0 by simp
    show ?thesis
    proof (rule eventually-mono)
      fix x assume  $\forall p \in set\ ps'. sgn (poly\ p\ x) = sgn (poly\ p\ x_0)$ 
      thus  $sign-changes\ ps'\ x = sign-changes\ ps'\ x_0$ 
        by (rule sign-changes-cong')
    next
      show eventually  $(\lambda x. \forall p \in set\ ps'. sgn (poly\ p\ x) = sgn (poly\ p\ x_0)) (at\ x_0)$ 
        by (force intro: eventually-ball-finite
            poly-neighbourhood-same-sign)
    qed
  qed
qed
qed
qed

```

lemma (in *quasi-sturm-seq*) *hd-nonzero-imp-sign-changes-const*:
assumes *poly* (*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 –
let $?pss = \text{split-sign-changes } ps \ x_0$
let $?f = \lambda pss \ x. \sum ps' \leftarrow pss. \text{sign-changes } ps' \ x$
{
fix *pss* **assume** $\bigwedge ps'. ps' \in \text{set } pss \implies$
eventually ($\lambda x. \text{sign-changes } ps' \ x = \text{sign-changes } ps' \ x_0$) (*at* x_0)
hence *eventually* ($\lambda x. ?f \ pss \ x = ?f \ pss \ x_0$) (*at* x_0)
proof (*induction pss*)
case (*Cons ps' pss*)
then show *?case*
apply (*rule eventually-mono*[*OF eventually-conj*])
apply (*auto simp add: Cons.prem*s)
done
qed *simp*
}
note *A = this*[*of ?pss*]
have *B: eventually* ($\lambda x. ?f \ ?pss \ x = ?f \ ?pss \ x_0$) (*at* x_0)
by (*rule A, rule hd-nonzero-imp-sign-changes-const-aux*[*OF assms*], *simp*)
note *C = split-sign-changes-correct-nbh*[*OF assms*]
note *D = split-sign-changes-correct*[*OF assms*]
note *E = eventually-conj*[*OF B C*]
show *?thesis* **by** (*rule eventually-mono*[*OF E*], *auto simp: D*)
qed

lemma (in *sturm-seq*) *p-nonzero-imp-sign-changes-const*:
poly p x₀ ≠ 0 \implies
eventually ($\lambda x. \text{sign-changes } ps \ x = \text{sign-changes } ps \ x_0$) (*at* x_0)
using *hd-nonzero-imp-sign-changes-const* **by** *simp*

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 *poly p x₀ = 0 p ≠ 0*
shows *eventually* ($\lambda x. \text{sign-changes } ps \ x =$
sign-changes } ps \ x₀ + (if x < x₀ then 1 else 0)) (*at* x_0)
proof –
from *ps-first-two* **obtain** $q \ ps'$ **where** [*simp*]: $ps = p \# q \# ps'$.
hence $ps!1 = q$ **by** *simp*
have *eventually* ($\lambda x. x \neq x_0$) (*at* x_0)
by (*simp add: eventually-at, rule exI*[*of - 1*], *simp*)
moreover from *p-squarefree* **and** *assms(1)* **have** *poly q x₀ ≠ 0* **by** *simp*
{
have *A: quasi-sturm-seq ps ..*
with *quasi-sturm-seq-Cons*[*of p q # ps'*]
interpret *quasi-sturm-seq q # ps'* **by** *simp*
from $\langle \text{poly } q \ x_0 \neq 0 \rangle$ **have** *eventually* ($\lambda x. \text{sign-changes } (q \# ps') \ x =$

```

      sign-changes (q#ps^) x0 (at x0)
    using hd-nonzero-imp-sign-changes-const[where x0=x0] by simp
  }
  moreover note poly-neighbourhood-without-roots[OF assms(2)] deriv[OF assms(1)]
  ultimately
    have A: eventually (λx. x ≠ x0 ∧ poly p x ≠ 0 ∧
      sgn (poly (p*ps!1) x) = (if x > x0 then 1 else -1) ∧
      sign-changes (q#ps^) x = sign-changes (q#ps^) x0 (at x0)
      by (simp only: ⟨ps!1 = q⟩, intro eventually-conj)
  show ?thesis
  proof (rule eventually-mono[OF A], clarify, goal-cases)
    case prems: (1 x)
    from zero-less-mult-pos have zero-less-mult-pos':
      ∧ a b. [(0::real) < a*b; 0 < b] ⇒ 0 < a
      by (subgoal-tac a*b = b*a, auto)
    from prems have poly q x ≠ 0 and q-sgn: sgn (poly q x) =
      (if x < x0 then -sgn (poly p x) else sgn (poly p x))
      by (auto simp add: sgn-real-def elim: linorder-neqE-linordered-idom
        dest: mult-neg-neg zero-less-mult-pos
        zero-less-mult-pos' split: if-split-asm)
    from sign-changes-distrib[OF ⟨poly q x ≠ 0⟩, of [p] ps^]
      have sign-changes ps x = sign-changes [p,q] x + sign-changes (q#ps^) x
      by simp
    also from q-sgn and ⟨poly p x ≠ 0⟩
      have sign-changes [p,q] x = (if x < x0 then 1 else 0)
      by (simp add: sign-changes-def sgn-zero-iff split: if-split-asm)
    also note prems(4)
    also from assms(1) have sign-changes (q#ps^) x0 = sign-changes ps x0
      by (simp add: sign-changes-def)
    finally show ?case by simp
  qed
qed

```

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 (*clarify*)

fix x **assume** $x > a \wedge x \leq b$

with *assms* **have** $\forall x'. x \leq x' \wedge x' \leq b \longrightarrow$

$\text{eventually } (\lambda \xi. f \xi = f x') \text{ (at } x')$ **by** *auto*

from *fun-eq-in-ivl*[*OF* $\langle x \leq b \rangle$ *this*] **show** $f x = f b$.

qed

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 –

have $\text{sign-changes } ps \ a - \text{int } (\text{sign-changes } ps \ b) =$
 $\text{card } \{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **using** $\langle a \leq b \rangle$

proof (*induction card* $\{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ *arbitrary: a b*
rule: less-induct)

case (*less a b*)

show *?case*

proof (*cases* $\exists x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0$)

case *False*

hence *no-roots*: $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} = \{\}$

(**is** *?roots=-*) **by** *auto*

hence *card-roots*: $\text{card } ?\text{roots} = (0::\text{int})$ **by** (*subst no-roots, simp*)

show *?thesis*

proof (*simp only: card-roots eq-iff-diff-eq-0[symmetric] of-nat-eq-iff,*
cases poly p a = 0)

case *False*

with *no-roots* **show** $\text{sign-changes } ps \ a = \text{sign-changes } ps \ b$

by (*force intro: fun-eq-in-ivl* $\langle a \leq b \rangle$

p-nonzero-imp-sign-changes-const)

next

case *True*

have $A: \forall x. a < x \wedge x \leq b \longrightarrow \text{sign-changes } ps \ x = \text{sign-changes } ps \ b$

apply (*rule count-roots-between-aux, fact, clarify*)

apply (*rule p-nonzero-imp-sign-changes-const*)

apply (*insert False, simp*)

done

have *eventually* $(\lambda x. x > a \longrightarrow$

$\text{sign-changes } ps \ x = \text{sign-changes } ps \ a)$ (*at a*)

apply (*rule eventually-mono* [*OF p-zero* [*OF* $\langle \text{poly } p \ a = 0 \rangle \langle p \neq$

$0 \rangle$]])

apply *force*

done

then obtain δ **where** *δ-props*:

$\delta > 0 \ \forall x. x > a \wedge x < a + \delta \longrightarrow$

$\text{sign-changes } ps \ x = \text{sign-changes } ps \ a$

by (*auto simp: eventually-at dist-real-def*)

show $\text{sign-changes } ps \ a = \text{sign-changes } ps \ b$

proof (*cases a = b*)

case *False*

define x **where** $x = \min (a + \delta / 2) \ b$

with *False* **have** $a < x \ x < a + \delta \ x \leq b$

using $\langle \delta > 0 \rangle \ \langle a \leq b \rangle$ **by** *simp-all*

from *δ-props* $\langle a < x \rangle \ \langle x < a + \delta \rangle$

have *sign-changes ps a = sign-changes ps x* **by** *simp*
also from $A \langle a < x \rangle \langle x \leq b \rangle$ **have** $\dots = \text{sign-changes ps } b$
by *blast*
finally show *?thesis* .
qed *simp*
qed

next

case *True*
from *poly-roots-finite[OF assms(1)]*
have *fin: finite {x. x > a ∧ x ≤ b ∧ poly p x = 0}*
by (*force intro: finite-subset*)
from *True* **have** $\{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\} \neq \{\}$ **by** *blast*
with *fin* **have** *card-greater-0:*
 $\text{card } \{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\} > 0$ **by** *fastforce*

define x_2 **where** $x_2 = \text{Min } \{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\}$
from *Min-in[OF fin]* **and** *True*
have $x_2\text{-props: } x_2 > a \ x_2 \leq b \ \text{poly } p \ x_2 = 0$
unfolding $x_2\text{-def}$ **by** *blast+*
from *Min-le[OF fin]* $x_2\text{-props}$
have $x_2\text{-le: } \bigwedge x'. \llbracket x' > a; x' \leq b; \text{poly } p \ x' = 0 \rrbracket \implies x_2 \leq x'$
unfolding $x_2\text{-def}$ **by** *simp*

have *left: {x. a < x ∧ x ≤ x₂ ∧ poly p x = 0} = {x_{2}}}*
using $x_2\text{-props } x_2\text{-le}$ **by** *force*
hence [*simp*]: $\text{card } \{x. a < x \wedge x \leq x_2 \wedge \text{poly } p \ x = 0\} = 1$ **by** *simp*

from *p-zero[OF ⟨poly p x₂ = 0⟩ ⟨p ≠ 0⟩,*
unfolded eventually-at dist-real-def] **guess** $\varepsilon \dots$
hence $\varepsilon\text{-props: } \varepsilon > 0$
 $\forall x. x \neq x_2 \wedge |x - x_2| < \varepsilon \implies$
 $\text{sign-changes ps } x = \text{sign-changes ps } x_2 +$
 $(\text{if } x < x_2 \text{ then } 1 \text{ else } 0)$ **by** *auto*
define x_1 **where** $x_1 = \max(x_2 - \varepsilon / 2) \ a$
have $|x_1 - x_2| < \varepsilon$ **using** $\langle \varepsilon > 0 \rangle$ $x_2\text{-props}$ **by** (*simp add: x₁-def*)
hence $\text{sign-changes ps } x_1 =$
 $(\text{if } x_1 < x_2 \text{ then } \text{sign-changes ps } x_2 + 1 \text{ else } \text{sign-changes ps } x_2)$
using $\varepsilon\text{-props}(2)$ **by** (*cases x₁ = x₂, auto*)
hence $\text{sign-changes ps } x_1 - \text{sign-changes ps } x_2 = 1$
unfolding $x_1\text{-def}$ **using** $x_2\text{-props } \langle \varepsilon > 0 \rangle$ **by** *simp*

also have $x_2 \notin \{x. a < x \wedge x \leq x_1 \wedge \text{poly } p \ x = 0\}$
unfolding $x_1\text{-def}$ **using** $\langle \varepsilon > 0 \rangle$ **by** *force*
with *left* **have** $\{x. a < x \wedge x \leq x_1 \wedge \text{poly } p \ x = 0\} = \{\}$ **by** *force*
with *less(1)[of a x₁]* **have** $\text{sign-changes ps } x_1 = \text{sign-changes ps } a$
unfolding $x_1\text{-def } \langle \varepsilon > 0 \rangle$ **by** (*force simp: card-greater-0*)

finally have *signs-left:*

sign-changes ps a - int (sign-changes ps x2) = 1 by simp

have $\{x. x > a \wedge x \leq b \wedge \text{poly } p \ x = 0\} =$
 $\{x. a < x \wedge x \leq x_2 \wedge \text{poly } p \ x = 0\} \cup$
 $\{x. x_2 < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **using** *x2-props* **by** *auto*

also note *left*

finally have *A*: $\text{card } \{x. x_2 < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} + 1 =$
 $\text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **using** *fin* **by** *simp*

hence $\text{card } \{x. x_2 < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} <$
 $\text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **by** *simp*

from *less(1)[OF this x2-props(2)]* **and** *A*

have *signs-right*: $\text{sign-changes ps } x_2 - \text{int (sign-changes ps } b) + 1 =$
 $\text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **by** *simp*

from *signs-left* **and** *signs-right* **show** *?thesis* **by** *simp*

qed

qed

thus *?thesis* **by** *simp*

qed

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 –

have $p \in \text{set ps}$ **using** *hd-in-set[OF ps-not-Nil]* **by** *simp*

have *finite (set ps)* **by** *simp*

from *polys-inf-sign-thresholds[OF this]* **guess** *l u* .

note *lu-props = this*

let $?u = \max a \ u$

{fix *x* **assume** $\text{poly } p \ x = 0$ **hence** $x \leq ?u$

using *lu-props(3)[OF <p ∈ set ps>, of x]* $\langle p \neq 0 \rangle$

by (*cases* $u \leq x$, *auto simp: sgn-zero-iff*)

} note $[\text{simp}] = \text{this}$

from *lu-props*

have $\text{map } (\lambda p. \text{sgn } (\text{poly } p \ ?u)) \ ps = \text{map } \text{poly-inf ps}$ **by** *simp*

hence $\text{sign-changes ps } a - \text{sign-changes-inf ps} =$

$\text{sign-changes ps } a - \text{sign-changes ps } ?u$

by (*simp-all only: sign-changes-def sign-changes-inf-def*)

also from *count-roots-between[OF assms]* *lu-props*

have $\dots = \text{card } \{x. a < x \wedge x \leq ?u \wedge \text{poly } p \ x = 0\}$ **by** *simp*

also have $\{x. a < x \wedge x \leq ?u \wedge \text{poly } p \ x = 0\} = \{x. a < x \wedge \text{poly } p \ x = 0\}$

using *lu-props* **by** *auto*

finally show *?thesis* .

qed

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 –

have $p \in \text{set } ps$ **using** *hd-in-set[OF ps-not-Nil]* **by** *simp*

have *finite (set ps)* **by** *simp*

from *polys-inf-sign-thresholds[OF this]* **guess** $l \ u$.

note $lu\text{-props} = \text{this}$

let $?l = \min \ a \ l$

{fix x **assume** $\text{poly } p \ x = 0$ **hence** $x > ?l$

using $lu\text{-props}(4)[OF \langle p \in \text{set } ps \rangle, \text{of } x] \langle p \neq 0 \rangle$

by (cases $l < x$, *auto simp: sgn-zero-iff*)

} note $[simp] = \text{this}$

from $lu\text{-props}$

have $\text{map } (\lambda p. \text{sgn } (\text{poly } p \ ?l)) \ ps = \text{map } \text{poly-neg-inf } ps$ **by** *simp*

hence $\text{sign-changes-neg-inf } ps - \text{sign-changes } ps \ a =$

$\text{sign-changes } ps \ ?l - \text{sign-changes } ps \ a$

by (*simp-all only: sign-changes-def sign-changes-neg-inf-def*)

also from *count-roots-between[OF assms]* $lu\text{-props}$

have $\dots = \text{card } \{x. ?l < x \wedge x \leq a \wedge \text{poly } p \ x = 0\}$ **by** *simp*

also have $\{x. ?l < x \wedge x \leq a \wedge \text{poly } p \ x = 0\} = \{x. a \geq x \wedge \text{poly } p \ x = 0\}$

using $lu\text{-props}$ **by** *auto*

finally show *?thesis* .

qed

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 –

have *finite (set ps)* **by** *simp*

from *polys-inf-sign-thresholds[OF this]* **guess** $l \ u$.

note $lu\text{-props} = \text{this}$

from $lu\text{-props}$

have $\text{map } (\lambda p. \text{sgn } (\text{poly } p \ l)) \ ps = \text{map } \text{poly-neg-inf } ps$

$\text{map } (\lambda p. \text{sgn } (\text{poly } p \ u)) \ ps = \text{map } \text{poly-inf } ps$ **by** *simp-all*

hence $\text{sign-changes-neg-inf } ps - \text{sign-changes-inf } ps =$

$\text{sign-changes } ps \ l - \text{sign-changes } ps \ u$

by (*simp-all only: sign-changes-def sign-changes-inf-def*

sign-changes-neg-inf-def)

also from *count-roots-between[OF assms]* $lu\text{-props}$

have $\dots = \text{card } \{x. l < x \wedge x \leq u \wedge \text{poly } p \ x = 0\}$ **by** *simp*

also have $\{x. l < x \wedge x \leq u \wedge \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0\}$

using $lu\text{-props}$ *assms* **by** *simp*

finally show *?thesis* .
qed

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.

function *sturm-aux* **where**
sturm-aux ($p :: \text{real poly}$) $q =$
(if degree $q = 0$ *then* $[p, q]$ *else* $p \# \text{sturm-aux } q \text{ } (- (p \bmod q))$
by (*pat-completeness, simp-all*)
termination by (*relation measure (degree \circ snd),*
simp-all add: o-def degree-mod-less')

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

Next, we show some simple facts about this construction:

lemma *sturm-0[simp]*: $\text{sturm } 0 = [0, 0]$
by (*unfold sturm-def, subst sturm-aux.simps, simp*)

lemma *[simp]*: $\text{sturm-aux } p \ q = [] \iff \text{False}$
by (*induction p q rule: sturm-aux.induct, subst sturm-aux.simps, auto*)

lemma *sturm-neq-Nil[simp]*: $\text{sturm } p \neq []$ **unfolding** *sturm-def* **by** *simp*

lemma *[simp]*: $\text{hd } (\text{sturm } p) = p$
unfolding *sturm-def* **by** (*subst sturm-aux.simps, simp*)

lemma *[simp]*: $p \in \text{set } (\text{sturm } p)$
using *hd-in-set[OF sturm-neq-Nil]* **by** *simp*

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

proof–

{**fix** q **have** $\text{length } (\text{sturm-aux } p \ q) \geq 2$
by (*induction p q rule: sturm-aux.induct, subst sturm-aux.simps, auto*)

}

thus *?thesis* **unfolding** *sturm-def* .

qed

