A Formalisation of Sturm's Theorem in Isabelle/HOL

Manuel Eberl ¡eberlm@in.tum.de¿
Institut für Informatik, TU München

September 12, 2013

Contents

theory *Sturm*
imports ~~/*src*/*HOL*/*Library*/*Poly-Deriv Sturm-Library*
begin

1 *sign-changes* function

definition *sign-changes* **where**
sign-changes ps $(x{::}real) =$
$\quad$ *length* (*group* (*filter* $(\lambda x.\ x \neq 0)$ (*map* $(\lambda p.\ sgn\ (poly\ p\ x))$ *ps*))) $- 1$

lemma *sign-changes-distrib*:
$\quad poly\ p\ x \neq 0 \Longrightarrow$
$\quad\quad sign\text{-}changes\ (ps_1\ @\ [p]\ @\ ps_2)\ x =$
$\quad\quad sign\text{-}changes\ (ps_1\ @\ [p])\ x + sign\text{-}changes\ ([p]\ @\ ps_2)\ x$
by (*simp add*: *sign-changes-def sgn-zero-iff*, *subst group-append*, *simp*)

```
lemma same-signs-imp-same-sign-changes:
  assumes length ps = length ps'
  assumes ∀ i < length ps. sgn (poly (ps!i) x) = sgn (poly (ps'!i) y)
  shows sign-changes ps x = sign-changes ps' y
proof−
 from assms(2) have A: map (λp. sgn (poly p x)) ps = map (λp. sgn (poly p y))
ps'
  proof (induction rule: list-induct2[OF assms(1)], simp)
    case (goal1 p ps p' ps')
      from goal1(3)
      have ∀ i<length ps. sgn (poly (ps ! i) x) =
                     sgn (poly (ps' ! i) y) by auto
      from goal1(2)[OF this] goal1(3) show ?case by auto
  qed
  show ?thesis unfolding sign-changes-def by (simp add: A)
qed

lemma same-signs-imp-same-sign-changes':
  assumes ∀ p ∈ set ps. sgn (poly p x) = sgn (poly p y)
  shows sign-changes ps x = sign-changes ps y
using assms by (intro same-signs-imp-same-sign-changes, simp-all)

lemma sign-changes-sturm-triple:
  assumes poly p x ≠ 0 and sgn (poly r x) = − sgn (poly p x)
  shows sign-changes [p,q,r] x = 1
unfolding sign-changes-def by (insert assms, auto simp: sgn-real-def)


definition sign-changes-inf where
sign-changes-inf ps =
    length (group (filter (λx. x ≠ 0) (map poly-inf ps))) − 1

definition sign-changes-neg-inf where
sign-changes-neg-inf ps =
    length (group (filter (λx. x ≠ 0) (map poly-neg-inf ps))) − 1
```

2 Definition of Sturm sequences locale

```
locale quasi-sturm-seq =
  fixes ps :: (real poly) list
  assumes last-ps-sgn-const[simp]:
      ⋀x y. sgn (poly (last ps) x) = sgn (poly (last ps) y)
  assumes ps-not-Nil[simp]: ps ≠ []
  assumes signs: ⋀i x. ⟦i < length ps − 2; poly (ps ! (i+1)) x = 0⟧
                   ⟹ (poly (ps ! (i+2)) x) ∗ (poly (ps ! i) x) < 0

locale sturm-seq = quasi-sturm-seq +
  fixes p :: real poly
  assumes hd-ps-p[simp]: hd ps = p
```

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

lemma *quasi-sturm-seq*: *quasi-sturm-seq ps* **..**

lemma *ps-first-two*:
obtains *q ps′* **where** *ps* = *p* # *q* # *ps′*
using *hd-ps-p length-ps-ge-2*
by (*cases ps*, *simp*, *clarsimp*, *rename-tac ps′*, *case-tac ps′*, *auto*)

lemma *ps-first*: *ps* ! *0* = *p* **by** (*rule ps-first-two*, *simp*)

lemma [*simp*]: *p* $\in$ *set ps* **using** *hd-in-set*[*OF ps-not-Nil*] **by** *simp*

end

locale *sturm-seq-squarefree* = *sturm-seq* +
assumes *p-squarefree*: $\bigwedge x$. $\neg$(*poly p x* = *0* $\wedge$ *poly* (*ps*!*1*) *x* = *0*)

lemma [*simp*]: $\neg$*quasi-sturm-seq* [] **by** (*simp add*: *quasi-sturm-seq-def*)

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** ‹*ps* $\neq$ []›
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

3 Auxiliary lemmas about roots and sign changes

lemma (**in** $-$) *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*⟧
$\Longrightarrow$ *sgn* (*poly* (*ps* ! (*i*+*2*)) *x*) = $-$ *sgn* (*poly* (*ps* ! *i*) *x*)

shows $\forall j \le i+1.\ poly\ (ps\ !\ j)\ x = 0$
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.prems*(*1*,*2*)
have $sgn\ (poly\ (ps\ !\ (i + 2))\ x) = -\ sgn\ (poly\ (ps\ !\ i)\ x)$
by (*intro assms*(*4*)) *simp-all*
with *Suc.prems*(*3*) **have** $poly\ (ps\ !\ i)\ x = 0$ **by** (*simp add*: *sgn-zero-iff*)
with *Suc.prems* **have** $\forall j \le i+1.\ poly\ (ps\ !\ j)\ x = 0$
by (*intro Suc.IH*, *simp-all*)
with *Suc.prems*(*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
$[p,q]\ \#$ *split-sign-changes* $(q\#r\#ps)\ x$)

lemma (**in** *quasi-sturm-seq*) *split-sign-changes-subset*[*dest*]:
$ps' \in set$ (*split-sign-changes ps x*) $\Longrightarrow set\ ps' \subseteq set\ ps$
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*:
$[\![\bigwedge p\ x.\ P\ [p]\ x;\ \bigwedge p\ q\ x.\ quasi\text{-}sturm\text{-}seq\ [p,q] \Longrightarrow P\ [p,q]\ x;$
$\bigwedge p\ q\ r\ ps\ x.\ quasi\text{-}sturm\text{-}seq\ (p\#q\#r\#ps) \Longrightarrow$
$[\![poly\ p\ x \neq 0 \Longrightarrow poly\ q\ x = 0 \Longrightarrow P\ (r\#ps)\ x;$
$poly\ q\ x \neq 0 \Longrightarrow P\ (q\#r\#ps)\ x;$
$poly\ p\ x = 0 \Longrightarrow P\ (q\#r\#ps)\ x]\!]$
$\Longrightarrow P\ (p\#q\#r\#ps)\ x]\!] \Longrightarrow P\ ps\ x$
proof−

case *goal1*
have *quasi-sturm-seq ps* **..**
with *goal1* **show** *?thesis*
proof (*induction ps x rule*: *split-sign-changes.induct*)
case (*goal3 p q r ps x*)
show *?case*
proof (*rule goal3*(*5*)[*OF goal3*(*6*)])
assume *A*: $poly\ p\ x \neq 0\ poly\ q\ x = 0$
from *goal3*(*6*) **have** *quasi-sturm-seq* (*r#ps*)
by (*force dest*: *quasi-sturm-seq-Cons*)
with *goal3 A* **show** *P* (*r # ps*) *x* **by** *blast*
next
assume *A*: $poly\ q\ x \neq 0$
from *goal3*(*6*) **have** *quasi-sturm-seq* (*q#r#ps*)
by (*force dest*: *quasi-sturm-seq-Cons*)
with *goal3 A* **show** *P* (*q # r # ps*) *x* **by** *blast*
next
assume *A*: $poly\ p\ x = 0$
from *goal3*(*6*) **have** *quasi-sturm-seq* (*q#r#ps*)
by (*force dest*: *quasi-sturm-seq-Cons*)
with *goal3 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 $poly\ (hd\ ps)\ x_0 \neq 0$
defines $sign\text{-}changes' \equiv \lambda ps\ x.$
$\sum ps'{\leftarrow}split\text{-}sign\text{-}changes\ ps\ x.\ sign\text{-}changes\ ps'\ x$
shows $sign\text{-}changes'\ ps\ x_0 = sign\text{-}changes\ ps\ x_0$
using *assms*(*1*)
proof (*induction* x_0 *rule*: *split-sign-changes-induct*)
case (*goal3 p q r ps* x_0)
hence $poly\ p\ x_0 \neq 0$ **by** *simp*
note *IH* = *goal3*(*2*,*3*,*4*)
show *?case*
proof (*cases* $poly\ q\ x_0 = 0$)
case *True*
from *goal3* **interpret** *quasi-sturm-seq p#q#r#ps* **by** *simp*
from *signs*[*of 0*] **and** *True* **have**
sgn-r-x0: $poly\ r\ x_0 * poly\ p\ x_0 < 0$ **by** *simp*
with *goal3* **have** $poly\ r\ x_0 \neq 0$ **by** *force*
from *sign-changes-distrib*[*OF this*, *of* [*p*,*q*] *ps*]
have *sign-changes* (*p#q#r#ps*) x_0 =
sign-changes ([*p*, *q*, *r*]) x_0 + *sign-changes* (*r # ps*) x_0 **by** *simp*
also have *sign-changes* (*r#ps*) x_0 = *sign-changes'* (*r#ps*) x_0
using ‹$poly\ q\ x_0 = 0$› ‹$poly\ p\ x_0 \neq 0$› *goal3*(*5*)‹$poly\ r\ x_0 \neq 0$›

by (*intro IH*(*1*)[*symmetric*], *simp-all*)
finally show *?thesis* **unfolding** *sign-changes'-def*
using *True* ‹*poly p* x_0 ≠ *0*› **by** *simp*
next
case *False*
from *sign-changes-distrib*[*OF this*, *of* [*p*] *r*#*ps*]
have *sign-changes* (*p*#*q*#*r*#*ps*) x_0 =
sign-changes ([*p*,*q*]) x_0 + *sign-changes* (*q*#*r*#*ps*) x_0 **by** *simp*
also have *sign-changes* (*q*#*r*#*ps*) x_0 = *sign-changes'* (*q*#*r*#*ps*) x_0
using ‹*poly q* x_0 ≠ *0*› ‹*poly p* x_0 ≠ *0*› *goal3*(*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*)

lemma (**in** *quasi-sturm-seq*) *split-sign-changes-correct-nbh*:
assumes *poly* (*hd ps*) x_0 ≠ *0*
defines *sign-changes'* ≡ λx_0 *ps x*.
∑*ps'*←*split-sign-changes ps* x_0. *sign-changes ps' x*
shows *eventually* (λ*x*. *sign-changes'* x_0 *ps x* = *sign-changes ps x*) (*at* x_0)
proof (*rule eventually-mono*)
case *goal1*
let *?ps-nz* = {*p* ∈ *set ps*. *poly p* x_0 ≠ *0*}
show *eventually* (λ*x*. ∀*p*∈*?ps-nz*. *sgn* (*poly p x*) = *sgn* (*poly p* x_0)) (*at* x_0)
by (*rule eventually-Ball-finite*, *auto intro*: *poly-neighbourhood-same-sign*)

