Descartes’ Rule of Signs

Manuel Eberl

September 13, 2023

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 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-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
1.1 Polynomials

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

**lemma pos-root-exI:**

assumes \( p 0 \ast \text{lead-coeff } p < (0 :: \text{real}) \)

obtains \( x \) where \( x > 0 \) poly \( p x = 0 \)

**proof**

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

**proof**

  - note that \((1)\) also from \( \text{poly-pinfty-gt-le} [OF \ (\text{lead-coeff } p > 0)] \) obtain \( x0 \)
    where \( \land x. \ x \geq x0 \longrightarrow \) 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 \( \exists x. \ x > 0 \land x < \text{max } x0 1 \land \) poly \( p x = 0 \)
  by \((\text{intro poly-IVT mult-neg-pos})\) auto
  
  thus \( \exists x > 0. \) poly \( p x = 0 \) by auto

qed

**show \( ?\text{thesis} \)**

**proof** \((\text{cases lead-coeff } p > 0)\)

  case True
  
  with assms have \( \text{poly } p 0 < 0 \)
  by \((\text{auto simp: mult-less-0-iff})\)
  from \( P[\text{OF True this}] \) that show \( ?\text{thesis} \)
  by blast

next

  case False
  
  from False assms have \( \text{poly } (-p) 0 < 0 \)
  by \((\text{auto simp: mult-less-0-iff})\)

  moreover from assms have \( p \neq 0 \)
  by auto

  with False have \( \text{lead-coeff } (-p) > 0 \)
  by \((\text{cases rule: linorder-cases[of lead-coeff } p 0])\)
  \((\text{simp-all add:})\)

  ultimately show \( ?\text{thesis using that } P[\text{of } -p] \) 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**

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 \((\text{simp add: reduce-root-def pcompose-pCons})\)
lemma reduce-root-nonzero [simp]:
\[ a \neq 0 \implies p \neq 0 \implies \text{reduce-root} \ a\ p \neq (0 :: 'a :: idom poly) \]

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 \( \text{psums} \ \text{xs} \) has as its \( i \)-th entry \( \sum_{j=0}^{i} x_{i} \).

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

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

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

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

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

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

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

lemma psums-nth: \( n < \text{length} \ \text{xs} \implies \text{psums} \ \text{xs} ! n = \sum_{i=0}^{n} x_{i} \)

proof (induction \text{xs} arbitrary: \text{n} rule: psums.induct[case-names Nil sng rec])

case (rec \text{x}_{y} \text{xs} \text{n})
show ?case
proof (cases \text{n})

case (Suc \text{m})
from Suc have \( \text{psums} \ (x \# y \# \text{xs}) ! n = \text{psums} \ ((x+y) \# \text{xs}) ! m \)
by simp
also from rec.prems Suc have \( \ldots = \sum_{i=m}^{n} ((x+y) \# \text{xs}) ! i \)
by (intro rec.IH) simp-all
also have \( \ldots = x + y + (\sum_{i=1..m} (y\#xs)!i) \)
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 :: \((\alpha :: \{sgn,zero\} list) \Rightarrow \text{nat}\) 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) \ge sign-changes xs
unfolding sign-changes-def by (simp add: remdups-adj-Cons split: list.split)

lemma sign-changes-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 \land sgn y = 1 \lor sgn x = 1 \land sgn y = -1
by (auto simp: mult-less-0-iff)
thus \(?thesis by \(\text{fastforce simp: sign-changes-def}\)
qed

lemma sign-changes-Cons-same:
fixes x y :: \('a :: linordered-idom\)
shows \( x \cdot y > 0 \implies \text{sign-changes} (x \# y \# xs) = \text{sign-changes} (y \# xs) \)
by (subst (asm) zero-less-mult-iff) (fastforce simp: sign-changes-def)