```

lemma [simp]: degree (last (sturm p)) = 0
proof -
  {fix q have degree (last (sturm-aux p q)) = 0
    by (induction p q rule: sturm-aux.induct, subst sturm-aux.simps, simp)
  }
  thus ?thesis unfolding sturm-def .
qed

```

```

lemma [simp]: sturm-aux p q ! 0 = p
  by (subst sturm-aux.simps, simp)
lemma [simp]: sturm-aux p q ! Suc 0 = q
  by (subst sturm-aux.simps, simp)

```

```

lemma [simp]: sturm p ! 0 = p
  unfolding sturm-def by simp
lemma [simp]: sturm p ! Suc 0 = pderiv p
  unfolding sturm-def by simp

```

```

lemma sturm-indices:
  assumes  $i < \text{length } (\text{sturm } p) - 2$ 
  shows  $\text{sturm } p!(i+2) = -(\text{sturm } p!i \text{ mod } \text{sturm } p!(i+1))$ 
proof -
  {fix ps q
  have  $\llbracket ps = \text{sturm-aux } p \ q; i < \text{length } ps - 2 \rrbracket$ 
     $\implies ps!(i+2) = -(ps!i \text{ mod } ps!(i+1))$ 
  proof (induction p q arbitrary: ps i rule: sturm-aux.induct)
  case (1 p q)
  show ?case
  proof (cases i = 0)
  case False
  then obtain  $i'$  where [simp]:  $i = \text{Suc } i'$  by (cases i, simp-all)
  hence  $\text{length } ps \geq 4$  using 1 by simp
  with 1(2) have deg:  $\text{degree } q \neq 0$ 
    by (subst (asm) sturm-aux.simps, simp split: if-split-asm)
  with 1(2) obtain  $ps'$  where [simp]:  $ps = p \# ps'$ 
    by (subst (asm) sturm-aux.simps, simp)
  with 1(2) deg have  $ps': ps' = \text{sturm-aux } q \ (-p \text{ mod } q)$ 
    by (subst (asm) sturm-aux.simps, simp)
  from  $\langle \text{length } ps \geq 4 \rangle$  and  $\langle ps = p \# ps' \rangle$  1(3) False
  have  $i - 1 < \text{length } ps' - 2$  by simp
  from 1(1)[OF deg ps' this]
  show ?thesis by simp
  next
  case True
  with 1(3) have  $\text{length } ps \geq 3$  by simp
  with 1(2) have  $\text{degree } q \neq 0$ 
    by (subst (asm) sturm-aux.simps, simp split: if-split-asm)
  }
  }

```