show ∀*x*. (∀*p*∈{*p* ∈ *set ps*. *poly p* x_0 ≠ *0*}. *sgn* (*poly p x*) = *sgn* (*poly p* x_0))
⟶
sign-changes' x_0 *ps x* = *sign-changes ps x*
proof (*clarify*)
fix *x* **assume** *nbh*: ∀*p*∈*?ps-nz*. *sgn* (*poly p x*) = *sgn* (*poly p* x_0)
thus *sign-changes'* x_0 *ps x* = *sign-changes ps x* **using** *assms*(*1*)
proof (*induction* x_0 *rule*: *split-sign-changes-induct*)
case (*goal3 p q r ps* x_0)
hence *poly p* x_0 ≠ *0* **by** *simp*
note *IH* = *goal3*(*2*,*3*,*4*)
show *?case*
proof (*cases poly q* x_0 = *0*)
case *True*
from *goal3* **interpret** *quasi-sturm-seq p*#*q*#*r*#*ps* **by** *simp*
from *signs*[*of 0*] **and** *True* **have**
sgn-r-x0: *poly r* x_0 ∗ *poly p* x_0 < *0* **by** *simp*
with *goal3* **have** *poly r* x_0 ≠ *0* **by** *force*
with *nbh goal3*(*5*) **have** *poly r x* ≠ *0* **by** (*auto simp*: *sgn-zero-iff*)
from *sign-changes-distrib*[*OF this*, *of* [*p*,*q*] *ps*]
have *sign-changes* (*p*#*q*#*r*#*ps*) *x* =

sign-changes ([*p*, *q*, *r*]) *x* + *sign-changes* (*r* # *ps*) *x* **by** *simp*
also have *sign-changes* (*r*#*ps*) *x* = *sign-changes′* x_0 (*r*#*ps*) *x*
using ‹*poly q* x_0 = *0*› *nbh* ‹*poly p* $x_0 \neq$ *0*› *goal3*(*5*)‹*poly r* $x_0 \neq$ *0*›
by (*intro IH*(*1*)[*symmetric*], *simp-all*)
finally show *?thesis* **unfolding** *sign-changes′-def*
using *True* ‹*poly p* $x_0 \neq$ *0*›**by** *simp*
next
case *False*
with *nbh goal3*(*5*) **have** *poly q x* $\neq$ *0* **by** (*auto simp*: *sgn-zero-iff*)
from *sign-changes-distrib*[*OF this*, *of* [*p*] *r*#*ps*]
have *sign-changes* (*p*#*q*#*r*#*ps*) *x* =
sign-changes ([*p*,*q*]) *x* + *sign-changes* (*q*#*r*#*ps*) *x* **by** *simp*
also have *sign-changes* (*q*#*r*#*ps*) *x* = *sign-changes′* x_0 (*q*#*r*#*ps*) *x*
using ‹*poly q* $x_0 \neq$ *0*› *nbh* ‹*poly p* $x_0 \neq$ *0*› *goal3*(*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 *poly* (*hd ps*) $x_0 \neq$ *0* **and** *ps′* $\in$ *set* (*split-sign-changes ps* x_0)
shows *eventually* (λ*x*. *sign-changes ps′ x* = *sign-changes ps′* x_0) (*at* x_0)
using *assms*
proof (*induction* x_0 *rule*: *split-sign-changes-induct*)
case (*goal1 p x*)
thus *?case* **by** (*simp add*: *sign-changes-def*)
next
case (*goal2 p q* x_0)
hence [*simp*]: *ps′* = [*p*,*q*] **by** *simp*
from *goal2* **have** *poly p* $x_0 \neq$ *0* **by** *simp*
from *goal2*(*1*) **interpret** *quasi-sturm-seq* [*p*,*q*] **.**
from *poly-neighbourhood-same-sign*[*OF* ‹*poly p* $x_0 \neq$ *0*›]
have *eventually* (λ*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* (λ*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*]
same-signs-imp-same-sign-changes′)
next
case (*goal3 p q r ps″* x_0)
hence *p-not-0*: *poly p* $x_0 \neq$ *0* **by** *simp*
note *sturm* = *goal3*(*1*)
note *IH* = *goal3*(*2*,*3*)

```
note ps''-props = goal3(6)
show ?case
proof (cases poly q x₀ = 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 x₀ * poly p x₀ < 0 by simp
    with p-not-0 have r-not-0: poly r x₀ ≠ 0 by force
    show ?thesis
    proof (cases ps' ∈ set (split-sign-changes (r # ps'') x₀))
      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 x₀ * poly p x₀ < 0 by simp
        from p-not-0 sgn-r
          have A: eventually (λx. sgn (poly p x) = sgn (poly p x₀) ∧
                              sgn (poly r x) = sgn (poly r x₀)) (at x₀)
              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 x₀)
                   and B: sgn (poly r x) = sgn (poly r x₀)
          have prod-neg: ⋀a (b::real). ⟦a>0; b>0; a*b<0⟧ ⟹ False
                         ⋀a (b::real). ⟦a<0; b<0; a*b<0⟧ ⟹ False
              by (drule mult-pos-pos, simp, simp,
                  drule mult-neg-neg, simp, simp)
          from A and ‹poly p x₀ ≠ 0› show poly p x ≠ 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 poly r x ≠ 0
              by (metis sgn-less sgn-times, 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: split-if-asm, intro conjI impI)
              using prod-neg[of poly r x poly p x] apply force+
              done

          show 1 = sign-changes ps' x₀
              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*, *clarify*)
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 same-signs-imp-same-sign-changes′*)
next
show *eventually* (λ*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

lemma (**in** *quasi-sturm-seq*) *hd-nonzero-imp-sign-changes-const*:
assumes *poly* (*hd ps*) x_0 $\neq$ *0*
shows *eventually* (λ*x*. *sign-changes ps x* = *sign-changes ps* x_0) (*at* x_0)
proof−
let *?pss* = *split-sign-changes ps* x_0
let *?f* = λ*pss x*. $\sum$ *ps′*$\leftarrow$*pss*. *sign-changes ps′ x*
{
fix *pss* **assume** $\bigwedge$*ps′*. *ps′*$\in$*set pss* $\Longrightarrow$
eventually (λ*x*. *sign-changes ps′ x* = *sign-changes ps′* x_0) (*at* x_0)
hence *eventually* (λ*x*. *?f pss x* = *?f pss* x_0) (*at* x_0)
proof (*induction pss*)
case (*Cons ps′ pss*)
have $\forall$ *x*. *?f pss x* = *?f pss* x_0 $\wedge$ *sign-changes ps′ x* = *sign-changes ps′* x_0
$\longrightarrow$ *?f* (*ps′*#*pss*) *x* = *?f* (*ps′*#*pss*) x_0 **by** *simp*
note *A* = *eventually-mono*[*OF this eventually-conj*]
show *?case* **by** (*rule A*, *simp-all add*: *Cons*)
qed *simp*
}
note *A* = *this*[*of ?pss*]
have *B*: *eventually* (λ*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

hide-fact *quasi-sturm-seq.hd-nonzero-imp-sign-changes-const-aux*

If x is not a root of p, the number of sign changes of the sequence remains constant in the neighbourhood of x.

lemma (**in** *sturm-seq*) *p-nonzero-imp-sign-changes-const*:
poly p $x_0 \neq 0 \Longrightarrow$
eventually ($\lambda x.$ *sign-changes ps x* = *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 decreases by 1 at x.

lemma (**in** *sturm-seq-squarefree*) *p-zero*:
assumes *poly p* $x_0 = 0$ $p \neq 0$
shows *eventually* ($\lambda x.$ *sign-changes ps x* =
sign-changes ps x_0 + (*if* $x<x_0$ *then 1 else 0*)) (*at* x_0)
proof−
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* $x_0 \neq 0$ **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 ‹*poly q* $x_0 \neq 0$› **have** *eventually* ($\lambda x.$ *sign-changes* (*q#ps′*) *x* =
sign-changes (*q#ps′*) x_0) (*at* x_0)
using *hd-nonzero-imp-sign-changes-const*[**where** x_0=x_0] **by** *simp*
}
moreover note *poly-neighbourhood-without-roots*[*OF assms*(*2*)] *deriv*[*OF assms*(*1*)]
ultimately
have *A*: *eventually* ($\lambda x.$ $x \neq x_0$ ∧ *poly p x* $\neq 0$ ∧
sgn (*poly* (*p*∗*ps*!*1*) *x*) = (*if* $x > x_0$ *then 1 else* −*1*) ∧
sign-changes (*q#ps′*) *x* = *sign-changes* (*q#ps′*) x_0) (*at* x_0)
by (*simp only*: ‹*ps*!*1* = *q*›, *intro eventually-conj*)
show *?thesis*
proof (*rule eventually-mono*[*OF - A*], *clarify*)
case (*goal1 x*)
from *zero-less-mult-pos* **have** *zero-less-mult-pos′*:
$\bigwedge a\ b.$ ⟦(*0*::*real*) < *a*∗*b*; *0* < *b*⟧ $\Longrightarrow$ *0* < *a*
by (*subgoal-tac a*∗*b* = *b*∗*a*, *auto*)
from *goal1* **have** *poly q x* $\neq 0$ **and** *q-sgn*: *sgn* (*poly q x*) =
(*if* $x < x_0$ *then* −*sgn* (*poly p x*) *else sgn* (*poly p x*))

by (*auto simp add*: *sgn-real-def elim*: *linorder-neqE-linordered-idom*
dest: *mult-pos-pos mult-neg-neg zero-less-mult-pos*
zero-less-mult-pos′ split: *split-if-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* $x<x_0$ *then 1 else 0*)
by (*simp add*: *sign-changes-def sgn-zero-iff split*: *split-if-asm*)
also note *goal1*(*4*)
also from *assms*(*1*) **have** *sign-changes* (*q*#*ps′*) x_0 = *sign-changes ps* x_0
by (*simp add*: *sign-changes-def*)
finally show *?case* **by** *simp*
qed
qed

lemma *count-roots-between-aux*:
assumes *a* ≤ *b*
assumes ∀*x*::*real*. *a* < *x* ∧ *x* ≤ *b* ⟶ *eventually* (λξ. *f* ξ = (*f x*::*nat*)) (*at x*)
shows ∀*x*. *a* < *x* ∧ *x* ≤ *b* ⟶ *f x* = *f b*
proof (*clarify*)
fix *x* **assume** *x* > *a x* ≤ *b*
with *assms* **have** ∀*x′*. *x* ≤ *x′* ∧ *x′* ≤ *b* ⟶
eventually (λξ. *f* ξ = *f x′*) (*at x′*) **by** *auto*
from *natfun-eq-in-ivl*[*OF* ‹*x* ≤ *b*› *this*] **show** *f x* = *f b* **.**
qed

If p is non-constant, the number of roots in an interval can be computed by the number of sign changes of the sequence at the border of the interval.

lemma (**in** *sturm-seq-squarefree*) *count-roots-between*:
assumes [*simp*]: *p* ≠ *0 a* ≤ *b*
shows *sign-changes ps a* − *sign-changes ps b* =
card {*x*. *x* > *a* ∧ *x* ≤ *b* ∧ *poly p x* = *0*}
proof−
have *sign-changes ps a* − *int* (*sign-changes ps b*) =
card {*x*. *x* > *a* ∧ *x* ≤ *b* ∧ *poly p x* = *0*} **using** ‹*a* ≤ *b*›
proof (*induction card* {*x*. *x* > *a* ∧ *x* ≤ *b* ∧ *poly p x* = *0*} *arbitrary*: *a b*
rule: *less-induct*)
case (*less a b*)
show *?case*
proof (*cases* ∃*x*. *a* < *x* ∧ *x* ≤ *b* ∧ *poly p x* = *0*)
case *False*
hence *no-roots*: {*x*. *a* < *x* ∧ *x* ≤ *b* ∧ *poly p x* = *0*} = {}
(**is** *?roots*=-) **by** *auto*
hence *card-roots*: *card ?roots* = (*0*::*int*) **by** (*subst no-roots*, *simp*)
show *?thesis*
proof (*simp only*: *card-roots eq-iff-diff-eq-0*[*symmetric*] *int-int-eq*,
cases poly p a = *0*)

