Algebraic Numbers in Isabelle/HOL

René Thiemann, Akihisa Yamada, and Sebastiaan Joosten

February 23, 2021

Abstract

Based on existing libraries for matrices, factorization of integer polynomials, and Sturm’s theorem, we formalized algebraic numbers in Isabelle/HOL. Our development serves as an implementation for real and complex numbers, and it admits to compute roots and completely factorize real and complex polynomials, provided that all coefficients are rational numbers. Moreover, we provide two implementations to display algebraic numbers, an injective one that reveals the representing polynomial, or an approximative one that only displays a fixed amount of digits.

To this end, we mechanized several results on resultants.

Contents

1 Introduction 3

2 Auxiliary Algorithms 5

3 Algebraic Numbers – Excluding Addition and Multiplication 5

3.1 Polynomial Evaluation of Integer and Rational Polynomials in Fields. 7

3.2 Algebraic Numbers – Definition, Inverse, and Roots 8

4 Resultants 28

4.1 Bivariate Polynomials 28

4.1.1 Evaluation of Bivariate Polynomials 28

4.1.2 Swapping the Order of Variables 31

4.2 Resultant 39

4.2.1 Sylvester matrices and vector representation of polynomials 39

4.2.2 Homomorphism and Resultant 47

*Supported by FWF (Austrian Science Fund) project Y757.
5 Algebraic Numbers: Addition and Multiplication 61
5.1 Addition of Algebraic Numbers 61
5.1.1 poly-add has desired root 62
5.1.2 poly-add is nonzero 64
5.1.3 Summary for addition 68
5.2 Division of Algebraic Numbers 71
5.2.1 Summary for division 76
5.3 Multiplication of Algebraic Numbers 76
5.4 Summary: Closure Properties of Algebraic Numbers 77
5.5 More on algebraic integers 78

6 Separation of Roots: Sturm 89
6.1 Interface for Separating Roots 89
6.2 Implementing Sturm on Rational Polynomials 92

7 Getting Small Representative Polynomials via Factorization 96

8 Real Algebraic Numbers 102
8.1 Real Algebraic Numbers – Innermost Layer 106
8.1.1 Basic Definitions 106
8.2 Real Algebraic Numbers = Rational + Irrational Real Algebraic Numbers 111
8.2.1 Definitions and Algorithms on Raw Type 111
8.2.2 Definitions and Algorithms on Quotient Type 112
8.2.3 Sign 112
8.2.4 Normalization: Bounds Close Together 112
8.2.5 Comparisons 126
8.2.6 Negation 128
8.2.7 Inverse 129
8.2.8 Floor 131
8.2.9 Generic Factorization and Bisection Framework 133
8.2.10 Addition 142
8.2.11 Multiplication 148
8.2.12 Root 155
8.2.13 Embedding of Rational Numbers 163
8.2.14 Definitions and Algorithms on Type with Invariant 171
8.3 Real Algebraic Numbers as Implementation for Real Numbers 179

9 Real Roots 180

10 Complex Roots of Real Valued Polynomials 196
10.1 Compare Instance for Complex Numbers 205
11 Interval Arithmetic
  11.1 Syntactic Class Instantiations .................................. 208
  11.2 Class Instantiations ............................................. 209
  11.3 Membership ....................................................... 209
  11.4 Convergence ..................................................... 212
  11.5 Complex Intervals ................................................ 214

12 Complex Algebraic Numbers
  12.1 Complex Roots .................................................. 218
  12.2 Factorization of Complex Polynomials ......................... 239

13 Real Factorization .................................................. 243

14 Show for Real Algebraic Numbers – Interface ....................... 249

15 Show for Real (Algebraic) Numbers – Approximate Representation 250

16 Show for Real (Algebraic) Numbers – Unique Representation ........ 251

17 Algebraic Number Tests ............................................. 254
  17.1 Stand-Alone Examples ........................................... 254
  17.2 Example Application: Compute Norms of Eigenvalues ........ 255

18 Explicit Constants for External Code ................................ 255
  18.1 Operations on Real Algebraic Numbers ......................... 256
  18.2 Operations on Complex Algebraic Numbers ..................... 256
  18.3 Export Constants in Haskell .................................... 257

1 Introduction

Introduction

Isabelle’s previous implementation of irrational numbers was limited: it only admitted numbers expressed in the form “$a + b\sqrt{c}$” for $a, b, c \in \mathbb{Q}$, and even computations like $\sqrt{2} \cdot \sqrt{3}$ led to a runtime error [3].

In this work, we provide full support for the real algebraic numbers, i.e., the real numbers that are expressed as roots of non-zero integer polynomials, and we also partially support complex algebraic numbers.

Most of the results on algebraic numbers have been taken from a textbook by Bhubaneswar Mishra [2]. Also Wikipedia provided valuable help.