```

with 1(2) have [simp]: sturm-aux p q ! Suc (Suc 0) = -(p mod q)
  by (subst sturm-aux.simps, simp)
from True have ps!i = p ps!(i+1) = q ps!(i+2) = -(p mod q)
  by (simp-all add: 1(2))
thus ?thesis by simp
qed
qed}
from this[OF sturm-def assms] show ?thesis .
qed

```

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*: $r \in \text{set } (\text{sturm-aux } p \ q) \implies \text{gcd } p \ q \ \text{dvd } r$

proof (*induction p q rule: sturm-aux.induct*)

case (1 p q)

show ?case

proof (*cases r = p*)

case False

with 1(2) **have** $r: r \in \text{set } (\text{sturm-aux } q \ (- (p \ \text{mod } q)))$

by (subst (asm) sturm-aux.simps, simp split: if-split-asm,
subst sturm-aux.simps, simp)

show ?thesis

proof (*cases degree q = 0*)

case False

hence $q \neq 0$ **by** force

with 1(1) [OF False r] **show** ?thesis

by (simp add: gcd-mod-right ac-simps)

next

case True

with 1(2) **and** $\langle r \neq p \rangle$ **have** $r = q$

by (subst (asm) sturm-aux.simps, simp)

thus ?thesis **by** simp

qed

qed simp

qed

lemma *sturm-gcd*: $r \in \text{set } (\text{sturm } p) \implies \text{gcd } p \ (\text{pderiv } p) \ \text{dvd } r$

unfolding sturm-def **by** (rule sturm-aux-gcd)

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 $i < \text{length } (\text{sturm } (p :: \text{real poly})) - 1$

assumes $\text{poly } (\text{sturm } p \ ! \ i) \ x = 0$

and $\text{poly } (\text{sturm } p \ ! \ (i + 1)) \ x = 0$

shows $\forall j \leq i+1. \text{poly } (\text{sturm } p ! j) x = 0$
using *assms(2)*
proof (*intro sturm-adjacent-root-aux[OF assms(1,2,3)], goal-cases*)
case *prems: (1 i x)*
let $?p = \text{sturm } p ! i$
let $?q = \text{sturm } p ! (i + 1)$
let $?r = \text{sturm } p ! (i + 2)$
from *sturm-indices[OF prems(2)]* **have** $?p = ?p \text{ div } ?q * ?q - ?r$
by (*simp add: div-mult-mod-eq*)
hence $\text{poly } ?p x = \text{poly } (?p \text{ div } ?q * ?q - ?r) x$ **by** *simp*
hence $\text{poly } ?p x = -\text{poly } ?r x$ **using** *prems(3)* **by** *simp*
thus $?case$ **by** (*simp add: sgn-minus*)
qed

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 $i < \text{length } (\text{sturm } (p :: \text{real poly})) - 1$
 $\text{poly } (\text{sturm } p ! i) x = 0 \text{ poly } (\text{sturm } p ! (i + 1)) x = 0$
shows $\neg \text{rsquarefree } p$
proof –
from *sturm-adjacent-root-propagate-left[OF assms]*
have $\text{poly } p x = 0 \text{ poly } (\text{pderiv } p) x = 0$ **by** *auto*
thus $?thesis$ **by** (*auto simp: rsquarefree-roots*)
qed

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 $(p :: \text{real poly}) \neq 0 \ q \neq 0$
assumes *q-pderiv*:
 $\text{eventually } (\lambda x. \text{sgn } (\text{poly } q x) = \text{sgn } (\text{poly } (\text{pderiv } p) x)) \text{ (at } x_0)$
assumes *p-0*: $\text{poly } p (x_0 :: \text{real}) = 0$
shows $\text{eventually } (\lambda x. \text{sgn } (\text{poly } (p*q) x) = (\text{if } x > x_0 \text{ then } 1 \text{ else } -1)) \text{ (at } x_0)$
proof –
have *A*: $\text{eventually } (\lambda x. \text{poly } p x \neq 0 \wedge \text{poly } q x \neq 0 \wedge$
 $\text{sgn } (\text{poly } q x) = \text{sgn } (\text{poly } (\text{pderiv } p) x)) \text{ (at } x_0)$
using $\langle p \neq 0 \rangle \ \langle q \neq 0 \rangle$
by (*intro poly-neighbourhood-same-sign q-pderiv*
 $\text{poly-neighbourhood-without-roots eventually-conj}$)
then obtain ε **where** ε -*props*: $\varepsilon > 0 \ \forall x. x \neq x_0 \wedge |x - x_0| < \varepsilon \longrightarrow$
 $\text{poly } p x \neq 0 \wedge \text{poly } q x \neq 0 \wedge \text{sgn } (\text{poly } (\text{pderiv } p) x) = \text{sgn } (\text{poly } q x)$
by (*auto simp: eventually-at dist-real-def*)
have *sqr-pos*: $\bigwedge x :: \text{real}. x \neq 0 \implies \text{sgn } x * \text{sgn } x = 1$
by (*auto simp: sgn-real-def*)
show $?thesis$

proof (*simp only: eventually-at dist-real-def, rule exI[of - ε],
intro conjI, fact ⟨ε > 0⟩, clarify*)
fix x **assume** $x \neq x_0 \mid x - x_0 \mid < \varepsilon$
with ε -props **have** [simp]: $\text{poly } p \ x \neq 0 \ \text{poly } q \ x \neq 0$
 $\text{sgn}(\text{poly}(pderiv\ p)\ x) = \text{sgn}(\text{poly } q \ x)$ **by** *auto*
show $\text{sgn}(\text{poly}(p * q)\ x) = (\text{if } x > x_0 \text{ then } 1 \text{ else } -1)$
proof (*cases* $x \geq x_0$)
case *True*
with $\langle x \neq x_0 \rangle$ **have** $x > x_0$ **by** *simp*
from *poly-MVT[OF this, of p]* **guess** $\xi \ ..$
note ξ -props = *this*
with $\langle |x - x_0| < \varepsilon \rangle \langle \text{poly } p \ x_0 = 0 \rangle \langle x > x_0 \rangle \varepsilon$ -props
have $|\xi - x_0| < \varepsilon \ \text{sgn}(\text{poly } p \ x) = \text{sgn}(x - x_0) * \text{sgn}(\text{poly } q \ \xi)$
by (*auto simp add: q-pderiv sgn-mult*)
moreover from ξ -props ε -props $\langle |x - x_0| < \varepsilon \rangle$
have $\forall t. \xi \leq t \wedge t \leq x \longrightarrow \text{poly } q \ t \neq 0$ **by** *auto*
hence $\text{sgn}(\text{poly } q \ \xi) = \text{sgn}(\text{poly } q \ x)$ **using** ξ -props ε -props
by (*intro no-roots-inbetween-imp-same-sign, simp-all*)
ultimately show ?thesis **using** *True* $\langle x \neq x_0 \rangle \varepsilon$ -props ξ -props
by (*auto simp: sgn-mult sqr-pos*)
next
case *False*
hence $x < x_0$ **by** *simp*
hence $\text{sgn}(\text{sgn}(x - x_0)) = -1$ **by** *simp*
from *poly-MVT[OF ⟨x < x_0⟩, of p]* **guess** $\xi \ ..$
note ξ -props = *this*
with $\langle |x - x_0| < \varepsilon \rangle \langle \text{poly } p \ x_0 = 0 \rangle \langle x < x_0 \rangle \varepsilon$ -props
have $|\xi - x_0| < \varepsilon \ \text{poly } p \ x = (x - x_0) * \text{poly}(pderiv\ p)\ \xi$
 $\text{poly } p \ \xi \neq 0$ **by** (*auto simp: field-simps*)
hence $\text{sgn}(\text{poly } p \ x) = \text{sgn}(x - x_0) * \text{sgn}(\text{poly } q \ \xi)$
using ε -props ξ -props **by** (*auto simp: q-pderiv sgn-mult*)
moreover from ξ -props ε -props $\langle |x - x_0| < \varepsilon \rangle$
have $\forall t. x \leq t \wedge t \leq \xi \longrightarrow \text{poly } q \ t \neq 0$ **by** *auto*
hence $\text{sgn}(\text{poly } q \ \xi) = \text{sgn}(\text{poly } q \ x)$ **using** ξ -props ε -props
by (*rule-tac sym, intro no-roots-inbetween-imp-same-sign, simp-all*)
ultimately show ?thesis **using** *False* $\langle x \neq x_0 \rangle$
by (*auto simp: sgn-mult sqr-pos*)
qed
qed
qed

lemma *sturm-firsttwo-signs*:

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

assumes *squarefree: rsquarefree p*

assumes *p-0: poly p (x_0::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 –

from *assms* **have** [simp]: $p \neq 0$ **by** (*auto simp add: rsquarefree-roots*)

with *squarefree p-0* **have** [*simp*]: $pderiv\ p \neq 0$
by (*auto simp add: rsquarefree-roots*)
from *assms* **show** *?thesis*
by (*intro sturm-firsttwo-signs-aux,*
simp-all add: rsquarefree-roots)
qed

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–

from *sturm-indices[OF i-in-range]*
have $sturm\ p\ !\ (i+2) = - (sturm\ p\ !\ i\ mod\ sturm\ p\ !\ (i+1))$
(is ?r = - (?p mod ?q)) .
hence $-?r = ?p\ mod\ ?q$ **by** *simp*
with *div-mult-mod-eq[of ?p ?q]* **have** $?p\ div\ ?q * ?q - ?r = ?p$ **by** *simp*
hence $poly\ (?p\ div\ ?q)\ x * poly\ ?q\ x - poly\ ?r\ x = poly\ ?p\ x$
by (*metis poly-diff poly-mult*)
with *q-0* **have** $r-x: poly\ ?r\ x = -poly\ ?p\ x$ **by** *simp*
moreover **have** $sqr-pos: \bigwedge x::real. x \neq 0 \implies x * x > 0$ **apply** (*case-tac x ≥ 0*)
by (*simp-all add: mult-neg-neg*)
from *sturm-adjacent-root-not-squarefree[of i p]* *assms r-x*
have $poly\ ?p\ x * poly\ ?p\ x > 0$ **by** (*force intro: sqr-pos*)
ultimately show $poly\ ?r\ x * poly\ ?p\ x < 0$ **by** *simp*
qed

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

show $sturm\ p \neq []$ **by** *simp*
show $hd\ (sturm\ p) = p$ **by** *simp*
show $length\ (sturm\ p) \geq 2$ **by** *simp*
from *assms* **show** $\bigwedge x. \neg(poly\ p\ x = 0 \wedge poly\ (sturm\ p\ !\ 1)\ x = 0)$
by (*simp add: rsquarefree-roots*)

next

fix $x :: real$ **and** $y :: real$
have $degree\ (last\ (sturm\ p)) = 0$ **by** *simp*
then obtain c **where** $last\ (sturm\ p) = [:c:]$
by (*cases last (sturm p), simp split: if-split-asm*)
thus $\bigwedge x\ y. sgn\ (poly\ (last\ (sturm\ p))\ x) =$

```

      sgn (poly (last (sturm p)) y) by simp
next
from sturm-firsttwo-signs[OF assms]
  show  $\bigwedge x_0. \text{poly } p \ x_0 = 0 \implies$ 
    eventually ( $\lambda x. \text{sgn } (\text{poly } (p * \text{sturm } p \ ! \ 1) \ x) =$ 
      (if  $x > x_0$  then 1 else -1)) (at  $x_0$ ) by simp
next
from sturm-signs[OF assms]
  show  $\bigwedge i \ x. \llbracket i < \text{length } (\text{sturm } p) - 2; \text{poly } (\text{sturm } p \ ! \ (i + 1)) \ x = 0 \rrbracket$ 
     $\implies \text{poly } (\text{sturm } p \ ! \ (i + 2)) \ x * \text{poly } (\text{sturm } p \ ! \ i) \ x < 0$  by simp
qed

```

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

lemma *sturm-squarefree-not-Nil*[simp]: $\text{sturm-squarefree } p \neq []$
by (simp add: sturm-squarefree-def)

lemma *sturm-seq-sturm-squarefree*:
assumes [simp]: $p \neq 0$
defines [simp]: $p' \equiv p \ \text{div } \text{gcd } p \ (p\text{deriv } p)$
shows *sturm-seq* (sturm-squarefree p) p'
proof
have *rsquarefree* p'
proof (subst *rsquarefree-roots*, clarify)
fix x **assume** $\text{poly } p' \ x = 0$ $\text{poly } (p\text{deriv } p') \ x = 0$
hence $[-x, 1:] \ \text{dvd } \text{gcd } p' \ (p\text{deriv } p')$ **by** (simp add: poly-eq-0-iff-dvd)
also from *poly-div-gcd-squarefree(1)*[OF *assms(1)*]
have $\text{gcd } p' \ (p\text{deriv } p') = 1$ **by** simp
finally show *False* **by** (simp add: poly-eq-0-iff-dvd[symmetric])
qed

from *sturm-seq-sturm*[OF $\langle \text{rsquarefree } p' \rangle$]
interpret *sturm-seq*: *sturm-seq* sturm-squarefree $p \ p'$
by (simp add: sturm-squarefree-def)

show $\bigwedge x \ y. \text{sgn } (\text{poly } (\text{last } (\text{sturm-squarefree } p)) \ x) =$
 $\text{sgn } (\text{poly } (\text{last } (\text{sturm-squarefree } p)) \ y)$ **by** simp
show *sturm-squarefree* $p \neq []$ **by** simp
show $\text{hd } (\text{sturm-squarefree } p) = p'$ **by** (simp add: sturm-squarefree-def)
show $\text{length } (\text{sturm-squarefree } p) \geq 2$ **by** simp