```
      case False
        with no-roots show sign-changes ps a = sign-changes ps b
          by (force intro: natfun-eq-in-ivl ‹a ≤ b›
                      p-nonzero-imp-sign-changes-const)
    next
      case True
       have A: ∀ x. a < x ∧ x ≤ b ⟶ sign-changes ps x = 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 (λx. x > a ⟶
              sign-changes ps x = sign-changes ps a) (at a)
          apply (rule eventually-mono) defer
          apply (rule p-zero[OF ‹poly p a = 0› ‹p ≠ 0›], force)
          done
        then obtain δ where δ-props:
          δ > 0 ∀ x. x > a ∧ x < a+δ ⟶
                      sign-changes ps x = sign-changes ps a
          by (auto simp: eventually-at dist-real-def)

        show sign-changes ps a = sign-changes ps b
        proof (cases a = b)
          case False
          def x ≡ min (a+δ/2) b
          with False have a < x x < a+δ x ≤ b
            using ‹δ > 0› ‹a ≤ b› by simp-all
          from δ-props ‹a < x› ‹x < a+δ›
            have sign-changes ps a = sign-changes ps x by simp
          also from A ‹a < x› ‹x ≤ b› have ... = 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 ∧ x ≤ b ∧ poly p x = 0} ≠ {} by blast
    with fin have card-greater-0:
        card {x. x > a ∧ x ≤ b ∧ poly p x = 0} > 0 by fastforce

    def x ≡ Min {x. x > a ∧ x ≤ b ∧ poly p x = 0}
    from Min-in[OF fin] and True
        have x-props: x > a x ≤ b poly p x = 0
        unfolding x-def by blast+
    from Min-le[OF fin] x-props
```

```
      have x-le: ⋀x'. ⟦x' > a; x' ≤ b; poly p x' = 0⟧ ⟹ x ≤ x'
        unfolding x-def by simp

    have left: {x'. a < x' ∧ x' ≤ x ∧ poly p x' = 0} = {x}
      using x-props x-le by force
    hence [simp]: card {x'. a < x' ∧ x' ≤ x ∧ poly p x' = 0} = 1 by simp

    from p-zero[OF ‹poly p x = 0› ‹p ≠ 0›,
        unfolded eventually-at dist-real-def] guess ε ..
    hence ε-props: ε > 0
        ∀x'. x' ≠ x ∧ |x' − x| < ε ⟶
            sign-changes ps x' = sign-changes ps x +
                (if x' < x then 1 else 0) by auto
    def x' ≡ max (x − ε / 2) a
    have |x' − x| < ε using ‹ε > 0› x-props by (simp add: x'-def)
    hence sign-changes ps x' =
        (if x' < x then sign-changes ps x + 1 else sign-changes ps x)
        using ε-props(2) by (cases x' = x, simp, force)
    hence sign-changes ps x' − sign-changes ps x = 1
        unfolding x'-def using x-props ‹ε > 0› by simp

    also have x ∉ {x''. a < x'' ∧ x'' ≤ x' ∧ poly p x'' = 0}
        unfolding x'-def using ‹ε > 0› by force
    with left have {x''. a < x'' ∧ x'' ≤ x' ∧ poly p x'' = 0} = {}
        by force
    with less(1)[of a x'] have sign-changes ps x' = sign-changes ps a
        unfolding x'-def ‹ε > 0› by (force simp: card-greater-0)

    finally have signs-left:
        sign-changes ps a − int (sign-changes ps x) = 1 by simp

    have {x. x > a ∧ x ≤ b ∧ poly p x = 0} =
          {x'. a < x' ∧ x' ≤ x ∧ poly p x' = 0} ∪
          {x'. x < x' ∧ x' ≤ b ∧ poly p x' = 0} using x-props by auto
    also note left
    finally have A: card {x'. x < x' ∧ x' ≤ b ∧ poly p x' = 0} + 1 =
        card {x. a < x ∧ x ≤ b ∧ poly p x = 0} using fin by simp
    hence card {x'. x < x' ∧ x' ≤ b ∧ poly p x' = 0} <
          card {x. a < x ∧ x ≤ b ∧ poly p x = 0} by simp
    from less(1)[OF this x-props(2)] and A
        have signs-right: sign-changes ps x − int (sign-changes ps b) + 1 =
            card {x'. x' > a ∧ x' ≤ b ∧ poly p x' = 0} by simp

    from signs-left and signs-right show ?thesis by simp
  qed
qed
thus ?thesis by simp
qed
```

The number of sign changes in the limits of the polynomials to positive (resp. negative) infinity can be used to compute the number of roots above or below a certain number, or the total number.

lemma (**in** *sturm-seq-squarefree*) *count-roots-above*:
  assumes $p \neq 0$
  shows *sign-changes ps a* $-$ *sign-changes-inf ps* $=$
    card $\{x.\ x > a \wedge$ *poly p x* $= 0\}$
proof−
  have $p \in$ *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** *poly p x* = *0* **hence** $x \leq$ *?u*
   using *lu-props*(*3*)[*OF* ‹$p \in$ *set ps*›, *of x*] ‹$p \neq 0$›
     by (*cases* $u \leq x$, *auto simp*: *sgn-zero-iff*)
  } **note** [*simp*] = *this*

  from *lu-props*
   have *map* ($\lambda p.$ *sgn* (*poly p ?u*)) *ps* = *map poly-inf ps* **by** *simp*
  hence *sign-changes ps a* $-$ *sign-changes-inf ps* $=$
    sign-changes ps a $-$ *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 ... = *card* $\{x.\ a < x \wedge x \leq$ *?u* $\wedge$ *poly p x* $= 0\}$ **by** *simp*
  also have $\{x.\ a < x \wedge x \leq$ *?u* $\wedge$ *poly p x* $= 0\} = \{x.\ a < x \wedge$ *poly p x* $= 0\}$
   using *lu-props* **by** *auto*
  finally show *?thesis* **.**
qed

lemma (**in** *sturm-seq-squarefree*) *count-roots-below*:
  assumes $p \neq 0$
  shows *sign-changes-neg-inf ps* $-$ *sign-changes ps a* $=$
    card $\{x.\ x \leq a \wedge$ *poly p x* $= 0\}$
proof−
  have $p \in$ *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 *?l* = *min a l*
  {**fix** *x* **assume** *poly p x* = *0* **hence** $x >$ *?l*
   using *lu-props*(*4*)[*OF* ‹$p \in$ *set ps*›, *of x*] ‹$p \neq 0$›
     by (*cases* $l < x$, *auto simp*: *sgn-zero-iff*)
  } **note** [*simp*] = *this*

  from *lu-props*
   have *map* ($\lambda p.$ *sgn* (*poly p ?l*)) *ps* = *map poly-neg-inf ps* **by** *simp*
  hence *sign-changes-neg-inf ps* $-$ *sign-changes ps a* $=$
    sign-changes ps ?l $-$ *sign-changes ps a*
   by (*simp-all only*: *sign-changes-def sign-changes-neg-inf-def*)

```
  also from count-roots-between[OF assms] lu-props
      have ... =  card {x. ?l < x ∧ x ≤ a ∧ poly p x = 0} by simp
  also have {x. ?l < x ∧ x ≤ a ∧ poly p x = 0} = {x. a ≥ x ∧ poly p x = 0}
      using lu-props by auto
  finally show ?thesis .
qed

lemma (in sturm-seq-squarefree) count-roots:
  assumes p ≠ 0
  shows sign-changes-neg-inf ps − sign-changes-inf ps =
           card {x. poly p x = 0}
proof−
  have finite (set ps) by simp
  from polys-inf-sign-thresholds[OF this] guess l u .
  note lu-props = this

  from lu-props
    have map (λp. sgn (poly p l)) ps = map poly-neg-inf ps
         map (λp. sgn (poly p u)) ps = map poly-inf ps by simp-all
  hence sign-changes-neg-inf ps − sign-changes-inf ps =
           sign-changes ps l − 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-props
      have ... =  card {x. l < x ∧ x ≤ u ∧ poly p x = 0} by simp
  also have {x. l < x ∧ x ≤ u ∧ poly p x = 0} = {x. poly p x = 0}
      using lu-props assms by simp
  finally show ?thesis .
qed
```

4 Canonical Sturm sequence

```
lemma degree-mod-less': degree q ≠ 0 ⟹ degree (p mod q) < degree q
  using assms degree-mod-less by force

function sturm-aux where
sturm-aux (p :: real poly) q =
    (if degree q = 0 then [p,q] else p # sturm-aux q (−(p mod q)))
  by (pat-completeness, simp-all)
termination by (relation measure (degree ∘ snd),
               simp-all add: o-def degree-mod-less')

declare sturm-aux.simps[simp del]

definition sturm where sturm p = sturm-aux p (pderiv p)

lemma sturm-0[simp]: sturm 0 = [0,0]
    by (unfold sturm-def, subst sturm-aux.simps, simp)
```

```
lemma [simp]: sturm-aux p q = [] ⟷ False
   by (induction p q rule: sturm-aux.induct, subst sturm-aux.simps, auto)

lemma sturm-neq-Nil[simp]: sturm p ≠ [] unfolding sturm-def by simp

lemma [simp]: hd (sturm p) = p
  unfolding sturm-def by (subst sturm-aux.simps, simp)