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

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

lemma sign-changes-induct [case-names nil sing zero nonzero]:
\[ \begin{align*}
P [] \land x. P [x] \land xs. P xs & \implies P (0\#xs) \\
\land x y xs. x \neq 0 & \implies P ((x + y) \# xs) \implies P (x \# y \# xs)
\end{align*} \]
shows \( P xs \)
proof (induction length xs arbitrary; xs rule: less-induct)
\[ \begin{align*}
\text{case (less zs)} & \\
\text{show ?case} & \\
\text{proof (cases xs rule: psums.cases)} & \\
\text{fix } x y xs' \text{ assume } xs = x \# y \# xs' & \\
\text{with assms less show ?thesis} & \text{by (cases } x = 0 \text{) auto}
\end{align*} \]
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 (unfolding \text{sign-changes-def} ..)
\[ \begin{align*}
\text{have } \text{sign-changes} (\text{map} \text{uminus} xs) = & \\
\text{length} (\text{remdups-adj} [x\leftarrow \text{map} \text{sgn} (\text{map} \text{uminus} xs). x \neq 0]) - 1 & \\
\text{unfolding } \text{sign-changes-def} & ..
\end{align*} \]
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 \( \text{remdups-adj} (\text{filter} (\lambda x. x \neq 0) \ldots) = \\
\text{map} \text{uminus} (\text{remdups-adj} (\text{filter} (\lambda x. x \neq 0) (\text{map} \text{sgn} xs))) \)
by (subst filter-map, subst remdups-adj-map-injective)
(simp-all add: o-def)
also have \( \text{length} \ldots - 1 = \text{sign-changes} \; \text{xs} \) by (simp add: sign-changes-def)

finally show \(?\text{thesis}\).  

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 \( x \neq (0 :: \text{a :: linordered-idom}) \)

shows \( \text{sign-changes} \; (\text{xs} @ x @ \text{ys}) = \text{sign-changes} \; (\text{xs} @ [x]) + \text{sign-changes} \; (x @ \text{ys}) \)

proof –

have \( \text{sign-changes} \; (\text{xs} @ x @ \text{ys}) = \text{length} \; (\text{remdups-adj} \; ([\text{x} \leftarrow \text{map} \; \text{sgn} \; \text{xs} . \; \text{x} \neq 0] @ \text{sgn} \; x @ [\text{x} \leftarrow \text{map} \; \text{sgn} \; \text{ys} . \; \text{x} \neq 0])) - 1 \)

by (simp add: sgn-0-0 assms sign-changes-def)

also have \( \ldots = \text{sign-changes} \; (\text{xs} @ [x]) + \text{sign-changes} \; (x @ \text{ys}) \)

by (subst remdups-adj-append) (simp add: sign-changes-def assms sgn-0-0)

finally show \(?\text{thesis}\).  

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 \( \text{xs} \neq ([]) :: \text{a :: linordered-idom list} \; \text{hd} \; \text{xs} \neq 0 \; \text{last} \; \text{xs} \neq 0 \)

shows \( \text{even} \; (\text{sign-changes} \; \text{xs}) \iff \text{sgn} \; (\text{hd} \; \text{xs}) = \text{sgn} \; (\text{last} \; \text{xs}) \)

using assms

proof (induction length \text{xs} arbitrary: \text{xs} rule: less-induct)

case (less \text{xs})

show \(?\text{case}\)

proof (cases \text{xs})

case (Cons \text{x} \text{xs}'

note \text{x} = \this

show \(?\text{thesis}\)

proof (cases \text{xs}')

case (Cons \text{y} \text{xs}'')

note \text{y} = \this

show \(?\text{thesis}\)

proof (rule linorder-cases[of \text{x} \text{y} 0])

assume \( \text{xy} : \text{x} \cdot \text{y} = 0 \)

with \( \text{x} \cdot \text{y} \; \text{less}(1,3,4) \) show \(?\text{thesis}\) by (auto simp: sign-changes-Cons-Cons-0)

next

assume \( \text{xy} : \text{x} \cdot \text{y} > 0 \)

with \( \text{less}(1,4) \) show \(?\text{thesis}\)

by (auto simp add: \text{xy} \; \text{sign-changes-Cons-Cons-same} \; \text{zero-less-mult-iff})

next

assume \( \text{xy} : \text{x} \cdot \text{y} < 0 \)
moreover from \( xy \) have \( \text{sgn} \ x = - \text{sgn} \ y \) by \((\text{auto simp: mult-less-0-iff})\)
moreover have \( \text{even} \ (\text{sign-changes} \ (y \# \ x s''\)) \iff \text{sgn} \ (hd \ (y \# \ x s'')) = \text{sgn} \ (\text{last} \ (y \# \ x s'')) \)
  using \( xy \ less\) \( \text{prems} \) by \((\text{intro less})\) \((\text{auto simp: x y})\)
moreover from \( xy \ less\) \( \text{prems} \)

have \( \text{sgn} \ y = \text{sgn} \ (\text{last} \ xs) \iff -\text{sgn} \ y \neq \text{sgn} \ (\text{last} \ xs) \)
by \((\text{auto simp: sgn-if})\)
ultimately show \( \text{thesis} \) by \((\text{auto simp: sign-changes-Cons-Cons-different x y})\)

qed

We now show that the \( \alpha \) and \( \beta \) in Arthan’s proof have the necessary properties: their difference is non-negative and even.

