

Descartes' Rule of Signs

Manuel Eberl

December 17, 2016

Abstract

In this work, we formally proved Descartes Rule of Signs, which relates the number of positive real roots of a polynomial with the number of sign changes in its coefficient list.

Our proof follows the simple inductive proof given by Arthan [1], which was also used by John Harrison in his HOL Light formalisation. We proved most of the lemmas for arbitrary linearly-ordered integrity domains (e.g. integers, rationals, reals); the main result, however, requires the intermediate value theorem and was therefore only proven for real polynomials.

Contents

1	Sign changes and Descartes' Rule of Signs	1
1.1	Polynomials	2
1.2	List of partial sums	3
1.3	Sign changes in a list	4
1.4	Arthan's lemma	7
1.5	Roots of a polynomial with a certain property	11
1.6	Coefficient sign changes of a polynomial	13
1.7	Proof of Descartes' sign rule	15

1 Sign changes and Descartes' Rule of Signs

```
theory Descartes-Sign-Rule
```

```
imports
```

```
  Complex-Main
```

```
  ~/src/HOL/Library/Polynomial
```

```
begin
```

```
lemma op-plus-0: (op + (0 :: 'a :: monoid-add)) = id
```

```
  by auto
```

```
lemma filter-dropWhile:
```

```
  filter (λx. ¬P x) (dropWhile P xs) = filter (λx. ¬P x) xs
```

```
  by (induction xs) simp-all
```

1.1 Polynomials

A real polynomial whose leading and constant coefficients have opposite non-zero signs must have a positive root.

lemma *pos-root-exI*:

assumes $\text{poly } p \ 0 * \text{lead-coeff } p < (0 :: \text{real})$

obtains x **where** $x > 0$ $\text{poly } p \ x = 0$

proof –

have $P: \exists x > 0. \text{poly } p \ x = (0 :: \text{real})$ **if** $\text{lead-coeff } p > 0$ $\text{poly } p \ 0 < 0$ **for** p

proof –

note $\text{that}(1)$

also from $\text{poly-pinfty-gt-lc}[OF \langle \text{lead-coeff } p > 0 \rangle]$ **obtain** $x0$

where $\bigwedge x. x \geq x0 \implies \text{poly } p \ x \geq \text{lead-coeff } p$ **by** *auto*

hence $\text{poly } p \ (\text{max } x0 \ 1) \geq \text{lead-coeff } p$ **by** *auto*

finally have $\text{poly } p \ (\text{max } x0 \ 1) > 0$.

with that have $\exists x. x > 0 \wedge x < \text{max } x0 \ 1 \wedge \text{poly } p \ x = 0$

by (*intro poly-IVT mult-neg-pos*) *auto*

thus $\exists x > 0. \text{poly } p \ x = 0$ **by** *auto*

qed

show *?thesis*

proof (*cases lead-coeff p > 0*)

case *True*

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

by (*auto simp: mult-less-0-iff*)

from $P[OF \ \text{True} \ \text{this}]$ **that** **show** *?thesis*

by *blast*

next

case *False*

from *False assms* **have** $\text{poly } (-p) \ 0 < 0$

by (*auto simp: mult-less-0-iff*)

moreover from *assms* **have** $p \neq 0$

by *auto*

with *False* **have** $\text{lead-coeff } (-p) > 0$

by (*cases rule: linorder-cases[of lead-coeff p 0]*)

(*simp-all add: lead-coeff-def*)

ultimately show *?thesis* **using that** $P[\text{of } -p]$ **by** *auto*

qed

qed

Substitute X with aX in a polynomial $p(X)$. This turns all the $X - a$ factors in p into factors of the form $X - 1$.

definition *reduce-root* **where**

$\text{reduce-root } a \ p = \text{pcompose } p \ [;0, a;]$

lemma *reduce-root-pCons*:

$\text{reduce-root } a \ (\text{pCons } c \ p) = \text{pCons } c \ (\text{smult } a \ (\text{reduce-root } a \ p))$

by (*simp add: reduce-root-def pcompose-pCons*)

lemma *reduce-root-nonzero* [simp]:
 $a \neq 0 \implies p \neq 0 \implies \text{reduce-root } a \ p \neq (0 :: 'a :: \text{idom } \text{poly})$
unfolding *reduce-root-def* **using** *pcompose-eq-0*[of p [:0, a:]]
by *auto*

1.2 List of partial sums

We first define, for a given list, the list of accumulated partial sums from left to right: the list *psums xs* has as its i -th entry $\sum_{j=0}^i \text{xs}_j$.

fun *psums* **where**
 $\text{psums } [] = []$
 $\text{psums } [x] = [x]$
 $\text{psums } (x \# y \# xs) = x \# \text{psums } ((x+y) \# xs)$

lemma *length-psums* [simp]: $\text{length } (\text{psums } xs) = \text{length } xs$
by (*induction xs rule: psums.induct*) *simp-all*

lemma *psums-Cons*:
 $\text{psums } (x \# xs) = (x :: 'a :: \text{semigroup-add}) \# \text{map } (op + x) (\text{psums } xs)$
by (*induction xs rule: psums.induct*) (*simp-all add: algebra-simps*)

lemma *last-psums*:
 $(xs :: 'a :: \text{monoid-add list}) \neq [] \implies \text{last } (\text{psums } xs) = \text{sum-list } xs$
by (*induction xs rule: psums.induct*)
(auto simp add: add.assoc [symmetric] psums-Cons o-def)

lemma *psums-0-Cons* [simp]:
 $\text{psums } (0 \# xs :: 'a :: \text{monoid-add list}) = 0 \# \text{psums } xs$
by (*induction xs rule: psums.induct*) (*simp-all add: algebra-simps*)

lemma *map-uminus-psums*:
fixes $xs :: 'a :: \text{ab-group-add list}$
shows $\text{map } \text{uminus } (\text{psums } xs) = \text{psums } (\text{map } \text{uminus } xs)$
by (*induction xs rule: psums.induct*) (*simp-all*)