lemma [simp]: p ∈ set (sturm p)
  using hd-in-set[OF sturm-neq-Nil] by simp

lemma [simp]: length (sturm p) ≥ 2
proof–
  {fix q have length (sturm-aux p q) ≥ 2
         by (induction p q rule: sturm-aux.induct, subst sturm-aux.simps, simp)
  }
  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 < length (sturm p) − 2
  shows sturm p!(i+2) = −(sturm p!i mod sturm p!(i+1))
proof–
 {fix ps q
  have ⟦ps = sturm-aux p q; i < length ps − 2⟧
          ⟹ ps!(i+2) = −(ps!i mod ps!(i+1))
  proof (induction p q arbitrary: ps i rule: sturm-aux.induct)
    case (goal1 p q)
      show ?case
      proof (cases i = 0)
        case False
```

then obtain i' **where** [*simp*]: $i = Suc\ i'$ **by** (*cases* i, *simp-all*)
hence *length ps* ≥ 4 **using** *goal1* **by** *simp*
with *goal1*(*2*) **have** *deg*: *degree* $q \neq 0$
by (*subst* (*asm*) *sturm-aux.simps*, *simp split*: *split-if-asm*)
with *goal1*(*2*) **obtain** ps' **where** [*simp*]: $ps = p\ \#\ ps'$
by (*subst* (*asm*) *sturm-aux.simps*, *simp*)
with *goal1*(*2*) *deg* **have** ps': $ps' = sturm\text{-}aux\ q\ (-(p\ mod\ q))$
by (*subst* (*asm*) *sturm-aux.simps*, *simp*)
from ‹*length ps* ≥ 4› **and** ‹$ps = p\ \#\ ps'$›*goal1*(*3*) *False*
have $i - 1 < length\ ps' - 2$ **by** *simp*
from *goal1*(*1*)[*OF deg ps' this*]
show *?thesis* **by** *simp*
next
case *True*
with *goal1*(*3*) **have** *length ps* ≥ 3 **by** *simp*
with *goal1*(*2*) **have** *degree* $q \neq 0$
by (*subst* (*asm*) *sturm-aux.simps*, *simp split*: *split-if-asm*)
with *goal1*(*2*) **have** [*simp*]: $sturm\text{-}aux\ p\ q\ !\ Suc\ (Suc\ 0) = -(p\ mod\ q)$
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*: *goal1*(*2*))
thus *?thesis* **by** *simp*
qed
qed}
from *this*[*OF sturm-def assms*] **show** *?thesis* **.**
qed

lemma *sturm-aux-gcd*: $r \in set\ (sturm\text{-}aux\ p\ q) \Longrightarrow gcd\ p\ q\ dvd\ r$
proof (*induction p q rule*: *sturm-aux.induct*)
case (*goal1 p q*)
show *?case*
proof (*cases* $r = p$)
case *False*
with *goal1*(*2*) **have** *r*: $r \in set\ (sturm\text{-}aux\ q\ (-(p\ mod\ q)))$
by (*subst* (*asm*) *sturm-aux.simps*, *simp split*: *split-if-asm*,
subst sturm-aux.simps, *simp*)
show *?thesis*
proof (*cases degree* $q = 0$)
case *False*
hence $q \neq 0$ **by** *force*
from *goal1*(*1*)[*OF False r*] **show** *?thesis*
by (*subst gcd-poly.simps*(*2*)[*OF* ‹$q \neq 0$›], *simp*)
next
case *True*
with *goal1*(*2*) **and** ‹$r \neq p$› **have** $r = q$
by (*subst* (*asm*) *sturm-aux.simps*, *simp*)
thus *?thesis* **by** *simp*
qed
qed *simp*

qed

lemma *sturm-gcd*: $r \in set\ (sturm\ p) \Longrightarrow gcd\ p\ (pderiv\ p)\ dvd\ r$
  unfolding *sturm-def* **by** (*rule sturm-aux-gcd*)

lemma *sturm-adjacent-root-propagate-left*:
 assumes $i < length\ (sturm\ (p :: real\ poly)) - 1$
 assumes $poly\ (sturm\ p\ !\ i)\ x = 0$
  and $poly\ (sturm\ p\ !\ (i + 1))\ x = 0$
 shows $\forall j{\le}i{+}1.\ poly\ (sturm\ p\ !\ j)\ x = 0$
using *assms*(*2*)
proof (*intro sturm-adjacent-root-aux*[*OF assms*(*1*,*2*,*3*)])
 case (*goal1 i x*)
  let $?p = sturm\ p\ !\ i$
  let $?q = sturm\ p\ !\ (i + 1)$
  let $?r = sturm\ p\ !\ (i + 2)$
  from *sturm-indices*[*OF goal1*(*2*)] **have** $?p = ?p\ div\ ?q * ?q - ?r$
   by (*simp add*: *mod-div-equality*)
  hence $poly\ ?p\ x = poly\ (?p\ div\ ?q * ?q - ?r)\ x$ **by** *simp*
  hence $poly\ ?p\ x = -poly\ ?r\ x$ **using** *goal1*(*3*) **by** *simp*
  thus *?case* **by** (*simp add*: *sgn-minus*)
qed

lemma *sturm-adjacent-root-not-squarefree*:
 assumes $i < length\ (sturm\ (p :: real\ poly)) - 1$
   $poly\ (sturm\ p\ !\ i)\ x = 0$ $poly\ (sturm\ p\ !\ (i + 1))\ x = 0$
 shows $\neg rsquarefree\ p$
proof−
 from *sturm-adjacent-root-propagate-left*[*OF assms*]
  have $poly\ p\ x = 0$ $poly\ (pderiv\ p)\ x = 0$ **by** *auto*
 thus *?thesis* **by** (*auto simp*: *rsquarefree-roots*)
qed

lemma *sturm-firsttwo-signs-aux*:
 assumes $(p :: real\ poly) \neq 0$ $q \neq 0$
 assumes *q-pderiv*:
  $eventually\ (\lambda x.\ sgn\ (poly\ q\ x) = sgn\ (poly\ (pderiv\ p)\ x))\ (at\ x_0)$
 assumes *p-0*: $poly\ p\ (x_0{::}real) = 0$
 shows $eventually\ (\lambda x.\ sgn\ (poly\ (p{*}q)\ x) = (if\ x > x_0\ then\ 1\ else\ -1))\ (at\ x_0)$
proof−
 have *A*: $eventually\ (\lambda x.\ poly\ p\ x \neq 0 \land poly\ q\ x \neq 0 \land$
    $sgn\ (poly\ q\ x) = sgn\ (poly\ (pderiv\ p)\ x))\ (at\ x_0)$
  using ‹$p \neq 0$› ‹$q \neq 0$›
  by (*intro poly-neighbourhood-same-sign q-pderiv*
    poly-neighbourhood-without-roots eventually-conj)
 then obtain ε **where** *ε-props*: $\varepsilon > 0$ $\forall x.\ x \neq x_0 \land |x - x_0| < \varepsilon \longrightarrow$

$poly\ p\ x \neq 0 \wedge poly\ q\ x \neq 0 \wedge sgn\ (poly\ (pderiv\ p)\ x) = sgn\ (poly\ q\ x)$
by (*auto simp*: *eventually-at dist-real-def*)
have *sqr-pos*: $\bigwedge x{::}real.\ x \neq 0 \Longrightarrow sgn\ x * sgn\ x = 1$
by (*auto simp*: *sgn-real-def*)

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

lemma *sturm-firsttwo-signs*:
fixes *ps* :: *real poly list*

assumes *squarefree*: *rsquarefree p*
assumes *p-0*: *poly p* $(x_0$::*real*$) = 0$
shows *eventually* (λ*x*. *sgn* (*poly* (*p* ∗ *sturm p* ! *1*) *x*) =
(*if* $x > x_0$ *then 1 else* −*1*)) (*at* x_0)
proof−
from *assms* **have** [*simp*]: *p* ≠ *0* **by** (*auto simp add*: *rsquarefree-roots*)
with *squarefree p-0* **have** [*simp*]: *pderiv p* ≠ *0*
by (*auto simp add*:*rsquarefree-roots*)
from *assms* **show** *?thesis*
by (*intro sturm-firsttwo-signs-aux*,
simp-all add: *rsquarefree-roots*)
qed

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 *mod-div-equality*[*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*: ⋀*x*::*real*. *x* ≠ *0* ⟹ *x* ∗ *x* > *0* **apply** (*case-tac x* ≥ *0*)
by (*simp-all add*: *mult-pos-pos 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

If *p* contains no multiple roots, *sturm p*, i.e. the canonical Sturm sequence for p, is a squarefree Sturm sequence that can be used to determine the number of roots of *p*.

lemma *sturm-seq-sturm*[*simp*]:
assumes *rsquarefree p*
shows *sturm-seq-squarefree* (*sturm p*) *p*
proof
show *sturm p* ≠ [] **by** *simp*
show *hd* (*sturm p*) = *p* **by** *simp*
show *length* (*sturm p*) ≥ *2* **by** *simp*
from *assms* **show** ⋀*x*. ¬(*poly p x* = *0* ∧ *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*: *split-if-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 $\bigwedge x_0$. *poly p* x_0 = *0* $\Longrightarrow$
      eventually (λx. *sgn* (*poly* (*p***sturm p* ! *1*) *x*) =
        (*if x* > x_0 *then 1 else* −*1*)) (*at* x_0) **by** *simp*
next
  from *sturm-signs*[*OF assms*]
    show $\bigwedge i\ x$. ⟦*i* < *length* (*sturm p*) − *2*; *poly* (*sturm p* ! (*i* + *1*)) *x* = *0*⟧
      $\Longrightarrow$ *poly* (*sturm p* ! (*i* + *2*)) *x* * *poly* (*sturm p* ! *i*) *x* < *0* **by** *simp*
qed

5 Canonical squarefree Sturm sequence

This removes multiple roots from *p* by dividing it by its gcd with its derivative. The resulting polynomials has the same roots as *p*, but with multiplicity 1.

definition *sturm-squarefree* **where**
  sturm-squarefree p = *sturm* (*p div* (*gcd p* (*pderiv p*)))

lemma *sturm-squarefree-not-Nil*[*simp*]: *sturm-squarefree p* ≠ []
  by (*simp add*: *sturm-squarefree-def*)

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

  from *sturm-seq-sturm*[*OF* ‹*rsquarefree p'*›]
    interpret *sturm-seq*: *sturm-seq-squarefree sturm-squarefree p p'*
    by (*simp add*: *sturm-squarefree-def*)

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

have [*simp*]: $sturm\text{-}squarefree\ p\ !\ 0 = p'$
  $sturm\text{-}squarefree\ p\ !\ Suc\ 0 = pderiv\ p'$
  by (*simp-all add*: *sturm-squarefree-def*)

from ‹*rsquarefree p'*›
  show $\bigwedge x.\ \neg\ (poly\ p'\ x = 0 \wedge poly\ (sturm\text{-}squarefree\ p\ !\ 1)\ x = 0)$
  by (*simp add*: *rsquarefree-roots*)

from *sturm-seq.signs* **show** $\bigwedge i\ x.\ [\![i < length\ (sturm\text{-}squarefree\ p) - 2;$
  $poly\ (sturm\text{-}squarefree\ p\ !\ (i + 1))\ x = 0]\!]$
  $\Longrightarrow poly\ (sturm\text{-}squarefree\ p\ !\ (i + 2))\ x *$
  $poly\ (sturm\text{-}squarefree\ p\ !\ i)\ x < 0$ **.**

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

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

lemma *sturm-squarefree'-adjacent-root-propagate-left*:
  assumes $p \neq 0$
  assumes $i < length\ (sturm\text{-}squarefree'\ (p :: real\ poly)) - 1$
  assumes $poly\ (sturm\text{-}squarefree'\ p\ !\ i)\ x = 0$
  and $poly\ (sturm\text{-}squarefree'\ p\ !\ (i + 1))\ x = 0$
  shows $\forall j \leq i+1.\ poly\ (sturm\text{-}squarefree'\ p\ !\ j)\ x = 0$
proof (*intro sturm-adjacent-root-aux*[*OF assms*(*2*,*3*,*4*)])
  case (*goal1 i x*)
  def $q \equiv sturm\ p\ !\ i$
  def $r \equiv sturm\ p\ !\ (Suc\ i)$
  def $s \equiv sturm\ p\ !\ (Suc\ (Suc\ i))$
  def $d \equiv gcd\ p\ (pderiv\ p)$
  def $q' \equiv q\ div\ d$ **and** $r' \equiv r\ div\ d$ **and** $s' \equiv s\ div\ d$
  from ‹$p \neq 0$› **have** $d \neq 0$ **unfolding** *d-def* **by** *simp*