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

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 \( \text{arthan-wlog} \) [consumes 3, case-names nonneg lift]:
  fixes \( xs :: 'a :: \{\text{linordered-idom}\} \) \( \text{list} \)
  assumes \( xs \neq [] \) \( \text{last} \ xs \neq 0 x + y + \text{sum-list} \ xs = 0 \)
  defines \( v \equiv \lambda x s. \text{int} \ (\text{sign-changes} \ xs) \)
  shows \( v \ (x \# y \# xs) - v \ ((x + y) \# xs) \geq \)
    \( v \ (\text{psums} \ (x \# y \# xs)) - v \ (\text{psums} \ ((x + y) \# xs)) \land \)
    \( \text{even} \ (v \ (x \# y \# xs) - v \ ((x + y) \# xs)) - \)

We now show that the \( \alpha \) and \( \beta \) in Arthan’s proof have the necessary properties: their difference is non-negative and even.
\[(v \, (\text{psums} \, (x \# y \# xs))) - v \, (\text{psums} \, ((x + y) \# xs)))\]

using assms\{(1 - 3)\}

proof (induction rule: arthan-wlog)

have uminus-v: \(v \, (\text{map} \, \text{uminus} \, xs) = v \, x s\) for \(xs\) by (simp add: v-def sign-changes-uminus)

case (lift \(x\, y\) \(xs\))

note lift \(2\)

also have \(v \, (\text{psums} \, (x\#y\#xs)) - v \, (\text{psums} \, ((x+y)\#xs)) = v \, (\text{psums} \, (- x \# - y \# \text{map} \, \text{uminus} \, xs)) - v \, (\text{psums} \, ((- x + - y) \# \text{map} \, \text{uminus} \, xs))\)

by (subst \(1\, 2\) uminus-v [symmetric]) (simp add: map-uminus-psums)

also have \(v \, (x \# y \# xs) - v \, ((x + y) \# xs) = v \, (- x \# - y \# \text{map} \, \text{uminus} \, xs) - v \, ((- x + - y) \# \text{map} \, \text{uminus} \, xs)\)

by (subst \(1\, 2\) uminus-v [symmetric]) simp

finally show \(?\)thesis .

next

case (nonneg \(x\, y\) \(xs\))

define \(p\) where \(p = (LEAST \, n. \, xs \, ! \, n \neq 0)\)

define \(xs1\) :: \(\text{a list} \, \text{where} \, xs1 = \text{replicate} \, p \, 0\)

define \(xs2\) where \(xs2 = \text{drop} \, (\text{Suc} \, p) \, xs\)

from nonneg have \(xs \, ! \, (\text{length} \, xs - 1) \neq 0\) by (simp add: last-conv-nth)

hence \(p \, \text{nz}: \, xs \, ! \, p \neq 0\) unfolding p-def by (rule LeastI)

\{
fix \(q\) assume \(q < p\) hence \(xs \, ! \, q = 0\)

using Least-le[of \(\lambda n. \, x s \, ! n \neq 0\)] unfolding p-def by force
\}

note less-p-zero = this

from Least-le[of \(\lambda n. \, x s \, ! n \neq 0\) \(\text{length} \, xs - 1\)] nonneg

have \(p \leq \text{length} \, xs - 1\) unfolding p-def by (auto simp: last-conv-nth)

with nonneg have \(p \, \text{less-length}: \, p < \text{length} \, xs\) by (cases \(xs\)) simp-all

from p-less-length less-p-zero have take \(p\, xs = \text{replicate} \, p \, 0\)

by (subst list-eq-iff-nth-eq) auto

with p-less-length have \(xs\)-decompose: \(xs = xs1 \oplus xs \, ! \, p \neq xs2\)

unfolding \(xs1\)-def \(xs2\)-def

by (subst append-take-drop-id [of \(p\), symmetric], subst Cons-nth-drop-Suc) simp-all

have \(v\)-decompose: \(v \, (xs' \oplus xs) = v \, (xs' \oplus [x s \, ! p]) + v \, (xs \, ! p \neq xs2)\) for \(xs'\)

proof

have \(xs' \oplus xs = (xs' \oplus xs1) \oplus xs \, ! \, p \neq xs2\) by (subst \(xs\)-decompose) simp