lemma *psums-replicate-0-append*:
 $\text{psums } (\text{replicate } n \ (0 :: 'a :: \text{monoid-add}) \ @ \ xs) =$
 $\text{replicate } n \ 0 \ @ \ \text{psums } xs$
by (*induction n*) (*simp-all add: psums-Cons op-plus-0*)

lemma *psums-nth*: $n < \text{length } xs \implies \text{psums } xs ! n = (\sum_{i \leq n} xs ! i)$
proof (*induction xs arbitrary: n rule: psums.induct[case-names Nil sng rec]*)
case (*rec x y xs n*)
show *?case*
proof (*cases n*)
case (*Suc m*)
from *Suc* **have** $\text{psums } (x \# y \# xs) ! n = \text{psums } ((x+y) \# xs) ! m$ **by** *simp*
also from *rec.prem* *Suc* **have** $\dots = (\sum_{i \leq m} ((x+y) \# xs) ! i)$
by (*intro rec.IH*) *simp-all*

also have $\dots = x + y + (\sum_{i=1..m}. (y \# xs) ! i)$
by (*auto simp: atLeast0AtMost [symmetric] sum-head-Suc[of 0]*)
also have $(\sum_{i=1..m}. (y \# xs) ! i) = (\sum_{i=Suc\ 1..Suc\ m}. (x \# y \# xs) ! i)$
by (*subst sum-shift-bounds-cl-Suc-ivl simp*)
also from *Suc* **have** $x + y + \dots = (\sum_{i \leq n}. (x \# y \# xs) ! i)$
by (*auto simp: atLeast0AtMost [symmetric] sum-head-Suc add-ac*)
finally show *?thesis* .
qed *simp*
qed *simp-all*

1.3 Sign changes in a list

Next, we define the number of sign changes in a sequence. Intuitively, this is the number of times that, when passing through the list, a sign change between one element and the next element occurs (while ignoring all zero entries).

We implement this by filtering all zeros from the list of signs, removing all adjacent equal elements and taking the length of the resulting list minus one.

definition *sign-changes* :: (*'a* :: {*sgn,zero*} *list*) \Rightarrow *nat* **where**
*sign-changes xs = length (remdups-adj (filter ($\lambda x. x \neq 0$) (map *sgn xs*))) - 1*

lemma *sign-changes-Nil* [*simp*]: *sign-changes [] = 0*
by (*simp add: sign-changes-def*)

lemma *sign-changes-singleton* [*simp*]: *sign-changes [x] = 0*
by (*simp add: sign-changes-def*)

lemma *sign-changes-cong*:
assumes *map sgn xs = map sgn ys*
shows *sign-changes xs = sign-changes ys*
using *assms unfolding sign-changes-def by simp*

lemma *sign-changes-Cons-ge*: *sign-changes (x # xs) \geq sign-changes xs*
unfolding *sign-changes-def by (simp add: remdups-adj-Cons split: list.split)*

lemma *sign-changes-Cons-Cons-different*:
fixes *x y :: 'a :: linordered-idom*
assumes *x * y < 0*
shows *sign-changes (x # y # xs) = 1 + sign-changes (y # xs)*

proof –
from *assms* **have** *sgn x = -1 \wedge sgn y = 1 \vee sgn x = 1 \wedge sgn y = -1*
by (*auto simp: mult-less-0-iff*)
thus *?thesis by (fastforce simp: sign-changes-def)*
qed

lemma *sign-changes-Cons-Cons-same*:
fixes *x y :: 'a :: linordered-idom*

shows $x * y > 0 \implies \text{sign-changes } (x \# y \# xs) = \text{sign-changes } (y \# xs)$
by (*subst (asm) zero-less-mult-iff*) (*fastforce simp: sign-changes-def*)

lemma *sign-changes-0-Cons* [*simp*]:
 $\text{sign-changes } (0 \# xs :: 'a :: \text{idom-abs-sgn list}) = \text{sign-changes } xs$
by (*simp add: sign-changes-def*)

lemma *sign-changes-two*:
fixes $x y :: 'a :: \text{linordered-idom}$
shows $\text{sign-changes } [x,y] =$
 $(\text{if } x > 0 \wedge y < 0 \vee x < 0 \wedge y > 0 \text{ then } 1 \text{ else } 0)$
by (*auto simp: sgn-if sign-changes-def mult-less-0-iff*)

lemma *sign-changes-induct* [*case-names nil sing zero nonzero*]:
assumes $P [] \wedge x. P [x] \wedge xs. P xs \implies P (0 \# xs)$
 $\wedge x y xs. x \neq 0 \implies P ((x + y) \# xs) \implies P (x \# y \# xs)$
shows $P xs$
proof (*induction length xs arbitrary: xs rule: less-induct*)
case (*less xs*)
show *?case*
proof (*cases xs rule: psums.cases*)
fix $x y xs'$ **assume** $xs = x \# y \# xs'$
with *assms less* **show** *?thesis* **by** (*cases x = 0*) *auto*
qed (*insert less assms, auto*)
qed

lemma *sign-changes-filter*:
fixes $xs :: 'a :: \text{linordered-idom list}$
shows $\text{sign-changes } (\text{filter } (\lambda x. x \neq 0) xs) = \text{sign-changes } xs$
by (*simp add: sign-changes-def filter-map o-def sgn-0-0*)

lemma *sign-changes-Cons-Cons-0*:
fixes $xs :: 'a :: \text{linordered-idom list}$
shows $\text{sign-changes } (x \# 0 \# xs) = \text{sign-changes } (x \# xs)$
by (*subst (1 2) sign-changes-filter [symmetric] simp-all*)

