Descartes' Rule of Signs

Manuel Eberl

March 17, 2025

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
  HOL−Computational-Algebra.Polynomial
begin

lemma *op-plus-0*: $((+)\ (0 :: 'a :: monoid\text{-}add)) = id$
  by *auto*

lemma *filter-dropWhile*:
  filter $(\lambda x.\ \neg P\ x)$ (*dropWhile* $P\ xs$) = *filter* $(\lambda x.\ \neg P\ x)\ xs$
  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 poly p 0 * lead-coeff p < (0 :: real)
  obtains x where x > 0 poly p x = 0
proof −
  have P: ∃x>0. poly p x = (0::real) if lead-coeff p > 0 poly p 0 < 0 for p
  proof −
    note that(1)
    also from poly-pinfty-gt-lc[OF ‹lead-coeff p > 0›] obtain x0
      where ⋀x. x ≥ x0 ⟹ poly p x ≥ lead-coeff p by auto
    hence poly p (max x0 1) ≥ lead-coeff p by auto
    finally have poly p (max x0 1) > 0 .
    with that have ∃x. x > 0 ∧ x < max x0 1 ∧ poly p x = 0
      by (intro poly-IVT mult-neg-pos) auto
    thus ∃x>0. poly p x = 0 by auto
  qed

  show ?thesis
  proof (cases lead-coeff p > 0)
    case True
    with assms have poly p 0 < 0
      by (auto simp: mult-less-0-iff)
    from P[OF True this] that show ?thesis
      by blast
  next
    case False
    from False assms have poly (−p) 0 < 0
      by (auto simp: mult-less-0-iff)
    moreover from assms have p ≠ 0
      by auto
    with False have lead-coeff (−p) > 0
      by (cases rule: linorder-cases[of lead-coeff p 0])
         (simp-all add:)
    ultimately show ?thesis using that P[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
  reduce-root a p = pcompose p [:0, a:]

lemma reduce-root-pCons:
  reduce-root a (pCons c p) = pCons c (smult a (reduce-root a p))
  by (simp add: reduce-root-def pcompose-pCons)
```

```
lemma reduce-root-nonzero [simp]:
  a ≠ 0 ⟹ p ≠ 0 ⟹ reduce-root a p ≠ (0 :: 'a :: idom 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 $\sum_{j=0}^{i} \mathrm{xs}_i$.

```
fun psums where
  psums [] = []
| psums [x] = [x]
| psums (x#y#xs) = x # psums ((x+y) # xs)

lemma length-psums [simp]: length (psums xs) = length xs
  by (induction xs rule: psums.induct) simp-all

lemma psums-Cons:
  psums (x#xs) = (x :: 'a :: semigroup-add) # map ((+) x) (psums xs)
  by (induction xs rule: psums.induct) (simp-all add: algebra-simps)

lemma last-psums:
  (xs :: 'a :: monoid-add list) ≠ [] ⟹ last (psums xs) = sum-list xs
  by (induction xs rule: psums.induct)
     (auto simp add: add.assoc [symmetric] psums-Cons o-def)

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

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

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

lemma psums-nth: n < length xs ⟹ psums xs ! n = (∑ i≤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 psums (x # y # xs) ! n = psums ((x+y) # xs) ! m by simp
    also from rec.prems Suc have ... = (∑ i≤m. ((x+y) # xs) ! i)
      by (intro rec.IH) simp-all
```

also have ... = $x + y + (\sum i{=}1..m.\ (y\#xs)\ !\ i)$
  by (*auto simp*: *atLeast0AtMost* [*symmetric*] *sum.atLeast-Suc-atMost*[*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** $x + y + \ldots = (\sum i{\le}n.\ (x\#y\#xs)\ !\ i)$
  by (*auto simp*: *atLeast0AtMost* [*symmetric*] *sum.atLeast-Suc-atMost 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*) ⇒ *nat* **where**
  sign-changes xs = *length* (*remdups-adj* (*filter* (λ*x*. *x* ≠ *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*) ≥ *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* ∧ *sgn y* = *1* ∨ *sgn x* = *1* ∧ *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 $x * y > 0 \Longrightarrow$ *sign-changes* $(x \# y \# xs)$ = *sign-changes* $(y \# xs)$
by (*subst* (*asm*) *zero-less-mult-iff*) (*fastforce simp*: *sign-changes-def*)