Concerning the real algebraic numbers, we first had to prove that they form a field. To show that the addition and multiplication of real algebraic numbers are also real algebraic numbers, we formalize the theory of resultants, which are the determinants of specific matrices, where the size
of these matrices depend on the degree of the polynomials. To this end, we utilized the matrix library provided in the Jordan-Normal-Form AFP-entry [4] where the matrix dimension can arbitrarily be chosen at runtime.

Given real algebraic numbers $x$ and $y$ expressed as the roots of polynomials, we compute a polynomial that has $x + y$ or $x \cdot y$ as its root via resultants. In order to guarantee that the resulting polynomial is non-zero, we needed the result that multivariate polynomials over fields form a unique factorization domain (UFD). To this end, we initially proved that polynomials over some UFD are again a UFD, relying upon results in HOL-algebra.

When performing actual computations with algebraic numbers, it is important to reduce the degree of the representing polynomials. To this end, we use the existing Berlekamp-Zassenhaus factorization algorithm. This is crucial for the default show-function for real algebraic numbers which requires the unique minimal polynomial representing the algebraic number – but an alternative which displays only an approximative value is also available.

In order to support tests on whether a given algebraic number is a rational number, we also make use of the fact that we compute the minimal polynomial.

The formalization of Sturm’s method [1] was crucial to separate the different roots of a fixed polynomial. We could nearly use it as it is, and just copied some function definition so that Sturm’s method now is available to separate the real roots of rational polynomial, where all computations are now performed over $\mathbb{Q}$.

With all the mentioned ingredients we implemented all arithmetic operations on real algebraic numbers, i.e., addition, subtraction, multiplication, division, comparison, $n$-th root, floor- and ceiling, and testing on membership in $\mathbb{Q}$. Moreover, we provide a method to create real algebraic numbers from a given rational polynomial, a method which computes precisely the set of real roots of a rational polynomial.

The absence of an equivalent to Sturm’s method for the complex numbers in Isabelle/HOL prevented us from having native support for complex algebraic numbers. Instead, we represent complex algebraic numbers as their real and imaginary part: note that a complex number is algebraic if and only if both the real and the imaginary part are real algebraic numbers. This equivalence also admitted us to design an algorithm which computes all complex roots of a rational polynomial. It first constructs a set of polynomials which represent all real and imaginary parts of all complex roots, yielding a superset of all roots, and afterwards the set just is just filtered.

By the fundamental theorem of algebra, we then also have a factorization algorithm for polynomials over $\mathbb{C}$ with rational coefficients.

Finally, for factorizing a rational polynomial over $\mathbb{R}$, we first factorize it over $\mathbb{C}$, and then combine each pair of complex conjugate roots.
As future it would be interesting to include the result that the set of complex algebraic numbers is algebraically closed, i.e., at the moment we are limited to determine the complex roots of a polynomial over $\mathbb{Q}$, and cannot determine the real or complex roots of an polynomial having arbitrary algebraic coefficients.

Finally, an analog to Sturm’s method for the complex numbers would be welcome, in order to have a smaller representation: for instance, currently the complex roots of $1 + x + x^3$ are computed as “root #1 of $1 + x + x^3$”, “(root #1 of $-\frac{1}{8} + \frac{1}{4}x + x^3$)+(root #1 of $-\frac{31}{64} + \frac{9}{16}x^2 - \frac{3}{2}x^4 + x^6)i”,” and “(root #1 of $-\frac{1}{8} + \frac{1}{4}x + x^3$)+(root #2 of $-\frac{31}{64} + \frac{9}{16}x^2 - \frac{3}{2}x^4 + x^6)i”.

2 Auxiliary Algorithms

3 Algebraic Numbers – Excluding Addition and Multiplication

This theory contains basic definition and results on algebraic numbers, namely that algebraic numbers are closed under negation, inversion, $n$-th roots, and that every rational number is algebraic. For all of these closure properties, corresponding polynomial witnesses are available.

Moreover, this theory contains the uniqueness result, that for every algebraic number there is exactly one content-free irreducible polynomial with positive leading coefficient for it. This result is stronger than similar ones which you find in many textbooks. The reason is that here we do not require a least degree construction.

This is essential, since given some content-free irreducible polynomial for $x$, how should we check whether the degree is optimal. In the formalized result, this is not required. The result is proven via GCDs, and that the GCD does not change when executed on the rational numbers or on the reals or complex numbers, and that the GCD of a rational polynomial can be expressed via the GCD of integer polynomials.