also have \(v \ldots = v \, (xs' \oplus [x s \, ! p]) + v \, (xs \, ! p \neq xs2)\) 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 \(\text{psums-decompose}: \, \text{psums} \, xs = \text{replicate} \, p \, 0 \oplus \text{psums} \, (xs \, ! p \neq xs2)\)

by (subst \(xs\)-decompose) (simp add: \(xs1\)-def \(\text{psums-repeat-0-append}\))

have \(v\)-\(\text{psums-decompose}: \, \text{sign-changes} \, (xs' \oplus \text{psums} \, xs) = \text{sign-changes} \, (xs' \oplus \text{psums} \, xs)\)
proof

fix \(xs'::'a\) list

have \(\text{sign-changes} (xs' @ psums \text{xs}) = \)

\(= \text{sign-changes} (xs' @ psums \text{xs} \# (xs \oplus \text{map}) (psums \text{xs}2)) \)

by (subst psums-decompose, subst (1 2) \text{sign-changes-filter} [\text{symmetric}])

(simp-all add: psums-Cons)

also have \(\ldots = \text{sign-changes} (xs' @ [xs \oplus p]) + \)

\(= \text{sign-changes} (xs \oplus p \# (xs \oplus \text{map}) (psums \text{xs}2)) \)

by (subst \text{sign-changes-decompose}[OF p-nz]) simp-all

finally show \(\text{sign-changes} (xs' @ psums \text{xs}) = \ldots \).

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 \(y \cdot 0\)])

assume \(y: y > 0\)

from \(x\) and this have \(xy: x + y > 0\) by (rule add-pos-pos)

with \(y\) have \(\text{sign-changes} ((x + y) \# \text{xs}) = \text{sign-changes} (y \# \text{xs})\)

by (intro \text{sign-changes-cong}) auto

moreover have \(\text{sign-changes} (x \# \text{psums} ((x + y) \# \text{xs})) = \)

\(= \text{sign-changes} (\text{psums} ((x+y) \# \text{xs}))\)

using \(xy\) by (subst (1 2) \text{psums-Cons}) (simp-all add: \text{sign-changes-Cons-Cons-same})

ultimately show \(?thesis\) using \(x y\)

by (simp add: v-def \text{algebra-simps} \text{sign-changes-Cons-Cons-same})

next

assume \(y: y = 0\)

with \(x\) show \(?thesis\)

by (simp add: v-def \text{sign-changes-Cons-Cons-0} \text{psums-Cons}

\(\text{o-def} \text{sign-changes-Cons-Cons-same})\)

next

assume \(y: y < 0\)

with \(x\) have \(\text{different}: x \cdot y < 0\) by (rule mult-pos-neg)

show \(?thesis\)

proof (rule linorder-cases[of \(x + y \cdot 0\)])

assume \(xy: x + y < 0\)

with \(x\) have \(\text{different'}: x \cdot (x + y) < 0\) by (rule mult-pos-neg)

have \((\lambda t. t + (x + y)) = ((+) (x + y))\) by (rule ext) simp

moreover from \(y\) \(xy\) have \(\text{sign-changes} ((x+y) \# \text{xs}) = \text{sign-changes} (y \# \text{xs})\)

by (intro \text{sign-changes-cong}) auto

ultimately show \(?thesis\) using \(xy\) \(\text{different'}\)

by (simp add: v-def \text{sign-changes-Cons-Cons-different} \text{psums-Cons o-def}
add-ac)
next
assume \(xy: x + y = 0\)
show \(?\text{case}\)
proof (cases \(xs \mid p > 0\))
  assume \(p: xs \mid p > 0\)
  from \(p y\) have different\': \(y * xs \mid p < 0\) by (intro mult-neg-pos)
with v-decompose[of \([x, y]\)] v-decompose[of \([x+y]\)] \(xy p\) different different'
  v-psums-decompose[of \([x]\)] v-psums-decompose[of \([]\)]
show \(?\text{thesis}\) 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 \(-(xs \mid p > 0)\)
with p-nz have \(p: xs \mid p < 0\) by simp
from \(p y\) have same: \(y * xs \mid p > 0\) by (intro mult-neg-neg)
from \(p x\) have different\': \(x * xs \mid p < 0\) 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 \(?\text{thesis}\) 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 \(?\text{case}\)
proof (cases \(xs \mid p > 0\))
  assume \(p: xs \mid p > 0\)
  from \(xy p\) have same\': \((x + y) * xs \mid p > 0\) by (intro mult-pos-pos)
  from \(p y\) have different\': \(y * xs \mid p < 0\) 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'
  show \(?\text{thesis}\) by (auto simp add: algebra-simps v-def psums-Cons o-def
    sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)
next
assume \(-(xs \mid p > 0)\)
with p-nz have \(p: xs \mid p < 0\) by simp
from \(xy p\) have different\': \((x + y) * xs \mid p < 0\) by (rule mult-pos-neg)
from \(y p\) have same\': \(y * xs \mid p > 0\) 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'
  show \(?\text{thesis}\) by (auto simp add: algebra-simps v-def psums-Cons o-def
    sign-changes-Cons-Cons-different sign-changes-Cons-Cons-same)
qed
Now we can prove the main lemma of the proof by induction over the list with our specialised induction rule for \textit{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.