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

lemma *sign-changes-two*:
fixes $x\ y :: 'a ::$ *linordered-idom*
shows *sign-changes* $[x,y]$ =
(*if* $x > 0 \wedge y < 0 \vee x < 0 \wedge y > 0$ *then 1 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 $P\ []\ \bigwedge x.\ P\ [x]\ \bigwedge xs.\ P\ xs \Longrightarrow P\ (0\#xs)$
$\bigwedge x\ y\ xs.\ x \neq 0 \Longrightarrow P\ ((x + y) \# xs) \Longrightarrow P\ (x \# y \# xs)$
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 $xs :: 'a ::$ *linordered-idom list*
shows *sign-changes* (*filter* $(\lambda x.\ x \neq 0)$ *xs*) = *sign-changes xs*
by (*simp add*: *sign-changes-def filter-map o-def sgn-0-0*)

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

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

```
  also have length ... − 1 = sign-changes xs by (simp add: sign-changes-def)
  finally show ?thesis .
qed

lemma sign-changes-replicate: sign-changes (replicate n x) = 0
  by (simp add: sign-changes-def remdups-adj-replicate filter-replicate)

lemma sign-changes-decompose:
  assumes x ≠ (0 :: 'a :: linordered-idom)
  shows   sign-changes (xs @ x # ys) =
            sign-changes (xs @ [x]) + sign-changes (x # ys)
proof −
  have sign-changes (xs @ x # ys) =
           length (remdups-adj ([x←map sgn xs . x ≠ 0] @
                     sgn x # [x←map sgn ys . x ≠ 0])) − 1
    by (simp add: sgn-0-0 assms sign-changes-def)
  also have ... = sign-changes (xs @ [x]) + 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 ≠ ([] :: 'a :: linordered-idom list) hd xs ≠ 0 last xs ≠ 0
  shows   even (sign-changes xs) ⟷ sgn (hd xs) = sgn (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 = this
    show ?thesis
    proof (cases xs')
      case (Cons y xs'')
      note y = 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** *sgn x* = − *sgn y* **by** (*auto simp*: *mult-less-0-iff*)
    moreover have *even* (*sign-changes* (*y* # *xs''*)) ⟷
              sgn (*hd* (*y* # *xs''*)) = *sgn* (*last* (*y* # *xs''*))
     using *xy less.prems* **by** (*intro less*) (*auto simp*: *x y*)
    moreover from *xy less.prems*
     have *sgn y* = *sgn* (*last xs*) ⟷ −*sgn y* ≠ *sgn* (*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* :: *linordered-idom list*
 assumes *xs* ≠ [] *last xs* ≠ *0 x* + *y* + *sum-list xs* = *0*
 assumes ⋀*x y xs*. *xs* ≠ [] ⟹ *last xs* ≠ *0* ⟹
      x + *y* + *sum-list xs* = *0* ⟹ *x* ≥ *0* ⟹ *P x y xs*
 assumes ⋀*x y xs*. *xs* ≠ [] ⟹ *P x y xs* ⟹ *P* (−*x*) (−*y*) (*map uminus xs*)
 shows *P x y xs*
proof (*cases x* ≥ *0*)
 assume *x*: ¬(*x* ≥ *0*)
 from *assms* **have** *map uminus xs* ≠ [] **by** *simp*
 moreover from *x assms*(*1*,*2*,*3*) **have***P* (−*x*) (−*y*) (*map uminus xs*)
  using *uminus-sum-list-map*[*of λx. 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*)) (*map uminus* (*map 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* :: {*linordered-idom*} *list*
 assumes *xs* ≠ [] *last xs* ≠ *0 x* + *y* + *sum-list xs* = *0*
 defines *v* ≡ *λxs. int* (*sign-changes xs*)
 shows *v* (*x* # *y* # *xs*) − *v* ((*x* + *y*) # *xs*) ≥
     v (*psums* (*x* # *y* # *xs*)) − *v* (*psums* ((*x* + *y*) # *xs*)) ∧
   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 ((+) (xs!p)) (psums xs2)) for xs'
  proof -
    fix xs' :: 'a list
    have sign-changes (xs' @ psums xs) =
           sign-changes (xs' @ xs ! p # map ((+) (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 ((+) (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 ¬(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 x 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 (λt. t + (x + y)) = ((+) (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 ¬(*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 $(\lambda t.\ t + (x + y)) = ((+)\ (x + y))$ **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 ¬(*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 $(\lambda t.\ t + (x + y)) = ((+)\ (x + y))$ **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 α where α = int (sign-changes (x # y # xs)) − int (sign-changes ((x