```
    from goal1(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 add: dvd-div-mult-self)
    with goal1(2) i-in-range have r′-0: poly r′ x = 0
       unfolding r′-def r-def d-def sturm-squarefree′-def Let-def by simp
    hence r-0: poly r x = 0 by (simp add: ‹r = r′ ∗ d›)
    from sturm-indices[OF i-in-range] have q = q div r ∗ r − s
       unfolding q-def r-def s-def by (simp add: mod-div-equality)
    hence q′ = (q div r ∗ r − s) div d by (simp add: q′-def)
    also have ... = (q div r ∗ r) div d − s′
       unfolding s′-def by (rule div-diff[symmetric], simp-all)
    also have ... = q 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 div r = q′ div r′ by (simp add: qrs-simps ‹d ≠ 0›)
    finally have poly q′ x = poly (q′ div r′ ∗ r′ − s′) x by simp
    also from r′-0 have ... = −poly s′ x by simp
    finally have poly s′ x = −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 ≠ 0
          i < length (sturm-squarefree′ (p :: real poly)) − 1
          poly (sturm-squarefree′ p ! i) x = 0
          poly (sturm-squarefree′ p ! (i + 1)) x = 0
  shows False
proof−
  def d ≡ gcd p (pderiv p)
  from sturm-squarefree′-adjacent-root-propagate-left[OF assms]
     have poly (sturm-squarefree′ p ! 0) x = 0
          poly (sturm-squarefree′ p ! 1) x = 0 by auto
  hence poly (p div d) x = 0 poly (pderiv p div d) x = 0
     using assms(2)
     unfolding sturm-squarefree′-def Let-def d-def by auto
  moreover from div-gcd-coprime-poly 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 ≠ 0
  assumes i-in-range: i < length (sturm-squarefree′ (p :: real poly)) − 2
  assumes q-0: poly (sturm-squarefree′ p ! (i+1)) x = 0 (is poly ?q x = 0)
```

shows *poly* (*sturm-squarefree′ p* ! (*i*+*2*)) *x* ∗
poly (*sturm-squarefree′ p* ! *i*) *x* < *0*
(**is** *poly ?r x* ∗ *poly ?p x* < *0*)
proof−
def *d* ≡ *gcd p* (*pderiv p*)
with ‹*p* ≠ *0*› **have** [*simp*]: *d* ≠ *0* **by** *simp*
from *poly-div-gcd-squarefree*(*1*)[*OF* ‹*p* ≠ *0*›]
coprime-imp-no-common-roots
have *rsquarefree*: *rsquarefree* (*p div d*)
by (*auto simp*: *rsquarefree-roots d-def*)

from *i-in-range* **have** *i-in-range′*: *i* < *length* (*sturm p*) − *2*
unfolding *sturm-squarefree′-def* **by** *simp*
hence *d dvd* (*sturm p* ! *i*) (**is** *d dvd ?p′*)
d dvd (*sturm p* ! (*Suc i*)) (**is** *d dvd ?q′*)
d dvd (*sturm p* ! (*Suc* (*Suc i*))) (**is** *d 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*: *poly ?q′ x* = *0* **by** *simp*
from *sturm-indices*[*OF i-in-range′*]
have *sturm p* ! (*i*+*2*) = − (*sturm p* ! *i mod sturm p* ! (*i*+*1*)) **.**
hence −*?r′* = *?p′ mod ?q′* **by** *simp*
with *mod-div-equality*[*of ?p′ ?q′*] **have** *?p′ div ?q′* ∗ *?q′* − *?r′* = *?p′* **by** *simp*
hence *d*∗(*?p div ?q* ∗ *?q* − *?r*) = *d*∗ *?p* **by** (*simp add*: *pqr-simps algebra-simps*)
hence *?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*

from *sturm-squarefree′-adjacent-roots*[*OF* ‹*p* ≠ *0*›] *i-in-range q-0*
have *poly ?p x* ≠ *0* **by** *force*
moreover have *sqr-pos*: ⋀*x*::*real*. *x* ≠ *0* ⟹ *x* ∗ *x* > *0* **apply** (*case-tac x* ≥
0)
by (*simp-all add*: *mult-pos-pos 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* / *q*.

lemma *sturm-seq-squarefree′*:
assumes (*p* :: *real poly*) ≠ *0*
defines *d* ≡ *gcd p* (*pderiv p*)
shows *sturm-seq-squarefree* (*sturm-squarefree′ p*) (*p div d*)
(**is** *sturm-seq-squarefree ?ps′ ?p′*)
proof
show *?ps′* ≠ [] *hd ?ps′* = *?p′ 2* ≤ *length ?ps′*
by (*simp-all add*: *sturm-squarefree′-def d-def hd-map*)

from *assms* **have** $d \neq 0$ **by** *simp*
{
  have *d dvd last* (*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*)
  moreover from *this* **have** *last ?ps′ dvd last* (*sturm p*) **by** *simp*
  moreover note *dvd-imp-degree-le*[*OF this*]
  ultimately have *degree* (*last ?ps′*) ≤ *degree* (*last* (*sturm p*))
    using ‹$d \neq 0$› **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*: *split-if-asm*)
  thus ⋀*x y*. *sgn* (*poly* (*last ?ps′*) *x*) = *sgn* (*poly* (*last ?ps′*) *y*) **by** *simp*
}

have *squarefree*: *rsquarefree ?p′* **using** ‹$p \neq 0$›
  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-poly* ‹$p \neq 0$› **by** *blast*
thus *squarefree′*:
  ⋀*x*. ¬ (*poly* (*p div d*) *x* = *0* ∧ *poly* (*sturm-squarefree′ p* ! *1*) *x* = *0*)
  using *coprime-imp-no-common-roots* **by** *simp*

from *sturm-squarefree′-signs*[*OF* ‹$p \neq 0$›]
  show ⋀*i x*. ⟦*i* < *length ?ps′* − *2*; *poly* (*?ps′* ! (*i* + *1*)) *x* = *0*⟧
    ⟹ *poly* (*?ps′* ! (*i* + *2*)) *x* ∗ *poly* (*?ps′* ! *i*) *x* < *0* **.**

have [*simp*]: *?p′* ≠ *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* ≠ *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* ‹$p \neq 0$›]
  unfolding *d-def* **by** *simp*
hence *pderiv p* ≠ *0* **using** ‹$p \neq 0$› **by** (*auto dest*: *pderiv-iszero*)
with ‹$p \neq 0$› ‹*poly p* x_0 = *0*›
  have *A*: *eventually* (λ*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* ‹$d \neq 0$›]]

show *eventually* (λx. *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*], *clarify*)
have [*intro*]:
$\bigwedge a$ (*b*::*real*). $b \neq 0 \Longrightarrow a < 0 \Longrightarrow a / (b * b) < 0$
$\bigwedge a$ (*b*::*real*). $b \neq 0 \Longrightarrow a > 0 \Longrightarrow a / (b * b) > 0$
by ((*case-tac* $b > 0$,
auto simp: *mult-pos-pos mult-neg-neg field-simps*) [])+
case (*goal1 x*)
hence [*simp*]: *poly d x* $*$ *poly d x* > 0
by (*cases poly d x* > 0, *auto simp*: *mult-pos-pos mult-neg-neg*)
from *poly-div-gcd-squarefree-aux*(*2*)[*OF* ‹*pderiv p* $\neq 0$›]
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** *goal1*
by (*auto simp*: *sgn-real-def poly-div not-less*[*symmetric*]
zero-less-divide-iff split: *split-if-asm*)
qed
qed

Critically, unless x is a multiple root of p (i.e. a root of both p and its derivative), the number of sign changes in the non-canonical Sturm sequence we defined is the same as the number of sign changes in the canonical Sturm sequence. Therefore we can use the canonical Sturm sequence even in the non-squarefree case if the borders of the interval we are interested in are not multiple roots of the polynomial.

lemma *sign-changes-mult-aux*:
assumes $d \neq$ (*0*::*real*)
shows *length* (*group* (*filter* (λx. $x \neq 0$) (*map* (*op* $*d \circ f$) *xs*))) =
length (*group* (*filter* (λx. $x \neq 0$) (*map f xs*)))
proof−
from *assms* **have** *inj*: *inj* (*op* $*d$) **by** (*auto intro*: *injI*)
from *assms* **have** [*simp*]: *filter* (λx. (*op*$*$ $d \circ f$) $x \neq 0$) = *filter* (λx. $f\, x \neq 0$)
filter ((λx. $x \neq 0$) $\circ f$) = *filter* (λx. $f\, x \neq 0$)
by (*simp-all add*: *o-def*)
have *filter* (λx. $x \neq 0$) (*map* (*op*$*$ $d \circ f$) *xs*) =
map (*op*$*$ $d \circ f$) (*filter* (λx. (*op*$*$ $d \circ f$) $x \neq 0$) *xs*)
by (*simp add*: *filter-map o-def*)
thus *?thesis* **using** *group-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* $\neq 0 \lor$ *poly* (*pderiv p*) $x \neq 0 \Longrightarrow$
sign-changes ps′ *x* = *sign-changes ps x*

$p \neq 0 \Longrightarrow$ *sign-changes-inf ps' = sign-changes-inf ps*
$p \neq 0 \Longrightarrow$ *sign-changes-neg-inf ps' = sign-changes-neg-inf ps*

proof−
def *d* ≡ *gcd p (pderiv p)*
def *p'* ≡ *p div d*
def *s'* ≡ *poly-inf d*
def *s''* ≡ *poly-neg-inf d*

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

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

{
fix *f* :: *real poly* ⇒ *real* **and** *s* :: *real*
assume *f*: $\bigwedge q$. *q* ∈ *set ps* ⟹ *f q = s ∗ f (q div d)* **and** *s*: *s* ≠ *0*
hence *inverse s* ≠ *0* **by** *simp*
{**fix** *q* **assume** *q* ∈ *set ps*
hence *f (q div d) = inverse s ∗ f q*
by (*subst f[of q]*, *simp-all add*: *s*)
} **note** *f' = this*
have *length (group [x←map f (map (λq. q div d) ps). x ≠ 0]) − 1 =*
length (group [x←map (λq. f (q div d)) ps . x ≠ 0]) − 1
by (*simp only*: *sign-changes-def o-def map-map*)
also have *map (λq. q div d) ps = ps'*
by (*simp add*: *ps-def ps'-def sturm-squarefree'-def Let-def d-def*)
also from *f'* **have** *map (λq. f (q div d)) ps =*
map (λx. (op∗(inverse s) ∘ f) x) ps **by** (*simp add*: *o-def*)
also note *sign-changes-mult-aux[OF ‹inverse s ≠ 0›, of f ps]*
finally have
length (group [x←map f ps' . x ≠ 0]) − 1 =
length (group [x←map f ps . x ≠ 0]) − 1 **by** *simp*
}
note *length-group = this*

{

fix x **assume** A: $poly\ p\ x \neq 0 \vee poly\ (pderiv\ p)\ x \neq 0$
have $d\ dvd\ p\ d\ dvd\ pderiv\ p$ **unfolding** *d-def* **by** *simp-all*
with A **have** $sgn\ (poly\ d\ x) \neq 0$
by (*auto simp add*: *sgn-zero-iff elim*: *dvdE*)
thus $sign\text{-}changes\ ps'\ x = sign\text{-}changes\ ps\ x$ **using** *signs*(*1*)
unfolding *sign-changes-def*
by (*intro length-group*[*of* $\lambda q.\ sgn\ (poly\ q\ x)$], *simp-all*)
}

assume $p \neq 0$
hence $d \neq 0$ **unfolding** *d-def* **by** *simp*
hence $s' \neq 0\ s'' \neq 0$ **unfolding** s'*-def* s''*-def* **by** *simp-all*
from *length-group*[*of poly-inf* s', *OF signs*(*2*) ‹$s' \neq 0$›]
show $sign\text{-}changes\text{-}inf\ ps' = sign\text{-}changes\text{-}inf\ ps$
unfolding *sign-changes-inf-def* **.**
from *length-group*[*of poly-neg-inf* s'', *OF signs*(*3*) ‹$s'' \neq 0$›]
show $sign\text{-}changes\text{-}neg\text{-}inf\ ps' = sign\text{-}changes\text{-}neg\text{-}inf\ ps$
unfolding *sign-changes-neg-inf-def* **.**
qed

7 Root-counting functions

definition *count-roots-between* **where**
$count\text{-}roots\text{-}between\ p\ a\ b = (if\ a \leq b \wedge p \neq 0\ then$
$(let\ ps = sturm\text{-}squarefree\ p$
$in\ sign\text{-}changes\ ps\ a - sign\text{-}changes\ ps\ b)\ else\ 0)$

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

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

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

lemma *count-roots-between-correct*:
$count\text{-}roots\text{-}between\ p\ a\ b = card\ \{x.\ a < x \wedge x \leq b \wedge poly\ p\ x = 0\}$
proof (*cases* $p \neq 0 \wedge a \leq b$)
case *False*
note $False' = this$
hence $card\ \{x.\ a < x \wedge x \leq b \wedge poly\ p\ x = 0\} = 0$