lemma *sign-changes-uminus*:
fixes $xs :: 'a :: \text{linordered-idom list}$
shows $\text{sign-changes } (\text{map } \text{uminus } xs) = \text{sign-changes } xs$
proof –
have $\text{sign-changes } (\text{map } \text{uminus } xs) =$
 $\text{length } (\text{remdups-adj } [x \leftarrow \text{map } \text{sgn } (\text{map } \text{uminus } xs) . x \neq 0]) - 1$
unfolding *sign-changes-def* ..
also have $\text{map } \text{sgn } (\text{map } \text{uminus } xs) = \text{map } \text{uminus } (\text{map } \text{sgn } xs)$
by (*auto simp: sgn-minus*)
also have $\text{remdups-adj } (\text{filter } (\lambda x. x \neq 0) \dots) =$
 $\text{map } \text{uminus } (\text{remdups-adj } (\text{filter } (\lambda x. x \neq 0) (\text{map } \text{sgn } xs)))$
by (*subst filter-map, subst remdups-adj-map-injective*)
(simp-all add: o-def)

also have $\text{length } \dots - 1 = \text{sign-changes } xs$ by (simp add: sign-changes-def)
 finally show ?thesis .
 qed

lemma *sign-changes-replicate*: $\text{sign-changes } (\text{replicate } n \ x) = 0$
 by (simp add: sign-changes-def remdups-adj-replicate filter-replicate)

lemma *sign-changes-decompose*:
 assumes $x \neq 0 :: 'a :: \text{linordered-idom}$
 shows $\text{sign-changes } (xs \ @ \ x \ \# \ ys) =$
 $\text{sign-changes } (xs \ @ \ [x]) + \text{sign-changes } (x \ \# \ ys)$

proof –
 have $\text{sign-changes } (xs \ @ \ x \ \# \ ys) =$
 $\text{length } (\text{remdups-adj } ([x \leftarrow \text{map } \text{sgn } xs . x \neq 0] \ @ \$
 $\text{sgn } x \ \# \ [x \leftarrow \text{map } \text{sgn } ys . x \neq 0])) - 1$
 by (simp add: sgn-0-0 assms sign-changes-def)
 also have $\dots = \text{sign-changes } (xs \ @ \ [x]) + \text{sign-changes } (x \ \# \ ys)$
 by (subst remdups-adj-append) (simp add: sign-changes-def assms sgn-0-0)
 finally show ?thesis .
 qed

If the first and the last entry of a list are non-zero, its number of sign changes is even if and only if the first and the last element have the same sign. This will be important later to establish the base case of Descartes' Rule. (if there are no positive roots, the number of sign changes is even)

lemma *even-sign-changes-iff*:
 assumes $xs \neq [] :: 'a :: \text{linordered-idom list}$ $\text{hd } xs \neq 0$ $\text{last } xs \neq 0$
 shows $\text{even } (\text{sign-changes } xs) \longleftrightarrow \text{sgn } (\text{hd } xs) = \text{sgn } (\text{last } xs)$
 using assms
proof (induction length xs arbitrary: xs rule: less-induct)
 case (less xs)
 show ?case
proof (cases xs)
 case (Cons x xs')
 note $x = \text{this}$
 show ?thesis
proof (cases xs')
 case (Cons y xs'')
 note $y = \text{this}$
 show ?thesis
proof (rule linorder-cases[of $x*y$ 0])
 assume $xy: x*y = 0$
 with $x \ y$ less(1,3,4) show ?thesis by (auto simp: sign-changes-Cons-Cons-0)
 next
 assume $xy: x*y > 0$
 with less(1,4) show ?thesis
 by (auto simp add: $x \ y$ sign-changes-Cons-Cons-same zero-less-mult-iff)
 next
 assume $xy: x*y < 0$

```

moreover from  $xy$  have  $\text{sgn } x = - \text{sgn } y$  by (auto simp: mult-less-0-iff)
moreover have even (sign-changes ( $y \# xs''$ ))  $\longleftrightarrow$ 
     $\text{sgn } (\text{hd } (y \# xs'')) = \text{sgn } (\text{last } (y \# xs''))$ 
using  $xy$  less.prems by (intro less) (auto simp: x y)
moreover from  $xy$  less.prems
have  $\text{sgn } y = \text{sgn } (\text{last } xs) \longleftrightarrow -\text{sgn } y \neq \text{sgn } (\text{last } xs)$ 
by (auto simp: sgn-if)
ultimately show ?thesis by (auto simp: sign-changes-Cons-Cons-different
 $x y$ )
qed
qed (auto simp: x)
qed (insert less.prems, simp-all)
qed

```

1.4 Arthan's lemma

```

context
begin

```

We first prove an auxiliary lemma that allows us to assume w.l.o.g. that the first element of the list is non-negative, similarly to what Arthan does in his proof.

```

private lemma arthan-wlog [consumes 3, case-names nonneg lift]:
fixes  $xs :: 'a :: \text{linordered-idom list}$ 
assumes  $xs \neq []$   $\text{last } xs \neq 0$   $x + y + \text{sum-list } xs = 0$ 
assumes  $\bigwedge x y xs. xs \neq [] \implies \text{last } xs \neq 0 \implies$ 
     $x + y + \text{sum-list } xs = 0 \implies x \geq 0 \implies P x y xs$ 
assumes  $\bigwedge x y xs. xs \neq [] \implies P x y xs \implies P (-x) (-y) (\text{map } \text{uminus } xs)$ 
shows  $P x y xs$ 
proof (cases  $x \geq 0$ )
assume  $x: \neg(x \geq 0)$ 
from assms have  $\text{map } \text{uminus } xs \neq []$  by simp
moreover from  $x$  assms(1,2,3) have  $P (-x) (-y) (\text{map } \text{uminus } xs)$ 
using uminus-sum-list-map[of  $\lambda x. x$  xs, symmetric]
by (intro assms) (auto simp: last-map algebra-simps o-def neg-eq-iff-add-eq-0)
ultimately have  $P (-(-x)) (-(-y)) (\text{map } \text{uminus } (\text{map } \text{uminus } xs))$  by (rule
assms)
thus ?thesis by (simp add: o-def)
qed (simp-all add: assms)