+ y) # xs))
   define β where β = int (sign-changes (psums (x # y # xs))) − int (sign-changes
(psums ((x+y) # xs)))
    from nonzero False have α ≥ β ∧ even (α − β) unfolding α-def β-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 ∧ 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 = (∑ x∈roots-with P p. order x p)

abbreviation pos-roots ≡ roots-with (λx. x > 0)
abbreviation count-pos-roots ≡ count-roots-with (λx. x > 0)
```

```
lemma finite-roots-with [simp]:
  (p :: 'a :: linordered-idom poly) ≠ 0 ⟹ finite (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 ≠ 0 P (a :: 'a :: linordered-idom)
  shows  count-roots-with P ([:a, −1:] ∗ p) = Suc (count-roots-with P p)
proof −
  define q where q = [:a, −1:] ∗ p
  from assms have a: a ∈ roots-with P q by (simp-all add: roots-with-def q-def)
   have q-nz: q ≠ 0 unfolding q-def by (rule no-zero-divisors) (simp-all add: assms)

   have count-roots-with P q = (∑x∈roots-with P q. order x q) by (simp add: count-roots-with-def)
  also from a q-nz have . . . = order a q + (∑x∈roots-with P q − {a}. order x q)
    by (subst sum.remove) simp-all
  also have order a q = order a [:a, −1:] + order a p unfolding q-def
    by (subst order-mult[OF no-zero-divisors]) (simp-all add: assms)
  also have 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 (∑x∈roots-with P q − {a}. order x q) = (∑x∈roots-with P q − {a}. order x p)
  proof (intro sum.cong refl)
    fix x assume x: x ∈ roots-with P q − {a}
    from assms have order x q = order x [:a, −1:] + order x p unfolding q-def
      by (subst order-mult[OF no-zero-divisors]) (simp-all add: assms)
    also from x have order x [:a, −1:] = 0 by (intro order-0I) simp-all
    finally show order x q = order x p by simp
  qed
  also from a q-nz have 1 + order a p + (∑x∈roots-with P q − {a}. order x p) =
                         1 + (∑x∈roots-with P q. order x p)
    by (subst add.assoc, subst sum.remove[symmetric]) simp-all
  also from q-nz have (∑x∈roots-with P q. order x p) = (∑x∈roots-with P p. order x p)
  proof (intro sum.mono-neutral-right)
    show roots-with P p ⊆ roots-with P q
      by (auto simp: roots-with-def q-def simp del: mult-pCons-left)
    show ∀x∈roots-with P q − roots-with P p. 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*) *coeff-sign-changes f* ≡ *sign-changes* (*coeffs f*)

We first show that when building a polynomial from a coefficient list, the coefficient sign 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 *Poly xs* = (*p* :: *'a* :: *linordered-idom poly*)
  shows *sign-changes xs* = *coeff-sign-changes p*
proof −
  have *coeffs p* = *coeffs* (*Poly xs*) **by** (*subst assms*) (*rule refl*)
  also have ... = *strip-while* ((=) *0*) *xs* **by** *simp*
  also have *filter* ((≠) *0*) ... = *filter* ((≠) *0*) *xs* **unfolding** *strip-while-def o-def*
   by (*subst rev-filter* [*symmetric*], *subst filter-dropWhile*) (*simp-all add*: *rev-filter*)
  also have *sign-changes* ... = *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* :: *linordered-idom*)
  shows *coeff-sign-changes* (*reduce-root a p*) = *coeff-sign-changes p*
proof (*intro sign-changes-cong*, *induction p*)
  case (*pCons c p*)
  have *map sgn* (*coeffs* (*reduce-root a* (*pCons c p*))) =
      cCons (*sgn c*) (*map sgn* (*coeffs* (*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 *cCons* (*sgn c*) (*map sgn* (*coeffs p*)) = *map sgn* (*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* :: *linordered-idom*)