Many results are taken from the textbook [2, pages 317ff].

theory Algebraic-Numbers-Prelim
imports
HOL-Computational-Algebra.Fundamental-Theorem-Algebra
Polynomial-Interpolation.Newton-Interpolation
Polynomial-Factorization.Gauss-Lemma
Berlekamp-Zassenhaus.Unique-Factorization-Poly
Polynomial-Factorization.Square-Free-Factorization

begin

lemma primitive-imp-unit-iff:
fixes p :: 'a :: \{comm-semiring-1,semiring-no-zero-divisors\} poly
assumes pr: primitive p
shows \( p \text{ dvd } 1 \iff \deg p = 0 \)

proof

assume \( \deg p = 0 \)

from \( \text{degree0-coffs[OF this]} \) obtain \( p0 \) where \( p: p = [p0:] \) by auto

then have \( \forall c \in \text{set(coffs p)}. \, p0 \text{ dvd } c \) by (simp add: cCons-def)

with \( pr \) have \( p0 \text{ dvd } 1 \) by (auto dest: primitiveD)

with \( p \) show \( p \text{ dvd } 1 \) by auto

next

assume \( p \text{ dvd } 1 \)

then show \( \deg p = 0 \) by (auto simp: poly-dvd-1)

qed

lemma dvd-all-coffs-imp-dvd:

assumes \( \forall a \in \text{set(coffs p)}. \, c \text{ dvd } a \)

shows \( [:c:] \text{ dvd } p \)

proof (insert assms, induct p)

case 0

then show \(?case by simp\)

next

case (pCons a p)

have \( p\text{Cons a} = [:a:] + p\text{Cons 0} p \) by simp

also have \( [:c:] \text{ dvd } ... \)

proof (rule dvd-add)

from pCons show \( [:c:] \text{ dvd } [:a:] \) by (auto simp: cCons-def)

from pCons have \( [:c:] \text{ dvd } p \) by auto

from Rings.dvd-mult[OF this]

show \( [:c:] \text{ dvd } p\text{Cons 0} p \) by (subst pCons-0-as-mult)

qed

finally show \(?case.\)

qed

lemma irreducible-content:

fixes \( p :: 'a::\{\text{comm-semiring-1,semiring-no-zero-divisors}\} \text{ poly} \)

assumes irreducible \( p \)

shows \( \deg p = 0 \lor \text{primitive } p \)

proof (rule ccontr)

assume \( \neg \thesis \)

then obtain \( c \) where \( c1: \neg c \text{ dvd } 1 \text{ and } \forall a \in \text{set(coffs p)}. \, c \text{ dvd } a \) by (auto elim: not-primitiveE)

from dvd-all-coffs-imp-dvd[OF this(2)]

obtain \( r \) where \( p: p = r * [:c:] \) by (elim dvdE, auto)

from irreducibleD[OF assms this] have \( r \text{ dvd } 1 \lor [:c:] \text{ dvd } 1 \) by auto

with \( c1 \) have \( r \text{ dvd } 1 \) unfolding const-poly-dvd-1 by auto

then have \( \deg r = 0 \) unfolding poly-dvd-1 by auto

with \( p \) have \( \deg p = 0 \) by auto

with \( \neg \thesis \) False by auto

qed

lemma linear-irreducible-field:

fixes \( p :: 'a :: \text{field poly} \)

---

6
assumes \( \text{deg} \): degree \( p = 1 \) shows irreducible \( p \)

proof (intro irreducibleI)
  from \( \text{deg} \) show \( p \neq 0 \) by auto
  from \( \text{deg} \) show \( \neg p \mid 1 \) by (auto simp: poly-dvd-1)
  fix \( a \) \( b \) assume \( p \): \( p = a \times b \)
  with \( p \neq 0 \) have \( a \neq 0 \) and \( b \neq 0 \) by auto
  from degree-mult-eq[OF this, folded \( p \)] assms
  consider \( \text{degree } a = 1 \) \( \text{degree } b = 0 \) | \( \text{degree } a = 0 \) \( \text{degree } b = 1 \) by force
  then show \( a \mid 1 \lor b \mid 1 \)
    by (cases; insert \( a \) \( b \), auto simp: primitive-imp-unit-iff)
qed

lemma linear-irreducible-int:
  fixes \( p \) :: int poly
  assumes \( \text{deg} \): degree \( p = 1 \) and \( cp \): content \( p \mid 1 \)
  shows irreducible \( p \)
proof (intro irreducibleI)
  from \( \text{deg} \) show \( p \neq 0 \) by auto
  from \( \text{deg} \) show \( \neg p \mid 1 \) by (auto simp: poly-dvd-1)
  fix \( a \) \( b \) assume \( p \): \( p = a \times b \)
  note \( \star = cp[\text{unfolded } p \mid \text{is-unit-content-iff}, \text{unfolded } \text{content-mult}] \)
  have \( a1: \text{content } a \mid 1 \) and \( b1: \text{content } b \mid 1 \)
    using content-ge-0-int[of \( a \)] pos-zmult-eq-1-iff-lemma[OF \( \star \)] by (auto simp: abs-mult)
  with \( p \neq 0 \) have \( a \neq 0 \) and \( b \neq 0 \) by auto
  from degree-mult-eq[OF this, folded \( p \)] assms
  consider \( \text{degree } a = 1 \) \( \text{degree } b = 0 \) | \( \text{degree } a = 0 \) \( \text{degree } b = 1 \) by force
  then show \( a \mid 1 \lor b \mid 1 \)
    by (cases; insert \( a1 \) \( b1 \), auto simp: primitive-imp-unit-iff)
qed