```

have [simp]: sturm-squarefree p ! 0 = p'
           sturm-squarefree p ! Suc 0 = pderiv p'
by (simp-all add: sturm-squarefree-def)

from ⟨rsquarefree p⟩
show  $\bigwedge x. \neg (poly\ p'\ x = 0 \wedge poly\ (sturm-squarefree\ p\ !\ 1)\ x = 0)$ 
by (simp add: rsquarefree-roots)

from sturm-seq.signs show  $\bigwedge i\ x. \llbracket i < length\ (sturm-squarefree\ p) - 2;$ 
            $poly\ (sturm-squarefree\ p\ !\ (i + 1))\ x = 0 \rrbracket$ 
            $\implies poly\ (sturm-squarefree\ p\ !\ (i + 2))\ x *$ 
            $poly\ (sturm-squarefree\ p\ !\ i)\ x < 0 .$ 

from sturm-seq.deriv show  $\bigwedge x_0. poly\ p'\ x_0 = 0 \implies$ 
           eventually  $(\lambda x. sgn\ (poly\ (p' * sturm-squarefree\ p\ !\ 1)\ x) =$ 
            $(if\ x > x_0\ then\ 1\ else\ -1))\ (at\ x_0) .$ 
qed

```

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 $map\ (\lambda p'. p' \div d)\ (sturm\ p))$

This construction also has all the desired properties:

lemma *sturm-squarefree'-adjacent-root-propagate-left*:
assumes $p \neq 0$
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 $\forall j \leq i+1. poly\ (sturm-squarefree'\ p\ !\ j)\ x = 0$
proof (*intro sturm-adjacent-root-aux[OF assms(2,3,4)], goal-cases*)
case *prems*: $(1\ i\ x)$
define q **where** $q = sturm\ p\ !\ i$
define r **where** $r = sturm\ p\ !\ (Suc\ i)$
define s **where** $s = sturm\ p\ !\ (Suc\ (Suc\ i))$
define d **where** $d = gcd\ p\ (pderiv\ p)$
define $q'\ r'\ s'$ **where** $q' = q \div d$ **and** $r' = r \div d$ **and** $s' = s \div d$
from $\langle p \neq 0 \rangle$ **have** $d \neq 0$ **unfolding** *d-def* **by** *simp*
from *prems*(1) **have** *i-in-range*: $i < length\ (sturm\ p) - 2$
unfolding *sturm-squarefree'-def* *Let-def* **by** *simp*
have [simp]: $d \ dvd\ q\ d\ dvd\ r\ d\ dvd\ s$ **unfolding** *q-def* *r-def* *s-def* *d-def*
using *i-in-range* **by** (*auto intro: sturm-gcd*)

hence *qrs-simps*: $q = q' * d r = r' * d s = s' * d$
unfolding *q'-def r'-def s'-def* **by** (*simp-all*)
with *prems(2) i-in-range* **have** $r'-0$: $\text{poly } r' x = 0$
unfolding *r'-def r-def d-def sturm-squarefree'-def Let-def* **by** *simp*
hence $r-0$: $\text{poly } r x = 0$ **by** (*simp add: ⟨r = r' * d⟩*)
from *sturm-indices[OF i-in-range]* **have** $q = q \text{ div } r * r - s$
unfolding *q-def r-def s-def* **by** (*simp add: div-mult-mod-eq*)
hence $q' = (q \text{ div } r * r - s) \text{ div } d$ **by** (*simp add: q'-def*)
also have $\dots = (q \text{ div } r * r) \text{ div } d - s'$
by (*simp add: s'-def poly-div-diff-left*)
also have $\dots = q \text{ div } r * r' - s'$
using *dvd-div-mult[OF ⟨d dvd r⟩, of q div r]*
by (*simp add: algebra-simps r'-def*)
also have $q \text{ div } r = q' \text{ div } r'$ **by** (*simp add: qrs-simps ⟨d ≠ 0⟩*)
finally have $\text{poly } q' x = \text{poly } (q' \text{ div } r' * r' - s') x$ **by** *simp*
also from $r'-0$ **have** $\dots = -\text{poly } s' x$ **by** *simp*
finally have $\text{poly } s' x = -\text{poly } q' x$ **by** *simp*
thus *?case using i-in-range*
unfolding *q'-def s'-def q-def s-def sturm-squarefree'-def Let-def*
by (*simp add: d-def sgn-minus*)

qed

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*

proof –

define d **where** $d = \text{gcd } p (\text{pderiv } p)$
from *sturm-squarefree'-adjacent-root-propagate-left[OF assms]*
have $\text{poly } (\text{sturm-squarefree}' p ! 0) x = 0$
 $\text{poly } (\text{sturm-squarefree}' p ! 1) x = 0$ **by** *auto*
hence $\text{poly } (p \text{ div } d) x = 0$ $\text{poly } (\text{pderiv } p \text{ div } d) x = 0$
using *assms(2)*
unfolding *sturm-squarefree'-def Let-def d-def* **by** *auto*
moreover from *div-gcd-coprime assms(1)*
have *coprime (p div d) (pderiv p div d)* **unfolding** *d-def* **by** *auto*
ultimately show *False using coprime-imp-no-common-roots* **by** *auto*

qed

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** $\text{poly } ?r x * \text{poly } ?p x < 0$)

proof –

```

define  $d$  where  $d = \text{gcd } p \text{ (pderiv } p)$ 
with  $\langle p \neq 0 \rangle$  have [simp]:  $d \neq 0$  by simp
from poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ]
  coprime-imp-no-common-roots
  have rsquarefree: rsquarefree ( $p \text{ div } d$ )
  by (auto simp: rsquarefree-roots d-def)

from i-in-range have i-in-range':  $i < \text{length } (\text{sturm } p) - 2$ 
  unfolding sturm-squarefree'-def by simp
hence  $d \text{ dvd } (\text{sturm } p ! i)$  (is  $d \text{ dvd } ?p'$ )
   $d \text{ dvd } (\text{sturm } p ! (\text{Suc } i))$  (is  $d \text{ dvd } ?q'$ )
   $d \text{ dvd } (\text{sturm } p ! (\text{Suc } (\text{Suc } i)))$  (is  $d \text{ dvd } ?r'$ )
  unfolding d-def by (auto intro: sturm-gcd)
hence pqr-simps:  $?p' = ?p * d \ ?q' = ?q * d \ ?r' = ?r * d$ 
  unfolding sturm-squarefree'-def Let-def d-def using i-in-range'
  by (auto simp: dvd-div-mult-self)
with q-0 have q'-0:  $\text{poly } ?q' x = 0$  by simp
from sturm-indices[OF i-in-range']
  have  $\text{sturm } p ! (i+2) = - (\text{sturm } p ! i \text{ mod } \text{sturm } p ! (i+1))$  .
hence  $-?r' = ?p' \text{ mod } ?q'$  by simp
with div-mult-mod-eq[of  $?p' \ ?q'$ ] have  $?p' \text{ div } ?q' * ?q' - ?r' = ?p'$  by simp
hence  $d * (?p \text{ div } ?q * ?q - ?r) = d * ?p$  by (simp add: pqr-simps algebra-simps)
hence  $?p \text{ div } ?q * ?q - ?r = ?p$  by simp
hence  $\text{poly } (?p \text{ div } ?q) x * \text{poly } ?q x - \text{poly } ?r x = \text{poly } ?p x$ 
  by (metis poly-diff poly-mult)
with q-0 have r-x:  $\text{poly } ?r x = -\text{poly } ?p x$  by simp

from sturm-squarefree'-adjacent-roots[OF  $\langle p \neq 0 \rangle$ ] i-in-range q-0
  have  $\text{poly } ?p x \neq 0$  by force
moreover have sqr-pos:  $\bigwedge x :: \text{real}. x \neq 0 \implies x * x > 0$  apply (case-tac  $x \geq 0$ )
  by (simp-all add: mult-neg-neg)
ultimately show ?thesis using r-x by simp
qed

```

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 sturm-seq ( $\text{sturm-squarefree}' p$ ) ( $p \text{ div } d$ )
    (is sturm-seq  $?ps' \ ?p'$ )
proof
  show  $?ps' \neq []$  hd  $?ps' = ?p' \ 2 \leq \text{length } ?ps'$ 
    by (simp-all add: sturm-squarefree'-def d-def hd-map)

from assms have  $d \neq 0$  by simp
  {
    have  $d \text{ dvd last } (\text{sturm } p)$  unfolding d-def
      by (rule sturm-gcd, simp)
  }

```

```

hence *:  $last (sturm p) = last ?ps' * d$ 
  by (simp add: sturm-squarefree'-def last-map d-def dvd-div-mult-self)
then have  $last ?ps' dvd last (sturm p)$  by simp
with *  $dvd-imp-degree-le[OF this]$  have  $degree (last ?ps') \leq degree (last (sturm p))$ 
  using  $\langle d \neq 0 \rangle$  by (cases last ?ps' = 0) auto
hence  $degree (last ?ps') = 0$  by simp
then obtain  $c$  where  $last ?ps' = [:c]$ 
  by (cases last ?ps', simp split: if-split-asm)
thus  $\bigwedge x y. sgn (poly (last ?ps') x) = sgn (poly (last ?ps') y)$  by simp
}

have squarefree:  $rsquarefree ?p'$  using  $\langle p \neq 0 \rangle$ 
  by (subst rsquarefree-roots, unfold d-def,
    intro allI coprime-imp-no-common-roots poly-div-gcd-squarefree)
have [simp]:  $sturm-squarefree' p ! Suc 0 = pderiv p div d$ 
  unfolding sturm-squarefree'-def Let-def sturm-def d-def
  by (subst sturm-aux.simps, simp)
have coprime:  $coprime ?p' (pderiv p div d)$ 
  unfolding d-def using div-gcd-coprime  $\langle p \neq 0 \rangle$  by blast
thus squarefree':
   $\bigwedge x. \neg (poly (p div d) x = 0 \wedge poly (sturm-squarefree' p ! 1) x = 0)$ 
  using coprime-imp-no-common-roots by simp

from sturm-squarefree'-signs[OF  $\langle p \neq 0 \rangle$ ]
  show  $\bigwedge i x. \llbracket i < length ?ps' - 2; poly (?ps' ! (i + 1)) x = 0 \rrbracket$ 
     $\implies poly (?ps' ! (i + 2)) x * poly (?ps' ! i) x < 0$  .

have [simp]:  $?p' \neq 0$  using squarefree by (simp add: rsquarefree-def)
have A:  $?p' = ?ps' ! 0 pderiv p div d = ?ps' ! 1$ 
  by (simp-all add: sturm-squarefree'-def Let-def d-def sturm-def,
    subst sturm-aux.simps, simp)
have [simp]:  $?ps' ! 0 \neq 0$  using squarefree
  by (auto simp: A rsquarefree-def)

fix  $x_0 :: real$ 
assume  $poly ?p' x_0 = 0$ 
hence  $poly p x_0 = 0$  using poly-div-gcd-squarefree(2)[OF  $\langle p \neq 0 \rangle$ ]
  unfolding d-def by simp
hence  $pderiv p \neq 0$  using  $\langle p \neq 0 \rangle$  by (auto dest: pderiv-iszero)
with  $\langle p \neq 0 \rangle \langle poly p x_0 = 0 \rangle$ 
  have A:  $eventually (\lambda x. sgn (poly (p * pderiv p) x) =$ 
     $(if x_0 < x then 1 else -1)) (at x_0)$ 
  by (intro sturm-firsttwo-signs-aux, simp-all)
note  $ev = eventually-conj[OF A poly-neighbourhood-without-roots[OF  $\langle d \neq 0 \rangle$ ]]$ 

show  $eventually (\lambda x. sgn (poly (p div d * sturm-squarefree' p ! 1) x) =$ 
   $(if x_0 < x then 1 else -1)) (at x_0)$ 
proof (rule eventually-mono[OF ev], goal-cases)