  shows *coeff-sign-changes* (*smult a p*) = *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 [*simp*]: *p* ≠ (*0* :: *real poly*) *poly p 0* ≠ *0*
  assumes *odd* (*coeff-sign-changes p*)
  obtains *x* **where** *x* > *0 poly p x* = *0*
proof −
  from ‹*poly p 0* ≠ *0*›
  have [*simp*]: *hd* (*coeffs p*) ≠ *0*
    by (*induct p*) *auto*
  from *assms* **have** ¬ *even* (*coeff-sign-changes p*)
    by *blast*
  also have *even* (*coeff-sign-changes p*) ⟷ *sgn* (*hd* (*coeffs p*)) = *sgn* (*lead-coeff p*)
    by (*auto simp add*: *even-sign-changes-iff last-coeffs-eq-coeff-degree*)
  finally have *sgn* (*hd* (*coeffs p*)) ∗ *sgn* (*lead-coeff p*) < *0*
    by (*auto simp*: *sgn-if split*: *if-split-asm*)
  also from ‹*p* ≠ *0*› **have** *hd* (*coeffs p*) = *poly p 0* **by** (*induction p*) *auto*
  finally have *poly p 0* ∗ *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* ≠ (*0* :: *real poly*)
  assumes *odd* (*coeff-sign-changes p*)
  obtains *x* **where** *x* > *0 poly p x* = *0*
proof −
  define *s* **where** *s* = *sgn* (*lead-coeff p*)
  define *n* **where** *n* = *order 0 p*
  define *r* **where** *r* = *p div* [:*0*, *1*:] ^ *n*
  have *p*: *p* = [:*0*, *1*:] ^ *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)
    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 $p(X) = a_0 + \ldots + a_n X^n$, 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)
  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*≤*i*. *coeff* (*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*)
then show *sum-list xs* = *0* **by** (*simp add*: *last-psums* [*symmetric*] *ys* [*symmetric*] *ys-def*)
show *last xs* ≠ *0*
by (*auto simp*: *xs-def nz last-coeffs-eq-coeff-degree simp del*: *mult-pCons-left*)
qed
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*: $g \neq 0$ **and** *pos*: $a > 0$
defines $v \equiv$ *coeff-sign-changes*
shows v ([:a, -1:] $*$ g) $-$ v g $>$ 0 $\wedge$ *odd* (v ([:a, -1:] $*$ g) $-$ v g)
proof $-$
have $0 < v$ ([:1, $-$ 1:] $*$ *reduce-root a g*) $-$ v (*reduce-root a g*) $\wedge$
odd (v ([:1, $-$ 1:] $*$ *reduce-root a g*) $-$ v (*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:] $*$ *reduce-root a g*) $=$ v (*smult a* ([:1, -1:] $*$ *reduce-root a g*))
unfolding *v-def* **by** (*simp add*: *coeff-sign-changes-smult pos*)
also have *smult a* ([:1, -1:] $*$ *reduce-root a g*) $=$ [:a:] $*$ [:1, -1:] $*$ *reduce-root a g*
by (*subst mult.assoc*) *simp*
also have [:a:] $*$ [:1, -1:] $=$ *reduce-root a* [:a, -1:]
by (*simp add*: *reduce-root-def pcompose-pCons*)
also have ... $*$ *reduce-root a g* $=$ *reduce-root a* ([:a, -1:] $*$ g)
unfolding *reduce-root-def* **by** (*simp only*: *pcompose-mult*)
also have v ... $=$ v ([:a, -1:] $*$ g) **by** (*simp add*: *v-def coeff-sign-changes-reduce-root pos*)
also have v (*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 :: *real poly*
assumes $p \neq 0$
shows *coeff-sign-changes p* $\geq$ *count-pos-roots p* $\wedge$
even (*coeff-sign-changes p* $-$ *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: $p \neq 0$ **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** *pos-roots p* $=$ {} **by** (*auto simp*: *roots-with-def*)
hence [*simp*]: *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 :: *real poly*
  assumes $p \neq 0$
  shows $\exists\, d.$ *even* $d \wedge$ *coeff-sign-changes* $p =$ *count-pos-roots* $p + d$
proof
  define d **where** $d =$ *coeff-sign-changes* $p -$ *count-pos-roots* p
  show *even* $d \wedge$ *coeff-sign-changes* $p =$ *count-pos-roots* $p + d$
    unfolding *d-def* **using** *descartes-sign-rule-aux*[*OF assms*] **by** *auto*
qed

end

References

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