lemma irreducible-connect-rev:
  fixes \( p \) :: \( a \) :: \{comm-semiring-1,semiring-no-zero-divisors\} poly
  assumes \( \text{irr} \): irreducible \( p \) and \( \text{deg} \): degree \( p > 0 \)
  shows irreducible,\( \_ \) \( p \)
proof(intro irreducible_4I \( \text{deg} \))
  fix \( q \) \( r \)
  assume \( \text{degq} \): degree \( q > 0 \) and \( \text{diff} \): degree \( q < \text{degree } p \) and \( p \): \( p = q \times r \)
  from \( \text{degq} \) have \( \neg q \mid 1 \) by (auto simp: poly-dvd-1)
  from irreducibleD[OF \( \text{irr } p \)] nu have \( r \mid 1 \) by auto
  then have \( \text{degree } r = 0 \) by (auto simp: poly-dvd-1)
  with \( \text{degq} \) \( \text{diff} \) show False unfolding \( p \) using degree-mult-le[of \( q \) \( r \)] by auto
qed

3.1 Polynomial Evaluation of Integer and Rational Polynomials in Fields.

abbreviation ipoly where ipoly \( f \) \( x \equiv \text{poly } (\text{of-int } \text{poly } f) \) \( x \)
lemma poly-map-poly-code[code-unfold]: poly (map-poly h p) x = fold-coeffs (λ a b. h a + x * b) p 0
  by (induct p, auto)

abbreviation real-of-int-poly :: int poly ⇒ real poly where
  real-of-int-poly ≡ of-int-poly

abbreviation real-of-rat-poly :: rat poly ⇒ real poly where
  real-of-rat-poly ≡ map-poly of-rat

lemma of-rat-of-int[simp]: of-rat ◦ of-int = of-int by auto

lemma ipoly-of-rat[simp]: ipoly p (of-rat y :: 'a :: {field, real-algebra-1}) = of-rat (ipoly p y)
proof
  have id: of-int = of-rat o of-int unfolding comp-def by auto
  show ?thesis by (subst id, subst map-poly-map-poly[symmetric], auto)
qed

lemma finite-ipoly-roots: assumes p ≠ 0
  shows finite {x :: real. ipoly p x = 0}
proof
  let ?p = real-of-int-poly p
  from assms have ?p ≠ 0 by auto
  thus ?thesis by (rule poly-roots-finite)
qed

3.2 Algebraic Numbers – Definition, Inverse, and Roots
A number x is algebraic iff it is the root of an integer polynomial. Whereas the Isabelle distribution this is defined via the embedding of integers in a field via $\mathbb{Z}$, we work with integer polynomials of type int and then use ipoly for evaluating the polynomial at a real or complex point.

lemma algebraic-altdef-ipoly:
  shows algebraic x ←→ (∃ p. ipoly p x = 0 ∧ p ≠ 0)
unfolding algebraic-def
proof (safe, goal-cases)
case (1 p)
  define the-int where the-int = (λx::'a. THE r. x = of-int r)
define p' where p' = map-poly the-int p
have of-int-the-int: of-int (the-int x) = x if x ∈ $\mathbb{Z}$ for x
  unfolding the-int-def by (rule sym, rule the1') (insert that, auto simp: Ints-def)
have the-int-0-iff: the-int \( x = 0 \iff x = 0 \) if \( x \in \mathbb{Z} \)

using of-the-int[OF that] by auto

have map-poly of-int \( p' = \text{map-poly} \ (\text{of-int} \circ \text{the-int}) \) \( p \)
  by (simp add: \( p' \)-def map-poly-map-poly)

also from 1 of-int-the-int have ... = \( p \)
  by (subst poly-eq-iff) (auto simp coeff-map-poly)

finally have \( p-p' : \text{map-poly} \ \text{of-int} \) \( p' = p \).

show \?case
proof (intro exI conjI notI)
  from 1 show \( \text{ipoly} \ \ p' \ x = 0 \) by (simp add: \( p-p' \))

next
  assume \( p' = 0 \)
  hence \( p = 0 \) by (simp add: \( p-p' \) [symmetric])

with \( \langle p \neq 0 \rangle \) show False by contradiction

qed

next

\( \text{case } (2 \ p) \)

thus \?case by (intro exI[of - map-poly \ text{of-int} \ p], auto)

qed

Definition of being algebraic with explicit witness polynomial.

