# A Formalisation of Sturm's Theorem 

Manuel Eberl

September 13, 2023


#### Abstract

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


## Contents

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

## 1 Miscellaneous

theory Misc-Polynomial<br>imports HOL-Computational-Algebra.Polynomial HOL-Computational-Algebra.Polynomial-Factorial Pure-ex.Guess<br>begin

### 1.1 Analysis

```
lemma fun-eq-in-ivl:
    assumes }a\leqb\forallx::real. a\leqx\wedgex\leqb\longrightarrow eventually (\lambda\xi.f\xi=fx)(at x
    shows fa=fb
proof (rule connected-local-const)
    show connected {a..b} a \in{a..b} b\in{a..b} using <a\leqb> by (auto intro:
connected-Icc)
    show }\forallaa\in{a..b}. eventually (\lambdab.f aa=f b) (at aa within {a..b}
    proof
        fix }x\mathrm{ assume }x\in{a..b
        with assms(2)[rule-format, of x]
        show eventually (\lambdab.fx=fb) (at x within {a..b})
            by (auto simp: eventually-at-filter elim: eventually-mono)
    qed
qed
```


### 1.2 Polynomials

### 1.2.1 General simplification lemmas

lemma pderiv-div:
assumes $[$ simp]: $q$ dvd $p q \neq 0$
shows pderiv $(p$ div $q)=(q *$ pderiv $p-p * \operatorname{pderiv} q) \operatorname{div}(q * q)$ $q * q \operatorname{dvd}(q *$ pderiv $p-p *$ pderiv $q)$
proof-
from assms obtain $r$ where $p=q * r$ unfolding dvd-def by blast
hence $q *$ pderiv $p-p *$ pderiv $q=(q * q) *$ pderiv $r$
by (simp add: algebra-simps pderiv-mult)
thus $q * q d v d(q *$ pderiv $p-p *$ pderiv $q)$ by simp
note $0=p$ deriv-mult $[$ of $q$ p div $q$ ]
have 1: $q *(p \operatorname{div} q)=p$
by (metis assms(1) assms(2) dvd-def nonzero-mult-div-cancel-left)
have f1: pderiv $(p \operatorname{div} q) *(q * q) \operatorname{div}(q * q)=\operatorname{pderiv}(p \operatorname{div} q)$
by $\operatorname{simp}$
have f2: pderiv $p=q * \operatorname{pderiv}(p \operatorname{div} q)+p \operatorname{div} q * \operatorname{deriv} q$
by (metis 0 1)
have $p *$ pderiv $q=$ pderiv $q *(q *(p \operatorname{div} q))$
by (metis 1 mult.commute)
then have $p *$ pderiv $q=q *(p$ div $q * \operatorname{pderiv} q)$
by fastforce
then have $q *$ pderiv $p-p * \operatorname{pderiv} q=q *(q * \operatorname{pderiv}(p \operatorname{div} q))$
using f2 by (metis add-diff-cancel-right' distrib-left)
then show pderiv $(p$ div $q)=(q *$ pderiv $p-p * \operatorname{pderiv} q) \operatorname{div}(q * q)$
using $f 1$ by (metis mult.commute mult.left-commute)
qed

### 1.2.2 Divisibility of polynomials

Two polynomials that are coprime have no common roots.

```
lemma coprime-imp-no-common-roots:
    \(\neg(\) poly \(p x=0 \wedge\) poly \(q x=0)\) if coprime \(p q\)
        for \(x::{ }^{\prime} a\) :: field
proof clarify
    assume poly \(p x=0\) poly \(q x=0\)
    then have \([:-x, 1:] \operatorname{dvd} p[:-x, 1:]\) dvd \(q\)
        by (simp-all add: poly-eq-O-iff-dvd)
    with that have is-unit \([:-x, 1:]\)
        by (rule coprime-common-divisor)
    then show False
        by (auto simp add: is-unit-pCons-iff)
qed
lemma poly-div:
    assumes poly \(q x \neq 0\) and ( \(q::^{\prime} a::\) field poly) \(d v d p\)
    shows poly ( \(p\) div \(q\) ) \(x=\) poly \(p x / p o l y ~ q x\)
proof-
    from assms have \([\) simp \(]: q \neq 0\) by force
    have poly \(q x * \operatorname{poly}(p\) div \(q) x=\operatorname{poly}(q *(p \operatorname{div} q)) x\) by \(\operatorname{simp}\)
    also have \(q *(p \operatorname{div} q)=p\)
        using assms by (simp add: div-mult-swap)
    finally show poly \((p\) div \(q) x=\) poly \(p x / \operatorname{poly} q x\)
        using assms by (simp add: field-simps)
qed
```

lemma poly-div-gcd-squarefree-aux:
assumes pderiv ( $p::\left({ }^{\prime} a::\{\right.$ field-char- 0, field-gcd $\left.\}\right)$ poly $) \neq 0$
defines $d \equiv \operatorname{gcd} p(p d e r i v p)$
shows coprime ( $p$ div d) (pderiv ( $p$ div d)) and
$\bigwedge x . \operatorname{poly}(p$ div $d) x=0 \longleftrightarrow$ poly $p x=0$
proof -
obtain $r s$ where bezout-coefficients $p($ pderiv $p)=(r, s)$
by (auto simp add: prod-eq-iff)
then have $r * p+s *$ pderiv $p=\operatorname{gcd} p(p d e r i v p)$
by (rule bezout-coefficients)
then have $r s: d=r * p+s *$ pderiv $p$
by ( simp add: $d$-def)
define $t$ where $t=p$ div $d$
define $p^{\prime}$ where $\left[\right.$ simp]: $p^{\prime}=$ pderiv $p$
define $d^{\prime}$ where $[\operatorname{simp}]: d^{\prime}=$ pderiv $d$
define $u$ where $u=p^{\prime}$ div d
have $A: p=t * d$ and $B: p^{\prime}=u * d$
by (simp-all add: $t$-def $u$-def d-def algebra-simps)
from poly-squarefree-decomp[OF assms(1) A B[unfolded p'-def] rs] show $\bigwedge x$. poly $(p$ div $d) x=0 \longleftrightarrow$ poly $p x=0$ by (auto simp: $t$-def)
from $r s$ have $C: s * t * d^{\prime}=d *(1-r * t-s * p d e r i v t)$
by (simp add: A B algebra-simps pderiv-mult)
from assms have $[$ simp $]: p \neq 0 d \neq 0 t \neq 0$ by (force, force, subst (asm) A, force)
have $\bigwedge x . \llbracket x d v d t ; x d v d($ pderiv $t) \rrbracket \Longrightarrow x d v d 1$
proof -
fix $x$ assume $x d v d t x d v d(p d e r i v ~ t)$
then obtain $v w$ where $v w$ :
$t=x * v$ pderiv $t=x * w$ unfolding dvd-def by blast
define $x^{\prime} v^{\prime}$ where $\left[\right.$ simp]: $x^{\prime}=$ pderiv $x$ and $\left[\right.$ simp]: $v^{\prime}=$ pderiv $v$
from $v w$ have $x * v^{\prime}+v * x^{\prime}=x * w$ by (simp add: pderiv-mult)
hence $v * x^{\prime}=x *\left(w-v^{\prime}\right)$ by (simp add: algebra-simps)
hence $x d v d v *$ pderiv $x$ by simp
then obtain $y$ where $y: v * x^{\prime}=x * y$ unfolding dvd-def by force
from $\langle t \neq 0\rangle$ and $v w$ have $x \neq 0$ by simp
have $x$-pow- $n$-dvd-d: $\bigwedge n . \widehat{x n} d v d d$ proof-
fix $n$ show $x^{\wedge} n d v d d$
proof (induction n, simp, rename-tac n, case-tac n)
fix $n$ assume $n=(0::$ nat $)$
from $v w$ and $C$ have $d=x *\left(d * r * v+d * s * w+s * v * d^{\prime}\right)$
by (simp add: algebra-simps)
with $\langle n=0$ show $x \uparrow S u c n d v d d$ by (force intro: $d v d I$ )
next
fix $n n^{\prime}$ assume $I H: ~ x \widehat{n} d v d d$ and $n=S u c n^{\prime}$
hence [simp]: Suc $n^{\prime}=n x * x n^{\prime}=x \widehat{n}$ by simp-all
define $c::$ 'a poly where $c=[:$ of-nat $n$ :]
from pderiv-power-Suc[of $x n^{\prime}$ ]
have $[$ simp $]$ : pderiv $(x \widehat{x})=c * \widehat{n^{\prime}} * x^{\prime}$ unfolding $c$-def
by $\operatorname{simp}$
from $I H$ obtain $z$ where $d: d=\widehat{\wedge} n * z$ unfolding dvd-def by blast define $z^{\prime}$ where $[\operatorname{simp}]: z^{\prime}=$ pderiv $z$
from $d\langle d \neq 0\rangle$ have $x$ n $\neq 0 z \neq 0$ by force+
from $C d$ have $x$ $n * z=z * r * v * x$ SSuc $n+z * s * c * x$ n $n *\left(v * x^{\prime}\right)+$
$s * v * z^{\prime} * x$ §Suc $n+s * z *\left(v * x^{\prime}\right) * x$ n $+s * z * v^{\prime} * x$ §Suc $n$
by (simp add: algebra-simps vw pderiv-mult)
also have $\ldots=\widehat{x} n * x *\left(z * r * v+z * s * c * y+s * v * z^{\prime}+s * z * y+s * z * v^{\prime}\right)$
by (simp only: y, simp add: algebra-simps)
finally have $z=x *\left(z * r * v+z * s * c * y+s * v * z^{\prime}+s * z * y+s * z * v^{\prime}\right)$ using $\langle x \wedge \neq 0$ 〉 by force

```
            hence }x\mathrm{ dvd z by (metis dvd-triv-left)
            with d show }x\mathrm{ `Suc n dvd d by simp
        qed
    qed
    have degree x=0
    proof (cases degree x, simp)
        case (Suc n)
            hence }x\not=0\mathrm{ by auto
            with Suc have degree ( }\mp@subsup{x}{}{`}(\mathrm{ Suc (degree d))) > degree d
                by (subst degree-power-eq, simp-all)
            moreover from x-pow-n-dvd-d[of Suc (degree d)] and «d # 0 >
                have degree ( }x~\mathrm{ Suc (degree d)) }\leq\mathrm{ degree d
                    by (simp add: dvd-imp-degree-le)
            ultimately show ?thesis by simp
    qed
    then obtain c where [simp]: x=[:c:] by (cases x, simp split: if-split-asm)
    moreover from }\langlex\not=0\rangle\mathrm{ have c}\not=0\mathrm{ by simp
    ultimately show }x\mathrm{ dvd 1 using dvdI[of 1 x [:inverse c:]]
        by simp
qed
then show coprime t (pderiv t)
    by (rule coprimeI)
qed
lemma normalize-field:
normalize ( }x::'\\::{\mathrm{ field,normalization-semidom}) =(if x=0 then 0 else 1)
by (auto simp: is-unit-normalize dvd-field-iff)
lemma normalize-field-eq-1 [simp]:
    x\not=0\Longrightarrow normalize ( }x::\mathrm{ ''a :: {field,normalization-semidom}) = 1
    by (simp add: normalize-field)
lemma unit-factor-field [simp]:
    unit-factor (x :: 'a :: {field,normalization-semidom}) = x
    by (cases x = 0) (auto simp: is-unit-unit-factor dvd-field-iff)
Dividing a polynomial by its gcd with its derivative yields a squarefree polynomial with the same roots.
```

```
lemma poly-div-gcd-squarefree:
```

lemma poly-div-gcd-squarefree:
assumes ( $p::\left({ }^{\prime} a::\{\right.$ field-char-0,field-gcd $\left.\}\right)$ poly $) \neq 0$
assumes ( $p::\left({ }^{\prime} a::\{\right.$ field-char-0,field-gcd $\left.\}\right)$ poly $) \neq 0$
defines $d \equiv \operatorname{gcd} p(p d e r i v p)$
defines $d \equiv \operatorname{gcd} p(p d e r i v p)$
shows coprime ( $p$ div d) (pderiv $(p$ div d)) (is ? $A$ ) and
shows coprime ( $p$ div d) (pderiv $(p$ div d)) (is ? $A$ ) and
$\bigwedge x . \operatorname{poly}(p$ div $d) x=0 \longleftrightarrow$ poly $p x=0($ is $\Lambda x . ? B x)$
$\bigwedge x . \operatorname{poly}(p$ div $d) x=0 \longleftrightarrow$ poly $p x=0($ is $\Lambda x . ? B x)$
proof-
proof-
have ? $A \wedge(\forall x . ? B x)$
have ? $A \wedge(\forall x . ? B x)$
proof (cases pderiv $p=0$ )
proof (cases pderiv $p=0$ )
case False

```
    case False
```

```
        from poly-div-gcd-squarefree-aux[OF this] show ?thesis
            unfolding d-def by auto
    next
    case True
        then obtain c where [simp]: p=[:c:] using pderiv-iszero by blast
        from assms(1) have c\not=0 by simp
        from True have d=smult (inverse c) p
            by (simp add: d-def normalize-poly-def map-poly-pCons field-simps)
        with }\langlep\not=0\rangle\langlec\not=0\rangle\mathrm{ have p div d= [:c:]
        by (simp add: pCons-one)
    with }\langlec\not=0\rangle\mathrm{ show ?thesis
        by (simp add: normalize-const-poly is-unit-triv)
    qed
    thus ?A and }\bigwedgex.?Bx\mathrm{ by simp-all
qed
```


### 1.2.3 Sign changes of a polynomial

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

```
lemma poly-different-sign-imp-root:
    assumes \(a<b\) and \(\operatorname{sgn}(\) poly \(p a) \neq \operatorname{sgn}(\) poly \(p(b::\) real \())\)
    shows \(\exists x . a \leq x \wedge x \leq b \wedge\) poly \(p x=0\)
proof (cases poly p \(a=0 \vee\) poly p \(b=0\) )
    case True
        thus ?thesis using assms(1)
            by (elim disjE, rule-tac exI \([\) of - a], simp,
                rule-tac exI [of - b], simp)
next
    case False
        hence [simp]: poly pa\(=0\) poly \(p b \neq 0\) by simp-all
        show ?thesis
        proof (cases poly pa<0)
            case True
            hence \(\operatorname{sgn}(\) poly \(p a)=-1\) by \(\operatorname{simp}\)
            with assms True have poly \(p b>0\)
                by (auto simp: sgn-real-def split: if-split-asm)
            from poly-IVT-pos[OF \(\langle a<b\rangle\) True this \(]\) guess \(x\)..
            thus ?thesis by (intro exI [of - x], simp)
        next
            case False
            hence poly pa>0 by (simp add: not-less less-eq-real-def)
            hence \(\operatorname{sgn}(\) poly \(p a)=1\) by \(\operatorname{simp}\)
            with assms False have poly \(p b<0\)
                    by (auto simp: sgn-real-def not-less
                        less-eq-real-def split: if-split-asm)
            from poly-IVT-neg[OF \(\langle a<b\rangle\langle p o l y p a>0\rangle\) this \(]\) guess \(x\)..
            thus ?thesis by (intro exI [of - x], simp)
        qed
qed
```

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

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

## 1．2．4 Limits of polynomials

```
lemma poly-neighbourhood-without-roots:
    assumes \((p::\) real poly) \(\neq 0\)
    shows eventually \((\lambda x\). poly \(p x \neq 0)\left(\right.\) at \(\left.x_{0}\right)\)
proof-
    \{
    fix \(\varepsilon::\) real assume \(\varepsilon>0\)
    have fin: finite \(\left\{x .\left|x-x_{0}\right|<\varepsilon \wedge x \neq x_{0} \wedge\right.\) poly \(\left.p x=0\right\}\)
        using poly-roots-finite \([\) OF assms] by simp
    with \(« \varepsilon>0\rangle\) have \(\exists \delta>0 . \delta \leq \varepsilon \wedge\left(\forall x .\left|x-x_{0}\right|<\delta \wedge x \neq x_{0} \longrightarrow\right.\) poly \(\left.p x \neq 0\right)\)
    proof (induction card \(\left\{x .\left|x-x_{0}\right|<\varepsilon \wedge x \neq x_{0} \wedge\right.\) poly p \(\left.x=0\right\}\)
                arbitrary: \(\varepsilon\) rule: less-induct)
    case (less \(\varepsilon\) )
    let ? \(A=\left\{x .\left|x-x_{0}\right|<\varepsilon \wedge x \neq x_{0} \wedge\right.\) poly \(\left.p x=0\right\}\)
    show ?case
        proof (cases card ?A)
        case 0
            hence ? \(A=\{ \}\) using less by auto
            thus ?thesis using less(2) by (rule-tac exI[of - \(\varepsilon\) ], auto)
        next
        case (Suc -)
            with less(3) have \(\left\{x .\left|x-x_{0}\right|<\varepsilon \wedge x \neq x_{0} \wedge\right.\) poly \(\left.p x=0\right\} \neq\{ \}\) by force
                then obtain \(x\) where \(x\)-props: \(\left|x-x_{0}\right|<\varepsilon x \neq x_{0}\) poly \(p x=0\) by blast
        define \(\varepsilon^{\prime}\) where \(\varepsilon^{\prime}=\left|x-x_{0}\right| / 2\)
        have \(\varepsilon^{\prime}>0 \varepsilon^{\prime}<\varepsilon\) unfolding \(\varepsilon^{\prime}\)-def using \(x\)-props by simp-all
        from \(x-\operatorname{props}(1,2)\) and \(\langle\varepsilon>0\rangle\)
                            have \(x \notin\left\{x^{\prime} .\left|x^{\prime}-x_{0}\right|<\varepsilon^{\prime} \wedge x^{\prime} \neq x_{0} \wedge\right.\) poly \(\left.p x^{\prime}=0\right\}(\) is \(-\notin ? B)\)
                            by (auto simp: \(\varepsilon^{\prime}\)-def)
            moreover from \(x\)-props
                have \(x \in\left\{x .\left|x-x_{0}\right|<\varepsilon \wedge x \neq x_{0} \wedge\right.\) poly \(\left.p x=0\right\}\) by blast
            ultimately have ? \(B \subset\) ? A by auto
            hence card ? \(B<\) card ?A finite ?B
                by (rule psubset-card-mono[OF less(3)],
                    blast intro: finite-subset[OF - less(3)])
            from less(1)[OF this(1)〈的> 0〉this(2)]
```

```
                show ?thesis using < < < < > by force
        qed
    qed
    }
    from this[of 1]
    show ?thesis by (auto simp: eventually-at dist-real-def)
qed
lemma poly-neighbourhood-same-sign:
    assumes poly p(x0 :: real)}\not=
    shows eventually ( }\lambdax\mathrm{ . sgn (poly p x) = sgn (poly p x 员) (at x ( )
proof -
    have cont: isCont ( }\lambdax.\operatorname{sgn}(poly p x)) \mp@subsup{x}{0}{
        by (rule isCont-sgn, rule poly-isCont, rule assms)
    then have eventually ( }\lambdax.|\operatorname{sgn}(\mathrm{ poly p x) - sgn (poly p x ( ) | < 1) (at x ( )
        by (auto simp: isCont-def tendsto-iff dist-real-def)
    then show ?thesis
        by (rule eventually-mono) (simp add: sgn-real-def split: if-split-asm)
qed
lemma poly-lhopital:
assumes poly \(p(x::\) real \()=0\) poly \(q x=0 q \neq 0\)
assumes \((\lambda x\). poly (pderiv \(p) x / \operatorname{poly}(\) pderiv \(q) x)-x \rightarrow y\)
shows \((\lambda x\). poly \(p x /\) poly \(q x)-x \rightarrow y\)
using assms
proof (rule-tac lhopital)
have isCont (poly \(p\) ) \(x\) isCont (poly \(q\) ) \(x\) by simp-all
with \(\operatorname{assms}(1,2)\) show poly \(p-x \rightarrow 0\) poly \(q-x \rightarrow 0\)
by (simp-all add: isCont-def)
from \(\langle q \neq 0\rangle\) and \(\langle\) poly \(q x=0\rangle\) have pderiv \(q \neq 0\) by (auto dest: pderiv-iszero)
from poly-neighbourhood-without-roots[OF this] show eventually \((\lambda x\). poly (pderiv q) \(x \neq 0)(\) at \(x)\).
qed (auto intro: poly-DERIV poly-neighbourhood-without-roots)
lemma poly-roots-bounds:
assumes \(p \neq 0\)
obtains \(l u\)
where \(l \leq(u::\) real \()\)
and poly \(p l \neq 0\)
and poly \(p u \neq 0\)
and \(\{x . x>l \wedge x \leq u \wedge\) poly \(p x=0\}=\{x\). poly \(p x=0\}\)
and \(\bigwedge x . x \leq l \Longrightarrow \operatorname{sgn}(\) poly \(p x)=\operatorname{sgn}(\) poly \(p l)\)
and \(\bigwedge x . x \geq u \Longrightarrow \operatorname{sgn}(\) poly \(p x)=\operatorname{sgn}(\) poly \(p u)\)
proof
from assms have finite \(\{x\). poly \(p x=0\}\) (is finite ?roots)
using poly-roots-finite by fast
```

let ?roots' $=$ insert 0 ?roots
define $l$ where $l=$ Min ? roots ${ }^{\prime}-1$
define $u$ where $u=$ Max ? ${ }^{\text {roots }}{ }^{\prime}+1$
from 〈finite ?roots〉 have $A$ : finite ?roots' by auto
from Min-le[OF this, of 0$]$ and Max-ge[OF this, of 0] show $l \leq u$ by (simp add: l-def $u$-def)
from Min-le $[O F A]$ have $l$-props: $\wedge x . x \leq l \Longrightarrow$ poly $p x \neq 0$ by (fastforce simp: l-def)
from Max-ge $[O F A]$ have $u$-props: $\backslash x . x \geq u \Longrightarrow$ poly p $x \neq 0$ by (fastforce simp: u-def)
from l-props $u$-props show $[$ simp $]$ : poly $p l \neq 0$ poly $p u \neq 0$ by auto
from $l$-props have $\Lambda x$. poly $p x=0 \Longrightarrow x>l$ by (metis not-le)
moreover from $u$-props have $\bigwedge x$. poly $p x=0 \Longrightarrow x \leq u$ by (metis linear)
ultimately show $\{x . x>l \wedge x \leq u \wedge$ poly $p x=0\}=$ ? roots by auto