```

```

have [intro]:
   $\wedge a (b::real). b \neq 0 \implies a < 0 \implies a / (b * b) < 0$ 
   $\wedge a (b::real). b \neq 0 \implies a > 0 \implies a / (b * b) > 0$ 
  by ((case-tac b > 0,
        auto simp: mult-neg-neg field-simps) [])+
case prems: (1 x)
hence [simp]: poly d x * poly d x > 0
  by (cases poly d x > 0, auto simp: mult-neg-neg)
from poly-div-gcd-squarefree-aux(2)[OF ‹pderiv p ≠ 0›]
  have poly (p div d) x = 0  $\longleftrightarrow$  poly p x = 0 by (simp add: d-def)
moreover have d dvd p d dvd pderiv p unfolding d-def by simp-all
ultimately show ?case using prems
  by (auto simp: sgn-real-def poly-div not-less[symmetric]
        zero-less-divide-iff split: if-split-asm)

```

qed

qed

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-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 ≠ (0::real)
shows length (remdups-adj (filter (λx. x ≠ 0) (map ((* d ∘ f) xs))) =
        length (remdups-adj (filter (λx. x ≠ 0) (map f xs)))

```

proof –

```

from assms have inj: inj ((* d) by (auto intro: injI)
from assms have [simp]: filter (λx. ((* d ∘ f) x ≠ 0) = filter (λx. f x ≠ 0)
        filter ((λx. x ≠ 0) ∘ f) = filter (λx. f x ≠ 0)
  by (simp-all add: o-def)
have filter (λx. x ≠ 0) (map ((* d ∘ f) xs) =
        map ((* d ∘ f) (filter (λx. ((* d ∘ f) x ≠ 0) xs)
  by (simp add: filter-map o-def)
thus ?thesis using remdups-adj-map-injective[OF inj] assms
  by (simp add: filter-map map-map[symmetric] del: map-map)

```

qed

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

```

fixes p :: real poly
defines ps ≡ sturm p and ps' ≡ sturm-squarefree' p
shows poly p x ≠ 0 ∨ poly (pderiv p) x ≠ 0  $\implies$ 
        sign-changes ps' x = sign-changes ps x

```


$p \neq 0 \implies \text{sign-changes-inf } ps' = \text{sign-changes-inf } ps$
 $p \neq 0 \implies \text{sign-changes-neg-inf } ps' = \text{sign-changes-neg-inf } ps$

proof –

define d **where** $d = \text{gcd } p \text{ (pderiv } p)$
define p' **where** $p' = p \text{ div } d$
define s' **where** $s' = \text{poly-inf } d$
define s'' **where** $s'' = \text{poly-neg-inf } d$

{
fix $x :: \text{real}$ **and** $q :: \text{real poly}$
assume $q \in \text{set } ps$
hence $d \text{ dvd } q$ **unfolding** $d\text{-def } ps\text{-def}$ **using** sturm-gcd **by** simp
hence $q\text{-prod}: q = (q \text{ div } d) * d$ **unfolding** $p'\text{-def } d\text{-def}$
by $(\text{simp add: algebra-simps dvd-mult-div-cancel})$

have $\text{poly } q \ x = \text{poly } d \ x * \text{poly } (q \text{ div } d) \ x$ **by** $(\text{subst } q\text{-prod}, \text{simp})$
hence $s1: \text{sgn } (\text{poly } q \ x) = \text{sgn } (\text{poly } d \ x) * \text{sgn } (\text{poly } (q \text{ div } d) \ x)$
by $(\text{subst } q\text{-prod}, \text{simp add: sgn-mult})$
from poly-inf-mult **have** $s2: \text{poly-inf } q = s' * \text{poly-inf } (q \text{ div } d)$
unfolding $s'\text{-def}$ **by** $(\text{subst } q\text{-prod}, \text{simp})$
from poly-inf-mult **have** $s3: \text{poly-neg-inf } q = s'' * \text{poly-neg-inf } (q \text{ div } d)$
unfolding $s''\text{-def}$ **by** $(\text{subst } q\text{-prod}, \text{simp})$
note $s1 \ s2 \ s3$
}

note $\text{signs} = \text{this}$

{
fix $f :: \text{real poly} \implies \text{real}$ **and** $s :: \text{real}$
assume $f: \bigwedge q. q \in \text{set } ps \implies f \ q = s * f \ (q \text{ div } d)$ **and** $s: s \neq 0$
hence $\text{inverse } s \neq 0$ **by** simp
{fix q **assume** $q \in \text{set } ps$
hence $f \ (q \text{ div } d) = \text{inverse } s * f \ q$
by $(\text{subst } f[\text{of } q], \text{simp-all add: } s)$
} **note** $f' = \text{this}$
have $\text{length } (\text{remdups-adj } [x \leftarrow \text{map } f \ (\text{map } (\lambda q. q \text{ div } d) \ ps). \ x \neq 0]) - 1 =$
 $\text{length } (\text{remdups-adj } [x \leftarrow \text{map } (\lambda q. f \ (q \text{ div } d)) \ ps . \ x \neq 0]) - 1$
by $(\text{simp only: sign-changes-def o-def map-map})$
also have $\text{map } (\lambda q. q \text{ div } d) \ ps = ps'$
by $(\text{simp add: ps-def ps'\text{-def sturm-squarefree'\text{-def Let-def } d\text{-def})}$
also from f' **have** $\text{map } (\lambda q. f \ (q \text{ div } d)) \ ps =$
 $\text{map } (\lambda x. ((*)(\text{inverse } s) \circ f) \ x) \ ps$ **by** (simp add: o-def)
also note $\text{sign-changes-mult-aux}[OF \langle \text{inverse } s \neq 0 \rangle, \text{of } f \ ps]$
finally have
 $\text{length } (\text{remdups-adj } [x \leftarrow \text{map } f \ ps' . \ x \neq 0]) - 1 =$
 $\text{length } (\text{remdups-adj } [x \leftarrow \text{map } f \ ps . \ x \neq 0]) - 1$ **by** simp
}

note $\text{length-remdups-adj} = \text{this}$

{

```

fix x assume A: poly p x ≠ 0 ∨ poly (pderiv p) x ≠ 0
have d dvd p d dvd pderiv p unfolding d-def by simp-all
with A have sgn (poly d x) ≠ 0
  by (auto simp add: sgn-zero-iff elim: dvdE)
thus sign-changes ps' x = sign-changes ps x using signs(1)
  unfolding sign-changes-def
  by (intro length-remdups-adj[of λq. sgn (poly q x)], simp-all)
}

```

```

assume p ≠ 0
hence d ≠ 0 unfolding d-def by simp
hence s' ≠ 0 s'' ≠ 0 unfolding s'-def s''-def by simp-all
from length-remdups-adj[of poly-inf s', OF signs(2) ⟨s' ≠ 0⟩]
  show sign-changes-inf ps' = sign-changes-inf ps
  unfolding sign-changes-inf-def .
from length-remdups-adj[of poly-neg-inf s'', OF signs(3) ⟨s'' ≠ 0⟩]
  show sign-changes-neg-inf ps' = sign-changes-neg-inf ps
  unfolding sign-changes-neg-inf-def .
qed

```

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

proof (cases p ≠ 0 ∧ a ≤ b)

case False

```

note False' = this
hence  $\text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} = 0$ 
proof (cases  $a < b$ )
  case True
    with False have [simp]:  $p = 0$  by simp
    have subset:  $\{a < .. < b\} \subseteq \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$  by auto
    from infinite-Ioo[OF True] have  $\neg \text{finite } \{a < .. < b\}$  .
    hence  $\neg \text{finite } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ 
      using finite-subset[OF subset] by blast
    thus ?thesis by simp
  next
    case False
      with False' show ?thesis by (auto simp: not-less card-eq-0-iff)
    qed
  thus ?thesis unfolding count-roots-between-def Let-def using False by auto
next
  case True
    hence  $p \neq 0 \wedge a \leq b$  by simp-all
    define p' where  $p' = p \ \text{div} \ (\text{gcd } p \ (\text{pderiv } p))$ 
    from poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ] have  $p' \neq 0$ 
      unfolding p'-def by clarsimp

    from sturm-seq-sturm-squarefree[OF  $\langle p \neq 0 \rangle$ ]
      interpret sturm-seq sturm-squarefree p p'
      unfolding p'-def .
    from poly-roots-finite[OF  $\langle p' \neq 0 \rangle$ ]
      have finite  $\{x. a < x \wedge x \leq b \wedge \text{poly } p' \ x = 0\}$  by fast
    have count-roots-between  $p \ a \ b = \text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p' \ x = 0\}$ 
      unfolding count-roots-between-def Let-def
      using True count-roots-between[OF  $\langle p' \neq 0 \rangle \langle a \leq b \rangle$ ] by simp
    also from poly-div-gcd-squarefree(2)[OF  $\langle p \neq 0 \rangle$ ]
      have  $\{x. a < x \wedge x \leq b \wedge \text{poly } p' \ x = 0\} =$ 
         $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$  unfolding p'-def by blast
    finally show ?thesis .
  qed

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 (cases  $p = 0$ )
  case True
    with finite-subset[of  $\{0 < .. < 1\} ?S$ ]
      have  $\neg \text{finite } \{x. \text{poly } p \ x = 0\}$  by (auto simp: infinite-Ioo)
      thus ?thesis by (simp add: count-roots-def True)
  next
    case False
      define p' where  $p' = p \ \text{div} \ (\text{gcd } p \ (\text{pderiv } p))$ 
      from poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ] have  $p' \neq 0$ 
        unfolding p'-def by clarsimp

```

```

from sturm-seq-sturm-squarefree[OF ‹ $p \neq 0$ ›]
  interpret sturm-seq sturm-squarefree  $p$   $p'$ 
  unfolding  $p'$ -def .
from count-roots[OF ‹ $p' \neq 0$ ›]
  have count-roots  $p = \text{card } \{x. \text{poly } p' x = 0\}$ 
  unfolding count-roots-def Let-def by (simp add: ‹ $p \neq 0$ ›)
also from poly-div-gcd-squarefree(2)[OF ‹ $p \neq 0$ ›]
  have  $\{x. \text{poly } p' x = 0\} = \{x. \text{poly } p x = 0\}$  unfolding  $p'$ -def by blast
finally show ?thesis .
qed

```

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 - = card ?S)
proof (cases  $p = 0$ )
  case True
  with finite-subset[of  $\{a <..<a+1\}$  ?S]
  have  $\neg \text{finite } \{x. x > a \wedge \text{poly } p x = 0\}$  by (auto simp: infinite-Ioo subset-eq)
  thus ?thesis by (simp add: count-roots-above-def True)
next
  case False
  define  $p'$  where  $p' = p \text{ div } (\text{gcd } p (\text{pderiv } p))$ 
  from poly-div-gcd-squarefree(1)[OF ‹ $p \neq 0$ ›] have  $p' \neq 0$ 
  unfolding  $p'$ -def by clarsimp