definition represents :: int poly \( \Rightarrow 'a \Rightarrow \text{field-char-0} \Rightarrow \text{bool} \) (infix represents 51)
  where \( p \ \text{represents} \ x \) = \( \text{ipoly} \ \ p \ x = 0 \ \wedge \ p \neq 0 \)

lemma representsI[intro]: \( \text{ipoly} \ \ p \ x = 0 \Rightarrow p \neq 0 \Rightarrow p \ \text{represents} \ x \)

unfolding represents-def by auto

lemma representsD:
  assumes \( p \ \text{represents} \ x \) shows \( p \neq 0 \) and \( \text{ipoly} \ \ p \ x = 0 \) using assms unfolding represents-def by auto

lemma representsE:
  assumes \( p \ \text{represents} \ x \) and \( p \neq 0 \Rightarrow \text{ipoly} \ \ p \ x = 0 \Rightarrow \text{thesis} \)

shows thesis using assms unfolding represents-def by auto

lemma represents-imp-degree:
  fixes \( x :: 'a :: \text{field-char-0} \)
  assumes \( p \ \text{represents} \ x \) shows \( \text{degree} \ p \neq 0 \)

proof
  from assms have \( p \neq 0 \) and \( \text{px} : \text{ipoly} \ \ p \ x = 0 \) by (auto dest:representsD)
  then have \( \text{of-int-poly} \ p :: 'a \poly \neq 0 \) by auto
  then have \( \text{degree} \ (\text{of-int-poly} \ p :: 'a \poly) \neq 0 \) by (fold poly-zero[OF px])
  then show \?thesis by auto

qed

lemma representsE-full[elim]:
  assumes rep: \( p \ \text{represents} \ x \)
  and main: \( p \neq 0 \Rightarrow \text{ipoly} \ \ p \ x = 0 \Rightarrow \text{degree} \ p \neq 0 \Rightarrow \text{thesis} \)

shows thesis
by (rule main, insert represents-imp-degree[OF rep] rep, auto elim: representsE)

lemma represents-of-rat[simp]: p represents (of-rat x) = p represents x by (auto elim!:representsE)
lemma represents-of-real[simp]: p represents (of-real x) = p represents x by (auto elim!:representsE)

lemma algebraic-iff-represents: algebraic x ←→ (∃ p. p represents x)
unfolding algebraic-altdef-ipoly represents-def ..

lemma represents-irr-non-0: assumes irr: irreducible p and ap: p represents x and x0: x ≠ 0 shows poly p 0 ≠ 0 proof have nu: ¬ [:0,1::int:] dvd 1 by (auto simp: poly-dvd-1) assume poly p 0 = 0 hence dvd: [:0,1:] dvd p by (unfold dvd-iff-poly-eq-0, simp)
then obtain q where pq: p = [:0,1:] * q by (elim dvdE)
from irreducibleD[OF irr this] nu have q dvd 1 by auto
from this obtain r where q = [:r:] r dvd 1 by (auto simp add: poly-dvd-1 dest: degree0-coeffs)
with pq have p = [:0,r:] by auto
with ap have x = 0 by (auto simp: of-int-hom.map-poly-pCons-hom)
with x0 show False by auto
qed

The polynomial encoding a rational number.
definition poly-rat :: rat ⇒ int poly where
poly-rat x = (case quotient-of x of (n,d) ⇒ [:−n,d:])
definition abs-int-poly :: int poly ⇒ int poly where
abs-int-poly p ≡ if lead-coeff p < 0 then −p else p
lemma pos-poly-abs-poly[simp]: shows lead-coeff (abs-int-poly p) > 0 ←→ p ≠ 0
proof have p ≠ 0 ←→ lead-coeff p * sgn (lead-coeff p) > 0 by (fold abs-sgn, auto)
then show ?thesis by (auto simp: abs-int-poly-def mult.commute)
qed

lemma abs-int-poly-0[simp]: abs-int-poly 0 = 0 by (auto simp: abs-int-poly-def)
lemma abs-int-poly-eq-0-iff[simp]: abs-int-poly p = 0 ←→ p = 0 by (auto simp: abs-int-poly-def sgn-eq-0-iff)
lemma degree-abs-int-poly[simp]: degree (abs-int-poly p) = degree p by (auto simp: abs-int-poly-def sgn-eq-0-iff)
lemma abs-int-poly-dvd[simp]: abs-int-poly p dvd q ←→ p dvd q
  by (unfold abs-int-poly-def, auto)

lemma (in idom) irreducible-uminus[simp]: irreducible (−x) ←→ irreducible x
  proof
    have −x = −1 ∗ x by simp
    also have irreducible ... ←→ irreducible x by (rule irreducible-mult-unit-left, auto)
    finally show ?thesis.
  qed

lemma irreducible-abs-int-poly[simp]:
  irreducible (abs-int-poly p) ←→ irreducible p
  by (unfold abs-int-poly-def, auto)

lemma coeff-abs-int-poly[simp]:
  coeff (abs-int-poly p) n = (if lead-coeff p < 0 then −coeff p n else coeff p n)
  by (simp add: abs-int-poly-def)