## \{

fix $x$ assume $A: x<l \operatorname{sgn}($ poly $p x) \neq \operatorname{sgn}($ poly $p l)$
with poly-IVT-pos[OF A(1), of p] poly-IVT-neg[OF A(1), of p] A(2)
have False by (auto split: if-split-asm
simp: sgn-real-def l-props not-less less-eq-real-def)
\}
thus $\bigwedge x . x \leq l \Longrightarrow \operatorname{sgn}($ poly $p x)=\operatorname{sgn}($ poly $p l)$
by (case-tac $x=l$, auto simp: less-eq-real-def)

## \{

fix $x$ assume $A: x>u \operatorname{sgn}($ poly $p x) \neq \operatorname{sgn}($ poly $p u)$
with $u$-props poly-IVT-neg[OF $A(1)$, of p] poly-IVT-pos $[O F A(1)$, of p] $A(2)$
have False by (auto split: if-split-asm
simp: sgn-real-def l-props not-less less-eq-real-def)
\}
thus $\wedge x . x \geq u \Longrightarrow \operatorname{sgn}($ poly $p x)=\operatorname{sgn}($ poly $p u)$
by (case-tac $x=u$, auto simp: less-eq-real-def)
qed

```
definition poly-inf :: ('a::real-normed-vector) poly \(\Rightarrow^{\prime} a\) where
    poly-inf \(p \equiv \operatorname{sgn}(\) coeff \(p(\) degree \(p))\)
definition poly-neg-inf :: ('a::real-normed-vector) poly \(\Rightarrow\) ' \(a\) where
    poly-neg-inf \(p \equiv\) if even (degree \(p\) ) then sgn (coeff \(p\) (degree \(p)\) )
        else -sgn (coeff \(p(\) degree \(p)\) )
lemma poly-inf-0-iff [simp]:
    poly-inf \(p=0 \longleftrightarrow p=0\) poly-neg-inf \(p=0 \longleftrightarrow p=0\)
    by (auto simp: poly-inf-def poly-neg-inf-def sgn-zero-iff)
lemma poly-inf-mult[simp]:
```

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

shows poly-inf $(p * q)=$ poly-inf $p * \operatorname{poly}-\inf q$
poly-neg-inf $(p * q)=$ poly-neg-inf $p *$ poly-neg-inf $q$
unfolding poly-inf-def poly-neg-inf-def
by ( cases $p=0 \vee q=0$, auto simp: sgn-zero-iff degree-mult-eq[of p q] coeff-mult-degree-sum Real-Vector-Spaces.sgn-mult) []$)+$
lemma poly-neq-0-at-infinity:
assumes $(p::$ real poly) $\neq 0$
shows eventually ( $\lambda x$. poly $p x \neq 0$ ) at-infinity

## proof -

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

```
lemma poly-limit-aux:
    fixes p :: real poly
    defines }n\equiv\mathrm{ degree }
    shows (( }\lambdax.\mathrm{ poly p x/ x^ n) }\longrightarrow\mathrm{ coeff p n) at-infinity
proof (subst filterlim-cong, rule refl, rule refl)
    show eventually (\lambdax. poly px/\widehat{ n}=(\sumi\leqn. coeff pi/ x`(n-i)))
                at-infinity
    proof (rule eventually-mono)
    show eventually ( }\lambdax::\mathrm{ real. }x\not=0)\mathrm{ at-infinity
        by (simp add: eventually-at-infinity, rule exI[of-1], auto)
    fix x :: real assume [simp]: x\not=0
    show poly p x/ x^ n = (\sumi\leqn. coeff pi/ x^(n - i))
            by (simp add: n-def sum-divide-distrib power-diff poly-altdef)
    qed
```

```
let ?a = \lambdai. if i=n then coeff p n else 0
have }\foralli\in{..n}. ((\lambdax. coeff pi/\mp@subsup{x}{}{`}(n-i))\longrightarrow ?a i) at-infinit
proof
    fix }i\mathrm{ assume }i\in{..n
    hence }i\leqn\mathrm{ by simp
    show ((\lambdax. coeff p i / x^ (n-i)) \longrightarrow?a i) at-infinity
    proof (cases i=n)
        case True
            thus ?thesis by (intro tendstoI, subst eventually-at-infinity,
                        intro exI[of-1], simp add: dist-real-def)
    next
        case False
            hence }n-i>0\mathrm{ using <i \ n> by simp
            from tendsto-inverse-0 and divide-real-def[of 1]
                    have ((\lambdax.1 / x :: real)\longrightarrow0) at-infinity by simp
            from tendsto-power[OF this, of n-i]
                    have ((\lambdax::real. 1 / x^ (n-i))\longrightarrow0) at-infinity
                    using <n - i> 0\rangle by (simp add: power-0-left power-one-over)
            from tendsto-mult-right-zero[OF this, of coeff pi]
                    have }((\lambdax.coeff pi/ x^ (n-i))\longrightarrow0) at-infinity
                        by (simp add: field-simps)
            thus ?thesis using False by simp
        qed
qed
hence (( }\lambdax.\sumi\leqn. coeff pi/ x`(n-i))\longrightarrow(\sumi\leqn. ?a i)) at-infinity
    by (force intro!: tendsto-sum)
also have (\sumi\leqn. ?a i) = coeff p n by (subst sum.delta, simp-all)
finally show (( }\lambdax.\sumi\leqn.coeff pi/ x`(n-i))\longrightarrow <oeff p n) at-infinity .
qed
lemma poly-at-top-at-top:
    fixes p :: real poly
    assumes degree p\geq1 coeff p(degree p)>0
    shows LIM x at-top. poly p x :> at-top
proof-
    let ? n = degree p
    define fg}\mathrm{ where fx= poly px/ x^?n and gx= x^ ?n for }x\mathrm{ :: real
    from poly-limit-aux have (f\longrightarrow coeff p (degree p)) at-top
        using tendsto-mono at-top-le-at-infinity unfolding f-def by blast
    moreover from assms
        have LIM x at-top.g x :> at-top
            by (auto simp add: g-def intro!: filterlim-pow-at-top filterlim-ident)
    ultimately have LIM x at-top. fx*gx :> at-top
        using filterlim-tendsto-pos-mult-at-top assms by simp
    also have eventually ( }\lambdax.fx*gx=\mathrm{ poly p x) at-top
        unfolding f-def g-def
```

by (subst eventually-at-top-linorder, rule exI[of - 1],
simp add: poly-altdef field-simps sum-distrib-left power-diff)
note filterlim-cong[OF refl refl this]
finally show ?thesis.
qed
lemma poly-at-bot-at-top:
fixes $p$ :: real poly
assumes degree $p \geq 1$ coeff $p($ degree $p)<0$
shows LIM $x$ at-top. poly $p x$ :> at-bot
proof -
from poly-at-top-at-top $[o f-p]$ and assms
have LIM $x$ at-top. -poly $p x:>$ at-top by simp
thus ?thesis by (simp add: filterlim-uminus-at-bot)
qed
lemma poly-lim-inf:
eventually $(\lambda x::$ real. sgn (poly $p x)=$ poly-inf $p$ ) at-top
proof (cases degree $p \geq 1$ )
case False
hence degree $p=0$ by simp
then obtain $c$ where $p=[: c:]$ by (cases $p$, auto split: if-split-asm)
thus ?thesis
by (simp add: eventually-at-top-linorder poly-inf-def)
next
case True
note $d e g=$ this
let ?lc $=$ coeff $p($ degree $p)$
from True have ?lc $\neq 0$ by force
show ?thesis
proof (cases ?lc $>0$ )
case True
from poly-at-top-at-top[OF deg this]
obtain $x_{0}$ where $\bigwedge x . x \geq x_{0} \Longrightarrow$ poly $p x \geq 1$
by (fastforce simp: filterlim-at-top
eventually-at-top-linorder less-eq-real-def)
hence $\bigwedge x . x \geq x_{0} \Longrightarrow \operatorname{sgn}($ poly $p x)=1$ by force
thus ?thesis by (simp only: eventually-at-top-linorder poly-inf-def, intro exI[of - $\left.x_{0}\right]$, simp add: True)
next
case False
hence $? l c<0$ using $\langle ? l c \neq 0\rangle$ by linarith
from poly-at-bot-at-top[OF deg this]
obtain $x_{0}$ where $\wedge x . x \geq x_{0} \Longrightarrow$ poly $p x \leq-1$
by (fastforce simp: filterlim-at-bot
eventually-at-top-linorder less-eq-real-def)
hence $\bigwedge x . x \geq x_{0} \Longrightarrow \operatorname{sgn}($ poly $p x)=-1$ by force
thus ?thesis by (simp only: eventually-at-top-linorder poly-inf-def, intro exI $\left[\right.$ of $\left.-x_{0}\right]$, simp add: <?lc $\left.<0\right\rangle$ )
qed
qed
lemma poly-at-top-or-bot-at-bot:
fixes $p$ :: real poly
assumes degree $p \geq 1$ coeff $p($ degree $p)>0$
shows LIM $x$ at-bot. poly $p x:>$ (if even (degree $p$ ) then at-top else at-bot) proof-
let $? n=$ degree $p$
define $f g$ where $f x=$ poly $p x / x^{\wedge} ? n$ and $g x=x^{\wedge} ? n$ for $x::$ real
from poly-limit-aux have $(f \longrightarrow$ coeff $p$ (degree $p)$ ) at-bot
using tendsto-mono at-bot-le-at-infinity by (force simp: $f$-def [abs-def])
moreover from assms
have LIM $x$ at-bot. $g x:>($ if even (degree $p$ ) then at-top else at-bot)
by (auto simp add: g-def split: if-split-asm intro: filterlim-pow-at-bot-even
filterlim-pow-at-bot-odd filterlim-ident)
ultimately have LIM $x$ at-bot. $f x * g x$ :>
(if even ? $n$ then at-top else at-bot)
by (auto simp: assms intro: filterlim-tendsto-pos-mult-at-top
filterlim-tendsto-pos-mult-at-bot)
also have eventually ( $\lambda x . f x * g x=$ poly $p x$ ) at-bot
unfolding $f$-def $g$-def
by (subst eventually-at-bot-linorder, rule exI[of - -1], simp add: poly-altdef field-simps sum-distrib-left power-diff)
note filterlim-cong[OF refl refl this]
finally show? ?thesis.
qed
lemma poly-at-bot-or-top-at-bot:
fixes $p$ :: real poly
assumes degree $p \geq 1$ coeff $p($ degree $p)<0$
shows LIM $x$ at-bot. poly $p x:>$ (if even (degree $p$ ) then at-bot else at-top)
proof-
from poly-at-top-or-bot-at-bot $[o f-p]$ and assms
have LIM x at-bot. -poly $p x$ :>
(if even (degree $p$ ) then at-top else at-bot) by simp
thus ?thesis by (auto simp: filterlim-uminus-at-bot)
qed
lemma poly-lim-neg-inf:
eventually ( $\lambda x::$ real. sgn (poly $p x)=$ poly-neg-inf $p$ ) at-bot
proof (cases degree $p \geq 1$ )
case False
hence degree $p=0$ by simp
then obtain $c$ where $p=[: c:]$ by (cases $p$, auto split: if-split-asm)
thus ?thesis
by (simp add: eventually-at-bot-linorder poly-neg-inf-def)

```
next
    case True
        note deg = this
    let ?lc = coeff p (degree p)
    from True have ?lc \not=0 by force
    show ?thesis
    proof (cases ?lc > 0)
        case True
            note lc-pos=this
            show ?thesis
            proof (cases even (degree p))
                case True
                    from poly-at-top-or-bot-at-bot[OF deg lc-pos] and True
                        obtain }\mp@subsup{x}{0}{}\mathrm{ where }\bigwedgex.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ poly p }x\geq
                        by (fastforce simp add: filterlim-at-top filterlim-at-bot
                        eventually-at-bot-linorder less-eq-real-def)
                            hence }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\operatorname{sgn}(\mathrm{ poly p x)=1 by force
                    thus ?thesis
                            by (simp add: True eventually-at-bot-linorder poly-neg-inf-def,
                            intro exI[of - x 的, simp add:lc-pos)
        next
                case False
                    from poly-at-top-or-bot-at-bot[OF deg lc-pos] and False
                    obtain }\mp@subsup{x}{0}{}\mathrm{ where }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ poly p x 
                        by (fastforce simp add: filterlim-at-bot
                                    eventually-at-bot-linorder less-eq-real-def)
                                    hence }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ sgn (poly p x)=-1 by force
                                    thus ?thesis
                                    by (simp add: False eventually-at-bot-linorder poly-neg-inf-def,
                                    intro exI[of - x 0], simp add:lc-pos)
        qed
    next
        case False
            hence lc-neg:?lc < 0 using <?lc \not=0> by linarith
            show ?thesis
            proof (cases even (degree p))
                case True
                with poly-at-bot-or-top-at-bot[OF deg lc-neg]
                    obtain }\mp@subsup{x}{0}{}\mathrm{ where }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ poly p }x\leq-
                        by (fastforce simp: filterlim-at-bot
                            eventually-at-bot-linorder less-eq-real-def)
                            hence }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ sgn (poly p x)=-1 by force
                        thus ?thesis
                                    by (simp only: True eventually-at-bot-linorder poly-neg-inf-def,
                                    intro exI[of - x 0 ], simp add:lc-neg)
            next
                        case False
                                with poly-at-bot-or-top-at-bot[OF deg lc-neg]
                        obtain }\mp@subsup{x}{0}{}\mathrm{ where }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\mathrm{ poly p }x\geq
```

```
                    by (fastforce simp: filterlim-at-top
                                    eventually-at-bot-linorder less-eq-real-def)
            hence }\x.x\leq\mp@subsup{x}{0}{}\Longrightarrow\operatorname{sgn}(\mathrm{ poly p x)=1 by force
            thus ?thesis
                by (simp only: False eventually-at-bot-linorder poly-neg-inf-def,
                                    intro exI[of - x 0], simp add:lc-neg)
        qed
    qed
qed
```


### 1.2.5 Signs of polynomials for sufficiently large values

## lemma polys-inf-sign-thresholds:

## assumes finite ( $p s$ :: real poly set)

obtains $l u$
where $l \leq u$
and $\bigwedge p . \llbracket p \in p s ; p \neq 0 \rrbracket \Longrightarrow$
$\{x . l<x \wedge x \leq u \wedge$ poly $p x=0\}=\{x$. poly $p x=0\}$
and $\bigwedge p x . \llbracket p \in p s ; x \geq u \rrbracket \Longrightarrow \operatorname{sgn}($ poly $p x)=$ poly-inf $p$
and $\wedge p x . \llbracket p \in p s ; x \leq l \rrbracket \Longrightarrow \operatorname{sgn}($ poly $p x)=$ poly-neg-inf $p$
proof goal-cases
case prems: 1
have $\exists l u . l \leq u \wedge(\forall p x . p \in p s \wedge x \geq u \longrightarrow \operatorname{sgn}($ poly $p x)=$ poly-inf $p) \wedge$
$(\forall p x . p \in p s \wedge x \leq l \longrightarrow \operatorname{sgn}($ poly $p x)=$ poly-neg-inf $p)$
(is $\exists l u$.? P ps $l u$ )
proof (induction rule: finite-subset-induct[OF $\operatorname{assms}(1)$, where $A=U N I V])$
case 1
show? case by simp
next
case 2
show ?case by (intro exI[of - 42], simp)
next
case prems: $(3 p p s)$
from prems(4) obtain $l u$ where lu-props: ?P ps $l u$ by blast from poly-lim-inf obtain $u^{\prime}$
where $u^{\prime}$-props: $\forall x \geq u^{\prime}$. sgn (poly $p x$ ) $=$ poly-inf $p$
by (force simp add: eventually-at-top-linorder)
from poly-lim-neg-inf obtain $l^{\prime}$
where $l^{\prime}$-props: $\forall x \leq l^{\prime}$. sgn (poly $\left.p x\right)=$ poly-neg-inf $p$
by (force simp add: eventually-at-bot-linorder)
show ?case

insert lu-props l'-props $u^{\prime}$-props, auto)
qed
then obtain $l u$ where lu-props: $l \leq u$
$\bigwedge p x . p \in p s \Longrightarrow u \leq x \Longrightarrow \operatorname{sgn}($ poly $p x)=$ poly-inf $p$
$\bigwedge p x . p \in p s \Longrightarrow x \leq l \Longrightarrow$ sgn $($ poly $p x)=$ poly-neg-inf $p$ by blast
moreover \{
fix $p x$ assume $A: p \in p s p \neq 0$ poly $p x=0$

```
    from A have l<x x<u
        by (auto simp: not-le[symmetric] dest: lu-props(2,3))
    }
    note }A=\mathrm{ this
    have }\p.p\inps\Longrightarrowp\not=0
            {x.l<x\wedge x\lequ^ poly p x=0}={x. poly p }x=0
        by (auto dest: A)
```

    from \(\operatorname{prems}[O F\) lu-props(1) this lu-props(2,3)] show thesis .
    qed

### 1.2.6 Positivity of polynomials

lemma poly-pos:
$(\forall x::$ real. poly $p x>0) \longleftrightarrow$ poly-inf $p=1 \wedge(\forall x$. poly $p x \neq 0)$
proof (intro iffI conjI)
assume $A: \forall x::$ real. poly $p x>0$
have $\wedge x$. poly $p(x::$ real $)>0 \Longrightarrow$ poly $p x \neq 0$ by simp
with $A$ show $\forall x::$ real. poly $p x \neq 0$ by simp
from poly-lim-inf obtain $x$ where $\operatorname{sgn}($ poly $p x)=$ poly-inf $p$ by (auto simp: eventually-at-top-linorder)
with $A$ show poly-inf $p=1$
by (simp add: sgn-real-def split: if-split-asm)
next
assume poly-inf $p=1 \wedge(\forall x$. poly $p x \neq 0)$
hence $A$ : poly-inf $p=1$ and $B:(\forall x$. poly $p x \neq 0)$ by simp-all
from poly-lim-inf obtain $x$ where $C$ : sgn $($ poly $p x)=$ poly-inf $p$
by (auto simp: eventually-at-top-linorder)
show $\forall x$. poly $p x>0$
proof (rule ccontr)
assume $\neg(\forall x$. poly $p x>0)$
then obtain $x^{\prime}$ where poly $p x^{\prime} \leq 0$ by (auto simp: not-less)
with $A$ and $C$ have $\operatorname{sgn}\left(\right.$ poly $\left.p x^{\prime}\right) \neq \operatorname{sgn}($ poly $p x)$
by (auto simp: sgn-real-def split: if-split-asm)
from poly-different-sign-imp-root' $[O F$ this $]$ and $B$
show False by blast
qed
qed
lemma poly-pos-greater:
$(\forall x::$ real. $x>a \longrightarrow$ poly $p x>0) \longleftrightarrow$ poly-inf $p=1 \wedge(\forall x . x>a \longrightarrow$ poly $p x \neq 0)$
proof (intro iffI conjI)
assume $A: \forall x::$ real. $x>a \longrightarrow$ poly $p x>0$
have $\wedge x$. poly $p(x::$ real $)>0 \Longrightarrow$ poly $p x \neq 0$ by simp
with $A$ show $\forall x::$ real. $x>a \longrightarrow$ poly $p x \neq 0$ by auto
from poly-lim-inf obtain $x_{0}$ where

```
\forallx\geq\mp@subsup{x}{0}{}.\operatorname{sgn}(\mathrm{ poly p x) = poly-inf p}
```