```

We now show that the α and β in Arthan's proof have the necessary properties: their difference is non-negative and even.

```

private lemma arthan-aux1:
fixes  $xs :: 'a :: \{\text{linordered-idom}\} \text{ list}$ 
assumes  $xs \neq []$   $\text{last } xs \neq 0$   $x + y + \text{sum-list } xs = 0$ 
defines  $v \equiv \lambda xs. \text{int } (\text{sign-changes } xs)$ 
shows  $v (x \# y \# xs) - v ((x + y) \# xs) \geq$ 
     $v (\text{psums } (x \# y \# xs)) - v (\text{psums } ((x + y) \# xs)) \wedge$ 
    even ( $v (x \# y \# xs) - v ((x + y) \# xs) -$ 

```

```

      (v (psums (x # y # xs)) - v (psums ((x + y) # xs)))
using assms(1-3)
proof (induction rule: arthan-wlog)
  have uminus-v: v (map uminus xs) = v xs for xs by (simp add: v-def sign-changes-uminus)

  case (lift x y xs)
  note lift(2)
  also have v (psums (x#y#xs)) - v (psums ((x+y)#xs)) =
    v (psums (- x # - y # map uminus xs)) -
    v (psums ((- x + - y) # map uminus xs))
    by (subst (1 2) uminus-v [symmetric] (simp add: map-uminus-psums))
  also have v (x # y # xs) - v ((x + y) # xs) =
    v (-x # -y # map uminus xs) - v ((-x + -y) # map uminus xs)
    by (subst (1 2) uminus-v [symmetric] simp)
  finally show ?case .
next
  case (nonneg x y xs)
  define p where p = (LEAST n. xs ! n ≠ 0)
  define xs1 :: 'a list where xs1 = replicate p 0
  define xs2 where xs2 = drop (Suc p) xs
  from nonneg have xs ! (length xs - 1) ≠ 0 by (simp add: last-conv-nth)
  hence p-nz: xs ! p ≠ 0 unfolding p-def by (rule LeastI)
  {
    fix q assume q < p hence xs ! q = 0
    using Least-le[of λn. xs ! n ≠ 0 q] unfolding p-def by force
  } note less-p-zero = this
  from Least-le[of λn. xs ! n ≠ 0 length xs - 1] nonneg
    have p ≤ length xs - 1 unfolding p-def by (auto simp: last-conv-nth)
  with nonneg have p-less-length: p < length xs by (cases xs) simp-all

  from p-less-length less-p-zero have take p xs = replicate p 0
    by (subst list-eq-iff-nth-eq) auto
  with p-less-length have xs-decompose: xs = xs1 @ xs ! p # xs2
    unfolding xs1-def xs2-def
    by (subst append-take-drop-id [of p, symmetric],
      subst Cons-nth-drop-Suc) simp-all

  have v-decompose: v (xs' @ xs) = v (xs' @ [xs ! p]) + v (xs ! p # xs2) for xs'
  proof -
    have xs' @ xs = (xs' @ xs1) @ xs ! p # xs2 by (subst xs-decompose) simp
    also have v ... = v (xs' @ [xs ! p]) + v (xs ! p # xs2) unfolding v-def
      by (subst sign-changes-decompose[OF p-nz],
        subst (1 2 3 4) sign-changes-filter [symmetric]) (simp-all add: xs1-def)
    finally show ?thesis .
  qed

  have psums-decompose: psums xs = replicate p 0 @ psums (xs!p # xs2)
    by (subst xs-decompose) (simp add: xs1-def psums-replicate-0-append)
  have v-psums-decompose: sign-changes (xs' @ psums xs) = sign-changes (xs' @

```



```

[xs!p]) +
  sign-changes (xs!p # map (op + (xs!p)) (psums xs2)) for xs'
proof –
  fix xs' :: 'a list
  have sign-changes (xs' @ psums xs) =
    sign-changes (xs' @ xs ! p # map (op + (xs!p)) (psums xs2))
  by (subst psums-decompose, subst (1 2) sign-changes-filter [symmetric])
    (simp-all add: psums-Cons)
  also have ... = sign-changes (xs' @ [xs!p]) +
    sign-changes (xs!p # map (op + (xs!p)) (psums xs2))
  by (subst sign-changes-decompose[OF p-nz]) simp-all
  finally show sign-changes (xs' @ psums xs) = ... .
qed

show ?case
proof (cases x > 0)
  assume  $\neg(x > 0)$ 
  with nonneg show ?thesis by (auto simp: v-def)
next
  assume x: x > 0
  show ?thesis
  proof (rule linorder-cases[of y 0])
    assume y: y > 0
    from x and this have xy: x + y > 0 by (rule add-pos-pos)
    with y have sign-changes ((x + y) # xs) = sign-changes (y # xs)
    by (intro sign-changes-cong) auto
    moreover have sign-changes (x # psums ((x + y) # xs)) =
      sign-changes (psums ((x+y) # xs))
    using xy by (subst (1 2) psums-Cons) (simp-all add: sign-changes-Cons-Cons-same)
    ultimately show ?thesis using x y
    by (simp add: v-def algebra-simps sign-changes-Cons-Cons-same)
  next
  assume y: y = 0
  with x show ?thesis
    by (simp add: v-def sign-changes-Cons-Cons-0 psums-Cons
      o-def sign-changes-Cons-Cons-same)
  next
  assume y: y < 0
  with x have different: x * y < 0 by (rule mult-pos-neg)
  show ?thesis
  proof (rule linorder-cases[of x + y 0])
    assume xy: x + y < 0
    with x have different': x * (x + y) < 0 by (rule mult-pos-neg)
    have ( $\lambda t. t + (x + y)$ ) = (op + (x + y)) by (rule ext) simp
    moreover from y xy have sign-changes ((x+y) # xs) = sign-changes (y
# xs)
    by (intro sign-changes-cong) auto
    ultimately show ?thesis using xy different different' y
    by (simp add: v-def sign-changes-Cons-Cons-different psums-Cons o-def