lemma lead-coeff-abs-int-poly[simp]:
  lead-coeff (abs-int-poly p) = abs (lead-coeff p)
  by auto

lemma ipoly-abs-int-poly-eq-zero-iff[simp]:
  ipoly (abs-int-poly p) (x :: 'a :: comm-ring-1) = 0 ←→ ipoly p x = 0
  by (auto simp: abs-int-poly-def sgn-eq-0-iff of-int-poly-hom.hom-uminus)

lemma abs-int-poly-represents[simp]:
  abs-int-poly p represents x ←→ p represents x by (auto elim!:representsE)

lemma content-pCons[simp]: content (pCons a p) = gcd a (content p)
  by (unfold content-def coeffs-pCons-eq-cCons cCons-def, auto)

lemma content-uminus[simp]:
  fixes p :: 'a :: ring-gcd poly shows content (−p) = content p
  by (induct p, auto)

lemma primitive-abs-int-poly[simp]:
  primitive (abs-int-poly p) ←→ primitive p
  by (auto simp: abs-int-poly-def)

lemma abs-int-poly-inv[simp]: smult (sgn (lead-coeff p)) (abs-int-poly p) = p
  by (cases lead-coeff p > 0, auto simp: abs-int-poly-def)
definition cf-pos :: int poly ⇒ bool where
  cf-pos p = (content p = 1 ∧ lead-coeff p > 0)

definition cf-pos-poly :: int poly ⇒ int poly where
  cf-pos-poly f = (let
c    c = content f;
d  d = (sgn (lead-coeff f) * c)
in sdiv-poly f d)

lemma sgn-is-unit[intro!]:
  fixes x :: 'a :: linordered-idom
  assumes x ≠ 0
  shows sgn x dvd 1 using assms by (cases x 0::'a rule: linorder-cases, auto)

lemma cf-pos-poly-0[simp]: cf-pos-poly 0 = 0 by (unfold cf-pos-poly-def sdiv-poly-def, auto)

lemma cf-pos-poly-eq-0[simp]: cf-pos-poly f = 0 ←→ f = 0
  proof (cases f = 0)
    case True
    thus ?thesis unfolding cf-pos-poly-def Let-def by (simp add: sdiv-poly-def)
  next
    case False
    then have lc0: lead-coeff f ≠ 0 by auto
    then have s0: sgn (lead-coeff f) ≠ 0 (is ?s ≠ 0) and content f ≠ 0 (is ?c ≠ 0)
      by (auto simp: sgn-0-0)
    then have sc0: ?s * ?c ≠ 0 by auto
    { fix i from content-dvd-coeff sgn-is-unit[OF lc0]
      have ?s * ?c dvd coeff f i by (auto simp: unit-dvd-iff)
      then have coeff f i div (?s * ?c) = 0 ←→ coeff f i = 0 by (auto simp: dvd-div-eq-0-iff)
    }
    note * = this
  qed

lemma cf-pos-poly-main: smult (sgn (lead-coeff f) * content f) (cf-pos-poly f) = f (is ?g1)
  and content-cf-pos-poly[simp]: content (cf-pos-poly f) = (if f = 0 then 0 else 1) (is ?g2)
  and lead-coeff-cf-pos-poly[simp]: lead-coeff (cf-pos-poly f) > 0 ←→ f ≠ 0 (is ?g3)
  and cf-pos-poly-dvd[simp]: cf-pos-poly f dvd f (is ?g4)
  proof (atomize(full), (cases f = 0; intro conjI))
    case True
    then show ?g1 ?g2 ?g3 ?g4 by simp-all
  next
    case f0: False
let \(?s = \text{sgn}\ \ (\text{lead-coeff } f)\)

have \(s\) : \(?s \in \{-1,1\}\) using \(f0\) unfolding \(\text{sgn-if}\) by auto

define \(g\) where \(g = \text{smult } ?s f\)
define \(d\) where \(d = ?s \times \text{content } f\)

have \(\text{content } g = \text{content } \{[:?s:] \times f\}\) unfolding \(\text{g-def}\) by simp
also have \(\ldots = \text{content } [:?s]: \times \text{content } f\) unfolding \(\text{content-mult}\) by simp
also have \(\text{content } [:?s:] = 1\) using \(s\) by (auto simp: \(\text{content-def}\))

finally have \(cg\): \(\text{content } g = \text{content } f\) by simp

from \(f0\) have \(d\) : \(\text{cf-pos-poly } f = \text{sdiv-poly } f \times d\) by (auto simp: \(\text{cf-pos-poly-def}\) \(\text{Let-def}\) \(\text{d-def}\))

let \(?g = \text{primitive-part } g\)
define \(ng\) where \(ng = \text{primitive-part } g\)