by (auto simp: eventually-at-top-linorder)
hence poly-inf $p=\operatorname{sgn}\left(\right.$ poly $\left.p\left(\max x_{0}(a+1)\right)\right)$ by simp
also from $A$ have $\ldots=1$ by force
finally show poly-inf $p=1$.
next
assume poly-inf $p=1 \wedge(\forall x . x>a \longrightarrow$ poly $p x \neq 0)$
hence $A$ : poly-inf $p=1$ and
$B:(\forall x . x>a \longrightarrow$ poly $p x \neq 0)$ by simp-all
from poly-lim-inf obtain $x_{0}$ where
$C: \forall x \geq x_{0} . \operatorname{sgn}($ poly $p x)=$ poly-inf $p$
by (auto simp: eventually-at-top-linorder)
hence $\operatorname{sgn}\left(\right.$ poly $\left.p\left(\max x_{0}(a+1)\right)\right)=$ poly-inf $p$ by $\operatorname{simp}$
with $A$ have $D$ : sgn $\left(\right.$ poly $\left.p\left(\max x_{0}(a+1)\right)\right)=1$ by simp
show $\forall x . x>a \longrightarrow$ poly $p x>0$
proof (rule ccontr)
assume $\neg(\forall x . x>a \longrightarrow$ poly $p x>0)$
then obtain $x^{\prime}$ where $x^{\prime}>$ a poly $p x^{\prime} \leq 0$ by (auto simp: not-less)
with $A$ and $D$ have $E$ : sgn $\left(\right.$ poly $\left.p x^{\prime}\right) \neq \operatorname{sgn}\left(\right.$ poly $\left.p\left(\max x_{0}(a+1)\right)\right)$
by (auto simp: sgn-real-def split: if-split-asm)
show False
apply (cases $x^{\prime} \max x_{0}(a+1)$ rule: linorder-cases)
using $B E\left\langle x^{\prime}>a\right\rangle$
apply (force dest!: poly-different-sign-imp-root $[$ of - p $]$ )+
done
qed
qed
lemma poly-pos-geq:
$(\forall x::$ real. $x \geq a \longrightarrow$ poly $p x>0) \longleftrightarrow$
poly-inf $p=1 \wedge(\forall x . x \geq a \longrightarrow$ poly $p x \neq 0)$
proof (intro iffI conjI)
assume $A: \forall x::$ real. $x \geq a \longrightarrow$ poly $p x>0$
hence $\forall x::$ real. $x>a \longrightarrow$ poly $p x>0$ by simp
also note poly-pos-greater
finally have poly-inf $p=1(\forall x>a$. poly $p x \neq 0)$ by simp-all
moreover from $A$ have poly $p a>0$ by simp
ultimately show poly-inf $p=1 \forall x \geq a$. poly $p x \neq 0$
by (auto simp: less-eq-real-def)
next
assume poly-inf $p=1 \wedge(\forall x . x \geq a \longrightarrow$ poly $p x \neq 0)$
hence $A$ : poly-inf $p=1$ and
B: poly $p a \neq 0$ and $C: \forall x>a$. poly $p x \neq 0$ by simp-all
from $A$ and $C$ and poly-pos-greater have $\forall x>a$. poly $p x>0$ by simp
moreover with $B$ C poly-IVT-pos[of $a a+1$ p] have poly $p a>0$ by force
ultimately show $\forall x \geq a$. poly $p x>0$ by (auto simp: less-eq-real-def)
qed
lemma poly-pos-less:

```
    (\forallx::real. }x<a\longrightarrow\mathrm{ poly p x>0) }
    poly-neg-inf p=1 ^(\forallx.x<a\longrightarrow poly p x\not=0)
proof (intro iffI conjI)
    assume A: }\forallx::real. x<a\longrightarrow poly p x>0
    have }\wedgex\mathrm{ . poly p (x::real)>0 poly p }x\not=0\mathrm{ by simp
    with A show }\forallx::real. x<a\longrightarrow poly p x\not=0 by aut
    from poly-lim-neg-inf obtain }\mp@subsup{x}{0}{}\mathrm{ where
        \forallx\leqx. . sgn (poly p x) = poly-neg-inf p
        by (auto simp: eventually-at-bot-linorder)
    hence poly-neg-inf p = sgn (poly p (min }\mp@subsup{x}{0}{}(a-1)))\mathrm{ by simp
    also from }A\mathrm{ have ... = 1 by force
    finally show poly-neg-inf p=1.
next
    assume poly-neg-inf p=1^(\forallx.x<a\longrightarrowpoly p x\not=0)
    hence A: poly-neg-inf p=1 and
        B: (\forallx.x<a\longrightarrow poly px\not=0) by simp-all
    from poly-lim-neg-inf obtain }\mp@subsup{x}{0}{}\mathrm{ where
        C:\forallx\leqx 稆.sgn (poly p x)= poly-neg-inf p
        by (auto simp: eventually-at-bot-linorder)
    hence sgn (poly p (min }\mp@subsup{x}{0}{}(a-1)))=\mathrm{ poly-neg-inf }p\mathrm{ by simp
    with }A\mathrm{ have D: sgn (poly p (min x x (a-1))) = 1 by simp
    show }\forallx.x<a\longrightarrow\mathrm{ poly p }x>
    proof (rule ccontr)
    assume }\neg(\forallx.x<a\longrightarrow\mathrm{ poly p x>0)
    then obtain \mp@subsup{x}{}{\prime}}\mathrm{ where }\mp@subsup{x}{}{\prime}<a poly p \mp@subsup{x}{}{\prime}\leq0 by (auto simp: not-less
    with }A\mathrm{ and D have E: sgn (poly p x')}\not=\operatorname{sgn}(\mathrm{ poly p (min x 
            by (auto simp: sgn-real-def split: if-split-asm)
    show False
            apply (cases \mp@subsup{x}{}{\prime}}\operatorname{min}\mp@subsup{x}{0}{}(a-1)\mathrm{ rule: linorder-cases)
            using B E〈\mp@subsup{x}{}{\prime}<a\rangle
                apply (auto dest!: poly-different-sign-imp-root[of - - p])+
            done
    qed
qed
lemma poly-pos-leq:
    (}\forallx::\mathrm{ real. }x\leqa\longrightarrow\mathrm{ poly p }x>0)
        poly-neg-inf p=1 ^(\forallx.x\leqa\longrightarrowpoly p x = 0)
proof (intro iffI conjI)
    assume A: \forallx::real. x\leqa\longrightarrow poly px>0
    hence }\forallx::real. x<a\longrightarrow poly p x>0 by sim
    also note poly-pos-less
    finally have poly-neg-inf p=1 ( }\forallx<a.poly p x\not=0) by simp-all
    moreover from A have poly pa>0 by simp
    ultimately show poly-neg-inf p=1 \forallx\leqa.poly p x\not=0
        by (auto simp: less-eq-real-def)
next
    assume poly-neg-inf p=1 ^(\forallx.x\leqa\longrightarrowpoly p x\not=0)
```

hence $A$ : poly-neg-inf $p=1$ and
B: poly $p a \neq 0$ and $C: \forall x<a$. poly $p x \neq 0$ by simp-all
from $A$ and $C$ and poly-pos-less have $\forall x<a$. poly $p x>0$ by simp moreover with $B$ C poly-IVT-neg $[$ of $a-1$ a p] have poly $p a>0$ by force ultimately show $\forall x \leq a$. poly $p x>0$ by (auto simp: less-eq-real-def)

## qed

lemma poly-pos-between-less-less:

```
    \((\forall x::\) real. \(a<x \wedge x<b \longrightarrow\) poly \(p x>0) \longleftrightarrow\)
    \((a \geq b \vee\) poly \(p((a+b) / 2)>0) \wedge(\forall x . a<x \wedge x<b \longrightarrow\) poly \(p x \neq 0)\)
proof (intro iffI conjI)
    assume \(A: \forall x . a<x \wedge x<b \longrightarrow\) poly \(p x>0\)
    have \(\wedge x\). poly \(p(x::\) real \()>0 \Longrightarrow\) poly \(p x \neq 0\) by simp
    with \(A\) show \(\forall x\) :: real. \(a<x \wedge x<b \longrightarrow\) poly \(p x \neq 0\) by auto
    from \(A\) show \(a \geq b \vee\) poly \(p((a+b) / 2)>0\) by (cases \(a<b\), auto)
next
    assume \((b \leq a \vee 0<\) poly \(p((a+b) / 2)) \wedge(\forall x . a<x \wedge x<b \longrightarrow\) poly \(p x \neq 0)\)
    hence \(A: b \leq a \vee 0<\) poly \(p((a+b) / 2)\) and
        \(B: \forall x . a<x \wedge x<b \longrightarrow\) poly \(p x \neq 0\) by simp-all
    show \(\forall x . a<x \wedge x<b \longrightarrow\) poly \(p x>0\)
    proof (cases \(a \geq b\), simp, clarify, rule-tac ccontr,
        simp only: not-le not-less)
    fix \(x\) assume \(a<b a<x x<b\) poly \(p x \leq 0\)
    with \(B\) have poly \(p x<0\) by (simp add: less-eq-real-def)
    moreover from \(A\) and \(\langle a<b\rangle\) have poly \(p((a+b) / 2)>0\) by simp
    ultimately have sgn \((\) poly \(p x) \neq \operatorname{sgn}(\) poly \(p((a+b) / 2))\) by simp
    thus False using \(B\)
        apply (cases \(x(a+b) / 2\) rule: linorder-cases)
        apply (drule poly-different-sign-imp-root \([\) of \(-p]\), assumption,
        insert \(\langle a<b\rangle\langle a<x\rangle\langle x<b\rangle\), force) []
        apply simp
        apply (drule poly-different-sign-imp-root \([o f-p]\), simp,
            insert \(\langle a<b\rangle\langle a<x\rangle\langle x<b\rangle\), force)
        done
    qed
qed
```

lemma poly-pos-between-less-leq:
$(\forall x::$ real. $a<x \wedge x \leq b \longrightarrow$ poly $p x>0) \longleftrightarrow$
$(a \geq b \vee$ poly $p b>0) \wedge(\forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x \neq 0)$
proof (intro iffI conjI)
assume $A: \forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x>0$
have $\wedge x$. poly $p(x::$ real $)>0 \Longrightarrow$ poly $p x \neq 0$ by simp
with $A$ show $\forall x:$ :real. $a<x \wedge x \leq b \longrightarrow$ poly $p x \neq 0$ by auto
from $A$ show $a \geq b \vee$ poly $p b>0$ by (cases $a<b$, auto)
next
assume $(b \leq a \vee 0<$ poly $p b) \wedge(\forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x \neq 0)$
hence $A: b \leq a \vee 0<$ poly $p b$ and $B: \forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x \neq 0$
by simp-all

```
    show }\forallx.a<x\wedgex\leqb\longrightarrow\mathrm{ poly p x>0
    proof (cases a\geqb, simp, clarify, rule-tac ccontr,
        simp only: not-le not-less)
    fix x assume a<b a<x x\leqb poly p x \leq 0
    with B have poly p x<0 by (simp add: less-eq-real-def)
    moreover from A and }\langlea<b\rangle\mathrm{ have poly p b>0 by simp
    ultimately have }x<b\mathrm{ using <x s b> by (auto simp: less-eq-real-def)
    from <poly p x < 0 > and <poly p b>0 <
        have sgn (poly p x)}\not=\operatorname{sgn}(\mathrm{ poly p b) by simp
    from poly-different-sign-imp-root[OF \langlex<b\ranglethis] and B and <x>a>
        show False by auto
    qed
qed
lemma poly-pos-between-leq-less:
    ( }\forall\mathrm{ x::real. a s x ^x<b P poly p x>0) 山
        (a\geqb\vee poly pa>0) ^( }\forallx.a\leqx\wedgex<b\longrightarrow\mathrm{ poly p x = 0)
proof (intro iffI conjI)
    assume A:\forallx.a\leqx^x<b\longrightarrow poly p x>0
    have }\bigwedgex\mathrm{ . poly p(x::real)>0 poly p }x\not=0\mathrm{ by simp
    with A show }\forallx::real. a\leqx\wedgex<b\longrightarrow poly px\not=0 by aut
    from A show }a\geqb\vee poly pa>0 by (cases a<b, auto
next
    assume (b\leqa\vee 0< poly pa) ^( }\forallx.a\leqx\wedgex<b\longrightarrow\mathrm{ poly p x = 0)
    hence A:b\leqa\vee0< poly pa and B:\forallx.a\leqx\wedge x<b\longrightarrow poly p x\not=0
        by simp-all
    show }\forallx.a\leqx\wedgex<b\longrightarrow\mathrm{ poly p x>0
    proof (cases a \geqb, simp, clarify, rule-tac ccontr,
                simp only: not-le not-less)
    fix x assume a<ba\leqx x < b poly p x \leq 0
    with B have poly p x<0 by (simp add: less-eq-real-def)
    moreover from }A\mathrm{ and }\langlea<b\rangle have poly pa>0 by sim
    ultimately have }x>a\mathrm{ using <x }\geqa\rangle\mathrm{ by (auto simp:less-eq-real-def)
    from <poly p x<0〉 and <poly p a>0`
                have sgn (poly pa)}\not=\operatorname{sgn}(\mathrm{ poly p x) by simp
    from poly-different-sign-imp-root[OF \langlex>a\ranglethis] and B and <x< < >
        show False by auto
    qed
qed
lemma poly-pos-between-leq-leq:
    ( }\forall\mathrm{ x::real. }a\leqx\wedgex\leqb\longrightarrow\mathrm{ poly p }x>0)
        (a>b\vee poly pa>0)^(\forallx.a\leqx\wedge x\leqb \longrightarrow poly p x\not=0)
proof (intro iffI conjI)
    assume A: \forallx. a s x^x\leqb \longrightarrow poly p x>0
    have }\x\mathrm{ . poly p (x::real)>0 > poly p }x\not=0\mathrm{ by simp
    with A show }\forallx::real. a\leqx\wedgex\leqb\longrightarrow\mathrm{ poly px}=0\mathrm{ by auto
    from A show a>b\vee poly pa>0 by (cases a \leqb, auto)
next
```

```
    assume (b<a\vee0< poly pa) ^( }\forallx.a\leqx\wedgex\leqb\longrightarrow\mathrm{ poly p x = 0)
    hence }A:b<a\vee0<\mathrm{ poly pa and B: }\forallx.a\leqx\wedgex\leqb\longrightarrow poly px\not=
        by simp-all
    show }\forallx.a\leqx\wedgex\leqb\longrightarrow\mathrm{ poly p x>0
    proof (cases a>b, simp, clarify, rule-tac ccontr,
        simp only: not-le not-less)
    fix x assume a\leqb a\leqx x \leq b poly p x \leq 0
    with }B\mathrm{ have poly p x<0 by (simp add: less-eq-real-def)
    moreover from A and <a\leqb\rangle have poly pa>0 by simp
    ultimately have }x>a\mathrm{ using <x }\geqa\rangle\mathrm{ by (auto simp:less-eq-real-def)
    from <poly p x<0〉 and <poly p a>0〉
        have sgn (poly pa)}\not=\operatorname{sgn}(\mathrm{ poly p x) by simp
    from poly-different-sign-imp-root[OF\langlex> a\rangle this] and B and <x\leqb\rangle
        show False by auto
    qed
qed
end
```


## 2 Proof of Sturm's Theorem

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


### 2.1 Sign changes of polynomial sequences

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

```
definition sign-changes where
sign-changes \(p s(x::\) real \()=\)
    length (remdups-adj \((\) filter \((\lambda x . x \neq 0)(\operatorname{map}(\lambda p . \operatorname{sgn}(\) poly \(p x)) p s)))-1\)
```

The number of sign changes of a sequence distributes over a list in the sense that the number of sign changes of a sequence $p_{1}, \ldots, p_{i}, \ldots, p_{n}$ at $x$ is the same as the sum of the sign changes of the sequence $p_{1}, \ldots, p_{i}$ and $p_{i}, \ldots, p_{n}$ as long as $p_{i}(x) \neq 0$.
lemma sign-changes-distrib:

```
poly \(p x \neq 0 \Longrightarrow\)
    sign-changes \(\left(p s_{1} @[p] @ p s_{2}\right) x=\)
    sign-changes \(\left(p s_{1} @[p]\right) x+\) sign-changes \(\left([p] @ p s_{2}\right) x\)
by (simp add: sign-changes-def sgn-zero-iff, subst remdups-adj-append, simp)
```

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

```
lemma sign-changes-cong:
    assumes length ps = length ps'
    assumes }\foralli<length ps.sgn (poly (ps!i) x)=\operatorname{sgn}(poly (ps'!i) y)
    shows sign-changes ps x = sign-changes ps'}
proof-
    from assms(2) have A: map ( \lambdap. sgn (poly p x)) ps = map ( \lambdap. sgn (poly p y))
ps'
    proof (induction rule: list-induct2[OF assms(1)])
        case 1
            then show ?case by simp
    next
        case (2 p ps p'ps')
            from 2(3)
            have }\foralli<length ps. sgn (poly (ps!i) x)
                        sgn (poly (ps'!i) y) by auto
            from 2(2)[OF this] 2(3) show ?case by auto
    qed
    show ?thesis unfolding sign-changes-def by (simp add: A)
qed
lemma sign-changes-cong':
    assumes }\forallp\in\mathrm{ set ps. sgn (poly p x)= sgn (poly p y)
    shows sign-changes ps x = sign-changes ps y
using assms by (intro sign-changes-cong, simp-all)
For a sequence of polynomials of length 3 , if the first and the third polynomial have opposite and nonzero sign at some \(x\), the number of sign changes is always 1 , irrespective of the sign of the second polynomial.
```

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

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

```
definition sign-changes-inf where
sign-changes-inf ps =
    length (remdups-adj \((\) filter \((\lambda x . x \neq 0)(\) map poly-inf \(p s)))-1\)
```

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

### 2.2 Definition of Sturm sequences locale

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

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

```
locale quasi-sturm-seq \(=\)
fixes \(p s\) :: (real poly) list
assumes last-ps-sgn-const[simp]:
    \(\wedge x y . \operatorname{sgn}(\) poly \((\) last \(p s) x)=\operatorname{sgn}(\) poly \((\) last ps) \(y)\)
    assumes \(p s\)-not-Nil[simp]: \(p s \neq[]\)
    assumes signs: \(\backslash i x . \llbracket i<\) length \(p s-2 ;\) poly \((p s!(i+1)) x=0 \rrbracket\)
    \(\Longrightarrow(\) poly \((p s!(i+2)) x) *(\) poly \((p s!i) x)<0\)
```

Now we define a Sturm sequence $p_{1}, \ldots, p_{n}$ of a polynomial $p$ in the following way:

- The sequence contains at least two elements.
- $p$ is the first polynomial, i. e. $p_{1}=p$.
- At any root $x$ of $p, p_{2}$ and $p$ have opposite sign left of $x$ and the same sign right of $x$ in some neighbourhood around $x$.
- The first two polynomials in the sequence have no common roots.
- If the middle one of three adjacent polynomials has a root at $x$, the other two have opposite and nonzero signs at $x$.

```
locale sturm-seq \(=\) quasi-sturm-seq +
    fixes \(p\) :: real poly
    assumes \(h d-p s-p[s i m p]: h d p s=p\)
    assumes length-ps-ge-2 2 simp]: length \(p s \geq 2\)
    assumes deriv: \(\backslash x_{0}\). poly p \(x_{0}=0 \Longrightarrow\)
        eventually \((\lambda x\). sgn \((\) poly \((p * p s!1) x)=\)
            (if \(x>x_{0}\) then 1 else -1 )) (at \(x_{0}\) )
    assumes \(p\)-squarefree: \(\backslash x\). \(\neg(\) poly \(p x=0 \wedge\) poly \((p s!1) x=0)\)
begin
Any Sturm sequence is obviously a Quasi-Sturm sequence.
lemma quasi-sturm-seq: quasi-sturm-seq ps ..
end
```

Any suffix of a Quasi-Sturm sequence is again a Quasi-Sturm sequence.
lemma quasi-sturm-seq-Cons: assumes quasi-sturm-seq $(p \# p s)$ and $p s \neq[]$
shows quasi-sturm-seq ps

```
proof (unfold-locales)
    show ps }\not=[]\mathrm{ by fact
next
    from assms(1) interpret quasi-sturm-seq p#ps .
    fix }x
    from last-ps-sgn-const and <ps \not= []>
        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 ix
    assume i< length ps - 2 and poly (ps!(i+1)) x = 0
    with signs[of i+1]
        show poly (ps!(i+2)) x* poly (ps!i) x<0 by simp
qed
```


### 2.3 Auxiliary lemmas about roots and sign changes

lemma sturm-adjacent-root-aux:
assumes $i<$ length ( $p s$ :: real poly list) -1
assumes poly $(p s!i) x=0$ and poly $(p s!(i+1)) x=0$
assumes $\bigwedge i x . \llbracket i<$ length $p s-2$; poly $(p s!(i+1)) x=0 \rrbracket$

$$
\Longrightarrow \operatorname{sgn}(\text { poly }(p s!(i+2)) x)=-\operatorname{sgn}(\text { poly }(p s!i) x)
$$

shows $\forall j \leq i+1$. poly $(p s!j) x=0$
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 $\operatorname{sgn}($ poly $(p s!(i+2)) x)=-\operatorname{sgn}(p o l y(p s!i) x)$
by (intro assms(4)) simp-all
with Suc.prems(3) have poly ( $p s!i$ ) $x=0$ by (simp add: sgn-zero-iff)
with Suc.prems have $\forall j \leq i+1$. poly $(p s!j) x=0$
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 px\not=0^ poly qx=0 then
    [p,q,r] # split-sign-changes (r#ps)x
    else
```