```

```

add-ac)
next
  assume  $xy: x + y = 0$ 
  show ?case
  proof (cases  $xs ! p > 0$ )
    assume  $p: xs ! p > 0$ 
    from  $p y$  have  $different': y * xs ! p < 0$  by (intro mult-neg-pos)
    with  $v-decompose[of [x, y]] v-decompose[of [x+y]] x xy p different different'$ 

       $v-psums-decompose[of [x]] v-psums-decompose[of []]$ 
    show ?thesis by (auto simp add: algebra-simps v-def sign-changes-Cons-Cons-0

      sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)

  next
    assume  $\neg(xs ! p > 0)$ 
    with  $p-nz$  have  $p: xs ! p < 0$  by simp
    from  $p y$  have  $same: y * xs ! p > 0$  by (intro mult-neg-neg)
    from  $p x$  have  $different': x * xs ! p < 0$  by (intro mult-pos-neg)
    from  $v-decompose[of [x, y]] v-decompose[of [x+y]] xy different different'$ 
  same
     $v-psums-decompose[of [x]] v-psums-decompose[of []]$ 
  show ?thesis by (auto simp add: algebra-simps v-def sign-changes-Cons-Cons-0

    sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)

qed
next
  assume  $xy: x + y > 0$ 
  from  $x$  and  $this$  have  $same: x * (x + y) > 0$  by (rule mult-pos-pos)
  show ?case
  proof (cases  $xs ! p > 0$ )
    assume  $p: xs ! p > 0$ 
    from  $xy p$  have  $same': (x + y) * xs ! p > 0$  by (intro mult-pos-pos)
    from  $p y$  have  $different': y * xs ! p < 0$  by (intro mult-neg-pos)
    have  $(\lambda t. t + (x + y)) = (op + (x + y))$  by (rule ext) simp
    with  $v-decompose[of [x, y]] v-decompose[of [x+y]] different different' same$ 
  same'
    show ?thesis by (auto simp add: algebra-simps v-def psums-Cons o-def

      sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)

  next
    assume  $\neg(xs ! p > 0)$ 
    with  $p-nz$  have  $p: xs ! p < 0$  by simp
    from  $xy p$  have  $different': (x + y) * xs ! p < 0$  by (rule mult-pos-neg)
    from  $y p$  have  $same': y * xs ! p > 0$  by (rule mult-neg-neg)
    have  $(\lambda t. t + (x + y)) = (op + (x + y))$  by (rule ext) simp
    with  $v-decompose[of [x, y]] v-decompose[of [x+y]] different different' same$ 
  same'
    show ?thesis by (auto simp add: algebra-simps v-def psums-Cons o-def

      sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)

qed

```

qed
 qed
 qed
 qed

Now we can prove the main lemma of the proof by induction over the list with our specialised induction rule for *sign-changes*. It states that for a non-empty list whose last element is non-zero and whose sum is zero, the difference of the sign changes in the list and in the list of its partial sums is odd and positive.

lemma *arthan*:

```

fixes xs :: 'a :: linordered-idom list
assumes xs ≠ [] last xs ≠ 0 sum-list xs = 0
shows sign-changes xs > sign-changes (psums xs) ∧
       odd (sign-changes xs - sign-changes (psums xs))
using assms
proof (induction xs rule: sign-changes-induct)
case (nonzero x y xs)
show ?case
proof (cases xs = [])
case False
define  $\alpha$  where  $\alpha = \text{int } (\text{sign-changes } (x \# y \# xs)) - \text{int } (\text{sign-changes } ((x + y) \# xs))$ 
define  $\beta$  where  $\beta = \text{int } (\text{sign-changes } (\text{psums } (x \# y \# xs))) - \text{int } (\text{sign-changes } (\text{psums } ((x+y) \# xs)))$ 
from nonzero False have  $\alpha \geq \beta \wedge \text{even } (\alpha - \beta)$  unfolding  $\alpha$ -def  $\beta$ -def
by (intro arthan-aux1) auto
from False and nonzero.prems have
  sign-changes (psums ((x + y) # xs)) < sign-changes ((x + y) # xs) ∧
  odd (sign-changes ((x + y) # xs) - sign-changes (psums ((x + y) # xs)))
by (intro nonzero.IH) (auto simp: add.assoc)
with arthan-aux1 [of xs x y] nonzero(4,5) False(1) show ?thesis by force
qed (insert nonzero.prems, auto split: if-split-asm simp: sign-changes-two add-eq-0-iff)
qed (auto split: if-split-asm simp: add-eq-0-iff)
end

```

1.5 Roots of a polynomial with a certain property

The set of roots of a polynomial p that fulfil a given property P :

definition *roots-with* $P p = \{x. P x \wedge \text{poly } p x = 0\}$

The number of roots of a polynomial p with a given property P , where multiple roots are counted multiple times.

definition *count-roots-with* $P p = (\sum_{x \in \text{roots-with } P p. \text{order } x p}$)

abbreviation *pos-roots* $\equiv \text{roots-with } (\lambda x. x > 0)$

abbreviation *count-pos-roots* $\equiv \text{count-roots-with } (\lambda x. x > 0)$

lemma *finite-roots-with* [simp]:
 $(p :: 'a :: \text{linordered-idom poly}) \neq 0 \implies \text{finite} (\text{roots-with } P p)$
by (rule *finite-subset*[*OF - poly-roots-finite*[of p]]) (auto simp: *roots-with-def*)

lemma *count-roots-with-times-root*:
assumes $p \neq 0$ $P (a :: 'a :: \text{linordered-idom})$
shows $\text{count-roots-with } P ([:a, -1:] * p) = \text{Suc} (\text{count-roots-with } P p)$
proof –
define q **where** $q = [:a, -1:] * p$
from *assms* **have** $a \in \text{roots-with } P q$ **by** (simp-all add: *roots-with-def q-def*)
have $q\text{-nz}: q \neq 0$ **unfolding** *q-def* **by** (rule *no-zero-divisors*) (simp-all add: *assms*)