note \(d\)
also have \(\text{sdv-poly } f \times d = \text{sdv-poly } g \times (\text{content } g)\) unfolding \(\text{cg}\) unfolding \(\text{g-def}\) \(\text{d-def}\) by (rule \(\text{poly-eqI}\), unfold \(\text{coeff-sdv-poly}\) \(\text{coeff-smult}\), insert \(s\), auto simp: \(\text{div-minus-right}\))
finally have \(fg\) : \(\text{cf-pos-poly } f = \text{primitive-part } g\) unfolding \(\text{primitive-part-alt-def}\)

have \(\text{lead-coeff } f \neq 0\) using \(f0\) by auto
hence \(g\) : \(\text{lead-coeff } g > 0\) unfolding \(\text{g-def}\) \(\text{lead-coeff-smult}\)
  by (meson \(\text{linorder-neqE-linordered-idom}\) \(\text{sgn-greater}\) \(\text{sgn-less}\) \(\text{zero-less-mult-iff}\))

hence \(g0\): \(g \neq 0\) by auto
from \(f0\) \(\text{content-primitive-part}[OF this]\)
show \(?g2\) unfolding \(fg\) by auto

from \(g0\) have \(\text{content } g \neq 0\) by simp
with \(\text{arg-cong}[OF \text{content-times-primitive-part}[af g, of \text{lead-coeff}, unfolded \text{lead-coeff-smult}]\)
  \(\text{lg content-ge-0-in}[of g]\) have \(\text{lg}\): \(\text{lead-coeff } ng > 0\) unfolding \(\text{ng-def}\)
  by (metis \(\text{dual-order}\) \(\text{antisym}\) \(\text{dual-order}\) \(\text{strict-implies-order}\) \(\text{zero-less-mult-iff}\))

with \(f0\) show \(?g3\) unfolding \(fg\) \(\text{ng-def}\) by auto

have \(dl\): \(d \neq 0\) using \(s\) \(\text{f0}\) by (force simp add: \(\text{d-def}\))

have \(\text{smult } d \times (\text{cf-pos-poly } f) = \text{smult } ?s \times (\text{smult } (\text{content } f)) \times (\text{sdv-poly } (\text{smult } ?s f)) \times (\text{content } f))\)
  unfolding \(\text{fg}\) \(\text{primitive-part-alt-def}\) \(\text{cg}\) by (simp add: \(\text{g-def}\) \(\text{d-def}\))
also have \(\text{sdv-poly } (\text{smult } ?s f) \times (\text{content } f) = \text{smult } ?s \times (\text{sdv-poly } f \times (\text{content } f))\)
  using \(s\) by (metis \(\text{cg}\) \(\text{g-def}\) \(\text{primitive-part-alt-def}\) \(\text{primitive-part-smult-int}\) \(\text{sgn-sgn}\))
finally have \(\text{smult } d \times (\text{cf-pos-poly } f) = \text{smult } (\text{content } f) \times (\text{primitive-part } f)\)
  unfolding \(\text{primitive-part-alt-def}\) using \(s\) by auto
also have \(\ldots = f\) by (rule \(\text{content-times-primitive-part}\))
finally have \(df\): \(\text{smult } d \times (\text{cf-pos-poly } f) = f\).
with \(dl\) show \(?g4\) by (auto simp: \(\text{d-def}\))
from \(df\) have \(\star = \text{cf-pos-poly } f \times [:d:]\) by simp
from \(\text{dvdI}[OF this]\) show \(?g4.\)

qed

lemma irreducible-connect-int:
  fixes \(p\) :: int poly
  assumes \(ir\): irreducible\(_d\) \(p\) and \(c\): \(\text{content } p = 1\)
shows irreducible \( p \)

using \( c \) primitive-iff-content-eq-1 ir irreducible-primitive-connect by blast

lemma

fixes \( x \) :: 'a :: 

shows ipoly-cf-pos-poly-eq-0[simp]: ipoly (cf-pos-poly \( p \)) \( x = 0 \) \( \iff \) ipoly \( p \) \( x = 0 \)

and degree-cf-pos-poly[simp]: degree (cf-pos-poly \( p \)) = degree \( p \)

and cf-pos-cf-pos-poly[intro]: \( p \neq 0 \implies cf-pos(cf-pos-poly \( p \))

proof–

show degree (cf-pos-poly \( p \)) = degree \( p \)

by (subst(3) cf-pos-poly-main[symmetric], auto simp:sgn-eq-0-iff)