```
\([p, q] \#\) split-sign-changes \((q \# r \# p s) x)\)
```

lemma (in quasi-sturm-seq) split-sign-changes-subset[dest]:
$p s^{\prime} \in$ set (split-sign-changes $\left.p s x\right) \Longrightarrow$ set $p s^{\prime} \subseteq$ set $p s$
apply (insert ps-not-Nil)
apply (induction ps $x$ rule: split-sign-changes.induct)
apply (simp, simp, rename-tac p q r ps $x$,
case-tac poly $p x \neq 0 \wedge$ poly $q x=0$, auto)
done
A custom induction rule for split-sign-changes that uses the fact that all the intermediate parameters in calls of split-sign-changes are quasi-Sturm sequences.
lemma (in quasi-sturm-seq) split-sign-changes-induct:
$\llbracket \bigwedge p x . P[p] x ; \bigwedge p q x . q u a s i-$ sturm-seq $[p, q] \Longrightarrow P[p, q] x ;$
$\bigwedge p$ q rps x. quasi-sturm-seq $(p \# q \# r \# p s) \Longrightarrow$
«poly $p x \neq 0 \Longrightarrow$ poly $q x=0 \Longrightarrow P(r \# p s) x$;
poly $q x \neq 0 \Longrightarrow P(q \# r \# p s) x$;
poly $p x=0 \Longrightarrow P(q \# r \# p s) x \rrbracket$
$\Longrightarrow P(p \# q \# r \# p s) x \rrbracket \Longrightarrow P p s x$
proof goal-cases
case prems: 1
have quasi-sturm-seq ps ..
with prems show ?thesis
proof (induction ps $x$ rule: split-sign-changes.induct)
case ( $3 p q r p s x$ )
show ?case
proof (rule 3(5)[OF 3(6)])
assume $A$ : poly $p x \neq 0$ poly $q x=0$
from 3(6) have quasi-sturm-seq ( $r \# p s$ )
by (force dest: quasi-sturm-seq-Cons)
with $3 A$ show $P(r \# p s) x$ by blast next
assume $A$ : poly q $x \neq 0$
from 3(6) have quasi-sturm-seq ( $q \# r \# p s$ )
by (force dest: quasi-sturm-seq-Cons)
with $3 A$ show $P(q \# r \# p s) x$ by blast next
assume $A$ : poly $p x=0$
from 3(6) have quasi-sturm-seq ( $q \# r \# p s$ )
by (force dest: quasi-sturm-seq-Cons)
with $3 A$ show $P(q \# r \# p s) x$ 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-changes \({ }^{\prime} \equiv \lambda p s x\).
    \(\sum p s^{\prime} \leftarrow\) split-sign-changes ps \(x\). sign-changes \(p s^{\prime} x\)
    shows sign-changes' ps \(x_{0}=\) sign-changes ps \(x_{0}\)
using assms(1)
proof (induction \(x_{0}\) rule: split-sign-changes-induct)
case ( 3 p q r ps \(x_{0}\) )
    hence poly \(p x_{0} \neq 0\) by simp
    note \(I H=3(2,3,4)\)
    show? case
    proof (cases poly q \(x_{0}=0\) )
    case True
        from 3 interpret quasi-sturm-seq \(p \# q \# r \# p s\) by simp
        from signs \([\) of 0\(]\) and True have
                sgn-r-x0: poly \(r x_{0} *\) poly \(p x_{0}<0\) by simp
            with 3 have poly \(r x_{0} \neq 0\) by force
            from sign-changes-distrib[OF this, of \([p, q] p s]\)
                have sign-changes \((p \# q \# r \# p s) x_{0}=\)
                            sign-changes \(([p, q, r]) x_{0}+\) sign-changes \((r \# p s) x_{0}\) by simp
            also have sign-changes \((r \# p s) x_{0}=\) sign-changes \(^{\prime}(r \# p s) x_{0}\)
                using 〈poly \(q x_{0}=0\) 〉〈poly p \(x_{0} \neq 0\) 〉 3(5) \(\left\langle\right.\) poly \(\left.r x_{0} \neq 0\right\rangle\)
                by (intro IH(1)[symmetric], simp-all)
            finally show ?thesis unfolding sign-changes'-def
                using True \(\left\langle\right.\) poly \(p x_{0} \neq 0\) 〉 by simp
    next
        case False
            from sign-changes-distrib[OF this, of \([p] r \# p s]\)
                have sign-changes \((p \# q \# r \# p s) x_{0}=\)
                    sign-changes \(([p, q]) x_{0}+\) sign-changes \((q \# r \# p s) x_{0}\) by simp
            also have sign-changes \((q \# r \# p s) x_{0}=\) sign-changes \(^{\prime}(q \# r \# p s) x_{0}\)
                using 〈poly \(q x_{0} \neq 0\) 〉〈poly p \(x_{0} \neq 0\) 〉 \(3(5)\)
                by (intro IH(2)[symmetric], simp-all)
            finally show ?thesis unfolding sign-changes'-def
                using False by simp
    qed
qed (simp-all add: sign-changes-def sign-changes'-def)
```

We now prove that if $p(x) \neq 0$ ，the number of sign changes of a Sturm sequence of $p$ at $x$ is constant in a neighbourhood of $x$ ．
lemma（in quasi－sturm－seq）split－sign－changes－correct－nbh：
assumes poly（hd ps）$x_{0} \neq 0$
defines sign－changes ${ }^{\prime} \equiv \lambda x_{0}$ ps $x$ ．
$\sum p s^{\prime} \leftarrow$ split－sign－changes ps $x_{0}$ ．sign－changes $p s^{\prime} x$
shows eventually（ $\lambda x$ ．sign－changes＇$x_{0}$ ps $x=$ sign－changes $\left.p s x\right)\left(\right.$ at $\left.x_{0}\right)$
proof（rule eventually－mono）
show eventually $\left(\lambda x . \forall p \in\left\{p \in\right.\right.$ set ps．poly $\left.p x_{0} \neq 0\right\}$ ．sgn $($ poly $p x)=\operatorname{sgn}($ poly $p x_{0}$ ））（at $x_{0}$ ）
by（rule eventually－ball－finite，auto intro：poly－neighbourhood－same－sign）
next

```
fix \(x\)
show \(\left(\forall p \in\left\{p \in\right.\right.\) set ps. poly \(\left.p x_{0} \neq 0\right\}\). sgn \((\) poly \(p x)=\operatorname{sgn}\left(\right.\) poly \(\left.\left.p x_{0}\right)\right) \Longrightarrow\)
        sign-changes \({ }^{\prime} x_{0}\) ps \(x=\) sign-changes \(p s x\)
    proof -
    fix \(x\) assume \(n b h: \forall p \in\left\{p \in\right.\) set \(p\). poly \(\left.p x_{0} \neq 0\right\}\). sgn (poly \(\left.p x\right)=\operatorname{sgn}(\) poly
p \(x_{0}\) )
    thus sign-changes' \(x_{0}\) ps \(x=\) sign-changes ps \(x\) using \(\operatorname{assms}(1)\)
    proof (induction \(x_{0}\) rule: split-sign-changes-induct)
    case ( 3 p q r ps \(x_{0}\) )
        hence poly \(p x_{0} \neq 0\) by simp
        note \(I H=3(2,3,4)\)
        show ?case
        proof (cases poly \(q x_{0}=0\) )
            case True
                from 3 interpret quasi-sturm-seq \(p \# q \# r \# p s\) by simp
                    from signs [of 0\(]\) and True have
                    sgn- \(r-x 0\) : poly \(r x_{0} *\) poly \(p x_{0}<0\) by simp
            with 3 have poly \(r x_{0} \neq 0\) by force
            with nbh \(3(5)\) have poly \(r x \neq 0\) by (auto simp: sgn-zero-iff)
            from sign-changes-distrib[OF this, of \([p, q] p s]\)
                have sign-changes \((p \# q \# r \# p s) x=\)
                        sign-changes \(([p, q, r]) x+\) sign-changes \((r \# p s) x\) by simp
            also have sign-changes \((r \# p s) x=\) sign-changes' \(x_{0}(r \# p s) x\)
                using «poly \(q x_{0}=0\) nbh «poly p \(x_{0} \neq 0\) 〉 \(3(5)\left\langle\right.\) poly \(r x_{0} \neq 0\) 〉
                by (intro \(\operatorname{IH}(1)[\) symmetric], simp-all)
            finally show ?thesis unfolding sign-changes'-def
                using True \(\left\langle\right.\) poly p \(x_{0} \neq 0 〉\) by simp
        next
            case False
            with nbh 3(5) have poly \(q x \neq 0\) by (auto simp: sgn-zero-iff)
            from sign-changes-distrib[OF this, of \([p] r \# p s]\)
                have sign-changes \((p \# q \# r \# p s) x=\)
                        sign-changes \(([p, q]) x+\) sign-changes \((q \# r \# p s) x\) by \(\operatorname{simp}\)
            also have sign-changes \((q \# r \# p s) x=\) sign-changes' \(x_{0}(q \# r \# p s) x\)
                using 〈poly \(q x_{0} \neq 0\) 〉 nbh 〈poly p \(x_{0} \neq 0\) 〉 \(3(5)\)
                by (intro \(I H\) (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 $(h d p s) x_{0} \neq 0$ and $p s^{\prime} \in$ set (split-sign-changes ps $x_{0}$ )
shows eventually $\left(\lambda x\right.$. sign-changes $p s^{\prime} x=$ sign-changes $\left.p s^{\prime} x_{0}\right)\left(\right.$ at $\left.x_{0}\right)$
using assms

```
proof (induction \(x_{0}\) rule: split-sign-changes-induct)
    case (1px)
        thus ?case by (simp add: sign-changes-def)
next
    case (2 \(p\) q \(x_{0}\) )
    hence \(\left[\operatorname{simp} p: p s^{\prime}=[p, q]\right.\) by \(\operatorname{simp}\)
    from 2 have poly \(p x_{0} \neq 0\) by simp
    from 2(1) interpret quasi-sturm-seq \([p, q]\).
    from poly-neighbourhood-same-sign \(\left[O F\left\langle p o l y ~ p ~ x_{0} \neq 0\right\rangle\right]\)
        have eventually \(\left(\lambda x\right.\). sgn \((\) poly \(p x)=\operatorname{sgn}\left(\right.\) poly \(\left.\left.p x_{0}\right)\right)\left(\right.\) at \(\left.x_{0}\right)\).
    moreover from last-ps-sgn-const
            have sgn-q: \(\bigwedge x\). sgn \((\) poly \(q x)=\operatorname{sgn}\left(\right.\) poly \(\left.q x_{0}\right)\) by simp
    ultimately have \(A\) : eventually \((\lambda x . \forall p \in \operatorname{set}[p, q]\). sgn \((\) poly \(p x)=\)
                                    \(\operatorname{sgn}\left(\right.\) poly \(\left.\left.p x_{0}\right)\right)\left(\right.\) at \(\left.x_{0}\right)\) by \(\operatorname{simp}\)
    thus ?case by (force intro: eventually-mono[OF A]
                                    sign-changes-cong')
next
    case ( 3 p qrps \({ }^{\prime \prime} x_{0}\) )
    hence \(p\)-not- 0 : poly \(p x_{0} \neq 0\) by simp
    note sturm \(=3(1)\)
    note \(I H=3(2,3)\)
    note \(p s^{\prime \prime}\)-props \(=3(6)\)
    show ?case
    proof (cases poly \(q x_{0}=0\) )
        case True
            note \(q-0=\) this
            from sturm interpret quasi-sturm-seq \(p \# q \# r \# p s^{\prime \prime}\).
            from signs \([\) of 0\(]\) and \(q-0\)
                    have signs': poly \(r x_{0} *\) poly \(p x_{0}<0\) by simp
            with \(p\)-not- 0 have \(r\)-not- 0 : poly \(r x_{0} \neq 0\) by force
            show ?thesis
            proof (cases ps \(s^{\prime} \in\) set (split-sign-changes \(\left.\left(r \# p s^{\prime \prime}\right) x_{0}\right)\) )
                case True
                    show ?thesis by (rule \(I H(1)\), fact, fact, simp add: r-not-0, fact)
                    next
                    case False
                                    with \(p s^{\prime \prime}\)-props \(p\)-not- \(0 q\) - 0 have \(p s^{\prime}\)-props: \(p s^{\prime}=[p, q, r]\) by simp
                                    from signs \([o f 0]\) and \(q-0\)
                            have sgn-r: poly \(r x_{0} *\) poly \(p x_{0}<0\) by simp
                    from \(p\)-not-0 sgn-r
                                    have \(A\) : eventually \(\left(\lambda x\right.\). sgn \((\) poly \(p x)=\operatorname{sgn}\left(\right.\) poly \(\left.p x_{0}\right) \wedge\)
                                    \(\operatorname{sgn}(\) poly \(r x)=\operatorname{sgn}\left(\right.\) poly \(\left.\left.r x_{0}\right)\right)\left(\right.\) at \(\left.x_{0}\right)\)
                                    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 \(A: \operatorname{sgn}(\) poly \(p x)=\operatorname{sgn}\left(\right.\) poly \(\left.p x_{0}\right)\)
                                    and \(B: \operatorname{sgn}(\) poly \(r x)=\operatorname{sgn}\left(\right.\) poly \(\left.r x_{0}\right)\)
```

```
                    have prod-neg: \a (b::real).\llbracketa>0;b>0;a*b<0\rrbracket\Longrightarrow False
                    \a(b::real). \llbracketa<0;b<0;a*b<0\rrbracket\Longrightarrow False
                    by (drule mult-pos-pos, simp, simp,
                drule mult-neg-neg, simp, simp)
            from A and <poly p x 0 # 0 \ show poly p x\not=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\not=0
                        by (metis sgn-less sgn-mult, metis sgn-0-0)
                    with sgn-r show sgn-r': sgn (poly r x) = - sgn (poly p x)
                        apply (simp add: sgn-real-def not-le not-less
                                    split: if-split-asm, intro conjI impI)
                                    using prod-neg[of poly r x poly p x] apply force+
                done
            show 1 = sign-changes ps' }\mp@subsup{x}{0}{
                        by (subst ps'-props, subst sign-changes-sturm-triple,
                fact, metis A B sgn-r', simp)
            qed
        qed
    next
    case False
        note q-not-0 = this
        show ?thesis
        proof (cases ps' }\in\mathrm{ set (split-sign-changes (q#r# ps')}\mp@subsup{)}{}{\prime
        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]: }\forallp\in\mathrm{ set ps'. poly p }\mp@subsup{x}{0}{}\not=
                    using q-not-0 p-not-0 by simp
            show ?thesis
            proof (rule eventually-mono)
                fix }x\mathrm{ assume }\forallp\in\operatorname{set ps'. sgn (poly p x) = sgn (poly p x ( )
                thus sign-changes ps' }x=\mathrm{ sign-changes ps' }\mp@subsup{x}{0}{
                    by (rule sign-changes-cong')
            next
                show eventually ( }\lambdax.\forallp\inset ps'
                        sgn (poly p x) = sgn (poly p x < )) (at \mp@subsup{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 ( \(h d p s\) ) \(x_{0} \neq 0\)
    shows eventually ( \(\lambda x\). sign-changes ps \(x=\) sign-changes ps \(x_{0}\) ) (at \(\left.x_{0}\right)\)
proof-
    let ?pss \(=\) split-sign-changes ps \(x_{0}\)
    let ? \(f=\lambda\) pss \(x . \sum p s^{\prime} \leftarrow\) pss. sign-changes \(p s^{\prime} x\)
    \{
        fix pss assume \(\bigwedge p s^{\prime} . p s^{\prime} \in\) set pss \(\Longrightarrow\)
            eventually \(\left(\lambda x\right.\). sign-changes \(p s^{\prime} x=\) sign-changes \(\left.p s^{\prime} x_{0}\right)\left(\right.\) at \(\left.x_{0}\right)\)
        hence eventually ( \(\lambda x\). ?f pss \(x=\) ?f pss \(x_{0}\) ) (at \(x_{0}\) )
        proof (induction pss)
            case (Cons ps' pss)
            then show? case
                apply (rule eventually-mono[OF eventually-conj])
                apply (auto simp add: Cons.prems)
                done
        qed simp
    \}
    note \(A=\) this[of ?pss]
    have \(B\) : eventually ( \(\lambda x\). ?f ?pss \(x=\) ?f ?pss \(x_{0}\) ) (at \(x_{0}\) )
        by (rule \(A\), rule hd-nonzero-imp-sign-changes-const-aux[OF assms], simp)
    note \(C=\) split-sign-changes-correct-nbh[OF assms]
    note \(D=\) split-sign-changes-correct \([O F\) assms \(]\)
    note \(E=\) eventually-conj \([\) OF \(B C]\)
    show ?thesis by (rule eventually-mono \([\) OF \(E]\), auto simp: \(D\) )
qed
lemma (in sturm-seq) p-nonzero-imp-sign-changes-const:
    poly \(p x_{0} \neq 0 \Longrightarrow\)
        eventually ( \(\lambda x\). sign-changes ps \(x=\) sign-changes ps \(\left.x_{0}\right)\left(\right.\) at \(\left.x_{0}\right)\)
    using hd-nonzero-imp-sign-changes-const by simp
```

If $x$ is a root of $p$ and $p$ is not the zero polynomial, the number of sign changes of a Sturm chain of $p$ decreases by 1 at $x$.
lemma (in sturm-seq) p-zero:
assumes poly $p x_{0}=0 p \neq 0$
shows eventually ( $\lambda x$. sign-changes ps $x=$
sign-changes ps $x_{0}+\left(\right.$ if $x<x_{0}$ then 1 else 0$\left.)\right)\left(\right.$ at $\left.x_{0}\right)$
proof-
from $p s$-first-two obtain $q p s^{\prime}$ where $[s i m p]: p s=p \# q \# p s^{\prime}$.
hence $p s!1=q$ by $\operatorname{simp}$
have eventually $\left(\lambda x . x \neq x_{0}\right)\left(\right.$ at $\left.x_{0}\right)$
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 $p q \# p s$ ]
interpret quasi-sturm-seq $q \# p s^{\prime}$ by simp
from $\left.\prec p o l y ~ q x_{0} \neq 0\right\rangle$ have eventually $\left(\lambda x\right.$. sign-changes $\left(q \# p s^{\prime}\right) x=$

```
        sign-changes (q#ps') \mp@subsup{x}{0}{})(\mathrm{ at }\mp@subsup{x}{0}{})
    using hd-nonzero-imp-sign-changes-const[where }\mp@subsup{x}{0}{}=\mp@subsup{x}{0}{}]\mathrm{ by simp
}
moreover note poly-neighbourhood-without-roots[OF assms(2)] deriv[OF assms(1)]
ultimately
    have A: eventually ( }\lambdax.x\not=\mp@subsup{x}{0}{}\wedge\mathrm{ poly p x = 0^
```



```
                                    sign-changes (q#ps') x = sign-changes (q#ps') ( x ) (at x x )
            by (simp only: <ps!1 = q>, intro eventually-conj)
    show ?thesis
    proof (rule eventually-mono[OF A], clarify, goal-cases)
    case prems: (1 x)
    from zero-less-mult-pos have zero-less-mult-pos':
        \ab. \llbracket(0::real)<a*b;0<b\rrbracket\Longrightarrow0<a
        by (subgoal-tac a*b=b*a, auto)
    from prems have poly q }x\not=0\mathrm{ and q-sgn: sgn (poly q x)=
                (if }x<\mp@subsup{x}{0}{}\mathrm{ then -sgn (poly p x) else sgn (poly p x))
        by (auto simp add: sgn-real-def elim: linorder-neqE-linordered-idom
                    dest: mult-neg-neg zero-less-mult-pos
                    zero-less-mult-pos' split: if-split-asm)
    from sign-changes-distrib[OF <poly q x = 0`, of [p] ps']
        have sign-changes ps x = sign-changes [p,q] x + sign-changes (q#ps')x
            by simp
    also from q-sgn and <poly p x =0 \
        have sign-changes [p,q] x=( if }x<\mp@subsup{x}{0}{}\mathrm{ then 1 else 0)
        by (simp add: sign-changes-def sgn-zero-iff split: if-split-asm)
    also note prems(4)
    also from assms(1) have sign-changes (q#ps') \mp@subsup{x}{0}{}=\mathrm{ sign-changes ps }\mp@subsup{x}{0}{}
        by (simp add: sign-changes-def)
    finally show?case by simp
qed
qed
```

With these two results, we can now show that if $p$ is nonzero, the number of roots in an interval of the form $(a ; b]$ is the difference of the sign changes of a Sturm sequence of $p$ at $a$ and $b$.
First, however, we prove the following auxiliary lemma that shows that if a function $f: \mathbb{R} \rightarrow \mathbb{N}$ is locally constant at any $x \in(a ; b]$, it is constant across the entire interval $(a ; b]$ :
lemma count-roots-between-aux:
assumes $a \leq b$
assumes $\forall x::$ real. $a<x \wedge x \leq b \longrightarrow$ eventually $(\lambda \xi . f \xi=(f x::$ nat $)$ ) (at $x)$
shows $\forall x . a<x \wedge x \leq b \longrightarrow f x=f b$
proof (clarify)
fix $x$ assume $x>a x \leq b$
with assms have $\forall x^{\prime} . x \leq x^{\prime} \wedge x^{\prime} \leq b \longrightarrow$
eventually $\left(\lambda \xi . f \xi=f x^{\prime}\right)\left(\right.$ at $\left.x^{\prime}\right)$ by auto
from fun-eq-in-ivl[OF $\langle x \leq b\rangle$ this $]$ show $f x=f b$.
qed

Now we can prove the actual root-counting theorem:

```
theorem (in sturm-seq) count-roots-between:
    assumes [simp]: \(p \neq 0 a \leq b\)
    shows sign-changes ps \(a-\) sign-changes ps \(b=\)
            card \(\{x . x>a \wedge x \leq b \wedge\) poly \(p x=0\}\)
proof-
    have sign-changes ps \(a-\) int (sign-changes ps \(b)=\)
                                    card \(\{x . x>a \wedge x \leq b \wedge\) poly \(p x=0\}\) using \(\langle a \leq b\rangle\)
    proof (induction card \(\{x . x>a \wedge x \leq b \wedge\) poly \(p x=0\}\) arbitrary: \(a b\)
                rule: less-induct)
    case (less a b)
        show ?case
        proof (cases \(\exists x . a<x \wedge x \leq b \wedge\) poly \(p x=0\) )
            case False
                hence no-roots: \(\{x . a<x \wedge x \leq b \wedge\) 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] of-nat-eq-iff,
                        cases poly \(p a=0\) )
                case False
                            with no-roots show sign-changes ps \(a=\) sign-changes \(p s b\)
                            by (force intro: fun-eq-in-ivl \(\langle a \leq b\rangle\)
                                    p-nonzero-imp-sign-changes-const)
                    next
                        case True
                        have \(A: \forall x . a<x \wedge x \leq b \longrightarrow\) sign-changes \(p s x=\) sign-changes \(p s b\)
                            apply (rule count-roots-between-aux, fact, clarify)
                            apply (rule p-nonzero-imp-sign-changes-const)
                            apply (insert False, simp)
                                done
                                    have eventually ( \(\lambda x . x>a \longrightarrow\)
                                    sign-changes \(p s x=\) sign-changes ps a) (at a)
                            apply (rule eventually-mono [OF p-zero[OF «poly p \(a=0\rangle\langle p \neq\)
0) ]])
                        apply force
                        done
                then obtain \(\delta\) where \(\delta\)-props:
                    \(\delta>0 \forall x . x>a \wedge x<a+\delta \longrightarrow\)
                        sign-changes \(p\) s \(x=\) sign-changes \(p s a\)
                        by (auto simp: eventually-at dist-real-def)
                show sign-changes ps \(a=\) sign-changes \(p\) s \(b\)
            proof (cases \(a=b\) )
                case False
                        define \(x\) where \(x=\min (a+\delta / 2) b\)
                        with False have \(a<x x<a+\delta x \leq b\)
                            using \(\langle\delta>0\rangle\langle a \leq b\rangle\) by simp-all
                    from \(\delta\)-props \(\langle a<x\rangle\langle x<a+\delta\rangle\)
```

have sign-changes ps $a=$ sign-changes $p s x$ by simp
also from $A\langle a\langle x\rangle\langle x \leq b\rangle$ have $\ldots=$ sign-changes $p s b$
by blast
finally show? ?thesis .
qed $\operatorname{simp}$
qed
next
case True
from poly-roots-finite[OF assms(1)]
have fin: finite $\{x . x>a \wedge x \leq b \wedge$ poly $p x=0\}$
by (force intro: finite-subset)
from True have $\{x . x>a \wedge x \leq b \wedge$ poly $p x=0\} \neq\{ \}$ by blast with fin have card-greater- 0 :
card $\{x . x>a \wedge x \leq b \wedge$ poly $p x=0\}>0$ by fastforce
define $x_{2}$ where $x_{2}=\operatorname{Min}\{x . x>a \wedge x \leq b \wedge$ poly $p x=0\}$
from Min-in[OF fin] and True
have $x_{2}$-props: $x_{2}>a x_{2} \leq b$ poly $p x_{2}=0$
unfolding $x_{2}$-def by blast+
from Min-le[OF fin] $x_{2}$-props
have $x_{2}$-le: $\bigwedge x^{\prime} . \llbracket x^{\prime}>a ; x^{\prime} \leq b ;$ poly $p x^{\prime}=0 \rrbracket \Longrightarrow x_{2} \leq x^{\prime}$
unfolding $x_{2}$-def by simp
have left: $\left\{x . a<x \wedge x \leq x_{2} \wedge\right.$ poly $\left.p x=0\right\}=\left\{x_{2}\right\}$
using $x_{2}$-props $x_{2}$-le by force
hence [simp]: card $\left\{x . a<x \wedge x \leq x_{2} \wedge\right.$ poly $\left.p x=0\right\}=1$ by simp
from $p$-zero $\left[\right.$ OF $\left\langle\right.$ poly $\left.p x_{2}=0\right\rangle\langle p \neq 0\rangle$, unfolded eventually-at dist-real-def] guess $\varepsilon$..
hence $\varepsilon$-props: $\varepsilon>0$

$$
\forall x . x \neq x_{2} \wedge\left|x-x_{2}\right|<\varepsilon \longrightarrow
$$

sign-changes ps $x=$ sign-changes ps $x_{2}+$ (if $x<x_{2}$ then 1 else 0 ) by auto
define $x_{1}$ where $x_{1}=\max \left(x_{2}-\varepsilon /\right.$ 2) $a$
have $\left|x_{1}-x_{2}\right|<\varepsilon$ using $\langle\varepsilon>0\rangle x_{2}$-props by (simp add: $x_{1}$-def)
hence sign-changes ps $x_{1}=$
(if $x_{1}<x_{2}$ then sign-changes ps $x_{2}+1$ else sign-changes ps $x_{2}$ )
using $\varepsilon$-props(2) by (cases $x_{1}=x_{2}$, auto)
hence sign-changes ps $x_{1}-$ sign-changes ps $x_{2}=1$
unfolding $x_{1}$-def using $x_{2}$-props $\langle\varepsilon>0\rangle$ by simp
also have $x_{2} \notin\left\{x . a<x \wedge x \leq x_{1} \wedge\right.$ poly $\left.p x=0\right\}$
unfolding $x_{1}$-def using $\langle\varepsilon>0\rangle$ by force
with left have $\left\{x . a<x \wedge x \leq x_{1} \wedge\right.$ poly $\left.p x=0\right\}=\{ \}$ by force
with less (1)[of a $\left.x_{1}\right]$ have sign-changes ps $x_{1}=$ sign-changes $p s a$
unfolding $x_{1}$-def $\langle\varepsilon>0\rangle$ by (force simp: card-greater- 0 )
finally have signs-left:

$$
\text { sign-changes ps } a-i n t\left(\text { sign-changes } p s x_{2}\right)=1 \text { by } \operatorname{simp}
$$

have $\{x . x>a \wedge x \leq b \wedge$ poly $p x=0\}=$
$\left\{x . a<x \wedge x \leq x_{2} \wedge\right.$ poly $\left.p x=0\right\} \cup$
$\left\{x . x_{2}<x \wedge x \leq b \wedge\right.$ poly $\left.p x=0\right\}$ using $x_{2}$-props by auto
also note left
finally have $A$ : card $\left\{x . x_{2}<x \wedge x \leq b \wedge\right.$ poly $\left.p x=0\right\}+1=$ card $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$ using fin by simp
hence card $\left\{x . x_{2}<x \wedge x \leq b \wedge\right.$ poly $\left.p x=0\right\}<$
card $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$ by simp
from less(1)[OF this $\left.x_{2}-\operatorname{props(2)}\right]$ and $A$
have signs-right: sign-changes ps $x_{2}-i n t($ sign-changes ps $b)+1=$ card $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$ by simp
from signs-left and signs-right show?thesis by simp qed
qed
thus?thesis by simp
qed
By applying this result to a sufficiently large upper bound, we can effectively count the number of roots "between $a$ and infinity", i. e. the roots greater than $a$ :

```
lemma (in sturm-seq) count-roots-above:
    assumes \(p \neq 0\)
    shows sign-changes ps a-sign-changes-inf ps \(=\)
            card \(\{x . x>a \wedge\) poly \(p x=0\}\)
proof-
    have \(p \in\) set \(p s\) 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 ? \(u=\max a \quad u\)
    \{fix \(x\) assume poly \(p x=0\) hence \(x \leq ? u\)
    using lu-props(3)[OF \(\langle p \in\) set \(p s\rangle\), of \(x]\langle p \neq 0\rangle\)
        by (cases \(u \leq x\), auto simp: sgn-zero-iff)
    \(\}\) note \([\) simp \(]=\) this
    from lu-props
        have map ( \(\lambda\). sgn (poly \(p\) ? u) ) ps = map poly-inf ps by simp
    hence sign-changes ps a-sign-changes-inf \(p s=\)
                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 \(\ldots=\operatorname{card}\{x . a<x \wedge x \leq ? u \wedge\) 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
```

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

```
lemma (in sturm-seq) count-roots-below:
    assumes \(p \neq 0\)
    shows sign-changes-neg-inf ps - sign-changes ps \(a=\)
                card \(\{x . x \leq a \wedge\) poly \(p x=0\}\)
proof-
    have \(p \in\) set \(p s\) using hd-in-set[OF ps-not-Nil] by simp
    have finite (set ps) by simp
    from polys-inf-sign-thresholds \([O F\) this] guess \(l u\).
    note lu-props \(=\) this
    let \(? l=\min\) a \(l\)
    \{fix \(x\) assume poly \(p x=0\) hence \(x>\) ?l
    using \(l u-p r o p s(4)[O F\langle p \in\) set \(p s\rangle\), of \(x]\langle p \neq 0\rangle\)
        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 \(p s a=\)
                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 \(\ldots=\operatorname{card}\{x . ? l<x \wedge x \leq a \wedge\) poly \(p x=0\}\) by simp
    also have \(\{x\). ? \(l<x \wedge x \leq a \wedge\) poly \(p x=0\}=\{x . a \geq x \wedge\) poly \(p x=0\}\)
            using lu-props by auto
    finally show? ?thesis.
qed
lemma (in sturm-seq) count-roots:
    assumes \(p \neq 0\)
    shows sign-changes-neg-inf ps - sign-changes-inf ps \(=\)
                card \(\{x\). poly \(p x=0\}\)
proof-
    have finite (set ps) by simp
    from polys-inf-sign-thresholds \([\) OF this] guess \(l u\).
    note \(l u\)-props \(=\) this
    from lu-props
        have \(\operatorname{map}(\lambda p . \operatorname{sgn}(\) poly \(p l)) p s=\) map poly-neg-inf \(p s\)
            map \((\lambda p\). sgn (poly \(p u))\) ps = map poly-inf \(p s\) 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 \(\ldots=\operatorname{card}\{x . l<x \wedge x \leq u \wedge\) poly \(p x=0\}\) by simp
    also have \(\{x . l<x \wedge x \leq u \wedge\) poly \(p x=0\}=\{x\). poly \(p x=0\}\)
        using lu-props assms by simp
```

finally show? ?thesis.
qed

### 2.4 Constructing Sturm sequences

### 2.5 The canonical Sturm sequence

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

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

We break off the sequence at the first constant polynomial.

```
function sturm-aux where
sturm-aux ( \(p\) :: real poly) \(q=\)
    (if degree \(q=0\) then \([p, q]\) else \(p \#\) sturm-aux \(q(-(p \bmod q))\) )
    by (pat-completeness, simp-all)
termination by (relation measure (degree \(\circ\) snd),
    simp-all add: o-def degree-mod-less')
definition sturm where sturm \(p=\) sturm-aux \(p\) (pderiv \(p\) )
Next, we show some simple facts about this construction:
lemma sturm- \(O[\) simp \(]\) : sturm \(0=[0,0]\)
    by (unfold sturm-def, subst sturm-aux.simps, simp)
lemma [simp]: sturm-aux \(p \quad q=[] \longleftrightarrow\) False
    by (induction \(p\) q rule: sturm-aux.induct, subst sturm-aux.simps, auto)
lemma sturm-neq-Nil[simp]: sturm \(p \neq[]\) unfolding sturm-def by simp
lemma \([\) simp \(]: h d(\) sturm \(p)=p\)
    unfolding sturm-def by (subst sturm-aux.simps, simp)
lemma \([\) simp \(]: p \in \operatorname{set}(\) sturm \(p\) )
    using hd-in-set[OF sturm-neq-Nil] by simp
lemma \([\) simp \(]\) : length \((\) sturm \(p) \geq 2\)
proof-
    \{fix \(q\) have length (sturm-aux \(p q\) ) \(\geq 2\)
                by (induction \(p\) q rule: sturm-aux.induct, subst sturm-aux.simps, auto)
    \}
    thus ?thesis unfolding sturm-def .
qed
```

```
lemma [simp]: degree (last (sturm p)) =0
proof-
    {fix q have degree (last (sturm-aux p q)) =0
            by (induction p q rule: sturm-aux.induct, subst sturm-aux.simps, simp)
    }
    thus ?thesis unfolding sturm-def .
qed
lemma [simp]: sturm-aux p q!0 = p
    by (subst sturm-aux.simps, simp)
lemma [simp]: sturm-aux p q!Suc 0 = q
    by (subst sturm-aux.simps, simp)
lemma [simp]: sturm p!0 = p
    unfolding sturm-def by simp
lemma [simp]: sturm p!Suc 0 = pderiv p
    unfolding sturm-def by simp
lemma sturm-indices:
    assumes i< length (sturm p) - 2
    shows sturm p!(i+2) = -(sturm p!i mod sturm p!(i+1))
proof-
{fix ps q
    have \llbracketps= sturm-aux p q;i< length ps - 2\rrbracket
                            \Longrightarrow p s ! ( i + 2 ) = - ( p s ! i ~ m o d ~ p s ! ( i + 1 ) )
proof (induction p q arbitrary: ps i rule: sturm-aux.induct)
    case (1 pq)
        show ?case
        proof (cases i=0)
            case False
                            then obtain }\mp@subsup{i}{}{\prime}\mathrm{ where [simp]: i=Suc i' by (cases i, simp-all)
                            hence length ps\geq4 using 1 by simp
                            with 1(2) have deg: degree q}\not=
                            by (subst (asm) sturm-aux.simps, simp split: if-split-asm)
                            with 1(2) obtain ps' where [simp]: ps=p# ps'
                            by (subst (asm) sturm-aux.simps, simp)
                            with 1(2) deg have ps':ps' = sturm-aux q (-(p mod q))
                            by (subst (asm) sturm-aux.simps, simp)
                            from〈length ps \geq4〉 and <ps = p# p\mp@subsup{s}{}{\prime}> 1(3) False
                            have }i-1<length ps' - 2 by sim
                    from 1(1)[OF deg ps' this]
                            show ?thesis by simp
        next
            case True
                    with 1(3) have length ps \geq3 by simp
                        with 1(2) have degree q}=
                            by (subst (asm) sturm-aux.simps, simp split: if-split-asm)
```

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

If the Sturm sequence construction is applied to polynomials $p$ and $q$, the greatest common divisor of $p$ and $q$ a divisor of every element in the sequence. This is obvious from the similarity to Euclid's algorithm for computing the GCD.
lemma sturm-aux-gcd: $r \in \operatorname{set}($ sturm-aux $p q) \Longrightarrow g c d p q d v d r$ proof (induction $p$ q rule: sturm-aux.induct)
case (1 p q)
show ?case
proof (cases $r=p$ )
case False
with 1 (2) have $r: r \in \operatorname{set}($ sturm-aux $q(-(p \bmod q)))$
by (subst (asm) sturm-aux.simps, simp split: if-split-asm, subst sturm-aux.simps, simp)
show ?thesis
proof (cases degree $q=0$ )
case False
hence $q \neq 0$ by force
with 1 (1) [OF False r] show ?thesis by (simp add: gcd-mod-right ac-simps)
next
case True with 1(2) and $\langle r \neq p\rangle$ have $r=q$
by (subst (asm) sturm-aux.simps, simp) thus ?thesis by simp
qed
qed $\operatorname{simp}$
qed
lemma sturm-gcd: $r \in \operatorname{set}($ sturm $p) \Longrightarrow g c d p(p d e r i v p) d v d r$ unfolding sturm-def by (rule sturm-aux-gcd)

If two adjacent polynomials in the result of the canonical Sturm chain construction both have a root at some $x$, this $x$ is a root of all polynomials in the sequence.
lemma sturm-adjacent-root-propagate-left:
assumes $i<$ length (sturm ( $p::$ real poly)) - 1
assumes poly (sturm $p!i) x=0$
and poly $(\operatorname{sturm} p!(i+1)) x=0$

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

Consequently, if this is the case in the canonical Sturm chain of $p, p$ must have multiple roots.
lemma sturm-adjacent-root-not-squarefree:
assumes $i<$ 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
Since the second element of the sequence is chosen to be the derivative of $p$, $p_{1}$ and $p_{2}$ fulfil the property demanded by the definition of a Sturm sequence that they locally have opposite sign left of a root $x$ of $p$ and the same sign to the right of $x$.

```
lemma sturm-firsttwo-signs-aux:
    assumes \((p::\) real poly) \(\neq 0 q \neq 0\)
    assumes \(q\)-pderiv:
    eventually \((\lambda x . \operatorname{sgn}(\) poly \(q x)=\operatorname{sgn}(\) poly \((p d e r i v p) x))\left(\right.\) at \(\left.x_{0}\right)\)
    assumes \(p\) - \(0:\) poly \(p\left(x_{0}::\right.\) real \()=0\)
    shows eventually \(\left(\lambda x\right.\). sgn \((\) poly \((p * q) x)=\left(\right.\) if \(x>x_{0}\) then 1 else -1\(\left.)\right)\left(\right.\) at \(\left.x_{0}\right)\)
proof-
    have \(A\) : eventually \((\lambda x\). poly \(p x \neq 0 \wedge\) poly \(q x \neq 0 \wedge\)
                        \(\operatorname{sgn}(\) poly \(q x)=\operatorname{sgn}(\) poly \((\) pderiv \(p) x))\left(\right.\) at \(\left.x_{0}\right)\)
        using \(\langle p \neq 0\rangle\langle q \neq 0\rangle\)
        by (intro poly-neighbourhood-same-sign \(q\)-pderiv
            poly-neighbourhood-without-roots eventually-conj)
    then obtain \(\varepsilon\) where \(\varepsilon\)-props: \(\varepsilon>0 \forall x . x \neq x_{0} \wedge\left|x-x_{0}\right|<\varepsilon \longrightarrow\)
        poly \(p x \neq 0 \wedge\) poly \(q x \neq 0 \wedge \operatorname{sgn}(\) poly \((\) pderiv \(p) x)=\operatorname{sgn}(\) poly \(q x)\)
        by (auto simp: eventually-at dist-real-def)
    have sqr-pos: \(\bigwedge x::\) real. \(x \neq 0 \Longrightarrow \operatorname{sgn} x * \operatorname{sgn} x=1\)
        by (auto simp: sgn-real-def)
    show ?thesis
```

```
    proof (simp only: eventually-at dist-real-def, rule exI[of - \(\varepsilon]\),
        intro conjI, fact \(\langle\varepsilon>0\rangle\), clarify)
    fix \(x\) assume \(x \neq x_{0}\left|x-x_{0}\right|<\varepsilon\)
    with \(\varepsilon\)-props have \([\) simp \(]\) : poly \(p x \neq 0\) poly \(q x \neq 0\)
        sgn \((\) poly \((\) pderiv \(p) x)=\operatorname{sgn}(\) poly \(q x)\) by auto
    show sgn \((p o l y(p * q) x)=\left(\right.\) if \(x>x_{0}\) then 1 else -1\()\)
    proof (cases \(x \geq x_{0}\) )
        case True
            with \(\left\langle x \neq x_{0}\right\rangle\) have \(x>x_{0}\) by simp
            from poly-MVT[OF this, of \(p]\) guess \(\xi\)..
            note \(\xi\)-props \(=\) this
            with \(\langle | x-x_{0}|<\varepsilon\rangle\left\langle\right.\) poly \(\left.p x_{0}=0\right\rangle\left\langle x>x_{0}\right\rangle \varepsilon\)-props
                have \(\left|\xi-x_{0}\right|<\varepsilon \operatorname{sgn}(\) poly \(p x)=\operatorname{sgn}\left(x-x_{0}\right) * \operatorname{sgn}(\) poly \(q \xi)\)
                by (auto simp add: \(q\)-pderiv sgn-mult)
            moreover from \(\xi\)-props \(\varepsilon\)-props \(\langle | x-x_{0}|<\varepsilon\rangle\)
                have \(\forall t . \xi \leq t \wedge t \leq x \longrightarrow\) poly \(q t \neq 0\) by auto
            hence \(\operatorname{sgn}(\) poly \(q \xi)=\operatorname{sgn}(\) poly \(q x)\) using \(\xi\)-props \(\varepsilon\)-props
                by (intro no-roots-inbetween-imp-same-sign, simp-all)
            ultimately show ?thesis using True \(\left\langle x \neq x_{0}\right\rangle \varepsilon\)-props \(\xi\)-props
                by (auto simp: sgn-mult sqr-pos)
    next
        case False
            hence \(x<x_{0}\) by simp
            hence \(\operatorname{sgn}\) : \(\operatorname{sgn}\left(x-x_{0}\right)=-1\) by \(\operatorname{simp}\)
            from poly-MVT[OF \(\left\langle x<x_{0}\right\rangle\), of \(\left.p\right]\) guess \(\xi\)..
            note \(\xi\)-props \(=\) this
            with \(\langle | x-x_{0}|<\varepsilon\rangle\left\langle\right.\) poly p \(\left.x_{0}=0\right\rangle\left\langle x<x_{0}\right\rangle \varepsilon\)-props
                have \(\left|\xi-x_{0}\right|<\varepsilon\) poly \(p x=\left(x-x_{0}\right) *\) poly \((\) pderiv \(p) \xi\)
                    poly \(p \xi \neq 0\) by (auto simp: field-simps)
            hence \(\operatorname{sgn}(\) poly \(p x)=\operatorname{sgn}\left(x-x_{0}\right) * \operatorname{sgn}(\) poly \(q \xi)\)
                using \(\varepsilon\)-props \(\xi\)-props by (auto simp: \(q\)-pderiv sgn-mult)
            moreover from \(\xi\)-props \(\varepsilon\)-props \(\langle | x-x_{0}|<\varepsilon\rangle\)
                have \(\forall t . x \leq t \wedge t \leq \xi \longrightarrow\) poly \(q t \neq 0\) by auto
            hence sgn \((\) poly \(q \xi)=\operatorname{sgn}(\) poly \(q x)\) using \(\xi\)-props \(\varepsilon\)-props
                by (rule-tac sym, intro no-roots-inbetween-imp-same-sign, simp-all)
            ultimately show ?thesis using False \(\left\langle x \neq x_{0}\right\rangle\)
                by (auto simp: sgn-mult sqr-pos)
    qed
    qed
qed
lemma sturm-firsttwo-signs:
    fixes \(p s\) :: real poly list
    assumes squarefree: rsquarefree \(p\)
    assumes \(p\) - 0 : poly \(p\left(x_{0}::\right.\) real \()=0\)
    shows eventually \((\lambda x\). sgn \((\) poly \((p * \operatorname{sturm} p!1) x)=\)
                (if \(x>x_{0}\) then 1 else -1\()\) ) (at \(\left.x_{0}\right)\)
proof-
    from assms have \([\) simp \(]: p \neq 0\) by (auto simp add: rsquarefree-roots)
```

with squarefree $p-0$ have $[$ simp $]$ : pderiv $p \neq 0$
by (auto simp add:rsquarefree-roots)
from assms show ?thesis
by (intro sturm-firsttwo-signs-aux,
simp-all add: rsquarefree-roots)
qed
The construction also obviously fulfils the property about three adjacent polynomials in the sequence.