have $\text{count-roots-with } P q = (\sum_{x \in \text{roots-with } P q} \text{order } x q)$ **by** (simp add: *count-roots-with-def*)
also from $a \text{ } q\text{-nz}$ **have** $\dots = \text{order } a q + (\sum_{x \in \text{roots-with } P q - \{a\}} \text{order } x q)$
by (subst *sum.remove*) *simp-all*
also have $\text{order } a q = \text{order } a [:a, -1:] + \text{order } a p$ **unfolding** *q-def*
by (subst *order-mult*[*OF no-zero-divisors*]) (simp-all add: *assms*)
also have $\text{order } a [:a, -1:] = 1$
by (subst *order-smult* [of -1 , *symmetric*])
(*insert order-power-n-n*[of a 1], *simp-all add: order-1*)
also have $(\sum_{x \in \text{roots-with } P q - \{a\}} \text{order } x q) = (\sum_{x \in \text{roots-with } P q - \{a\}} \text{order } x p)$
order x p
proof (*intro sum.cong refl*)
fix x **assume** $x \in \text{roots-with } P q - \{a\}$
from *assms* **have** $\text{order } x q = \text{order } x [:a, -1:] + \text{order } x p$ **unfolding** *q-def*
by (subst *order-mult*[*OF no-zero-divisors*]) (simp-all add: *assms*)
also from x **have** $\text{order } x [:a, -1:] = 0$ **by** (*intro order-0I*) *simp-all*
finally show $\text{order } x q = \text{order } x p$ **by** *simp*
qed
also from $a \text{ } q\text{-nz}$ **have** $1 + \text{order } a p + (\sum_{x \in \text{roots-with } P q - \{a\}} \text{order } x p)$
 $=$
 $1 + (\sum_{x \in \text{roots-with } P q} \text{order } x p)$
by (*subst add.assoc*, *subst sum.remove*[*symmetric*]) *simp-all*
also from $q\text{-nz}$ **have** $(\sum_{x \in \text{roots-with } P q} \text{order } x p) = (\sum_{x \in \text{roots-with } P p} \text{order } x p)$
order x p
proof (*intro sum.mono-neutral-right*)
show $\text{roots-with } P p \subseteq \text{roots-with } P q$
by (auto simp: *roots-with-def q-def simp del: mult-pCons-left*)
show $\forall x \in \text{roots-with } P q - \text{roots-with } P p. \text{order } x p = 0$
by (auto simp: *roots-with-def q-def order-root simp del: mult-pCons-left*)
qed *simp-all*
finally show *?thesis* **by** (simp add: *q-def count-roots-with-def*)
qed

1.6 Coefficient sign changes of a polynomial

abbreviation (*input*) $\text{coeff-sign-changes } f \equiv \text{sign-changes } (\text{coeffs } f)$

We first show that when building a polynomial from a coefficient list, the coefficient sign changes of the resulting polynomial are the same as the same sign changes in the list.

Note that constructing a polynomial from a list removes all trailing zeros.

lemma *sign-changes-coeff-sign-changes*:

assumes $\text{Poly } xs = (p :: 'a :: \text{linordered-idom } \text{poly})$

shows $\text{sign-changes } xs = \text{coeff-sign-changes } p$

proof –

have $\text{coeffs } p = \text{coeffs } (\text{Poly } xs)$ **by** (*subst assms*) (*rule refl*)

also have $\dots = \text{strip-while } (op= 0) xs$ **by** *simp*

also have $\text{filter } (op\neq 0) \dots = \text{filter } (op\neq 0) xs$ **unfolding** *strip-while-def o-def*

by (*subst rev-filter [symmetric]*, *subst filter-dropWhile*) (*simp-all add: rev-filter*)

also have $\text{sign-changes } \dots = \text{sign-changes } xs$ **by** (*simp add: sign-changes-filter*)

finally show *?thesis* **by** (*simp add: sign-changes-filter*)

qed

By applying *reduce-root a*, we can assume w.l.o.g. that the root in question is 1, since applying root reduction does not change the number of sign changes.

lemma *coeff-sign-changes-reduce-root*:

assumes $a > (0 :: 'a :: \text{linordered-idom})$

shows $\text{coeff-sign-changes } (\text{reduce-root } a p) = \text{coeff-sign-changes } p$

proof (*intro sign-changes-cong, induction p*)

case (*pCons c p*)

have $\text{map } \text{sgn } (\text{coeffs } (\text{reduce-root } a (pCons c p))) =$

$\text{cCons } (\text{sgn } c) (\text{map } \text{sgn } (\text{coeffs } (\text{reduce-root } a p)))$

using *assms* **by** (*auto simp add: cCons-def sgn-0-0 sgn-mult reduce-root-pCons coeffs-smult*)

also note *pCons.IH*

also have $\text{cCons } (\text{sgn } c) (\text{map } \text{sgn } (\text{coeffs } p)) = \text{map } \text{sgn } (\text{coeffs } (pCons c p))$

using *assms* **by** (*auto simp add: cCons-def sgn-0-0*)

finally show *?case* .

qed (*simp-all add: reduce-root-def*)

Multiplying a polynomial with a positive constant also does not change the number of sign changes. (in fact, any non-zero constant would also work, but the proof is slightly more difficult and positive constants suffice in our use case)

lemma *coeff-sign-changes-smult*:

assumes $a > (0 :: 'a :: \text{linordered-idom})$

shows $\text{coeff-sign-changes } (\text{smult } a p) = \text{coeff-sign-changes } p$

using *assms* **by** (*auto intro!: sign-changes-cong simp: sgn-mult coeffs-smult*)

context

begin

We now show that a polynomial with an odd number of sign changes contains a positive root. We first assume that the constant coefficient is non-zero. Then it is clear that the polynomial's sign at 0 will be the sign of the constant coefficient, whereas the polynomial's sign for sufficiently large inputs will be the sign of the leading coefficient.