```

```

from sturm-seq-sturm-squarefree[OF ‹ $p \neq 0$ ›]
  interpret sturm-seq sturm-squarefree  $p$   $p'$ 
  unfolding  $p'$ -def .
from count-roots-above[OF ‹ $p' \neq 0$ ›]
  have count-roots-above  $p$   $a = \text{card } \{x. x > a \wedge \text{poly } p' x = 0\}$ 
  unfolding count-roots-above-def Let-def by (simp add: ‹ $p \neq 0$ ›)
also from poly-div-gcd-squarefree(2)[OF ‹ $p \neq 0$ ›]
  have  $\{x. x > a \wedge \text{poly } p' x = 0\} = \{x. x > a \wedge \text{poly } p x = 0\}$ 
  unfolding  $p'$ -def by blast
finally show ?thesis .
qed

```

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 - = card ?S)
proof (cases  $p = 0$ )
  case True
  with finite-subset[of  $\{a - 1 <..<a\}$  ?S]
  have  $\neg \text{finite } \{x. x \leq a \wedge \text{poly } p x = 0\}$  by (auto simp: infinite-Ioo subset-eq)
  thus ?thesis by (simp add: count-roots-below-def True)
next

```

```

case False
define  $p'$  where  $p' = p \text{ div } (\text{gcd } p \text{ (pderiv } p))$ 
from poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ] have  $p' \neq 0$ 
  unfolding  $p'$ -def by clarsimp

from sturm-seq-sturm-squarefree[OF  $\langle p \neq 0 \rangle$ ]
  interpret sturm-seq sturm-squarefree  $p \ p'$ 
  unfolding  $p'$ -def .
from count-roots-below[OF  $\langle p' \neq 0 \rangle$ ]
  have count-roots-below  $p \ a = \text{card } \{x. x \leq a \wedge \text{poly } p' \ x = 0\}$ 
  unfolding count-roots-below-def Let-def by (simp add:  $\langle p \neq 0 \rangle$ )
also from poly-div-gcd-squarefree(2)[OF  $\langle p \neq 0 \rangle$ ]
  have  $\{x. x \leq a \wedge \text{poly } p' \ x = 0\} = \{x. x \leq a \wedge \text{poly } p \ x = 0\}$ 
  unfolding  $p'$ -def by blast
finally show ?thesis .
qed

```

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 sign-changes  $ps \ a - \text{sign-changes } ps \ b$ )
        else (let  $ps = \text{sturm-squarefree } p$ 
          in sign-changes  $ps \ a - \text{sign-changes } ps \ b$ ))
proof (cases  $a > b \vee p = 0$ )
  case True
    thus ?thesis by (auto simp add: count-roots-between-def Let-def)
next
  case False
    note False1 = this
    hence  $a \leq b \ p \neq 0$  by simp-all
    thus ?thesis
    proof (cases  $(\text{poly } p \ a \neq 0 \vee \text{poly } (\text{pderiv } p) \ a \neq 0) \wedge$ 
       $(\text{poly } p \ b \neq 0 \vee \text{poly } (\text{pderiv } p) \ b \neq 0)$ )
    case False
      thus ?thesis using False1
      by (auto simp add: Let-def count-roots-between-def)
    next
    case True
      hence A: poly  $p \ a \neq 0 \vee \text{poly } (\text{pderiv } p) \ a \neq 0$  and
        B: poly  $p \ b \neq 0 \vee \text{poly } (\text{pderiv } p) \ b \neq 0$  by auto
      define  $d$  where  $d = \text{gcd } p \ (\text{pderiv } p)$ 
      from  $\langle p \neq 0 \rangle$  have [simp]:  $p \ \text{div} \ d \neq 0$ 
      using poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ] by (auto simp add: d-def)

```

```

from sturm-seq-sturm-squarefree'[OF ⟨p ≠ 0⟩]
  interpret sturm-seq sturm-squarefree' p p div d
  unfolding sturm-squarefree'-def Let-def d-def .
note count-roots-between-correct
also have {x. a < x ∧ x ≤ b ∧ poly p x = 0} =
  {x. a < x ∧ x ≤ b ∧ poly (p div d) x = 0}
  unfolding d-def using poly-div-gcd-squarefree(2)[OF ⟨p ≠ 0⟩] by simp
also note count-roots-between[OF ⟨p div d ≠ 0⟩ ⟨a ≤ b⟩, symmetric]
also note sturm-sturm-squarefree'-same-sign-changes(1)[OF A]
also note sturm-sturm-squarefree'-same-sign-changes(1)[OF B]
finally show ?thesis using True False by (simp add: Let-def)
qed
qed

```

```

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 (cases p = 0, simp add: count-roots-def)
case False
  define d where d = gcd p (pderiv p)
  from ⟨p ≠ 0⟩ have [simp]: p div d ≠ 0
    using poly-div-gcd-squarefree(1)[OF ⟨p ≠ 0⟩] by (auto simp add: d-def)
  from sturm-seq-sturm-squarefree'[OF ⟨p ≠ 0⟩]
    interpret sturm-seq sturm-squarefree' p p div d
    unfolding sturm-squarefree'-def Let-def d-def .

  note count-roots-correct
  also have {x. poly p x = 0} = {x. poly (p div d) x = 0}
    unfolding d-def using poly-div-gcd-squarefree(2)[OF ⟨p ≠ 0⟩] by simp
  also note count-roots[OF ⟨p div d ≠ 0⟩, symmetric]
  also note sturm-sturm-squarefree'-same-sign-changes(2)[OF ⟨p ≠ 0⟩]
  also note sturm-sturm-squarefree'-same-sign-changes(3)[OF ⟨p ≠ 0⟩]
  finally show ?thesis using False unfolding Let-def by simp
qed

```

```

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 (cases p = 0)

```

```

case True
  thus ?thesis by (auto simp add: count-roots-above-def Let-def)
next
case False
  note False1 = this
  hence  $p \neq 0$  by simp-all
  thus ?thesis
  proof (cases (poly p a  $\neq$  0  $\vee$  poly (pderiv p) a  $\neq$  0))
  case False
    thus ?thesis using False1
    by (auto simp add: Let-def count-roots-above-def)
  next
  case True
    hence  $A$ : poly p a  $\neq$  0  $\vee$  poly (pderiv p) a  $\neq$  0 by simp
    define  $d$  where  $d = \text{gcd } p \text{ (pderiv } p)$ 
    from  $\langle p \neq 0 \rangle$  have [simp]:  $p \text{ div } d \neq 0$ 
      using poly-div-gcd-squarefree(1)[OF  $\langle p \neq 0 \rangle$ ] by (auto simp add: d-def)
    from sturm-seq-sturm-squarefree'[OF  $\langle p \neq 0 \rangle$ ]
      interpret sturm-seq sturm-squarefree' p p div d
      unfolding sturm-squarefree'-def Let-def d-def .
    note count-roots-above-correct
    also have  $\{x. a < x \wedge \text{poly } p x = 0\} =$ 
       $\{x. a < x \wedge \text{poly } (p \text{ div } d) x = 0\}$ 
      unfolding d-def using poly-div-gcd-squarefree(2)[OF  $\langle p \neq 0 \rangle$ ] by simp
    also note count-roots-above[OF  $\langle p \text{ div } d \neq 0 \rangle$ , symmetric]
    also note sturm-sturm-squarefree'-same-sign-changes(1)[OF  $A$ ]
    also note sturm-sturm-squarefree'-same-sign-changes(2)[OF  $\langle p \neq 0 \rangle$ ]
    finally show ?thesis using True False by (simp add: Let-def)
  qed
qed

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  $\neq$  0  $\vee$  poly q a  $\neq$  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 (cases p = 0)
case True
  thus ?thesis by (auto simp add: count-roots-below-def Let-def)
next
case False
  note False1 = this
  hence  $p \neq 0$  by simp-all
  thus ?thesis
  proof (cases (poly p a  $\neq$  0  $\vee$  poly (pderiv p) a  $\neq$  0))

```

```

case False
  thus ?thesis using False1
    by (auto simp add: Let-def count-roots-below-def)
next
case True
  hence A: poly p a ≠ 0 ∨ poly (pderiv p) a ≠ 0 by simp
  define d where d = gcd p (pderiv p)
  from ⟨p ≠ 0⟩ have [simp]: p div d ≠ 0
    using poly-div-gcd-squarefree(1)[OF ⟨p ≠ 0⟩] by (auto simp add: d-def)
  from sturm-seq-sturm-squarefree'[OF ⟨p ≠ 0⟩]
    interpret sturm-seq sturm-squarefree' p p div d
    unfolding sturm-squarefree'-def Let-def d-def .
  note count-roots-below-correct
  also have {x. x ≤ a ∧ poly p x = 0} =
    {x. x ≤ a ∧ poly (p div d) x = 0}
    unfolding d-def using poly-div-gcd-squarefree(2)[OF ⟨p ≠ 0⟩] by simp
  also note count-roots-below[OF ⟨p div d ≠ 0⟩, symmetric]
  also note sturm-sturm-squarefree'-same-sign-changes(1)[OF A]
  also note sturm-sturm-squarefree'-same-sign-changes(3)[OF ⟨p ≠ 0⟩]
  finally show ?thesis using True False by (simp add: Let-def)
qed
qed
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
by (simp add: count-roots-between-correct)

```

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 (cases (a ≤ b ∧ poly p a = 0 ∧ p ≠ 0) ∨ (a = b ∧ p = 0))

case False

note False' = this

thus ?thesis


```

proof (cases p = 0)
  case False
    with False' have poly p a ≠ 0 ∨ a > b by auto
    hence {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0} =
      {x. a < x ∧ x ≤ b ∧ poly p x = 0}
    by (auto simp: less-eq-real-def)
    thus ?thesis using poly-card-roots-less-leq False'
      by (auto split: if-split-asm)
  next
    case True
      have {x. a ≤ x ∧ x ≤ b} = {a..b}
        {x. a < x ∧ x ≤ b} = {a<..b} by auto
      with True False have card {x. a < x ∧ x ≤ b} = 0 card {x. a ≤ x ∧ x ≤
b} = 0
        by (auto simp add: card-eq-0-iff infinite-Ioc infinite-Icc)
      with True False show ?thesis
        using count-roots-between-correct by (simp add: )
    qed
  next
    case True
      note True' = this
      have fin: finite {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0}
      proof (cases p = 0)
        case True
          with True' have a = b by simp
          hence {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0} = {b} using True by auto
          thus ?thesis by simp
        next
          case False
            from poly-roots-finite[OF this] show ?thesis by fast
          qed
        with True have {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0} =
          insert a {x. a < x ∧ x ≤ b ∧ poly p x = 0} by auto
        hence card {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0} =
          Suc (card {x. a < x ∧ x ≤ b ∧ poly p x = 0}) using fin by force
        thus ?thesis using True count-roots-between-correct by simp
      qed
  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 (cases poly p b = 0 ∧ a < b ∧ p ≠ 0)
    case False
      note False' = this
      show ?thesis
      proof (cases p = 0)
        case True
          have [simp]: {x. a < x ∧ x < b} = {a<..b}