```
lemma sturm-signs:
    assumes squarefree: rsquarefree \(p\)
    assumes \(i\)-in-range: \(i<\) length (sturm ( \(p::\) real poly)) - 2
    assumes \(q\) - 0 : poly (sturm \(p!(i+1)) x=0\) (is poly ? \(q x=0\) )
    shows poly (sturm \(p!(i+2)) x * \operatorname{poly}(\) 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 \(p!i \bmod\) sturm \(p!(i+1))\)
            (is ? \(r=-(? p \bmod ? q))\).
    hence \(-? r=? p \bmod ? q\) by simp
    with div-mult-mod-eq[of ?p ?q] have ?p div ? \(q * ? q-? r=? p\) by simp
    hence poly (?p div ?q) \(x *\) poly ? \(q x-\) poly ?r \(x=\) poly ?p \(x\)
            by (metis poly-diff poly-mult)
    with \(q-0\) have \(r-x\) : poly ? \(r x=-\) poly ? \(p x\) by simp
    moreover have sqr-pos: \(\bigwedge x::\) real. \(x \neq 0 \Longrightarrow x * x>0\) apply (case-tac \(x \geq 0\) )
        by (simp-all add: mult-neg-neg)
    from sturm-adjacent-root-not-squarefree \([\) of \(i p]\) assms \(r\)-x
            have poly ? \(p x *\) poly ? \(p x>0\) by (force intro: sqr-pos)
    ultimately show poly ? \(\mathrm{r} x *\) poly ? \(p x<0\) by simp
qed
```

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

```
lemma sturm-seq-sturm[simp]:
    assumes rsquarefree \(p\)
    shows sturm-seq (sturm \(p\) ) \(p\)
proof
    show sturm \(p \neq[]\) by \(\operatorname{simp}\)
    show \(h d(\) sturm \(p)=p\) by \(\operatorname{simp}\)
    show length (sturm \(p\) ) 22 by simp
    from assms show \(\bigwedge x\). \(\neg(\) poly \(p x=0 \wedge \operatorname{poly}(\operatorname{sturm} p!1) x=0)\)
        by (simp add: rsquarefree-roots)
next
    fix \(x\) :: real and \(y\) :: real
    have degree \((\) last \((\operatorname{sturm} p))=0\) by simp
    then obtain \(c\) where last (sturm \(p\) ) \(=[: c:]\)
        by (cases last (sturm p), simp split: if-split-asm)
    thus \(\bigwedge x y\). sgn \((\) poly \((\) last \((\) sturm \(p)) x)=\)
```

```
        sgn (poly (last (sturm p)) y) by simp
next
    from sturm-firsttwo-signs[OF assms]
            show }\\mp@subsup{x}{0}{}.\mathrm{ poly p x 
                eventually ( }\lambdax\mathrm{ . sgn (poly ( }p*\mathrm{ sturm p!1) x)=
                            (if }x>\mp@subsup{x}{0}{}\mathrm{ then 1 else -1)) (at }\mp@subsup{x}{0}{})\mathrm{ by simp
next
    from sturm-signs[OF assms]
        show \i x. \llbracketi< length (sturm p) - 2; poly (sturm p! (i+1)) x=0\rrbracket
            \Longrightarrowpoly (sturm p!(i+2)) x * poly (sturm p!i) x<0 by simp
qed
```


### 2.5.1 Canonical squarefree Sturm sequence

The previous construction does not work for polynomials with multiple roots, but we can simply "divide away" multiple roots by dividing $p$ by the GCD of $p$ and $p^{\prime}$. The resulting polynomial has the same roots as $p$, but with multiplicity 1 , allowing us to again use the canonical construction.
definition sturm-squarefree where
sturm-squarefree $p=\operatorname{sturm}(p \operatorname{div}(\operatorname{gcd} p(p d e r i v ~ p)))$
lemma sturm-squarefree-not-Nil[simp]: sturm-squarefree $p \neq[]$
by (simp add: sturm-squarefree-def)
lemma sturm-seq-sturm-squarefree:
assumes $[$ simp]: $p \neq 0$
defines $[$ simp $]: p^{\prime} \equiv p$ div gcd $p(p d e r i v ~ p)$
shows sturm-seq (sturm-squarefree $p$ ) $p^{\prime}$
proof
have rsquarefree $p^{\prime}$
proof (subst rsquarefree-roots, clarify)
fix $x$ assume poly $p^{\prime} x=0$ poly (pderiv $\left.p^{\prime}\right) x=0$
hence $[:-x, 1:]$ dvd gcd $p^{\prime}\left(p d e r i v ~ p^{\prime}\right)$ by (simp add: poly-eq- $\left.0-i f f-d v d\right)$
also from poly-div-gcd-squarefree(1)[OF assms(1)]
have $g c d p^{\prime}\left(\right.$ pderiv $\left.p^{\prime}\right)=1$ by simp
finally show False by (simp add: poly-eq-0-iff-dvd[symmetric])
qed
from sturm-seq-sturm [OF 〈rsquarefree $\left.p^{\prime}\right\rangle$ ]
interpret sturm-seq: sturm-seq sturm-squarefree $p p^{\prime}$
by (simp add: sturm-squarefree-def)
show $\bigwedge x y$. sgn $($ poly $($ last $($ sturm-squarefree $p)) x)=$ sgn (poly (last (sturm-squarefree $p$ )) y) by simp
show sturm-squarefree $p \neq[]$ by simp
show $h d$ (sturm-squarefree $p$ ) $=p^{\prime}$ by (simp add: sturm-squarefree-def)
show length (sturm-squarefree $p$ ) $\geq 2$ by simp
have [simp]: sturm-squarefree $p!0=p^{\prime}$
sturm-squarefree $p!$ Suc $0=$ pderiv $p^{\prime}$
by (simp-all add: sturm-squarefree-def)
from 〈rsquarefree $p^{\prime}$ 〉
show $\bigwedge x . \neg\left(\right.$ poly $p^{\prime} x=0 \wedge$ poly $($ sturm-squarefree $\left.p!1) x=0\right)$
by (simp add: rsquarefree-roots)
from sturm-seq.signs show $\bigwedge i x . \llbracket i<$ length (sturm-squarefree $p$ ) - 2;
poly (sturm-squarefree $p!(i+1)) x=0 \rrbracket$
$\Longrightarrow$ poly (sturm-squarefree $p!(i+2)) x *$ poly (sturm-squarefree $p!i) x<0$.
from sturm-seq.deriv show $\bigwedge x_{0}$. poly $p^{\prime} x_{0}=0 \Longrightarrow$
eventually $\left(\lambda x\right.$. sgn $\left(\right.$ poly $\left(p^{\prime} *\right.$ sturm-squarefree $\left.\left.p!1\right) x\right)=$ $\left(\right.$ if $x>x_{0}$ then 1 else -1$\left.)\right)\left(\right.$ at $\left.x_{0}\right)$.
qed

### 2.5.2 Optimisation for multiple roots

We can also define the following non-canonical Sturm sequence that is obtained by taking the canonical Sturm sequence of $p$ (possibly with multiple roots) and then dividing the entire sequence by the GCD of $p$ and its derivative.
definition sturm-squarefree' where
sturm-squarefree' $p=($ let $d=g c d p(p d e r i v p)$

$$
\text { in map }\left(\lambda p^{\prime} \cdot p^{\prime} \text { div d) }(\text { sturm } p)\right)
$$

This construction also has all the desired properties:

```
lemma sturm-squarefree'-adjacent-root-propagate-left:
    assumes p\not=0
    assumes i< length (sturm-squarefree' (p :: real poly)) - 1
    assumes poly (sturm-squarefree' p!i)x=0
        and poly (sturm-squarefree' }p!(i+1))x=
    shows }\forallj\leqi+1. poly (sturm-squarefree' p!j)x=
proof (intro sturm-adjacent-root-aux[OF assms(2,3,4)], goal-cases)
    case prems: (1 i x)
        define q}\mathrm{ where q= sturm p!i
        define }r\mathrm{ where }r=\operatorname{sturm}p!(Suc i
    define s where s=sturm p!(Suc (Suc i))
    define d}\mathrm{ where d=gcd p (pderiv p)
    define q' }\mp@subsup{r}{}{\prime}\mp@subsup{s}{}{\prime}\mathrm{ where }\mp@subsup{q}{}{\prime}=q\mathrm{ div d and r'}=r\mathrm{ div d and }\mp@subsup{s}{}{\prime}=s\mathrm{ div d
    from }\langlep\not=0\rangle\mathrm{ have }d\not=0\mathrm{ unfolding d-def by simp
    from prems(1) have i-in-range: i< length (sturm p) - 2
        unfolding sturm-squarefree'-def Let-def by simp
    have [simp]: d dvd q d dvd r d dvd s unfolding q-def r-def s-def d-def
        using i-in-range by (auto intro: sturm-gcd)
```

```
    hence qrs-simps: }q=\mp@subsup{q}{}{\prime}*dr=\mp@subsup{r}{}{\prime}*ds=\mp@subsup{s}{}{\prime}*
    unfolding }\mp@subsup{q}{}{\prime}\mathrm{ -def r'}\mp@subsup{r}{}{\prime}\mathrm{ -def s}\mp@subsup{s}{}{\prime}\mathrm{ -def by (simp-all)
    with prems(2) i-in-range have r'-0: poly r 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\mathrm{ div r*r-s
        unfolding q-def r-def s-def by (simp add: div-mult-mod-eq)
    hence }\mp@subsup{q}{}{\prime}=(q\mathrm{ div r *r-s) div d by (simp add: q'-def)
    also have ... = (q div r*r) div d - s'
        by (simp add: s'-def poly-div-diff-left)
    also have ... = q div r* r' - s'
        using dvd-div-mult[OF <d dvd r\rangle, 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\not=0`
    finally have poly }\mp@subsup{q}{}{\prime}x=poly ( q' div r r * r r' - s') x by sim
    also from r}\mp@subsup{r}{}{\prime}-0\mathrm{ have ... = -poly s' }x\mathrm{ by simp
    finally have poly s' }\mp@subsup{s}{}{\prime}=-\mathrm{ poly }\mp@subsup{q}{}{\prime}x\mathrm{ by simp
    thus ?case using i-in-range
        unfolding }\mp@subsup{q}{}{\prime}\mathrm{ -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\not=0
            i< length (sturm-squarefree' (p :: real poly)) - 1
            poly (sturm-squarefree' }p!i)x=
            poly (sturm-squarefree' p!(i+1)) x=0
    shows False
proof -
    define d where 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\mathrm{ poly (pderiv p div d) }x=
            using assms(2)
            unfolding sturm-squarefree'-def Let-def d-def by auto
    moreover from div-gcd-coprime assms(1)
            have coprime (p div d) (pderiv p div d) unfolding d-def by auto
    ultimately show False using coprime-imp-no-common-roots by auto
qed
lemma sturm-squarefree'-signs:
    assumes p\not=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
```

```
define \(d\) where \(d=g c d p\) (pderiv \(p\) )
with \(\langle p \neq 0\rangle\) have \([\operatorname{simp}]\) : \(d \neq 0\) by simp
from poly-div-gcd-squarefree (1)[OF \(\langle p \neq 0\rangle\) ]
    coprime-imp-no-common-roots
    have rsquarefree: rsquarefree ( \(p\) div d)
    by (auto simp: rsquarefree-roots \(d\)-def)
from \(i\)-in-range have \(i\)-in-range': \(i<\operatorname{length}(\) sturm \(p\) ) - 2
    unfolding sturm-squarefree' \({ }^{\prime}\) def by simp
hence \(d\) dvd (sturm \(p!i\) ) (is \(d\) dvd ? \(p\) ')
        \(d\) dvd (sturm \(p!(S u c i))(\) is \(d d v d ? q\) )
        \(d\) dvd (sturm \(p!(S u c(S u c i)))\) (is \(d\) dvd ? \(r^{\prime}\) )
    unfolding \(d\)-def by (auto intro: sturm-gcd)
hence pqr-simps: ? \(p^{\prime}=? p * d ? q^{\prime}=? q * d ? r^{\prime}=? r * d\)
    unfolding sturm-squarefree'-def Let-def d-def using i-in-range'
    by (auto simp: dvd-div-mult-self)
with \(q-0\) have \(q^{\prime}-0\) : poly ? \(q^{\prime} x=0\) by simp
from sturm-indices \([O F i\)-in-range \(]\)
    have sturm \(p!(i+2)=-(\) sturm \(p!i \bmod\) sturm \(p!(i+1))\).
hence \(-? r^{\prime}=? p^{\prime} \bmod ? q^{\prime}\) by simp
with div-mult-mod-eq[of ? \(\left.p^{\prime} ? q^{\prime}\right]\) have \(? p^{\prime}\) div ? \(q^{\prime} * ? q^{\prime}-? r^{\prime}=? p^{\prime}\) 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 \(\operatorname{simp}\)
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 \([O F\langle p \neq 0\rangle] i\)-in-range \(q\)-0
    have poly ? \(p x \neq 0\) by force
moreover have sqr-pos: \(\bigwedge x:\) :real. \(x \neq 0 \Longrightarrow x * x>0\) apply (case-tac \(x \geq 0\) )
    by (simp-all add: mult-neg-neg)
ultimately show ?thesis using \(r-x\) by simp
qed
```

This approach indeed also yields a valid squarefree Sturm sequence for the polynomial $p / \operatorname{gcd}\left(p, p^{\prime}\right)$.
lemma sturm-seq-sturm-squarefree':
assumes $(p::$ real poly) $\neq 0$
defines $d \equiv \operatorname{gcd} p(p d e r i v p)$
shows sturm-seq (sturm-squarefree' $p$ ) ( $p$ div d)
(is sturm-seq ?ps ${ }^{\prime}$ ? $p^{\prime}$ )
proof
show $? p s^{\prime} \neq[] h d ? p s^{\prime}=? p^{\prime} 2 \leq$ length $? p s^{\prime}$
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 ？$p s^{\prime} * d$
by（simp add：sturm－squarefree＇－def last－map d－def dvd－div－mult－self）
then have last ？ps＇dvd last（sturm p）by simp
with $*$ dvd－imp－degree－le $[O F$ this $]$ have degree（last ？ps＇）$\leq$ degree（last（sturm
p））
using $\langle d \neq 0\rangle$ by（cases last ？ps $s^{\prime}=0$ ）auto
hence degree（last ？ps ${ }^{\prime}$ ）$=0$ by simp
then obtain $c$ where last ？$p s^{\prime}=[: c:]$
by（cases last ？ps＇，simp split：if－split－asm）
thus $\wedge x y$ ．sgn（poly（last ？ps＇）$x$ ）$=$ sgn（poly（last ？ps＇）y）by simp
\}
have squarefree：rsquarefree $? p^{\prime}$ using $\langle p \neq 0$ 〉
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 ？$p^{\prime}$（pderiv $p$ div d）
unfolding $d$－def using div－gcd－coprime $\langle p \neq 0\rangle$ by blast
thus squarefree＇：
$\wedge x . \neg($ poly $(p$ div d）$x=0 \wedge$ poly（sturm－squarefree＇$p!1) x=0)$
using coprime－imp－no－common－roots by simp
from sturm－squarefree＇－signs $[$ OF $\langle p \neq 0\rangle]$
show $\wedge i x . \llbracket i<$ length ？ps＇$-2 ;$ poly $\left(? p s^{\prime}!(i+1)\right) x=0 \rrbracket$
$\Longrightarrow$ poly（？ps＇$!(i+2)) x *$ poly $\left(? p s^{\prime}!i\right) x<0$ ．
have $[$ simp $]: ? p^{\prime} \neq 0$ using squarefree by（simp add：rsquarefree－def）
have $A: ? p^{\prime}=? p s^{\prime}!0$ pderiv $p$ div $d=? p s^{\prime}!1$
by（simp－all add：sturm－squarefree＇－def Let－def d－def sturm－def， subst sturm－aux．simps，simp）
have $[$ simp $]: ? p s s^{\prime}!0 \neq 0$ using squarefree
by（auto simp：A rsquarefree－def）
fix $x_{0}$ ：：real
assume poly ？$p^{\prime} x_{0}=0$
hence poly $p x_{0}=0$ using poly－div－gcd－squarefree（2）［OF $\left.\langle p \neq 0\rangle\right]$
unfolding $d$－def by simp
hence pderiv $p \neq 0$ using $\langle p \neq 0$ 〉 by（auto dest：pderiv－iszero）
with $\left\langle p \neq 0\right.$ 〉 〈poly $p x_{0}=0$ 〉
have A：eventually $(\lambda x . \operatorname{sgn}($ poly $(p *$ pderiv $p) x)=$

$$
\left.\left(\text { if } x_{0}<x \text { then } 1 \text { else }-1\right)\right)\left(\text { at } x_{0}\right)
$$

by（intro sturm－firsttwo－signs－aux，simp－all）
note ev $=$ eventually－conj $[$ OF A poly－neighbourhood－without－roots $[$ OF $\langle d \neq 0\rangle]]$
show eventually $(\lambda x$ ．sgn（poly $(p$ div $d *$ sturm－squarefree＇$p!1) x)=$ （if $x_{0}<x$ then 1 else -1 ））（at $x_{0}$ ）
proof（rule eventually－mono［OF ev］，goal－cases）

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

This construction is obviously more expensive to compute than the one that first divides $p$ by $\operatorname{gcd}\left(p, p^{\prime}\right)$ and then applies the canonical construction. In this construction, we first compute the canonical Sturm sequence of $p$ as if it had no multiple roots and then divide by the GCD. However, it can be seen quite easily that unless $x$ is a multiple root of $p$, i. e. as long as $\operatorname{gcd}\left(P, P^{\prime}\right) \neq 0$, the number of sign changes in a sequence of polynomials does not actually change when we divide the polynomials by $\operatorname{gcd}\left(p, p^{\prime}\right)$.
Therefore we can use the canonical Sturm sequence even in the non-squarefree case as long as the borders of the interval we are interested in are not multiple roots of the polynomial.