```
    proof (cases a < b)
      case True
        with False have [simp]: p = 0 by simp
        have subset: {a<..<b} ⊆ {x. a < x ∧ x ≤ b ∧ poly p x = 0} by auto
        from real-infinite-interval[OF True] have ¬finite {a<..<b} .
        hence ¬finite {x. a < x ∧ x ≤ b ∧ 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 ≠ 0 a ≤ b by simp-all
  def p' ≡ p div (gcd p (pderiv p))
  from poly-div-gcd-squarefree(1)[OF ‹p ≠ 0›] have p' ≠ 0
      unfolding p'-def by clarsimp

  from sturm-seq-squarefree[OF ‹p ≠ 0›]
      interpret sturm-seq-squarefree sturm-squarefree p p'
      unfolding p'-def .
  from poly-roots-finite[OF ‹p' ≠ 0›]
      have finite {x. a < x ∧ x ≤ b ∧ poly p' x = 0} by fast
  have count-roots-between p a b = card {x. a < x ∧ x ≤ b ∧ poly p' x = 0}
      unfolding count-roots-between-def Let-def
      using True count-roots-between[OF ‹p' ≠ 0› ‹a ≤ b›] by simp
  also from poly-div-gcd-squarefree(2)[OF ‹p ≠ 0›]
      have {x. a < x ∧ x ≤ b ∧ poly p' x = 0} =
            {x. a < x ∧ x ≤ b ∧ poly p x = 0} unfolding p'-def by blast
  finally show ?thesis .
qed

lemma count-roots-correct:
  fixes p :: real poly
  shows count-roots p = card {x. poly p x = 0} (is - = card ?S)
proof (cases p = 0)
  case True
    with real-infinite-interval[of 0 1] finite-subset[of {0<..<1} ?S]
        have ¬finite {x. poly p x = 0} by force
    thus ?thesis by (simp add: count-roots-def True)
next
  case False
  def p' ≡ p div (gcd p (pderiv p))
  from poly-div-gcd-squarefree(1)[OF ‹p ≠ 0›] have p' ≠ 0
      unfolding p'-def by clarsimp

  from sturm-seq-squarefree[OF ‹p ≠ 0›]
```

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

lemma *count-roots-above-correct*:
fixes *p* :: *real poly*
shows *count-roots-above p a* = *card* {*x*. *x* > *a* ∧ *poly p x* = *0*}
(**is** - = *card ?S*)
proof (*cases p* = *0*)
case *True*
with *real-infinite-interval*[*of a a*+*1*] *finite-subset*[*of* {*a*<..<*a*+*1*} *?S*]
have ¬*finite* {*x*. *x* > *a* ∧ *poly p x* = *0*} **by** *force*
thus *?thesis* **by** (*simp add*: *count-roots-above-def True*)
next
case *False*
def *p′* ≡ *p div* (*gcd p* (*pderiv p*))
from *poly-div-gcd-squarefree*(*1*)[*OF* ‹*p* ≠ *0*›] **have** *p′* ≠ *0*
unfolding *p′-def* **by** *clarsimp*

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

lemma *count-roots-below-correct*:
fixes *p* :: *real poly*
shows *count-roots-below p a* = *card* {*x*. *x* ≤ *a* ∧ *poly p x* = *0*}
(**is** - = *card ?S*)
proof (*cases p* = *0*)
case *True*
with *real-infinite-interval*[*of a* − *1 a*]
finite-subset[*of* {*a* − *1*<..<*a*} *?S*]
have ¬*finite* {*x*. *x* ≤ *a* ∧ *poly p x* = *0*} **by** *force*
thus *?thesis* **by** (*simp add*: *count-roots-below-def True*)
next
case *False*

def $p' \equiv p\ div\ (gcd\ p\ (pderiv\ p))$
from *poly-div-gcd-squarefree*(*1*)[*OF* ‹$p \neq 0$›] **have** $p' \neq 0$
unfolding *p′-def* **by** *clarsimp*

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

lemma *count-roots-between*[*code*]:
$count\text{-}roots\text{-}between\ p\ a\ b =$
$(let\ q = pderiv\ p$
$in\ if\ a > b \vee p = 0\ then\ 0$
$else\ if\ (poly\ p\ a \neq 0 \vee poly\ q\ a \neq 0) \wedge (poly\ p\ b \neq 0 \vee poly\ q\ b \neq 0)$
$then\ (let\ ps = sturm\ p$
$in\ sign\text{-}changes\ ps\ a - sign\text{-}changes\ ps\ b)$
$else\ (let\ ps = sturm\text{-}squarefree\ p$
$in\ sign\text{-}changes\ ps\ a - sign\text{-}changes\ ps\ b))$
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* $(poly\ p\ a \neq 0 \vee poly\ (pderiv\ p)\ a \neq 0) \wedge$
$(poly\ p\ b \neq 0 \vee poly\ (pderiv\ p)\ b \neq 0))$
case *False*
thus *?thesis* **using** *False1*
by (*auto simp add*: *Let-def count-roots-between-def*)
next
case *True*
hence *A*: $poly\ p\ a \neq 0 \vee poly\ (pderiv\ p)\ a \neq 0$ **and**
B: $poly\ p\ b \neq 0 \vee poly\ (pderiv\ p)\ b \neq 0$ **by** *auto*
def $d \equiv gcd\ p\ (pderiv\ p)$
from ‹$p \neq 0$› **have** [*simp*]: $p\ div\ d \neq 0$
using *poly-div-gcd-squarefree*(*1*)[*OF* ‹$p \neq 0$›] **by** (*auto simp add*: *d-def*)
from *sturm-seq-squarefree′*[*OF* ‹$p \neq 0$›]
interpret *sturm-seq-squarefree 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
    def 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-squarefree'[OF ‹p ≠ 0›]
      interpret sturm-seq-squarefree 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 ≠ 0 by simp-all
    thus ?thesis
    proof (cases (poly p a ≠ 0 ∨ poly (pderiv p) a ≠ 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 ≠ 0 ∨ poly (pderiv p) a ≠ 0 by simp
      def 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-squarefree'[OF ‹p ≠ 0›]
          interpret sturm-seq-squarefree sturm-squarefree' p p div d
          unfolding sturm-squarefree'-def Let-def d-def .
      note count-roots-above-correct
      also have {x. a < x ∧ poly p x = 0} =
                 {x. a < 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-above[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(2)[OF ‹p ≠ 0›]
      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 ≠ 0 ∨ poly q a ≠ 0
            then (let ps = sturm p
                  in sign-changes-neg-inf ps − sign-changes ps a)
            else (let ps = sturm-squarefree p
                  in sign-changes-neg-inf ps − sign-changes ps a))
proof (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 ≠ 0 by simp-all
    thus ?thesis
    proof (cases (poly p a ≠ 0 ∨ poly (pderiv p) a ≠ 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
      def 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-squarefree'[OF ‹p ≠ 0›]
          interpret sturm-seq-squarefree 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
theory Sturm-Method
imports Sturm
begin
```