\textbf{lemma} \textit{arthan}:

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

\section*{1.5 Roots of a polynomial with a certain property}

The set of roots of a polynomial \textit{p} that fulfil a given property \textit{P}:

\textbf{definition} \textit{roots-with} \textit{P} \textit{p} = \{ \textit{x}. \textit{P} \textit{x} \land \textit{poly} \textit{p} \textit{x} = 0 \}\n
The number of roots of a polynomial \textit{p} with a given property \textit{P}, where multiple roots are counted multiple times.

\textbf{definition} \textit{count-roots-with} \textit{P} \textit{p} = (\sum \textit{x} \in \textit{roots-with} \textit{P} \textit{p}. \textit{order} \textit{x} \textit{p})

\textbf{abbreviation} \textit{pos-roots} \equiv \textit{roots-with}(λ\textit{x}. \textit{x} > 0)
\textbf{abbreviation} \textit{count-pos-roots} \equiv \textit{count-roots-with}(λ\textit{x}. \textit{x} > 0)
lemma finite-roots-with [simp]:
\((p :: 'a :: linordered_idom poly) \neq 0 \implies \text{finite (roots-with } P \, p)\)
by (rule finite-subset[OF poly-roots-finite[of p]]) (auto simp: roots-with-def)

lemma count-roots-with-times-root:
assumes \(p \neq 0\) \(P\) \((\text{a :: }'\text{a :: linordered_idom})\)
shows count-roots-with \(P\) \([\text{a, }-1:] * p\) = Suc (count-roots-with \(P\) \(p\))
proof
  define \(q\) where \(q = \text{[a, }-1:] * p\)
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 \(\sum x \in \text{roots-with } P \, q\) = \(\sum x \in \text{roots-with } P \, p\)
  proof
    fix \(x\) assume \(x \in \text{roots-with } P \, q\)
    from assms have \(\text{order } x \, q\) = \(\text{order } x \, p\) by (simp-all add: roots-with-def q-def)
    also have \(\text{order } a \text{-nz: } a \neq 0\)
      unfolding q-def
      by (rule no-zero-divisors) (simp-all add: assms)
    also have \(\sum x \in \text{roots-with } P \, q\) = \(\sum x \in \text{roots-with } P \, p\)
    proof
      fix \(x\) assume \(x \in \text{roots-with } P \, q\)
      from assms have \(\text{order } x \, q\) = \(\text{order } x \, p\) by (simp-all add: roots-with-def q-def)
      finally show \(\text{order } x \, q\) = \(\text{order } x \, p\) by simp
    qed
  qed
  also have \(\sum x \in \text{roots-with } P \, q\) = \(\sum x \in \text{roots-with } P \, p\)
  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 \(\forall x \in \text{roots-with } P \, q \neq \text{roots-with } P \, p\) \(\text{order } x \, p\) = 0
      by (auto simp: roots-with-def q-def order-root simp del: mult-pCons-left)
    qed
  finally show \(\text{count-roots-with } P \, q\) = Suc (count-roots-with \(P\) \(p\))
    by (simp add: 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** \(\text{sign-changes-coeff-sign-changes}\):

- **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** \(\ldots = \text{strip-while} \ ((\approx) 0) \ xs\) by simp
- **also have** \(\text{filter} \ ((\neq) 0) \ldots = \text{filter} \ ((\neq) 0) \ xs\) unfolding \(\text{strip-while-def} \ o\text{-def}\)
  by (subst rev-filter [symmetric], subst filter-dropWhile) (simp-all add: rev-filter)
- **also have** \(\text{sign-changes} \ldots = \text{sign-changes} \ xs\) by (simp add: \(\text{sign-changes-filter}\))
- **finally show** \(?\text{thesis}\) by (simp add: \(\text{sign-changes-filter}\))