```
lemma sign-changes-mult-aux:
    assumes \(d \neq(0::\) real \()\)
    shows length \((\) remdups-adj \((f i l t e r ~(\lambda x . x \neq 0)(\operatorname{map}((*) d \circ f) x s)))=\)
        length (remdups-adj \((\) filter \((\lambda x . x \neq 0)(\operatorname{map} f x s)))\)
proof-
    from assms have inj: inj ((*) d) by (auto intro: injI)
    from assms have [simp]: filter \((\lambda x .((*) d \circ f) x \neq 0)=\) filter \((\lambda x . f x \neq 0)\)
                        filter \(((\lambda x . x \neq 0) \circ f)=\) filter \((\lambda x . f x \neq 0)\)
        by (simp-all add: o-def)
    have filter \((\lambda x . x \neq 0)(\operatorname{map}((*) d \circ f) x s)=\)
        \(\operatorname{map}((*) d \circ f)(f i l t e r(\lambda x .((*) d \circ f) x \neq 0) x s)\)
        by (simp add: filter-map o-def)
    thus ?thesis using remdups-adj-map-injective[OF inj] assms
        by (simp add: filter-map map-map[symmetric] del: map-map)
qed
lemma sturm-sturm-squarefree'-same-sign-changes:
    fixes \(p::\) real poly
    defines \(p s \equiv\) sturm \(p\) and \(p s^{\prime} \equiv\) sturm-squarefree' \(p\)
    shows poly \(p x \neq 0 \vee\) poly (pderiv \(p\) ) \(x \neq 0 \Longrightarrow\)
                sign-changes \(p s^{\prime} x=\) sign-changes \(p s x\)
```

```
    p\not=0\Longrightarrowsign-changes-inf ps'}=\mathrm{ sign-changes-inf ps
    p\not=0\Longrightarrow sign-changes-neg-inf ps'}=\mathrm{ sign-changes-neg-inf ps
proof
    define d where d=gcd p(pderiv p)
    define p}\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\mathrm{ div d
    define }\mp@subsup{s}{}{\prime}\mathrm{ where }\mp@subsup{s}{}{\prime}=\mathrm{ poly-inf d
    define }\mp@subsup{s}{}{\prime\prime}\mathrm{ where }\mp@subsup{s}{}{\prime\prime}=\mathrm{ poly-neg-inf d
{
    fix }x\mathrm{ :: real and q :: real poly
    assume q\in 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 (qdiv 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 }\mp@subsup{s}{}{\prime}\mathrm{ -def by (subst q-prod, simp)
    from poly-inf-mult have s3: poly-neg-inf q = s' * poly-neg-inf (q div d)
        unfolding }\mp@subsup{s}{}{\prime\prime}\mathrm{ -def by (subst q-prod, simp)
    note s1 s2 s3
}
note signs = this
{
    fix f :: real poly }=>\mathrm{ real and s :: real
    assume f:\bigwedgeq. q\in set ps\Longrightarrowfq=s*f(qdivd) and s:s\not=0
    hence inverse s\not=0 by simp
    {fix q}\mathrm{ assume q}\in\mathrm{ set ps
    hence f(q div d) = inverse s*fq
            by (subst f[of q], simp-all add: s)
    } note f' = this
    have length (remdups-adj [x\leftarrowmap f (map (\lambdaq.q div d) ps). x\not=0]) - 1=
                length (remdups-adj [x\leftarrowmap (\lambdaq.f(qdivd)) ps.x\not=0]) - 1
    by (simp only: sign-changes-def o-def map-map)
    also have map ( }\lambdaq.q\mathrm{ div d) ps=ps'
        by (simp add: ps-def ps'-def sturm-squarefree'-def Let-def d-def)
    also from f}\mp@subsup{}{}{\prime}\mathrm{ have map ( }\lambdaq.f(qdivd)) ps
                            map (\lambdax. ((*)(inverse s)\circf) x) ps by (simp add:o-def)
    also note sign-changes-mult-aux[OF〈inverse s}\not=0\mathrm{ \, of f ps]
    finally have
        length (remdups-adj [x\leftarrowmap f ps'. x = 0]) - 1=
        length(remdups-adj [x\leftarrowmap f ps.x\not=0]) - 1 by simp
}
note length-remdups-adj = this
```

\{

```
    fix }x\mathrm{ assume A: poly p x =0 v poly (pderiv p) x =0
    have d dvd p d dvd pderiv p unfolding d-def by simp-all
    with }A\mathrm{ have sgn (poly dx)}\not=
    by (auto simp add: sgn-zero-iff elim: dvdE)
    thus sign-changes ps' }x=\mathrm{ sign-changes ps x using signs(1)
    unfolding sign-changes-def
    by (intro length-remdups-adj[of \lambdaq. sgn (poly q x)], simp-all)
}
assume p\not=0
hence d\not=0 unfolding d-def by simp
hence s'\not=0 s"}=0\mathrm{ unfolding }\mp@subsup{s}{}{\prime}\mathrm{ -def s''-def by simp-all
from length-remdups-adj[of poly-inf s}\mp@subsup{s}{}{\prime}\mathrm{ ,OF signs(2) <s'}=00\rangle
    show sign-changes-inf ps' = sign-changes-inf ps
    unfolding sign-changes-inf-def .
from length-remdups-adj[of poly-neg-inf s'",OF signs(3)<s" 看 0〉]
    show sign-changes-neg-inf ps' = sign-changes-neg-inf ps
    unfolding sign-changes-neg-inf-def .
qed
```


### 2.6 Root-counting functions

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

```
definition count-roots-between where
count-roots-between pab= (if a\leqb^p\not=0 then
    (let ps=sturm-squarefree p
    in sign-changes ps a - sign-changes ps b) else 0)
```

definition count-roots where
count-roots $p=($ if ( $p::$ real poly $)=0$ then 0 else
(let $p s=$ sturm-squarefree $p$
in sign-changes-neg-inf ps - sign-changes-inf ps))
definition count-roots-above where
count-roots-above $p a=($ if ( $p:$ :real poly) $=0$ then 0 else
(let $p s=$ sturm-squarefree $p$
in sign-changes ps a-sign-changes-inf ps))
definition count-roots-below where
count-roots-below $p a=($ if ( $p:$ : real poly) $=0$ then 0 else
(let ps = sturm-squarefree $p$
in sign-changes-neg-inf ps - sign-changes ps a))
lemma count-roots-between-correct:
count-roots-between p a $b=$ card $\{x . a<x \wedge x \leq b \wedge$ 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\leqb^ poly px=0}=0
    proof (cases a<b)
    case True
        with False have [simp]: p=0 by simp
        have subset: {a<..<b}\subseteq{x.a<x\wedge x\leqb^ poly p x=0} by auto
        from infinite-Ioo[OF True] have }\neg\mathrm{ finite {a<..<b} .
```



```
            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\not=0 a\leqb by simp-all
    define }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\operatorname{div}(gcd p(pderiv p)
    from poly-div-gcd-squarefree(1)[OF <p}\not=0\rangle] have p'\not=
        unfolding p'-def by clarsimp
    from sturm-seq-sturm-squarefree[OF <p\not=0>]
        interpret sturm-seq sturm-squarefree p p
        unfolding p}\mp@subsup{p}{}{\prime}-def 
    from poly-roots-finite[OF <p'\not=0\rangle]
        have finite {x.a<x^x\leqb^ poly p' x=0} by fast
    have count-roots-between pab=card {x. a<x\wedge x\leqb^ poly p' x=0}
        unfolding count-roots-between-def Let-def
        using True count-roots-between [OF \langlep' }=0\rangle\langlea\leqb\rangle] by sim
    also from poly-div-gcd-squarefree(2)[OF <p\not=0\rangle]
    have {x.a<x\wedgex\leqb^ poly p' x=0}=
        {x.a<x\wedge x\leqb^ poly px=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 finite-subset[of {0<..<1} ?S]
        have \negfinite {x. poly px=0} by (auto simp: infinite-Ioo)
        thus ?thesis by (simp add: count-roots-def True)
next
    case False
    define }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\operatorname{div}(gcd p(pderiv p)
    from poly-div-gcd-squarefree(1)[OF <p\not=0`] have p'
        unfolding p'-def by clarsimp
```

```
    from sturm-seq-sturm-squarefree[OF <p\not=0>]
        interpret sturm-seq sturm-squarefree p p
        unfolding p
    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}\not=0\rangle
    also from poly-div-gcd-squarefree(2)[OF \langlep\not=0\rangle]
    have {x. poly p' }x=0}={x.poly px=0} unfolding p'-def by blas
    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 finite-subset[of {a<..<a+1} ?S]
        have \negfinite {x. x>a^ poly px=0} by (auto simp: infinite-Ioo subset-eq)
    thus ?thesis by (simp add: count-roots-above-def True)
next
    case False
    define }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\operatorname{div}(gcd p(pderiv p)
    from poly-div-gcd-squarefree(1)[OF <p}\not=0\rangle] have p'\not=
        unfolding p'-def by clarsimp
    from sturm-seq-sturm-squarefree[OF <p\not=0\rangle]
        interpret sturm-seq sturm-squarefree p p'
        unfolding p'-def .
    from count-roots-above[OF < p' }=0\rangle
        have count-roots-above p a = card {x. x>a\wedge poly p' }x=0
        unfolding count-roots-above-def Let-def by (simp add: <p\not=0\rangle)
    also from poly-div-gcd-squarefree(2)[OF <p\not=0>]
        have {x. x>a\wedge poly p' x=0}={x. x>a\wedge poly px=0}
        unfolding p}\mp@subsup{p}{}{\prime}\mathrm{ -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\leqa^ poly p x=0}
        (is - = card ?S)
proof (cases p=0)
    case True
        with finite-subset[of {a-1<..<a} ?S]
```



```
        thus ?thesis by (simp add: count-roots-below-def True)
next
```

```
    case False
    define }\mp@subsup{p}{}{\prime}\mathrm{ where }\mp@subsup{p}{}{\prime}=p\operatorname{div}(gcd p(pderiv p)
    from poly-div-gcd-squarefree(1)[OF <p\not=0\rangle] have p}\mp@subsup{p}{}{\prime}\not=
        unfolding p'-def by clarsimp
    from sturm-seq-sturm-squarefree[OF <p\not=0\rangle]
    interpret sturm-seq sturm-squarefree p p
    unfolding }\mp@subsup{p}{}{\prime}\mathrm{ -def .
from count-roots-below[OF < p
    have count-roots-below pa= card {x. x \leq a ^ poly p'x=0}
    unfolding count-roots-below-def Let-def by (simp add: <p\not=0\rangle)
also from poly-div-gcd-squarefree(2)[OF <p\not=0>]
    have {x.x\leqa^ poly p' x=0}={x. x\leqa^ poly px=0}
    unfolding p}\mp@subsup{p}{}{\prime}\mathrm{ -def by blast
finally show ?thesis.
qed
```

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

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

```
    from sturm-seq-sturm-squarefree'[OF <p\not=0>]
        interpret sturm-seq sturm-squarefree' p p div d
        unfolding sturm-squarefree'-def Let-def d-def .
    note count-roots-between-correct
    also have {x.a<x^x\leqb^ poly p x=0}=
            {x.a<x\wedgex\leqb
        unfolding d-def using poly-div-gcd-squarefree(2)[OF <p\not=0\rangle] by simp
    also note count-roots-between[OF <p div d}\not=0\rangle\langlea\leqb\rangle\mathrm{ , 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::\mathrm{ :real poly) =
    (if p}=0\mathrm{ then 0
        else let ps = sturm p
            in sign-changes-neg-inf ps - sign-changes-inf ps)
proof (cases p=0, simp add: count-roots-def)
    case False
    define d where d=gcd p(pderiv p)
    from }\langlep\not=0\rangle\mathrm{ have [simp]: p div d}\not=
        using poly-div-gcd-squarefree(1)[OF <p\not=0`] by (auto simp add: d-def)
    from sturm-seq-sturm-squarefree'[OF}\langlep\not=0\rangle
        interpret sturm-seq sturm-squarefree' p p div d
        unfolding sturm-squarefree'-def Let-def d-def .
    note count-roots-correct
    also have {x.poly p x=0} ={x.poly (p div d) x=0}
        unfolding d-def using poly-div-gcd-squarefree(2)[OF}\langlep\not=0\rangle] by sim
    also note count-roots[OF<p div d}\not=0\rangle\mathrm{ , symmetric]
    also note sturm-sturm-squarefree'-same-sign-changes(2)[OF }\langlep\not=0\rangle
    also note sturm-sturm-squarefree'-same-sign-changes(3)[OF <p\not=0\rangle}
    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\not=0 \vee poly q a\not=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\not=0 by simp-all
    thus ?thesis
    proof (cases (poly p a\not=0\vee poly (pderiv p) a\not=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\not=0\vee poly (pderiv p) a\not=0 by simp
            define d where d=gcd p(pderiv p)
            from }\langlep\not=0\rangle\mathrm{ have [simp]: p div d}\not=
                    using poly-div-gcd-squarefree(1)[OF <p\not=0\rangle] by (auto simp add: d-def)
            from sturm-seq-sturm-squarefree'[OF <p\not=0\rangle]
                    interpret sturm-seq sturm-squarefree' p p div d
                    unfolding sturm-squarefree'-def Let-def d-def .
    note count-roots-above-correct
    also have {x.a<x\wedge poly p x=0}=
                    {x.a<x^poly (p div d) x=0}
                    unfolding d-def using poly-div-gcd-squarefree(2)[OF <p\not=0\rangle] by simp
    also note count-roots-above[OF <p div d\not=0\rangle, symmetric]
    also note sturm-sturm-squarefree'-same-sign-changes(1)[OF A]
    also note sturm-sturm-squarefree'-same-sign-changes(2)[OF <p}\not=0\rangle
    finally show ?thesis using True False by (simp add: Let-def)
    qed
qed
lemma count-roots-below-code[code]:
    count-roots-below p a =
    (let q= pderiv p
            in if p}=0\mathrm{ then 0
            else if poly p a\not=0\vee poly q a\not=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\not=0 by simp-all
        thus ?thesis
        proof (cases (poly p a = 0 \vee poly (pderiv p) a\not=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\not=0 0 poly (pderiv p) a\not=0 by simp
    define d where d=gcd p (pderiv p)
    from }\langlep\not=0\rangle\mathrm{ have [simp]: p div d}\not=
        using poly-div-gcd-squarefree(1)[OF <p\not=0\rangle] by (auto simp add: d-def)
    from sturm-seq-sturm-squarefree'[OF <p\not=0>]
        interpret sturm-seq sturm-squarefree' p p div d
            unfolding sturm-squarefree'-def Let-def d-def .
    note count-roots-below-correct
    also have {x. x\leqa^ poly p x=0}=
            {x. x\leqa^ poly (p div d) x=0}
        unfolding d-def using poly-div-gcd-squarefree(2)[OF <p\not= 0〉] by simp
    also note count-roots-below[OF <p div d 
    also note sturm-sturm-squarefree'-same-sign-changes(1)[OF A]
    also note sturm-sturm-squarefree'-same-sign-changes(3)[OF}\langlep\not=0\rangle
    finally show ?thesis using True False by (simp add: Let-def)
    qed
qed
end
```


## 3 The "sturm" proof method

theory Sturm-Method
imports Sturm-Theorem
begin

### 3.1 Preliminary lemmas

In this subsection, we prove lemmas that reduce root counting and related statements to simple, computable expressions using the count-roots function family.
lemma poly-card-roots-less-leq:
card $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}=$ count-roots-between $p a b$
by (simp add: count-roots-between-correct)
lemma poly-card-roots-leq-leq:
card $\{x . a \leq x \wedge x \leq b \wedge$ poly $p x=0\}=$
( count-roots-between pab+
(if $(a \leq b \wedge$ poly $p a=0 \wedge p \neq 0) \vee(a=b \wedge p=0)$ then 1 else 0$))$
proof (cases $(a \leq b \wedge$ poly $p a=0 \wedge p \neq 0) \vee(a=b \wedge p=0))$
case False
note False ${ }^{\prime}=$ this
thus ?thesis

```
    proof (cases p=0)
```

    case False
        with False' have poly \(p a \neq 0 \vee a>b\) by auto
        hence \(\{x . a \leq x \wedge x \leq b \wedge\) poly \(p x=0\}=\)
                \(\{x . a<x \wedge x \leq b \wedge\) poly \(p x=0\}\)
        by (auto simp: less-eq-real-def)
        thus ?thesis using poly-card-roots-less-leq False'
            by (auto split: if-split-asm)
        next
        case True
            have \(\{x . a \leq x \wedge x \leq b\}=\{a . . b\}\)
                    \(\{x . a<\bar{x} \wedge x \leq b\}=\{a<. . b\}\) by auto
        with True False have card \(\{x . a<x \wedge x \leq b\}=0\) card \(\{x . a \leq x \wedge x \leq\)
    $b\}=0$
by (auto simp add: card-eq-0-iff infinite-Ioc infinite-Icc)
with True False show ?thesis
using count-roots-between-correct by simp
qed
next
case True
note True ${ }^{\prime}=$ this
have fin: finite $\{x . a \leq x \wedge x \leq b \wedge$ poly $p x=0\}$
proof (cases $p=0$ )
case True
with True ${ }^{\prime}$ have $a=b$ by simp
hence $\{x . a \leq x \wedge x \leq b \wedge$ 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 $\{x . a \leq x \wedge x \leq b \wedge$ poly $p x=0\}=$
insert $a\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$ by auto
hence card $\{x . a \leq x \wedge x \leq b \wedge$ poly $p x=0\}=$
Suc (card $\{x . a<x \wedge x \leq b \wedge$ 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 $\{x . a<x \wedge x<b \wedge$ poly $p x=0\}=$
( count-roots-between $p a b-$
(if poly $p b=0 \wedge a<b \wedge p \neq 0$ then 1 else 0 ))
proof (cases poly $p b=0 \wedge a<b \wedge p \neq 0$ )
case False
note False ${ }^{\prime}=$ this
show ?thesis
proof (cases $p=0$ )
case True
have $[\operatorname{simp}]:\{x . a<x \wedge x<b\}=\{a<. .<b\}$

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

with True False have card $\{x . a<x \wedge x \leq b\}=0$ card $\{x . a<x \wedge x<$ $b\}=0$
by (auto simp add: card-eq-0-iff infinite-Ioo infinite-Ioc)
with True False' show ?thesis
by (auto simp: count-roots-between-correct)

## next

case False
with False ${ }^{\prime}$ have $\{x . a<x \wedge x<b \wedge$ poly $p x=0\}=$ $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$
by (auto simp: less-eq-real-def)
thus ?thesis using poly-card-roots-less-leq False by auto qed
next
case True
with poly-roots-finite
have fin: finite $\{x . a<x \wedge x<b \wedge$ poly $p x=0\}$ by fast
from True have $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}=$
insert $b\{x . a<x \wedge x<b \wedge$ poly $p x=0\}$ by auto
hence $\operatorname{Suc}(\operatorname{card}\{x . a<x \wedge x<b \wedge$ poly $p x=0\})=$
card $\{x . a<x \wedge x \leq b \wedge$ 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. $a \leq x \wedge x<b \wedge$ poly $p x=0\}=$
( count-roots-between pab+
(if $p \neq 0 \wedge a<b \wedge$ poly $p a=0$ then 1 else 0 ) -
(if $p \neq 0 \wedge a<b \wedge$ poly $p b=0$ then 1 else 0 ))
proof (cases $p=0 \vee a \geq b$ )
case True
note True ${ }^{\prime}=$ this
show ?thesis
proof (cases $a \geq b$ )
case False
hence $\{x . a<x \wedge x \leq b\}=\{a<. . b\}$
$\{x . a \leq x \wedge x<b\}=\{a . .<b\}$ by auto
with True False have card $\{x . a<x \wedge x \leq b\}=0$ card $\{x . a \leq x \wedge x<$
$b\}=0$
by (auto simp add: card-eq-0-iff infinite-Ico infinite-Ioc)
with False True' show ?thesis
by (simp add: count-roots-between-correct)
next
case True
with True ${ }^{\prime}$ have $\{x . a \leq x \wedge x<b \wedge$ poly $p x=0\}=$ $\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$
by (auto simp: less-eq-real-def)
thus ?thesis using poly-card-roots-less-leq True by simp
case False
let ? $A=\{x . a \leq x \wedge x<b \wedge$ poly $p x=0\}$
let ? $B=\{x . a<x \wedge x \leq b \wedge$ poly $p x=0\}$
let ? $C=\{x . x=b \wedge$ poly $p x=0\}$
let $? D=\{x . x=a \wedge$ poly $p a=0\}$
have $C D$-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 $? A=(? B \cup ? D)-? C$ 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 count-roots-correct by simp
lemma poly-no-roots:
$(\forall x$. poly $p x \neq 0) \longleftrightarrow(p \neq 0 \wedge$ count-roots $p=0)$
by (auto simp: count-roots-correct dest: poly-roots-finite)
lemma poly-pos:
$(\forall x$. poly $p x>0) \longleftrightarrow($
$p \neq 0 \wedge$ poly-inf $p=1 \wedge$ count-roots $p=0)$
by (simp only: Let-def poly-pos poly-no-roots, blast)
lemma poly-card-roots-greater:
card $\{x::$ real. $x>a \wedge$ poly $p x=0\}=$ count-roots-above $p a$
using count-roots-above-correct by simp
lemma poly-card-roots-leq:
card $\{x::$ real. $x \leq a \wedge$ poly $p x=0\}=$ count-roots-below $p a$
using count-roots-below-correct by simp
lemma poly-card-roots-geq:
card $\{x::$ real. $x \geq a \wedge$ poly $p x=0\}=($
count-roots-above $p a+($ if poly $p a=0 \wedge p \neq 0$ then 1 else 0$)$ )
proof (cases poly pa=0^pキ0)
case False
hence card $\{x . x \geq a \wedge$ poly $p x=0\}=\operatorname{card}\{x . x>a \wedge$ poly $p x=0\}$
proof (cases rule: disjE)

```
        assume \(p=0\)
        have \(\neg\) finite \(\{a<. .<a+1\}\)
        by (metis infinite-Ioo less-add-one)
    moreover have \(\{a<. .<a+1\} \subseteq\{x . x \geq a \wedge\) poly \(p x=0\}\)
            \(\{a<. .<a+1\} \subseteq\{x . x>a \wedge\) poly \(p x=0\}\)
            using \(\langle p=0\rangle\) by auto
        ultimately have \(\neg\) finite \(\{x . x \geq a \wedge\) poly \(p x=0\}\)
                    \(\neg\) finite \(\{x . x>a \wedge\) poly \(p x=0\}\)
        by (auto dest!: finite-subset \([\) of \(\{a<. .<a+1\}]\) simp: infinite-Ioo)
    thus ?thesis by simp
    next
        assume poly \(p a \neq 0\)
        hence \(\{x . x \geq a \wedge\) poly \(p x=0\}=\{x . x>a \wedge\) poly \(p x=0\}\)
            by (auto simp: less-eq-real-def)
        thus ?thesis by simp
    qed auto
    thus ?thesis using False
        by (auto intro: poly-card-roots-greater)
next
    case True
        hence finite \(\{x . x>a \wedge\) poly \(p x=0\}\) using poly-roots-finite by force
        moreover have \(\{x . x \geq a \wedge\) poly \(p x=0\}=\)
                    insert \(a\{x . x>a \wedge\) poly \(p x=0\}\) using True by auto
    ultimately have card \(\{x . x \geq a \wedge\) poly \(p x=0\}=\)
                    Suc (card \(\{x . x>a \wedge\) poly \(p x=0\})\)
        using card-insert-disjoint by auto
    thus ?thesis using True by (auto intro: poly-card-roots-greater)
qed
lemma poly-card-roots-less:
    card \(\{x::\) real. \(x<a \wedge\) poly \(p x=0\}=\)
        (count-roots-below pa-(if poly pa=0^pキ0 then 1 else 0 ))
proof (cases poly pa=0^pキ0)
    case False
        hence card \(\{x . x<a \wedge\) poly \(p x=0\}=\operatorname{card}\{x . x \leq a \wedge\) poly \(p x=0\}\)
        proof (cases rule: disjE)
            assume \(p=0\)
            have \(\neg\) finite \(\{a-1<. .<a\}\)
            by (metis infinite-Ioo diff-add-cancel less-add-one)
            moreover have \(\{a-1<. .<a\} \subseteq\{x . x \leq a \wedge\) poly \(p x=0\}\)
                        \(\{a-1<. .<a\} \subseteq\{x . x<a \wedge\) poly \(p x=0\}\)
            using \(\langle p=0\rangle\) by auto
        ultimately have \(\neg\) finite \(\{x . x \leq a \wedge\) poly \(p x=0\}\)
                            \(\neg\) finite \(\{x . x<a \wedge\) poly \(p x=0\}\)
            by (auto dest: finite-subset[of \(\{a-1<. .<a\}]\) simp: infinite-Ioo)
        thus ?thesis by simp
    next
        assume poly p \(a \neq 0\)
        hence \(\{x . x<a \wedge\) poly \(p x=0\}=\{x . x \leq a \wedge\) poly \(p x=0\}\)
```