8 Setup for the sturm method

```
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} =
       (let p = p in 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 assms False'
            by (auto split: split-if-asm)
    next
```

```
      case True
        have {x. a ≤ x ∧ x ≤ b} = {a..b}
             {x. a < x ∧ x ≤ b} = {a<..b} by auto
        with True False show ?thesis
            using count-roots-between-correct
            by (simp add: real-interval-card-eq)
    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} =
      (let p = p in 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 ∧ x ≤ b} = {a<..b} by auto
        from True False' assms show ?thesis
            by (auto simp: count-roots-between-correct real-interval-card-eq)
    next
      case False
        with False' have {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 assms False by auto
  qed
next
```

```
  case True
    with poly-roots-finite
        have fin: finite {x. a < x ∧ x < b ∧ poly p x = 0} by fast
    from True have {x. a < x ∧ x ≤ b ∧ poly p x = 0} =
        insert b {x. a < x ∧ x < b ∧ poly p x = 0} by auto
    hence Suc (card {x. a < x ∧ x < b ∧ poly p x = 0}) =
          card {x. a < x ∧ x ≤ b ∧ 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:
  card {x::real. a ≤ x ∧ x < b ∧ poly p x = 0} =
      (let p = p in count-roots-between p a b +
      (if p ≠ 0 ∧ a < b ∧ poly p a = 0 then 1 else 0) −
      (if p ≠ 0 ∧ a < b ∧ poly p b = 0 then 1 else 0))
proof (cases p = 0 ∨ a ≥ b)
  case True
    note True' = this
    show ?thesis
    proof (cases a ≥ b)
      case False
        hence {x. a < x ∧ x ≤ b} = {a<..b}
              {x. a ≤ x ∧ x < b} = {a..<b} by auto
        with False True' show ?thesis
            by (simp add: count-roots-between-correct real-interval-card-eq)
    next
      case True
        with True' have {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 True by simp
  qed
next
  case False
    let ?A = {x. a ≤ x ∧ x < b ∧ poly p x = 0}
    let ?B = {x. a < x ∧ x ≤ b ∧ poly p x = 0}
    let ?C = {x. x = b ∧ poly p x = 0}
    let ?D = {x. x = a ∧ poly p a = 0}
    have CD-if: ?C = (if poly p b = 0 then {b} else {})
                ?D = (if poly p a = 0 then {a} 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 ∪ ?D) − ?C by (auto simp: less-eq-real-def)
    with False have card ?A = card ?B + (if poly p a = 0 then 1 else 0) −
                    (if poly p b = 0 then 1 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*:
card $\{x{::}real.\ poly\ p\ x = 0\}$ = *count-roots p*
using *assms count-roots-correct* **by** *simp*

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

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

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

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

lemma *poly-card-roots-geq*:
card $\{x{::}real.\ x \geq a \land poly\ p\ x = 0\} = (let\ p = p\ in$
$count\text{-}roots\text{-}above\ p\ a + (if\ poly\ p\ a = 0 \land p \neq 0\ then\ 1\ else\ 0))$
proof (*cases* $poly\ p\ a = 0 \land p \neq 0$)
case *False*
hence *card* $\{x.\ x \geq a \land poly\ p\ x = 0\}$ = *card* $\{x.\ x > a \land poly\ p\ x = 0\}$
proof (*cases rule*: *disjE*)
assume $p = 0$
have $\neg finite\ \{a{<}..{<}a{+}1\}$ **using** *real-infinite-interval* **by** *simp*
moreover have $\{a{<}..{<}a{+}1\} \subseteq \{x.\ x \geq a \land poly\ p\ x = 0\}$
$\{a{<}..{<}a{+}1\} \subseteq \{x.\ x > a \land poly\ p\ x = 0\}$
using ‹$p = 0$› **by** *auto*
ultimately have $\neg finite\ \{x.\ x \geq a \land poly\ p\ x = 0\}$
$\neg finite\ \{x.\ x > a \land poly\ p\ x = 0\}$
by (*auto dest*: *finite-subset*[*of* $\{a{<}..{<}a{+}1\}$])
thus *?thesis* **by** *simp*
next
assume $poly\ p\ a \neq 0$
hence $\{x.\ x \geq a \land poly\ p\ x = 0\} = \{x.\ x > a \land poly\ p\ x = 0\}$
by (*auto simp*: *less-eq-real-def*)
thus *?thesis* **by** *simp*
qed *auto*
thus *?thesis* **using** *assms False*
by (*auto intro*: *poly-card-roots-greater*)

next
  case *True*
    hence $finite\ \{x.\ x > a \land poly\ p\ x = 0\}$ **using** *poly-roots-finite* **by** *force*
    moreover have $\{x.\ x \geq a \land poly\ p\ x = 0\} =$
        $insert\ a\ \{x.\ x > a \land poly\ p\ x = 0\}$ **using** *True* **by** *auto*
    ultimately have $card\ \{x.\ x \geq a \land poly\ p\ x = 0\} =$
        $Suc\ (card\ \{x.\ x > a \land poly\ p\ x = 0\})$
      using *card-insert-disjoint* **by** *auto*
    thus *?thesis* **using** *assms True* **by** (*auto intro*: *poly-card-roots-greater*)
qed

lemma *poly-card-roots-less*:
  $card\ \{x{::}real.\ x < a \land poly\ p\ x = 0\} = (let\ p = p\ in$
    $count\text{-}roots\text{-}below\ p\ a - (if\ poly\ p\ a = 0 \land p \neq 0\ then\ 1\ else\ 0))$
proof $(cases\ poly\ p\ a = 0 \land p \neq 0)$
  case *False*
    hence $card\ \{x.\ x < a \land poly\ p\ x = 0\} = card\ \{x.\ x \leq a \land poly\ p\ x = 0\}$
    proof (*cases rule*: *disjE*)
      assume $p = 0$
      have $\neg finite\ \{a - 1<..<a\}$ **using** *real-infinite-interval* **by** *simp*
      moreover have $\{a - 1<..<a\} \subseteq \{x.\ x \leq a \land poly\ p\ x = 0\}$
          $\{a - 1<..<a\} \subseteq \{x.\ x < a \land poly\ p\ x = 0\}$
        using $\langle p = 0\rangle$ **by** *auto*
      ultimately have $\neg finite\ \{x.\ x \leq a \land poly\ p\ x = 0\}$
          $\neg finite\ \{x.\ x < a \land poly\ p\ x = 0\}$
        by (*auto dest*: *finite-subset*[*of* $\{a - 1<..<a\}$])
      thus *?thesis* **by** *simp*
    next
      assume $poly\ p\ a \neq 0$
      hence $\{x.\ x < a \land poly\ p\ x = 0\} = \{x.\ x \leq a \land poly\ p\ x = 0\}$
        by (*auto simp*: *less-eq-real-def*)
      thus *?thesis* **by** *simp*
    qed *auto*
    thus *?thesis* **using** *assms False*
      by (*auto intro*: *poly-card-roots-leq*)
next
  case *True*
    hence $finite\ \{x.\ x < a \land poly\ p\ x = 0\}$ **using** *poly-roots-finite* **by** *force*
    moreover have $\{x.\ x \leq a \land poly\ p\ x = 0\} =$
        $insert\ a\ \{x.\ x < a \land poly\ p\ x = 0\}$ **using** *True* **by** *auto*
    ultimately have $Suc\ (card\ \{x.\ x < a \land poly\ p\ x = 0\}) =$
        $(card\ \{x.\ x \leq a \land poly\ p\ x = 0\})$
      using *card-insert-disjoint* **by** *auto*
    also note *count-roots-below-correct*[*symmetric*]
    finally show *?thesis* **using** *assms True* **by** *simp*
qed

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

$(\forall x.\ a < x \wedge x \leq b \longrightarrow poly\ p\ x \neq 0) \longleftrightarrow (let\ p = p\ in$
$(a \geq b \vee (p \neq 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b = 0)))$
by (*auto simp*: *count-roots-between-correct card-eq-0-iff not-le*
intro: *poly-roots-finite*)

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

lemma *poly-no-roots-leq-leq*:
$(\forall x.\ a \leq x \wedge x \leq b \longrightarrow poly\ p\ x \neq 0) \longleftrightarrow (let\ p = p\ in$
$(a > b \vee (p \neq 0 \wedge poly\ p\ a \neq 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b = 0)))$
apply (*intro iffI*)
apply (*force simp add*: *count-roots-between-correct card-eq-0-iff*)
apply (*unfold Let-def*)
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*
intro: *poly-roots-finite*)
done

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

lemma *poly-no-roots-less-less*:
$(\forall x.\ a < x \wedge x < b \longrightarrow poly\ p\ x \neq 0) \longleftrightarrow (let\ p = p\ in$
$(a \geq b \vee p \neq 0 \wedge count\text{-}roots\text{-}between\ p\ a\ b =$
$(if\ poly\ p\ b = 0\ then\ 1\ else\ 0)))$
proof
case *goal1*
note $A = this$
thus *?case*
proof (*cases* $a \geq b$, *simp*)
case *goal1*
with A **have** [*simp*]: $p \neq 0$ **using** *dense*[*of a b*] **by** *auto*
have B: $\{x.\ a < x \wedge x \leq b \wedge poly\ p\ x = 0\} =$
$\{x.\ a < x \wedge x < b \wedge poly\ p\ x = 0\} \cup$
$(if\ poly\ p\ b = 0\ then\ \{b\}\ else\ \{\})$ **using** *goal1* **by** *auto*
have $count\text{-}roots\text{-}between\ p\ a\ b =$
$card\ \{x.\ a < x \wedge x < b \wedge poly\ p\ x = 0\} +$
$(if\ poly\ p\ b = 0\ then\ 1\ else\ 0)$

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** $\{x.\ a < x \wedge x < b \wedge$ *poly p x* $= 0\} = \{\}$ **by** *simp*
finally show *?thesis* **by** *auto*
qed
next
case *goal2*
hence *card* $\{x.\ a < x \wedge x < b \wedge$ *poly p x* $= 0\} = 0$
by (*subst poly-card-roots-less-less*, *auto simp*: *count-roots-between-def*)
thus *?case* **using** *goal2*
by (*cases* $p = 0$, *simp*, *subst* (*asm*) *card-eq-0-iff*,
auto intro: *poly-roots-finite*)
qed

lemma *poly-pos-between-less-less*:
$(\forall x.\ a < x \wedge x < b \longrightarrow$ *poly p x* $> 0) \longleftrightarrow$ (*let* $p = p$ *in*
$(a \geq b \vee (p \neq 0 \wedge$ *poly p* $((a+b)/2) > 0 \wedge$
count-roots-between p a b $=$ (*if poly p b* $= 0$ *then* 1 *else* 0))))
by (*simp only*: *poly-pos-between-less-less Let-def*
poly-no-roots-less-less, *blast*)

lemma *poly-no-roots-leq-less*:
$(\forall x.\ a \leq x \wedge x < b \longrightarrow$ *poly p x* $\neq 0) \longleftrightarrow$ (*let* $p = p$ *in*
$(a \geq b \vee p \neq 0 \wedge$ *poly p a* $\neq 0 \wedge$ *count-roots-between p a b* $=$
(*if* $a < b \wedge$ *poly p b* $= 0$ *then* 1 *else* 0)))
proof
case *goal1*
hence $\forall x.\ a < x \wedge x < b \longrightarrow$ *poly p x* $\neq 0$ **by** *simp*
thus *?case* **using** *goal1* **by** (*subst* (*asm*) *poly-no-roots-less-less*, *auto*)
next
case *goal2*
hence $(b \leq a \vee p \neq 0 \wedge$ *count-roots-between p a b* $=$
(*if poly p b* $= 0$ *then* 1 *else* 0)) **by** *auto*
thus *?case* **using** *goal2* **unfolding** *Let-def*
by (*subst* (*asm*) *poly-no-roots-less-less*[*symmetric*, *unfolded Let-def*],
auto split: *split-if-asm simp*: *less-eq-real-def*)
qed

lemma *poly-pos-between-leq-less*:
$(\forall x.\ a \leq x \wedge x < b \longrightarrow$ *poly p x* $> 0) \longleftrightarrow$ (*let* $p = p$ *in*
$(a \geq b \vee (p \neq 0 \wedge$ *poly p a* $> 0 \wedge$ *count-roots-between p a b* $=$
(*if* $a < b \wedge$ *poly p b* $= 0$ *then* 1 *else* 0))))
by (*simp only*: *poly-pos-between-leq-less Let-def*
poly-no-roots-leq-less, *force*)

lemma *poly-no-roots-greater*:
$(\forall x.\ x > a \longrightarrow$ *poly p x* $\neq 0) \longleftrightarrow$ (*let* $p = p$ *in*