Moreover, we have shown before that in a list with an odd number of sign changes and non-zero initial and last coefficients, the initial coefficient and the last coefficient have opposite and non-zero signs. Then, the polynomial obviously has a positive root.

private lemma *odd-coeff-sign-changes-imp-pos-roots-aux*:

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

assumes *odd* (*coeff-sign-changes* p)

obtains x **where** $x > 0$ *poly* $p \ x = 0$

proof –

from *assms* **have** [*simp*]: $p \neq 0$ **by** *auto*

from $\langle \text{poly } p \ 0 \neq 0 \rangle$ **have** *const-coeff*: $\text{hd } (\text{coeffs } p) \neq 0$ **by** (*induction* p) *auto*

from *assms* **have** $\neg \text{even } (\text{coeff-sign-changes } p)$ **by** *blast*

also have *even* (*coeff-sign-changes* p) $\longleftrightarrow \text{sgn } (\text{hd } (\text{coeffs } p)) = \text{sgn } (\text{last } (\text{coeffs } p))$

using *const-coeff* *assms* **by** (*intro even-sign-changes-iff*) (*auto simp: last-coeffs-not-0*)

also have $\text{last } (\text{coeffs } p) = \text{lead-coeff } p$

by (*simp add: last-coeffs-eq-coeff-degree lead-coeff-def*)

finally have $\text{sgn } (\text{hd } (\text{coeffs } p)) * \text{sgn } (\text{lead-coeff } p) < 0$

using *const-coeff* **by** (*auto simp: sgn-if lead-coeff-def split: if-split-asm*)

also from $\langle p \neq 0 \rangle$ **have** $\text{hd } (\text{coeffs } p) = \text{poly } p \ 0$ **by** (*induction* p) *auto*

finally have $\text{poly } p \ 0 * \text{lead-coeff } p < 0$ **by** (*auto simp: mult-less-0-iff*)

from *pos-root-exI*[*OF this*] **that show** *?thesis* **by** *blast*

qed

We can now show the statement without the restriction to a non-zero constant coefficient. We can do this by simply factoring p into the form $p \cdot x^n$, where n is chosen as large as possible. This corresponds to stripping all initial zeros of the coefficient list, which obviously changes neither the existence of positive roots nor the number of coefficient sign changes.

lemma *odd-coeff-sign-changes-imp-pos-roots*:

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

assumes *odd* (*coeff-sign-changes* p)

obtains x **where** $x > 0$ *poly* $p \ x = 0$

proof –

define s **where** $s = \text{sgn } (\text{lead-coeff } p)$

define n **where** $n = \text{order } 0 \ p$

define r **where** $r = p \ \text{div } [:0, 1:] \wedge n$

have $p: p = [:0, 1:] \wedge n * r$ **unfolding** *r-def n-def*

```

    using order-1[of 0 p] by (simp del: mult-pCons-left)
  from assms p have r-nz: r ≠ 0 by auto

  obtain x where x > 0 poly r x = 0
  proof (rule odd-coeff-sign-changes-imp-pos-roots-aux)
    show r ≠ 0 by fact
    have order 0 p = order 0 p + order 0 r
      by (subst p, insert order-power-n-n[of 0::real n] r-nz)
        (simp del: mult-pCons-left add: order-mult n-def)
    hence order 0 r = 0 by simp
    with r-nz show nz: poly r 0 ≠ 0 by (simp add: order-root)

    note ⟨odd (coeff-sign-changes p)⟩
    also have p = [:0, 1:] ^ n * r by (simp add: p)
    also have [:0, 1:] ^ n = monom 1 n
      by (induction n) (simp-all add: monom-Suc monom-0 one-poly-def)
    also have coeffs (monom 1 n * r) = replicate n 0 @ coeffs r
      by (induction n) (simp-all add: monom-Suc cCons-def r-nz monom-0)
    also have sign-changes ... = coeff-sign-changes r
      by (subst (1 2) sign-changes-filter [symmetric]) simp
    finally show odd (coeff-sign-changes r) .
  qed
  thus ?thesis by (intro that[of x]) (simp-all add: p)
  qed
end

```

1.7 Proof of Descartes' sign rule

For a polynomial $p(X) = a_0 + \dots + a_n X^n$, we have $[X^i](1 - X)p(X) = (\sum_{j=0}^i a_j)$.

```

lemma coeff-poly-times-one-minus-x:
  fixes g :: 'a :: linordered-idom poly
  shows coeff g n = (∑ i ≤ n. coeff (g * [:1, -1:] i) i)
  by (induction n) simp-all

```

We apply the previous lemma to the coefficient list of a polynomial and show: given a polynomial $p(X)$ and $q(X) = (1 - X)p(X)$, the coefficient list of $p(X)$ is the list of partial sums of the coefficient list of $q(X)$.

```

lemma Poly-times-one-minus-x-eq-psums:
  fixes xs :: 'a :: linordered-idom list
  assumes [simp]: length xs = length ys
  assumes Poly xs = Poly ys * [:1, -1:]
  shows ys = psums xs
proof (rule nth-equalityI; safe?)
  fix i assume i: i < length ys
  hence ys ! i = coeff (Poly ys) i

```

```

  by (simp add: nth-default-def)
  also from coeff-poly-times-one-minus-x[of Poly ys i] assms
  have ... = ( $\sum_{j \leq i} \text{coeff } (\text{Poly } xs) j$ ) by simp
  also from i have ... = psums xs ! i
  by (auto simp: nth-default-def psums-nth)
  finally show ys ! i = psums xs ! i .
qed simp-all