**qed**

By applying \(\text{reduce-root} \ a\), 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** \(\text{coeff-sign-changes-reduce-root}\):

- **assumes** \(a > (0 :: 'a :: \text{linordered-idom})\)
- **shows** \(\text{coeff-sign-changes} \ (\text{reduce-root} \ a \ p) = \text{coeff-sign-changes} \ p\)

**proof** (intro \(\text{sign-changes-cong}\), induction \(p\))

**case** \((\text{pCons} \ c \ p)\)

- **have** \(\text{map} \ \text{sgn} \ (\text{coeffs} \ (\text{reduce-root} \ a \ (\text{pCons} \ c \ p))) = \text{cCons} \ (\text{sgn} \ c) \ (\text{map} \ \text{sgn} \ (\text{coeffs} \ (\text{reduce-root} \ a \ p)))\)
  using assms by (auto simp add: cCons-def sgn-0-0 sgn-mult reduce-root-pCons coeffs-smult)
- **also note** \(\text{pCons.IH}\)
- **also have** \(\text{cCons} \ (\text{sgn} \ c) \ (\text{map} \ \text{sgn} \ (\text{coeffs} \ p)) = \text{map} \ \text{sgn} \ (\text{coeffs} \ (\text{pCons} \ c \ p))\)
  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** \(\text{coeff-sign-changes-smult}\):

- **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!: \(\text{sign-changes-cong}\) simp: sgn-mult coeffs-smult)

**context**
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-coef-sign-changes-imp-pos-roots-aux`)
  show `r ≠ 0` by fact
  have `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 + . . . + a_n X^n`, we have

```
(\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`)

15
also from coeff-poly-times-one-minus-x[of Poly ys i] assms
  have ... = (∑ 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-coefs 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)
  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 \( a · (1 − X) · p(aX) \).

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 \( v \equiv \text{coeff-sign-changes} \)
shows \( v ([:a, -1:] \ast g) - v g > 0 \land \text{odd} (v ([:a, -1:] \ast g) - v g) \)

proof
have \( 0 < v ([:1, -1:] \ast \text{reduce-root} a g) - v (\text{reduce-root} a g) \land \text{odd} (v ([:1, -1:] \ast \text{reduce-root} a g) - v (\text{reduce-root} a g)) \)
  using \( nz \) pos unfolding \( v \)-def by (intro sign-changes-poly-times-one-minus-x)
simp-all
also have \( v ([:1, -1:] \ast \text{reduce-root} a g) = v (\text{smult} a ([:1, -1:] \ast \text{reduce-root} a g)) \)
unfolding \( v \)-def by (simp add: coeff-sign-changes-smult pos)
also have \( \text{smult} a ([:1, -1:] \ast \text{reduce-root} a g) = [:a:] \ast [:1, -1:] \ast \text{reduce-root} a g \) by (subst mult.assoc) simp
also have \( [:a:] \ast [:1, -1:] = \text{reduce-root} a [:a, -1:] \)
by (simp add: reduce-root-def pcompose-pCons)
also have \( \ldots \ast \text{reduce-root} a g = \text{reduce-root} a ([:a, -1:] \ast g) \)
unfolding reduce-root-def by (simp only: pcompose-mult)
also have \( v \ldots = v ([:a, -1:] \ast g) \) 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 \( \text{thesis} \).
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 \( \text{coeff-sign-changes} p \geq \text{count-pos-roots} p \land \text{even} (\text{coeff-sign-changes} p - \text{count-pos-roots} p) \)
using \( \text{assms} \)
proof (induction \( p \) rule: poly-root-induct[where \( P = \lambda a. a > 0 \)])
case (root \( a \) \( p \))
define \( q \) where \( q = [:a, -1:] \ast p \)
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 \( \text{case} \) 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 \( \text{case using no-roots} \ (p \neq 0) \) 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** \( \exists d. \text{even } d \land \text{coeff-sign-changes } p = \text{count-pos-roots } p + d \)

**Proof**

- **Define** \( d \) where \( d = \text{coeff-sign-changes } p - \text{count-pos-roots } p \)
- **Show** \( \text{even } d \land \text{coeff-sign-changes } p = \text{count-pos-roots } p + d \)

**Unfolding** \( d \)-def **Using** descartes-sign-rule-aux[OF assms] **By** auto

**QED**

**End**

**References**