$(p \neq 0 \wedge$ *count-roots-above p a* $= 0$))

proof −
  have $\forall x.\ \neg\ a < x \Longrightarrow$ *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*:
  $(\forall x.\ x > a \longrightarrow poly\ p\ x > 0) \longleftrightarrow (let\ p = p\ in$
    $p \neq 0 \wedge poly\text{-}inf\ p = 1 \wedge count\text{-}roots\text{-}above\ p\ a = 0)$
  unfolding *Let-def*
  by (*subst poly-pos-greater*, *subst poly-no-roots-greater*, *force*)

lemma *poly-no-roots-leq*:
  $(\forall x.\ x \leq a \longrightarrow poly\ p\ x \neq 0) \longleftrightarrow$
    $(let\ p = p\ in\ (p \neq 0 \wedge count\text{-}roots\text{-}below\ p\ a = 0))$
    by (*auto simp*: *Let-def count-roots-below-correct card-eq-0-iff*
      intro: *poly-roots-finite*)

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

lemma *poly-no-roots-geq*:
  $(\forall x.\ x \geq a \longrightarrow poly\ p\ x \neq 0) \longleftrightarrow$
    $(let\ p = p\ in\ (p \neq 0 \wedge poly\ p\ a \neq 0 \wedge count\text{-}roots\text{-}above\ p\ a = 0))$
proof
  case *goal1*
  hence $\forall x{>}a.\ poly\ p\ x \neq 0$ **by** *simp*
  thus *?case* **using** *goal1* **by** (*subst* (*asm*) *poly-no-roots-greater*, *auto*)
next
  case *goal2*
  hence $(p \neq 0 \wedge count\text{-}roots\text{-}above\ p\ a = 0)$ **by** *simp*
  thus *?case* **using** *goal2*
    by (*subst* (*asm*) *poly-no-roots-greater*[*symmetric*, *unfolded Let-def*],
      auto simp: *less-eq-real-def*)
qed

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

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

case *goal1*
hence $\{x.\ x \leq a \wedge poly\ p\ x = 0\} = (if\ poly\ p\ a = 0\ then\ \{a\}\ else\ \{\})$
by (*auto simp*: *less-eq-real-def*)
moreover have $\forall x.\ \neg\ x < a \Longrightarrow False$ **by** (*metis lt-ex*)
ultimately show *?case* **using** *goal1* **by** (*auto simp*: *count-roots-below-correct*)
next
case *goal2*
have *A*: $\{x.\ x \leq a \wedge poly\ p\ x = 0\} = \{x.\ x < a \wedge poly\ p\ x = 0\} \cup$
$(if\ poly\ p\ a = 0\ then\ \{a\}\ else\ \{\})$ **by** (*auto simp*: *less-eq-real-def*)
have *count-roots-below p a* $=$ *card* $\{x.\ x < a \wedge poly\ p\ x = 0\} +$
$(if\ poly\ p\ a = 0\ then\ 1\ else\ 0)$ **using** *goal2*
by (*subst count-roots-below-correct*, *subst A*, *subst card-Un-disjoint*,
auto intro: *poly-roots-finite*)
with *goal2* **have** *card* $\{x.\ x < a \wedge poly\ p\ x = 0\} = 0$ **by** *simp*
thus *?case* **using** *goal2*
by (*subst* (*asm*) *card-eq-0-iff*, *auto intro*: *poly-roots-finite*)
qed

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

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

lemma *sturm-id-PR-prio0*:
$\{x::real.\ P\ x\} = \{x::real.\ (PR\text{-}TAG\ P)\ x\}$
$(\forall x::real.\ f\ x < g\ x) = (\forall x::real.\ PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ P\ x) = (\forall x::real.\ \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
by (*simp-all add*: *PR-TAG-def*)

lemma *sturm-id-PR-prio1*:

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

lemma *sturm-id-PR-prio2*:
$\{x::real.\ x > a \wedge x \leq b \wedge P\ x\} =$
$\{x::real.\ x > a \wedge x \leq b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x \geq a \wedge x \leq b \wedge P\ x\} =$
$\{x::real.\ x \geq a \wedge x \leq b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x \geq a \wedge x < b \wedge P\ x\} =$
$\{x::real.\ x \geq a \wedge x < b \wedge PR\text{-}TAG\ P\ x\}$
$\{x::real.\ x > a \wedge x < b \wedge P\ x\} =$
$\{x::real.\ x > a \wedge x < b \wedge PR\text{-}TAG\ P\ x\}$
$(\forall x::real.\ a < x \wedge x \leq b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a < x \wedge x \leq b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a \leq x \wedge x \leq b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a \leq x \wedge x \leq b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a \leq x \wedge x < b \longrightarrow f\ x < g\ x) =$
$(\forall x::real.\ a \leq x \wedge x < b \longrightarrow PR\text{-}TAG\ (\lambda x.\ f\ x < g\ x)\ x)$
$(\forall x::real.\ a < x \wedge x \leq b \longrightarrow P\ x) =$
$(\forall x::real.\ a < x \wedge x \leq b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real.\ a \leq x \wedge x \leq b \longrightarrow P\ x) =$
$(\forall x::real.\ a \leq x \wedge x \leq b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real.\ a \leq x \wedge x < b \longrightarrow P\ x) =$
$(\forall x::real.\ a \leq x \wedge x < b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow P\ x) =$
$(\forall x::real.\ a < x \wedge x < b \longrightarrow \neg(PR\text{-}TAG\ (\lambda x.\ \neg P\ x))\ x)$
by (*simp-all add*: *PR-TAG-def*)

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

$[\![\mathit{PR\text{-}TAG}\ P = (\lambda x.\ \mathit{poly}\ p\ x = 0);\ \mathit{PR\text{-}TAG}\ Q = (\lambda x.\ \mathit{poly}\ q\ x = 0)]\!]$
$\quad \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ P\ x \vee Q\ x) = (\lambda x.\ \mathit{poly}\ (p{*}q)\ x = 0)$ **and**

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

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

lemma *PR-TAG-intro-prio1*:
fixes $f :: \mathit{real} \Rightarrow \mathit{real}$
shows
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x = 0) = (\lambda x.\ \mathit{poly}\ p\ x = 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x \neq 0) = (\lambda x.\ \mathit{poly}\ p\ x \neq 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ 0 = f\ x) = (\lambda x.\ \mathit{poly}\ p\ x = 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ 0 \neq f\ x) = (\lambda x.\ \mathit{poly}\ p\ x \neq 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x \geq 0) = (\lambda x.\ \mathit{poly}\ p\ x \geq 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x > 0) = (\lambda x.\ \mathit{poly}\ p\ x > 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x \leq 0) = (\lambda x.\ \mathit{poly}\ (-p)\ x \geq 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x < 0) = (\lambda x.\ \mathit{poly}\ (-p)\ x > 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow$
$\quad \mathit{PR\text{-}TAG}\ (\lambda x.\ 0 \leq f\ x) = (\lambda x.\ \mathit{poly}\ (-p)\ x \leq 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x) \Longrightarrow$
$\quad \mathit{PR\text{-}TAG}\ (\lambda x.\ 0 < f\ x) = (\lambda x.\ \mathit{poly}\ (-p)\ x < 0)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x)$
$\quad \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ a * f\ x) = (\lambda x.\ \mathit{poly}\ (\mathit{smult}\ a\ p)\ x)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x)$
$\quad \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x * a) = (\lambda x.\ \mathit{poly}\ (\mathit{smult}\ a\ p)\ x)$
$\mathit{PR\text{-}TAG}\ f = (\lambda x.\ \mathit{poly}\ p\ x)$
$\quad \Longrightarrow \mathit{PR\text{-}TAG}\ (\lambda x.\ f\ x\ /\ a) = (\lambda x.\ \mathit{poly}\ (\mathit{smult}\ (\mathit{inverse}\ a)\ p)\ x)$

PR-TAG ($\lambda x.\ x \hat{} n$ :: *real*) = ($\lambda x.$ *poly* (*monom 1 n*) *x*)
using *assms* **by** (*intro ext*, *simp-all add*: *PR-TAG-def field-simps*
poly-monom divide-real-def)

lemma *PR-TAG-intro-prio2*:
PR-TAG ($\lambda x.$ *1* / *b*) = ($\lambda x.$ *inverse b*)
PR-TAG ($\lambda x.$ *a* / *b*) = ($\lambda x.$ *a* / *b*)
PR-TAG ($\lambda x.$ *a* / *b* ∗ $x \hat{} n$ :: *real*) = ($\lambda x.$ *poly* (*monom* (*a*/*b*) *n*) *x*)
PR-TAG ($\lambda x.$ $x \hat{} n$ ∗ *a* / *b* :: *real*) = ($\lambda x.$ *poly* (*monom* (*a*/*b*) *n*) *x*)
PR-TAG ($\lambda x.$ *a* ∗ $x \hat{} n$ :: *real*) = ($\lambda x.$ *poly* (*monom a n*) *x*)
PR-TAG ($\lambda x.$ $x \hat{} n$ ∗ *a* :: *real*) = ($\lambda x.$ *poly* (*monom a n*) *x*)
PR-TAG ($\lambda x.$ $x \hat{} n$ / *a* :: *real*) = ($\lambda x.$ *poly* (*monom* (*inverse a*) *n*) *x*)
PR-TAG ($\lambda x.$ *f x*ˆ(*Suc* (*Suc 0*)) :: *real*) = ($\lambda x.$ *poly p x*)
$\Longrightarrow$ *PR-TAG* ($\lambda x.$ *f x* ∗ *f x* :: *real*) = ($\lambda x.$ *poly p x*)
PR-TAG ($\lambda x.$ (*f x*) ˆ*Suc n* :: *real*) = ($\lambda x.$ *poly p x*)
$\Longrightarrow$ *PR-TAG* ($\lambda x.$ (*f x*) ˆ*n* ∗ *f x* :: *real*) = ($\lambda x.$ *poly p x*)
PR-TAG ($\lambda x.$ (*f x*) ˆ*Suc n* :: *real*) = ($\lambda x.$ *poly p x*)
$\Longrightarrow$ *PR-TAG* ($\lambda x.$ *f x* ∗ (*f x*) ˆ*n* :: *real*) = ($\lambda x.$ *poly p x*)
PR-TAG ($\lambda x.$ (*f x*) ˆ(*m*+*n*) :: *real*) = ($\lambda x.$ *poly p x*)
$\Longrightarrow$ *PR-TAG* ($\lambda x.$ (*f x*) ˆ*m* ∗ (*f x*) ˆ*n* :: *real*) = ($\lambda x.$ *poly p x*)
using *assms* **by** (*intro ext*, *simp-all add*: *PR-TAG-def field-simps*
poly-monom power-add divide-real-def)

lemma *sturm-meta-spec*: ($\bigwedge x$::*real*. *P x*) $\Longrightarrow$ *P x* **by** *simp*
lemma *sturm-imp-conv*:
$(a < x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$
$(a \leq x \longrightarrow x < b \longrightarrow c) \longleftrightarrow (a \leq x \wedge x < b \longrightarrow c)$
$(a < x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow (a < x \wedge x \leq b \longrightarrow c)$
$(a \leq x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow (a \leq x \wedge x \leq b \longrightarrow c)$
$(x < b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x < b \longrightarrow c)$
$(x < b \longrightarrow a \leq x \longrightarrow c) \longleftrightarrow (a \leq x \wedge x < b \longrightarrow c)$
$(x \leq b \longrightarrow a < x \longrightarrow c) \longleftrightarrow (a < x \wedge x \leq b \longrightarrow c)$
$(x \leq b \longrightarrow a \leq x \longrightarrow c) \longleftrightarrow (a \leq x \wedge x \leq b \longrightarrow c)$
by *auto*

ML-file *sturm.ML*

method-setup *sturm* = ⟪
Scan.succeed (*fn ctxt* => *SIMPLE-METHOD′* (*Sturm.sturm-tac ctxt true*))
⟫

lemma
$\forall x$::*real*. $x \hat{} 2 + 1 \neq 0$
by *sturm*

lemma
fixes *x* :: *real*
shows $x \hat{} 2 + 1 \neq 0$ **by** *sturm*

lemma $(x::real) > 1 \Longrightarrow x\hat{}3 > 1$ **by** *sturm*

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

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

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

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

schematic-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 $(x::real)*x+1 \neq 0 \wedge (x\hat{}2+1)*(x\hat{}2+2) \neq 0$ **by** *sturm*

end