```

$\{x. a < x \wedge x \leq b\} = \{a <..b\}$ **by** *auto*
with *True False* **have** $\text{card } \{x. a < x \wedge x \leq b\} = 0$ $\text{card } \{x. a < x \wedge x < b\} = 0$
by (*auto simp add: card-eq-0-iff infinite-Ioo infinite-Ioc*)
with *True False'* **show** *?thesis*
by (*auto simp: count-roots-between-correct*)
next
case *False*
with *False'* **have** $\{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\} =$
 $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$
by (*auto simp: less-eq-real-def*)
thus *?thesis* **using** *poly-card-roots-less-leq False* **by** *auto*
qed
next
case *True*
with *poly-roots-finite*
have *fin: finite* $\{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\}$ **by** *fast*
from *True* **have** $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} =$
 $\text{insert } b \ \{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\}$ **by** *auto*
hence *Suc* $(\text{card } \{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\}) =$
 $\text{card } \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$ **using** *fin* **by** *force*
also note *count-roots-between-correct[symmetric]*
finally show *?thesis* **using** *True* **by** *simp*
qed

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

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

proof (*cases* $p = 0 \vee a \geq b$)

case *True*

note *True' = this*

show *?thesis*

proof (*cases* $a \geq b$)

case *False*

hence $\{x. a < x \wedge x \leq b\} = \{a <..b\}$

$\{x. a \leq x \wedge x < b\} = \{a..<b\}$ **by** *auto*

with *True False* **have** $\text{card } \{x. a < x \wedge x \leq b\} = 0$ $\text{card } \{x. a \leq x \wedge x < b\} = 0$
by (*auto simp add: card-eq-0-iff infinite-Ico infinite-Ioc*)

with *False True'* **show** *?thesis*

by (*simp add: count-roots-between-correct*)

next

case *True*

with *True'* **have** $\{x. a \leq x \wedge x < b \wedge \text{poly } p \ x = 0\} =$
 $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$

by (*auto simp: less-eq-real-def*)

thus *?thesis* **using** *poly-card-roots-less-leq True* **by** *simp*

qed
next
case *False*
let $?A = \{x. a \leq x \wedge x < b \wedge \text{poly } p \ x = 0\}$
let $?B = \{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\}$
let $?C = \{x. x = b \wedge \text{poly } p \ x = 0\}$
let $?D = \{x. x = a \wedge \text{poly } p \ a = 0\}$
have *CD-if*: $?C = (\text{if } \text{poly } p \ b = 0 \text{ then } \{b\} \text{ else } \{\})$
 $?D = (\text{if } \text{poly } p \ a = 0 \text{ then } \{a\} \text{ else } \{\})$ **by** *auto*
from *False poly-roots-finite*
have [*simp*]: *finite ?A finite ?B finite ?C finite ?D*
by (*fast, fast, simp-all*)

from *False* **have** $?A = (?B \cup ?D) - ?C$ **by** (*auto simp: less-eq-real-def*)
with *False* **have** $\text{card } ?A = \text{card } ?B + (\text{if } \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0) -$
 $(\text{if } \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0)$ **by** (*auto simp: CD-if*)
also note *count-roots-between-correct[symmetric]*
finally show *?thesis* **using** *False* **by** *simp*
qed

lemma *poly-card-roots*:
 $\text{card } \{x::\text{real}. \text{poly } p \ x = 0\} = \text{count-roots } p$
using *count-roots-correct* **by** *simp*

lemma *poly-no-roots*:
 $(\forall x. \text{poly } p \ x \neq 0) \longleftrightarrow (p \neq 0 \wedge \text{count-roots } p = 0)$
by (*auto simp: count-roots-correct dest: poly-roots-finite*)

lemma *poly-pos*:
 $(\forall x. \text{poly } p \ x > 0) \longleftrightarrow ($
 $p \neq 0 \wedge \text{poly-inf } p = 1 \wedge \text{count-roots } p = 0)$
by (*simp only: Let-def poly-pos poly-no-roots, blast*)

lemma *poly-card-roots-greater*:
 $\text{card } \{x::\text{real}. x > a \wedge \text{poly } p \ x = 0\} = \text{count-roots-above } p \ a$
using *count-roots-above-correct* **by** *simp*

lemma *poly-card-roots-leq*:
 $\text{card } \{x::\text{real}. x \leq a \wedge \text{poly } p \ x = 0\} = \text{count-roots-below } p \ a$
using *count-roots-below-correct* **by** *simp*

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 (*cases poly p a = 0 ∧ p ≠ 0*)

case *False*

hence $\text{card } \{x. x \geq a \wedge \text{poly } p \ x = 0\} = \text{card } \{x. x > a \wedge \text{poly } p \ x = 0\}$

proof (*cases rule: disjE*)

```

assume  $p = 0$ 
have  $\neg \text{finite } \{a <..<a+1\}$ 
  by (metis infinite-Ioo less-add-one)
moreover have  $\{a <..<a+1\} \subseteq \{x. x \geq a \wedge \text{poly } p x = 0\}$ 
   $\{a <..<a+1\} \subseteq \{x. x > a \wedge \text{poly } p x = 0\}$ 
  using  $\langle p = 0 \rangle$  by auto
ultimately have  $\neg \text{finite } \{x. x \geq a \wedge \text{poly } p x = 0\}$ 
   $\neg \text{finite } \{x. x > a \wedge \text{poly } p x = 0\}$ 
  by (auto dest!: finite-subset[of {a <..<a+1}] simp: infinite-Ioo)
thus ?thesis by simp
next
assume  $\text{poly } p a \neq 0$ 
hence  $\{x. x \geq a \wedge \text{poly } p x = 0\} = \{x. x > a \wedge \text{poly } p x = 0\}$ 
  by (auto simp: less-eq-real-def)
thus ?thesis by simp
qed auto
thus ?thesis using False
  by (auto intro: poly-card-roots-greater)
next
case True
hence finite  $\{x. x > a \wedge \text{poly } p x = 0\}$  using poly-roots-finite by force
moreover have  $\{x. x \geq a \wedge \text{poly } p x = 0\} =$ 
   $\text{insert } a \{x. x > a \wedge \text{poly } p x = 0\}$  using True by auto
ultimately have  $\text{card } \{x. x \geq a \wedge \text{poly } p x = 0\} =$ 
   $\text{Suc } (\text{card } \{x. x > a \wedge \text{poly } p x = 0\})$ 
  using card-insert-disjoint by auto
thus ?thesis using True by (auto intro: poly-card-roots-greater)
qed

lemma poly-card-roots-less:
 $\text{card } \{x::\text{real}. x < a \wedge \text{poly } p x = 0\} =$ 
  (count-roots-below p a - (if poly p a = 0 \wedge p \neq 0 then 1 else 0))
proof (cases poly p a = 0 \wedge p \neq 0)
case False
hence  $\text{card } \{x. x < a \wedge \text{poly } p x = 0\} = \text{card } \{x. x \leq a \wedge \text{poly } p x = 0\}$ 
proof (cases rule: disjE)
  assume  $p = 0$ 
  have  $\neg \text{finite } \{a - 1 <..<a\}$ 
    by (metis infinite-Ioo diff-add-cancel less-add-one)
  moreover have  $\{a - 1 <..<a\} \subseteq \{x. x \leq a \wedge \text{poly } p x = 0\}$ 
     $\{a - 1 <..<a\} \subseteq \{x. x < a \wedge \text{poly } p x = 0\}$ 
    using  $\langle p = 0 \rangle$  by auto
  ultimately have  $\neg \text{finite } \{x. x \leq a \wedge \text{poly } p x = 0\}$ 
     $\neg \text{finite } \{x. x < a \wedge \text{poly } p x = 0\}$ 
    by (auto dest: finite-subset[of {a - 1 <..<a}] simp: infinite-Ioo)
  thus ?thesis by simp
next
assume  $\text{poly } p a \neq 0$ 
hence  $\{x. x < a \wedge \text{poly } p x = 0\} = \{x. x \leq a \wedge \text{poly } p x = 0\}$ 

```

by (auto simp: less-eq-real-def)
 thus ?thesis by simp
 qed auto
 thus ?thesis using False
 by (auto intro: poly-card-roots-leq)
 next
 case True
 hence finite $\{x. x < a \wedge \text{poly } p \ x = 0\}$ using poly-roots-finite by force
 moreover have $\{x. x \leq a \wedge \text{poly } p \ x = 0\} =$
 insert a $\{x. x < a \wedge \text{poly } p \ x = 0\}$ using True by auto
 ultimately have Suc (card $\{x. x < a \wedge \text{poly } p \ x = 0\}$) =
 (card $\{x. x \leq a \wedge \text{poly } p \ x = 0\}$)
 using card-insert-disjoint by auto
 also note count-roots-below-correct[symmetric]
 finally show ?thesis using True by simp
 qed

lemma poly-no-roots-less-leq:
 $(\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$
 $((a \geq b \vee (p \neq 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$
 by (auto simp: count-roots-between-correct card-eq-0-iff not-le
 dest: poly-roots-finite)

lemma poly-pos-between-less-leq:
 $(\forall x. a < x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$
 $((a \geq b \vee (p \neq 0 \wedge \text{poly } p \ b > 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$
 by (simp only: poly-pos-between-less-leq Let-def
 poly-no-roots-less-leq, blast)

lemma poly-no-roots-leq-leq:
 $(\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x \neq 0) \longleftrightarrow$
 $((a > b \vee (p \neq 0 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-between } p \ a \ b = 0)))$
 apply (intro iffI)
 apply (force simp add: count-roots-between-correct card-eq-0-iff)
 apply (elim conjE disjE, simp, intro allI)
 apply (rename-tac x, case-tac x = a)
 apply (auto simp add: count-roots-between-correct card-eq-0-iff
 dest: poly-roots-finite)

done

lemma poly-pos-between-leq-leq:
 $(\forall x. a \leq x \wedge x \leq b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$
 $((a > b \vee (p \neq 0 \wedge \text{poly } p \ a > 0 \wedge$
 count-roots-between $p \ a \ b = 0)))$
 by (simp only: poly-pos-between-leq-leq Let-def poly-no-roots-leq-leq, force)

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 (*standard, goal-cases*)
case *A*: 1
show *?case*
proof (*cases* $a \geq b$)
case *True*
with *A* **show** *?thesis* **by** *simp*
next
case *False*
with *A* **have** [*simp*]: $p \neq 0$ **using** *dense*[*of a b*] **by** *auto*
have *B*: $\{x. a < x \wedge x \leq b \wedge \text{poly } p \ x = 0\} =$
 $\{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\} \cup$
 $(\text{if } \text{poly } p \ b = 0 \text{ then } \{b\} \text{ else } \{\})$ **using** *A False* **by** *auto*
have *count-roots-between* $p \ a \ b =$
 $\text{card } \{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\} +$
 $(\text{if } \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0)$
by (*subst count-roots-between-correct, subst B, subst card-Un-disjoint,*
rule finite-subset[OF - poly-roots-finite], blast, simp-all)
also from *A* **have** $\{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\} = \{\}$ **by** *simp*
finally show *?thesis* **by** *auto*
qed
next
case *prems*: 2
hence $\text{card } \{x. a < x \wedge x < b \wedge \text{poly } p \ x = 0\} = 0$
by (*subst poly-card-roots-less-less, auto simp: count-roots-between-def*)
thus *?case* **using** *prems*
by (*cases* $p = 0$, *simp, subst (asm) card-eq-0-iff,*
auto dest: poly-roots-finite)
qed