```

We can now apply our main lemma on the sign changes in lists to the coefficient lists of a nonzero polynomial $p(X)$ and $(1-X)p(X)$: the difference of the changes in the coefficient lists is odd and positive.

```

lemma sign-changes-poly-times-one-minus-x:
  fixes g :: 'a :: linordered-idom poly and a :: 'a
  assumes nz: g ≠ 0
  defines v ≡ coeff-sign-changes
  shows v ([:1, -1:] * g) - v g > 0 ∧ odd (v ([:1, -1:] * g) - v g)
proof -
  define xs where xs = coeffs ([:1, -1:] * g)
  define ys where ys = coeffs g @ [0]
  have ys: ys = psums xs
  proof (rule Poly-times-one-minus-x-eq-psums)
    show length xs = length ys unfolding xs-def ys-def
    by (simp add: length-coeffs nz degree-mult-eq no-zero-divisors del: mult-pCons-left)
    show Poly xs = Poly ys * [:1, - 1:] unfolding xs-def ys-def
    by (simp only: Poly-snoc Poly-coeffs) simp
  qed
  have sign-changes (psums xs) < sign-changes xs ∧
    odd (sign-changes xs - sign-changes (psums xs))
  proof (rule arthan)
    show xs ≠ [] by (auto simp: xs-def nz simp del: mult-pCons-left)
    thus sum-list xs = 0 by (simp add: last-psums [symmetric] ys [symmetric]
ys-def)
  qed (auto simp: last-coeffs-not-0 xs-def nz simp del: mult-pCons-left)
  with ys have sign-changes ys < sign-changes xs ∧
    odd (sign-changes xs - sign-changes ys) by simp
  also have sign-changes xs = v ([:1, -1:] * g) unfolding v-def
  by (intro sign-changes-coeff-sign-changes) (simp-all add: xs-def)
  also have sign-changes ys = v g unfolding v-def
  by (intro sign-changes-coeff-sign-changes) (simp-all add: ys-def Poly-snoc)
  finally show ?thesis by simp
qed

```

We can now lift the previous lemma to the case of $p(X)$ and $(a-X)p(X)$ by substituting X with aX , yielding the polynomials $p(aX)$ and $a \cdot (1-X) \cdot p(aX)$.

```

lemma sign-changes-poly-times-root-minus-x:
  fixes g :: 'a :: linordered-idom poly and a :: 'a
  assumes nz: g ≠ 0 and pos: a > 0
  defines v ≡ coeff-sign-changes

```



```

shows  $v ([:a, -1:] * g) - v g > 0 \wedge \text{odd} (v ([:a, -1:] * g) - v g)$ 
proof -
  have  $0 < v ([:1, -1:] * \text{reduce-root } a g) - v (\text{reduce-root } a g) \wedge$ 
     $\text{odd} (v ([:1, -1:] * \text{reduce-root } a g) - v (\text{reduce-root } a g))$ 
    using nz pos unfolding v-def by (intro sign-changes-poly-times-one-minus-x)
simp-all
  also have  $v ([:1, -1:] * \text{reduce-root } a g) = v (\text{smult } a ([:1, -1:] * \text{reduce-root } a g))$ 
    unfolding v-def by (simp add: coeff-sign-changes-smult pos)
  also have  $\text{smult } a ([:1, -1:] * \text{reduce-root } a g) = [:a:] * [:1, -1:] * \text{reduce-root } a g$ 
    by (subst mult.assoc simp)
  also have  $[:a:] * [:1, -1:] = \text{reduce-root } a [:a, -1:]$ 
    by (simp add: reduce-root-def pcompose-pCons)
  also have  $\dots * \text{reduce-root } a g = \text{reduce-root } a ([:a, -1:] * g)$ 
    unfolding reduce-root-def by (simp only: pcompose-mult)
  also have  $v \dots = v ([:a, -1:] * g)$  by (simp add: v-def coeff-sign-changes-reduce-root pos)
  also have  $v (\text{reduce-root } a g) = v g$  by (simp add: v-def coeff-sign-changes-reduce-root pos)
  finally show ?thesis .
qed

```

Finally, the difference of the number of coefficient sign changes and the number of positive roots is non-negative and even. This follows straightforwardly by induction over the roots.

lemma *descartes-sign-rule-aux:*

```

fixes  $p :: \text{real poly}$ 
assumes  $p \neq 0$ 
shows  $\text{coeff-sign-changes } p \geq \text{count-pos-roots } p \wedge$ 
   $\text{even} (\text{coeff-sign-changes } p - \text{count-pos-roots } p)$ 
using assms
proof (induction p rule: poly-root-induct[where P =  $\lambda a. a > 0$ ])
  case (root a p)
    define  $q$  where  $q = [:a, -1:] * p$ 
    from root.prems have  $p \neq 0$  by auto
    with root p sign-changes-poly-times-root-minus-x[of p a]
      count-roots-with-times-root[of p  $\lambda x. x > 0$  a] show ?case by (fold q-def)
fastforce
  next
    case (no-roots p)
    from no-roots have  $\text{pos-roots } p = \{\}$  by (auto simp: roots-with-def)
    hence [simp]:  $\text{count-pos-roots } p = 0$  by (simp add: count-roots-with-def)
    thus ?case using no-roots ( $p \neq 0$ ) odd-coeff-sign-changes-imp-pos-roots[of p]
      by (auto simp: roots-with-def)
qed simp-all

```

The main theorem is then an obvious consequence

theorem *descartes-sign-rule:*

```

fixes  $p :: \text{real poly}$ 
assumes  $p \neq 0$ 
shows  $\exists d. \text{even } d \wedge \text{coeff-sign-changes } p = \text{count-pos-roots } p + d$ 
proof
  define  $d$  where  $d = \text{coeff-sign-changes } p - \text{count-pos-roots } p$ 
  show  $\text{even } d \wedge \text{coeff-sign-changes } p = \text{count-pos-roots } p + d$ 
    unfolding  $d\text{-def}$  using  $\text{descartes-sign-rule-aux}[OF \text{ assms}]$  by auto
qed

end

```

References

- [1] R. D. Arthan. Descartes' rule of signs by an easy induction. 2007.