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

lemma poly-no-roots-less-leq:
$(\forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x \neq 0) \longleftrightarrow$
$((a \geq b \vee(p \neq 0 \wedge$ count-roots-between p a $b=0)))$
by (auto simp: count-roots-between-correct card-eq-0-iff not-le dest: poly-roots-finite)
lemma poly-pos-between-less-leq:
$(\forall x . a<x \wedge x \leq b \longrightarrow$ poly $p x>0) \longleftrightarrow$
$((a \geq b \vee(p \neq 0 \wedge$ poly $p b>0 \wedge$ count-roots-between $p a b=0)))$
by (simp only: poly-pos-between-less-leq Let-def poly-no-roots-less-leq, blast)
lemma poly-no-roots-leq-leq:
$(\forall x . a \leq x \wedge x \leq b \longrightarrow$ poly $p x \neq 0) \longleftrightarrow$
$((a>b \vee(p \neq 0 \wedge$ poly pa$a \neq 0 \wedge$ count-roots-between pab=0)))
apply (intro iffI)
apply (force simp add: count-roots-between-correct card-eq-0-iff)
apply (elim conjE disjE, simp, intro allI)
apply (rename-tac $x$, case-tac $x=a$ )
apply (auto simp add: count-roots-between-correct card-eq-0-iff dest: poly-roots-finite)
done
lemma poly-pos-between-leq-leq:
$(\forall x . a \leq x \wedge x \leq b \longrightarrow$ poly $p x>0) \longleftrightarrow$
$((a>b \vee(p \neq 0 \wedge$ poly $p a>0 \wedge$
count-roots-between pab=0)))
by (simp only: poly-pos-between-leq-leq Let-def poly-no-roots-leq-leq, force)

## lemma poly-no-roots-less-less:

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

$((a \geq b \vee p \neq 0 \wedge$ count-roots-between $p$ a $b=$ (if poly $p b=0$ then 1 else 0$)$ ))
proof (standard, goal-cases)
case $A: 1$
show ?case
proof (cases $a \geq b$ )
case True
with $A$ show ?thesis by simp
next
case False
with $A$ have $[\operatorname{simp}]: p \neq 0$ using dense $[o f$ 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 $A$ False by auto
have count-roots-between pab=
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 $[O F$ - 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 prems: 2
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 prems
by (cases $p=0$, simp, subst (asm) card-eq-0-iff, auto dest: poly-roots-finite)
qed
lemma poly-pos-between-less-less:

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

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

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 $p x \neq 0) \longleftrightarrow$
$((a \geq b \vee p \neq 0 \wedge$ poly $p a \neq 0 \wedge$ count-roots-between pab=
(if $a<b \wedge$ poly $p b=0$ then 1 else 0$)$ ))
proof (standard, goal-cases)
case prems: 1
hence $\forall x . a<x \wedge x<b \longrightarrow$ poly $p x \neq 0$ by simp
thus ?case using prems by (subst (asm) poly-no-roots-less-less, auto)

```
next
    case prems: 2
        hence ( }b\leqa\veep\not=0\wedge count-roots-between pab
                        (if poly p b=0 then 1 else 0)) by auto
        thus ?case using prems unfolding Let-def
            by (subst (asm) poly-no-roots-less-less[symmetric, unfolded Let-def],
            auto split: if-split-asm simp: less-eq-real-def)
qed
lemma poly-pos-between-leq-less:
    (\forallx.a\leqx\wedge x<b\longrightarrow poly p x>0)\longleftrightarrow
    ((a\geqb
            (if }a<b\wedge poly pb=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:
    (\forallx.x>a\longrightarrow poly p x = 0) \longleftrightarrow
            ((p\not=0\wedge count-roots-above p a=0))
proof-
    have }\forallx.\nega<x\Longrightarrow\mathrm{ 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:
    (\forallx. x>a \longrightarrow poly p x>0)\longleftrightarrow(
        p\not=0\wedge poly-inf p=1^ count-roots-above pa=0)
    unfolding Let-def
    by (subst poly-pos-greater, subst poly-no-roots-greater, force)
lemma poly-no-roots-leq:
    (}\forallx.x\leqa\longrightarrow\mathrm{ poly p }x\not=0)
        (( p\not=0\wedge count-roots-below pa=0))
    by (auto simp: Let-def count-roots-below-correct card-eq-0-iff
        intro: poly-roots-finite)
lemma poly-pos-leq:
    (}\forallx.x\leqa\longrightarrow\mathrm{ poly p x>0) }
    ( p\not=0\wedge poly-neg-inf p=1^count-roots-below pa=0)
    by (simp only: poly-pos-leq Let-def poly-no-roots-leq, blast)
lemma poly-no-roots-geq:
    (\forallx.x\geqa\longrightarrow poly p }x\not=0)
    ((p\not=0\wedge poly p a\not=0 ^ count-roots-above p a=0))
proof (standard, goal-cases)
```

```
    case prems: 1
    hence }\forallx>a. poly px\not=0 by sim
    thus ?case using prems by (subst (asm) poly-no-roots-greater, auto)
next
    case prems: 2
    hence ( }p\not=0\wedge\mathrm{ count-roots-above p a=0) by simp
    thus ?case using prems
        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 $p x>0) \longleftrightarrow$
$(p \neq 0 \wedge$ poly-inf $p=1 \wedge$ poly p $a \neq 0 \wedge$ count-roots-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 $p x \neq 0) \longleftrightarrow$
$((p \neq 0 \wedge$ count-roots-below $p a=($ if poly $p a=0$ then 1 else 0$)))$
proof (standard, goal-cases)
case prems: 1
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 prems by (auto simp: count-roots-below-correct)
next
case prems: 2
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 pa=card $\{x . x<a \wedge$ poly $p x=0\}+$
(if poly $p a=0$ then 1 else 0 ) using prems
by (subst count-roots-below-correct, subst A, subst card-Un-disjoint,
auto intro: poly-roots-finite)
with prems have card $\{x . x<a \wedge$ poly $p x=0\}=0$ by simp
thus ?case using prems
by (subst (asm) card-eq-0-iff, auto intro: poly-roots-finite)
qed
lemma poly-pos-less:
$(\forall x . x<a \longrightarrow$ poly $p x>0) \longleftrightarrow$
$(p \neq 0 \wedge$ poly-neg-inf $p=1 \wedge$ count-roots-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
```


### 3.2 Reification

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

## definition $P R-T A G x \equiv x$

lemma sturm-id-PR-prio0:
$\{x::$ real. $P x\}=\{x::$ real. $(P R-T A G P) x\}$
$(\forall x::$ real. $f x<g x)=(\forall x::$ real. PR-TAG $(\lambda x . f x<g x) x)$
$(\forall x::$ real. $P x)=(\forall x::$ real. $\neg(P R-T A G(\lambda x . \neg P x)) x)$
by (simp-all add: PR-TAG-def)
lemma sturm-id-PR-prio1:

```
\(\{x::\) real. \(x<a \wedge P x\}=\{x::\) real. \(x<a \wedge(P R-T A G P) x\}\)
\(\{x::\) real. \(x \leq a \wedge P x\}=\{x::\) real. \(x \leq a \wedge(P R-T A G P) x\}\)
\(\{x::\) real. \(x \geq b \wedge P x\}=\{x::\) real. \(x \geq b \wedge(P R-T A G P) x\}\)
\(\{x::\) real. \(x>b \wedge P x\}=\{x::\) real. \(x>b \wedge(P R-T A G P) x\}\)
\((\forall x::\) real \(<a . f x<g x)=(\forall x::\) real \(<a\). PR-TAG \((\lambda x . f x<g x) x)\)
\((\forall x::\) real \(\leq a . f x<g x)=(\forall x::\) real \(\leq a . P R-T A G(\lambda x . f x<g x) x)\)
\((\forall x::\) real \(>a . f x<g x)=(\forall x::\) real \(>a\). PR-TAG \((\lambda x . f x<g x) x)\)
\((\forall x::\) real \(\geq a . f x<g x)=(\forall x::\) real \(\geq a . P R-T A G(\lambda x . f x<g x) x)\)
\((\forall x::\) real \(<a . P x)=(\forall x::\) real \(<a . \neg(P R-T A G(\lambda x . \neg P x)) x)\)
\((\forall x::\) real \(>a . P x)=(\forall x::\) real \(>a . \neg(P R-T A G(\lambda x . \neg P x)) x)\)
\((\forall x::\) real \(\leq a . P x)=(\forall x::\) real \(\leq a . \neg(P R-T A G(\lambda x . \neg P x)) x)\)
\((\forall x::\) real \(\geq a . P x)=(\forall x::\) real \(\geq a . \neg(P R-T A G(\lambda x . \neg P x)) x)\)
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. $x>a \wedge x \leq b \wedge P R-T A G P x\}$
$\{x::$ real. $x \geq a \wedge x \leq b \wedge P x\}=$
$\{x::$ real. $x \geq a \wedge x \leq b \wedge P R-T A G P x\}$
$\{x::$ real. $x \geq a \wedge x<b \wedge P x\}=$
$\{x::$ real. $x \geq a \wedge x<b \wedge P R-T A G P x\}$
$\{x::$ real. $x>a \wedge x<b \wedge P x\}=$ $\{x::$ real. $x>a \wedge x<b \wedge P R-T A G P x\}$
( $\forall x$ :: real. $a<x \wedge x \leq b \longrightarrow f x<g x)=$
$(\forall x::$ real. $a<x \wedge x \leq b \longrightarrow P R-T A G(\lambda x . f x<g x) x)$
$(\forall x::$ real. $a \leq x \wedge x \leq b \longrightarrow f x<g x)=$
$(\forall x::$ real. $a \leq x \wedge x \leq b \longrightarrow P R-T A G(\lambda x . f x<g x) x)$ $(\forall x::$ real. $a<x \wedge x<b \longrightarrow f x<g x)=$ $(\forall x::$ real. $a<x \wedge x<b \longrightarrow P R-T A G(\lambda x . f x<g x) x)$ ( $\forall x$ :: real. $a \leq x \wedge x<b \longrightarrow f x<g x)=$ $(\forall x::$ real. $a \leq x \wedge x<b \longrightarrow P R-T A G(\lambda x . f x<g x) x)$ $(\forall x::$ real. $a<x \wedge x \leq b \longrightarrow P x)=$ $(\forall x::$ real. $a<x \wedge x \leq b \longrightarrow \neg(P R-T A G(\lambda x . \neg P x)) x)$ $(\forall x::$ real. $a \leq x \wedge x \leq b \longrightarrow P x)=$ $(\forall x::$ real. $a \leq x \wedge x \leq b \longrightarrow \neg(P R-T A G(\lambda x . \neg P x)) x)$ $(\forall x::$ real. $a \leq x \wedge x<b \longrightarrow P x)=$ $(\forall x::$ real. $a \leq x \wedge x<b \longrightarrow \neg(P R-T A G(\lambda x . \neg P x)) x)$ ( $\forall x::$ real. $a<x \wedge x<b \longrightarrow P x)=$ $(\forall x::$ real. $a<x \wedge x<b \longrightarrow \neg(P R-T A G(\lambda x . \neg P x)) x)$
by (simp-all add: PR-TAG-def)
lemma $P R$-TAG-intro-prio0:
fixes $P::$ real $\Rightarrow$ bool and $f::$ real $\Rightarrow$ real

## shows

$P R-T A G P=P^{\prime} \Longrightarrow P R-T A G(\lambda x . \neg(\neg P x))=P^{\prime}$
$\llbracket P R-T A G P=(\lambda x$. poly $p x=0) ; P R-T A G Q=(\lambda x$. poly $q x=0) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . P x \wedge Q x)=(\lambda x . p o l y(g c d p q) x=0)$ and
$\llbracket P R-T A G P=(\lambda x$. poly $p x=0) ; P R-T A G Q=(\lambda x$. poly $q x=0) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . P x \vee Q x)=(\lambda x . \operatorname{poly}(p * q) x=0)$ and
$\llbracket P R-T A G f=(\lambda x$. poly $p x) ; P R-T A G g=(\lambda x$. poly $q x) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . f x=g x)=(\lambda x$. poly $(p-q) x=0)$
$\llbracket P R-T A G f=(\lambda x$. poly $p x) ; P R-T A G g=(\lambda x$. poly $q x) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . f x \neq g x)=(\lambda x$. poly $(p-q) x \neq 0)$
$\llbracket P R-T A G f=(\lambda x$. poly $p x) ; P R-T A G g=(\lambda x$. poly $q x) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . f x<g x)=(\lambda x$. poly $(q-p) x>0)$
$\llbracket P R-T A G f=(\lambda x$. poly $p x) ; P R-T A G g=(\lambda x$. poly $q x) \rrbracket$ $\Longrightarrow P R-T A G(\lambda x . f x \leq g x)=(\lambda x . p o l y(q-p) x \geq 0)$

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

```
lemma PR-TAG-intro-prio1:
    fixes f :: real => real
    shows
    PR-TAG f = ( \lambdax. poly p x)\LongrightarrowPR-TAG (\lambdax.fx=0) = (\lambdax. poly p }x=0
    PR-TAG f = ( \lambdax. poly p x)\LongrightarrowPR-TAG (\lambdax.fx\not=0) = (\lambdax. poly p x\not=0)
    PR-TAGf = (\lambdax. poly p x)\LongrightarrowPR-TAG (\lambdax.0 = f x) = (\lambdax. poly p x = 0)
    PR-TAGf}=(\lambdax.poly px)\LongrightarrowPR-TAG (\lambdax.0\not=fx)=(\lambdax. poly p x\not=0)
    PR-TAGf}=(\lambdax.poly px)\LongrightarrowPR-TAG (\lambdax.fx\geq0)=(\lambdax. poly p x \geq 0)
    PR-TAGf}=(\lambdax.poly px)\LongrightarrowPR-TAG (\lambdax.fx>0)=(\lambdax. poly p x>0)
    PR-TAG f = ( \lambdax. poly px)\LongrightarrowPR-TAG ( \lambdax.fx\leq0) = ( \lambdax. poly (-p)x\geq0)
    PR-TAG f}=(\lambdax.poly p x)\LongrightarrowPR-TAG (\lambdax.fx<0)=(\lambdax.poly (-p)x>0
    PR-TAG f}=(\lambdax.poly px)
        PR-TAG ( }\lambdax.0\leqfx)=(\lambdax.poly (-p)x\leq0
    PR-TAG f}=(\lambdax.poly px)
        PR-TAG (\lambdax.0<fx) =( \lambdax. poly (-p) x<0)
    PR-TAG f = ( \lambdax. poly p x)
        \LongrightarrowPR-TAG (\lambdax.a*fx)=(\lambdax. poly (smult a p) x)
    PR-TAG f = ( \lambdax. poly p x)
        \LongrightarrowPR-TAG (\lambdax.fx*a)=(\lambdax. poly (smult a p) x)
    PR-TAG f = ( \lambdax. poly p x)
        \LongrightarrowPR-TAG (\lambdax.fx/a)=(\lambdax. poly (smult (inverse a) p) x)
    PR-TAG (\lambdax. x`n :: real) = ( }\lambdax.\mathrm{ poly (monom 1 n) x)
by (simp-all add: PR-TAG-def field-simps poly-monom)
lemma PR-TAG-intro-prio2:
    PR-TAG (\lambdax. 1 / b) = ( \lambdax. inverse b)
    PR-TAG (\lambdax.a/b) =(\lambdax.a/b)
    PR-TAG (\lambdax.a/b*x`n :: real) = (\lambdax. poly (monom (a/b) n) x)
    PR-TAG ( }\lambdax.x`n*a/b:: real)=(\lambdax.poly (monom (a/b) n) x)
    PR-TAG (\lambdax.a* x n :: real) = ( }\lambdax.\mathrm{ poly (monom a n) x)
    PR-TAG (\lambdax. x^n * a :: real) = ( \lambdax. poly (monom a n) x)
    PR-TAG (\lambdax. x`n / a :: real ) = (\lambdax. poly (monom (inverse a) n) x)
    PR-TAG (\lambdax.f x`(Suc (Suc 0)) :: real) = ( \lambdax. poly p x)
        CPR-TAG (\lambdax.fx*fx:: real)}=(\lambdax. poly p x)
    PR-TAG (\lambdax. (fx)^Suc n :: real) = ( }\lambdax\mathrm{ . poly p x)
        \Longrightarrow P R - T A G ( \lambda x . ( f x ) ` n * f x ~ : : ~ r e a l ) ~ = ~ ( \lambda x . ~ p o l y ~ p ~ x ) ~
    PR-TAG (\lambdax. (f x)^Suc n :: real) = ( \lambdax. poly p x)
        CPR-TAG (\lambdax.fx*(fx)`n :: real) = (\lambdax. poly p x)
    PR-TAG (\lambdax. (fx)^(m+n) :: real) = (\lambdax. poly p x)
        \Longrightarrow P R - T A G ( \lambda x . ( f x ) ` m * ( f x ) ` n ~ : : ~ r e a l ) = ( \lambda x . ~ p o l y ~ p ~ x ) ~
by (simp-all add: PR-TAG-def field-simps poly-monom power-add)
lemma sturm-meta-spec: \((\bigwedge x::\) real. \(P x) \Longrightarrow P x\) by simp
lemma sturm-imp-conv:
```

```
\((a<x \longrightarrow x<b \longrightarrow c) \longleftrightarrow(a<x \wedge x<b \longrightarrow c)\)
```

$(a<x \longrightarrow x<b \longrightarrow c) \longleftrightarrow(a<x \wedge x<b \longrightarrow c)$
$(a \leq x \longrightarrow x<b \longrightarrow c) \longleftrightarrow(a \leq x \wedge x<b \longrightarrow c)$
$(a \leq x \longrightarrow x<b \longrightarrow c) \longleftrightarrow(a \leq x \wedge x<b \longrightarrow c)$
$(a<x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow(a<x \wedge x \leq b \longrightarrow c)$
$(a<x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow(a<x \wedge x \leq b \longrightarrow c)$
$(a \leq x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow(a \leq x \wedge x \leq b \longrightarrow c)$

```
\((a \leq x \longrightarrow x \leq b \longrightarrow c) \longleftrightarrow(a \leq x \wedge x \leq b \longrightarrow c)\)
```

```
(x<b\longrightarrowa<x\longrightarrowc)\longleftrightarrow(a<x\wedgex<b\longrightarrowc)
(x<b\longrightarrowa\leqx\longrightarrowc)\longleftrightarrow(a\leqx^x<b\longrightarrowc)
(x\leqb\longrightarrowa<x\longrightarrowc)\longleftrightarrow(a<x\wedgex\leqb\longrightarrowc)
(x\leqb\longrightarrowa\leqx\longrightarrowc)\longleftrightarrow(a\leqx\wedge x \leqb\longrightarrowc)
by auto
```


### 3.3 Setup for the "sturm" method

ML-file 〈sturm.ML
method-setup sturm $=$ <
Scan.succeed (fn ctxt => SIMPLE-METHOD' (Sturm.sturm-tac ctxt true))
$\rightarrow$
end
theory Sturm
imports Sturm-Method
begin
end

## 4 Example usage of the "sturm" method

theory Sturm-Ex
imports ../Sturm
begin
In this section, we give a variety of statements about real polynomials that can b proven by the sturm method.

```
lemma
\(\forall x:\) :real. \(x^{\wedge} 2+1 \neq 0\)
by sturm
lemma
    fixes \(x\) :: real
    shows \(\times\) ^2 \(+1 \neq 0\) by sturm
lemma \((x::\) real \()>1 \Longrightarrow x\) 3 \(>1\) by sturm
lemma \(\forall x:\) :real. \(x * x \neq-1\) by sturm
schematic-goal \(A\) :
card \(\{x::\) real. \(-0.010831<x \wedge x<0.010831 \wedge\)
    \(1 / 120 * x \wedge 5+1 / 24 * x \wedge 4+1 / 6 * x\)-3 \(-49 / 16777216 * x\) へ2 \(-17 / 2097152 * x=\)
0\}
    \(=? n\)
    by sturm
```

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

## lemma

$$
\begin{aligned}
& \text { card }\{x:: \text { real. }-0.010831<x \wedge x<0.010831 \wedge \\
& \quad \text { poly }[: 0,-17 / 2097152,-49 / 16777216,1 / 6,1 / 24,1 / 120:] x=0\}=3
\end{aligned}
$$

by sturm
lemma $\forall x::$ real. $x * x \neq 0 \vee x * x-1 \neq 2 * x$ by sturm
lemma $(x::$ real $) * x+1 \neq 0 \wedge(x$ ^2 +1$) *(x$ ค2 +2$) \neq 0$ by sturm
3 examples related to continued fraction approximants to exp: LCP

```
lemma fixes x::real
    shows -7.29347719 \leq x 0 0 x^5 + 30*x^4 + 420*x^3 + 3360*x^2 +
15120*x + 30240
by sturm
lemma fixes x::real
    shows 0< x^6 + 42*x^5 + 840*x^4 + 10080*x`3 + 75600*x 2 + 332640*x
+665280
by sturm
```

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