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

by (*simp only: poly-pos-between-less-less Let-def*
poly-no-roots-less-less, blast)

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 (*standard, goal-cases*)
case *prems*: 1
hence $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p \ x \neq 0$ **by** *simp*
thus *?case* **using** *prems* **by** (*subst (asm) poly-no-roots-less-less, auto*)

next

case *prems*: 2

hence $(b \leq a \vee p \neq 0 \wedge \text{count-roots-between } p \ a \ b =$
 $(\text{if } \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0))$ **by** *auto*

thus ?*case using prems unfolding Let-def*

by (*subst (asm) poly-no-roots-less-less[symmetric, unfolded Let-def]*,
auto split: if-split-asm simp: less-eq-real-def)

qed

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

$(\forall x. a \leq x \wedge x < b \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$
 $((a \geq b \vee (p \neq 0 \wedge \text{poly } p \ a > 0 \wedge \text{count-roots-between } p \ a \ b =$
 $(\text{if } a < b \wedge \text{poly } p \ b = 0 \text{ then } 1 \text{ else } 0))))$

by (*simp only: poly-pos-between-leq-less Let-def*
poly-no-roots-leq-less, force)

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 –

have $\forall x. \neg a < x \implies \text{False}$ **by** (*metis gt-ex*)

thus ?*thesis by (auto simp: count-roots-above-correct card-eq-0-iff*
intro: poly-roots-finite)

qed

lemma *poly-pos-greater*:

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

unfolding *Let-def*

by (*subst poly-pos-greater, subst poly-no-roots-greater, force*)

lemma *poly-no-roots-leq*:

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

by (*auto simp: Let-def count-roots-below-correct card-eq-0-iff*
intro: poly-roots-finite)

lemma *poly-pos-leq*:

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

by (*simp only: poly-pos-leq Let-def poly-no-roots-leq, blast*)

lemma *poly-no-roots-geq*:

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

proof (*standard, goal-cases*)

case *prems: 1*
hence $\forall x > a. \text{poly } p \ x \neq 0$ **by** *simp*
thus *?case using prems by (subst (asm) poly-no-roots-greater, auto)*
next
case *prems: 2*
hence $(p \neq 0 \wedge \text{count-roots-above } p \ a = 0)$ **by** *simp*
thus *?case using prems*
by $(\text{subst } (asm) \text{ poly-no-roots-greater}[\text{symmetric, unfolded Let-def}],$
auto simp: less-eq-real-def)

qed

lemma *poly-pos-geq:*

$(\forall x. x \geq a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$
 $(p \neq 0 \wedge \text{poly-inf } p = 1 \wedge \text{poly } p \ a \neq 0 \wedge \text{count-roots-above } p \ a = 0)$
by $(\text{simp only: poly-pos-geq Let-def poly-no-roots-geq, blast})$

lemma *poly-no-roots-less:*

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

proof $(\text{standard, goal-cases})$

case *prems: 1*

hence $\{x. x \leq a \wedge \text{poly } p \ x = 0\} = (\text{if } \text{poly } p \ a = 0 \text{ then } \{a\} \text{ else } \{\})$
by $(\text{auto simp: less-eq-real-def})$

moreover **have** $\forall x. \neg x < a \implies \text{False}$ **by** (metis lt-ex)

ultimately show *?case using prems by (auto simp: count-roots-below-correct)*

next

case *prems: 2*

have $A: \{x. x \leq a \wedge \text{poly } p \ x = 0\} = \{x. x < a \wedge \text{poly } p \ x = 0\} \cup$
 $(\text{if } \text{poly } p \ a = 0 \text{ then } \{a\} \text{ else } \{\})$ **by** $(\text{auto simp: less-eq-real-def})$

have $\text{count-roots-below } p \ a = \text{card } \{x. x < a \wedge \text{poly } p \ x = 0\} +$
 $(\text{if } \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0)$ **using** *prems*

by $(\text{subst count-roots-below-correct, subst } A, \text{subst card-Un-disjoint,}$
auto intro: poly-roots-finite)

with *prems* **have** $\text{card } \{x. x < a \wedge \text{poly } p \ x = 0\} = 0$ **by** *simp*

thus *?case using prems*

by $(\text{subst } (asm) \text{ card-eq-0-iff, auto intro: poly-roots-finite})$

qed

lemma *poly-pos-less:*

$(\forall x. x < a \longrightarrow \text{poly } p \ x > 0) \longleftrightarrow$
 $(p \neq 0 \wedge \text{poly-neg-inf } p = 1 \wedge \text{count-roots-below } p \ a =$
 $(\text{if } \text{poly } p \ a = 0 \text{ then } 1 \text{ else } 0))$
by $(\text{simp only: poly-pos-less Let-def poly-no-roots-less, blast})$

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*:

$$\begin{aligned} \{x::real. P x\} &= \{x::real. (PR-TAG P) x\} \\ (\forall x::real. f x < g x) &= (\forall x::real. PR-TAG (\lambda x. f x < g x) x) \\ (\forall x::real. P x) &= (\forall x::real. \neg(PR-TAG (\lambda x. \neg P x)) x) \\ \text{by } &(\text{simp-all add: } PR-TAG\text{-def}) \end{aligned}$$

lemma *sturm-id-PR-prio1*:

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

lemma *sturm-id-PR-prio2*:

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

$(\forall x::real. a \leq x \wedge x \leq b \longrightarrow PR\text{-TAG } (\lambda x. f x < g x) x)$
 $(\forall x::real. a < x \wedge x < b \longrightarrow f x < g x) =$
 $(\forall x::real. a < x \wedge x < b \longrightarrow PR\text{-TAG } (\lambda x. f x < g x) x)$
 $(\forall x::real. a \leq x \wedge x < b \longrightarrow f x < g x) =$
 $(\forall x::real. a \leq x \wedge x < b \longrightarrow PR\text{-TAG } (\lambda x. f x < g x) x)$
 $(\forall x::real. a < x \wedge x \leq b \longrightarrow P x) =$
 $(\forall x::real. a < x \wedge x \leq b \longrightarrow \neg(PR\text{-TAG } (\lambda x. \neg P x) x))$
 $(\forall x::real. a \leq x \wedge x \leq b \longrightarrow P x) =$
 $(\forall x::real. a \leq x \wedge x \leq b \longrightarrow \neg(PR\text{-TAG } (\lambda x. \neg P x) x))$
 $(\forall x::real. a \leq x \wedge x < b \longrightarrow P x) =$
 $(\forall x::real. a \leq x \wedge x < b \longrightarrow \neg(PR\text{-TAG } (\lambda x. \neg P x) x))$
 $(\forall x::real. a < x \wedge x < b \longrightarrow P x) =$
 $(\forall x::real. a < x \wedge x < b \longrightarrow \neg(PR\text{-TAG } (\lambda x. \neg P x) x))$
by (*simp-all add: PR-TAG-def*)

lemma *PR-TAG-intro-prio0*:

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

shows

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

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

$PR\text{-TAG } f = (\lambda x. poly p x) \Longrightarrow PR\text{-TAG } (\lambda x. -f x) = (\lambda x. poly (-p) x)$
 $\llbracket PR\text{-TAG } f = (\lambda x. poly p x); PR\text{-TAG } g = (\lambda x. poly q x) \rrbracket$
 $\Longrightarrow PR\text{-TAG } (\lambda x. f x + g x) = (\lambda x. poly (p+q) x)$
 $\llbracket PR\text{-TAG } f = (\lambda x. poly p x); PR\text{-TAG } g = (\lambda x. poly q x) \rrbracket$
 $\Longrightarrow PR\text{-TAG } (\lambda x. f x - g x) = (\lambda x. poly (p-q) x)$
 $\llbracket PR\text{-TAG } f = (\lambda x. poly p x); PR\text{-TAG } g = (\lambda x. poly q x) \rrbracket$
 $\Longrightarrow PR\text{-TAG } (\lambda x. f x * g x) = (\lambda x. poly (p*q) x)$
 $PR\text{-TAG } f = (\lambda x. poly p x) \Longrightarrow PR\text{-TAG } (\lambda x. (f x) \widehat{n}) = (\lambda x. poly (p \widehat{n}) x)$
 $PR\text{-TAG } (\lambda x. poly p x :: real) = (\lambda x. poly p x)$
 $PR\text{-TAG } (\lambda x. x::real) = (\lambda x. poly [0,1:] x)$
 $PR\text{-TAG } (\lambda x. a::real) = (\lambda x. poly [a:] x)$
by (*simp-all add: PR-TAG-def poly-eq-0-iff-dvd field-simps*)

lemma *PR-TAG-intro-prio1*:

fixes $f :: \text{real} \Rightarrow \text{real}$

shows

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

by (*simp-all add: PR-TAG-def field-simps poly-monom*)

lemma *PR-TAG-intro-prio2*:

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

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

by (*simp-all add: PR-TAG-def field-simps poly-monom power-add*)

lemma *sturm-meta-spec*: $(\bigwedge x :: \text{real}. P \ x) \Longrightarrow P \ x$ **by** *simp*

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 → a < x → c) ↔ (a < x ∧ x < b → c)
(x < b → a ≤ x → c) ↔ (a ≤ x ∧ x < b → c)
(x ≤ b → a < x → c) ↔ (a < x ∧ x ≤ b → c)
(x ≤ b → a ≤ x → c) ↔ (a ≤ x ∧ x ≤ b → c)
by auto

```

3.3 Setup for the “sturm” method

ML-file `<sturm.ML>`

```

method-setup sturm = <
  Scan.succeed (fn ctxt => SIMPLE-METHOD' (Sturm.sturm-tac ctxt true))
>

```

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
  ∀x::real. x2 + 1 ≠ 0
by sturm

```

```

lemma
  fixes x :: real
  shows x2 + 1 ≠ 0 by sturm

```

```

lemma (x::real) > 1 ⇒ x3 > 1 by sturm

```

```

lemma ∀x::real. x*x ≠ -1 by sturm

```

schematic-goal A:

```

card {x::real. -0.010831 < x ∧ x < 0.010831 ∧
  1/120*x5 + 1/24*x4 + 1/6*x3 - 49/16777216*x2 - 17/2097152*x =
0}
= ?n
by sturm

```

lemma *card* { $x::real. x^3 + x = 2*x^2 \wedge x^3 - 6*x^2 + 11*x = 6$ } = 1
by *sturm*

schematic-goal *card* { $x::real. x^3 + x = 2*x^2 \vee x^3 - 6*x^2 + 11*x = 6$ }
= ?*n* **by** *sturm*

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
by *sturm*

lemma $\forall x::real. x*x \neq 0 \vee x*x - 1 \neq 2*x$ **by** *sturm*

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

3 examples related to continued fraction approximants to exp: LCP

lemma fixes $x::real$
shows $-7.29347719 \leq x \implies 0 < x^5 + 30*x^4 + 420*x^3 + 3360*x^2 +$
 $15120*x + 30240$
by *sturm*

lemma fixes $x::real$
shows $0 < x^6 + 42*x^5 + 840*x^4 + 10080*x^3 + 75600*x^2 + 332640*x$
 $+ 665280$
by *sturm*

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*
by *sturm*

end