{ assume \( p : p \neq 0 \)

show cf-pos (cf-pos-poly \( p \)) using cf-pos-poly-main \( p \) by (auto simp: cf-pos-def)

have (ipoly (cf-pos-poly \( p \)) \( x = 0 \)) = (ipoly \( p \) \( x = 0 \))

apply (subst(3) cf-pos-poly-main[symmetric]) by (auto simp: sgn-eq-0-iff hom-distribs)
}

then show (ipoly (cf-pos-poly \( p \)) \( x = 0 \)) = (ipoly \( p \) \( x = 0 \)) by (cases \( p = 0 \), auto)

qed

lemma cf-pos-poly-eq-1: cf-pos-poly \( f = 1 \) \( \iff \) degree \( f = 0 \wedge f \neq 0 \) (is ?l \( \iff \) ?r)

proof(intro iffI conjI)

assume ?r

then have df0: degree \( f = 0 \) and \( f0 : f \neq 0 \) by auto

from degree0-coeffs[OF df0] obtain \( f0 \) where \( f = [f0:] \) by auto

show cf-pos-poly \( f = 1 \) using \( f0 \) unfolding \( f \) cf-pos-poly-def Let-def sdiv-poly-def

by (auto simp: content-def mult-sgn-abs)

next

assume \( l : ?l \)

then have degree (cf-pos-poly \( f \)) = 0 by auto

then show degree \( f = 0 \) by simp

from \( l \) have cf-pos-poly \( f \neq 0 \) by auto

then show \( f \neq 0 \) by simp

qed

lemma irr-cf-poly-rat[simp]: irreducible (poly-rat \( x \))

lead-coeff (poly-rat \( x \)) > 0 primitive (poly-rat \( x \))

proof –

obtain \( n d \) where \( x : quotient-of x = (n,d) \) by force

hence id: poly-rat \( x = [-n,d] \) by (auto simp: poly-rat-def)

from quotient-of-denom-pos[OF \( x \)] have \( d : d > 0 \) by auto

show lead-coeff (poly-rat \( x \)) > 0 primitive (poly-rat \( x \))

unfolding id cf-pos-def using \( d \) quotient-of-coprime[OF \( x \)] by (auto simp: content-def)
from this[unfolded cf-pos-def]
show irr: irreducible (poly-rat x) unfolding id using d by (auto intro!: linear-irreducible-int)
qed

lemma poly-rat[simp]: ipoly (poly-rat x) (of-rat x :: 'a :: field-char-0) = 0 ipoly 
(poly-rat x) x = 0
poly-rat x ≠ 0 ipoly (poly-rat x) y = 0 ⟷ y = (of-rat x :: 'a)
proof –
from irr-cf-poly-rat(1)[of x] show poly-rat x ≠ 0
  unfolding Factorial-Ring.irreducible-def by auto
obtain n d where x: quotient-of x = (n,d) by force
hence id: poly-rat x = [-n,d:] by (auto simp: poly-rat-def)
from quotient-of-denom-pos[OF x] have d: d ≠ 0 by auto
have y * of-int d = of-int n ⟷ y = of-int n / of-int d using d
  by (simp add: eq-divide-imp)
with d id show ipoly (poly-rat x) (of-rat x) = 0 ipoly (poly-rat x) x = 0
ipoly (poly-rat x) y = 0 ⟷ y = (of-rat x :: 'a)
  by (auto simp: of-rat-minus of-rat-divide simp: quotient-of-div[OF x])
qed

lemma poly-rat-represents-of-rat: (poly-rat x) represents (of-rat x) by auto

lemma ipoly-smult-0-iff: assumes c: c ≠ 0
  shows (ipoly (smult c p) x = (0 :: real)) = (ipoly p x = 0)
  using c by (simp add: hom-distrib)

lemma not-irreducibleD: 
  assumes ¬ irreducible x and x ≠ 0 and ¬ x dvd 1 
  shows ∃ y z. x = y * z ∧ ¬ y dvd 1 ∧ ¬ z dvd 1 using assms
  apply (unfold Factorial-Ring.irreducible-def) by auto

lemma cf-pos-poly-represents[simp]: (cf-pos-poly p) represents x ⟷ p represents x
  unfolding represents-def by auto

lemma coprime-prod:
  a ≠ 0 ⟹ c ≠ 0 ⟹ coprime (a * b) (c * d) ⟹ coprime b (d::'a::{semiring-gcd})
  by auto

lemma smult-prod:
  smult a b = monom a 0 * b
  by (simp add: monom-0)

lemma degree-map-poly-2: 
  assumes f (lead-coeff p) ≠ 0
shows  \( \text{degree} (\text{map-poly } f \ p) = \text{degree } p \)

proof  
  (cases \( p = 0 \))
  
  case False  
  thus ?thesis
  unfolding \text{degree-eq-length-coeffs Polynomial.coeffs-map-poly}
  using \text{assms by (simp add:coeffs-def)}

qed auto

lemma irreducible-cf-pos-poly:
  assumes \( \text{irr} \): irreducible \( p \) and \( \text{deg} \): degree \( p \neq 0 \)
  shows irreducible (cf-pos-poly \( p \)) (is irreducible \( \_p \))

proof
  (unfold irreducible-altdef, intro conjI allI impI)
  from \( \text{irr} \) show \( \_p \neq 0 \) by auto
  from \( \text{deg} \) have degree \( \_p \neq 0 \) by simp
  then show \( \neg \_p \ 	ext{dvd } 1 \) unfolding \text{poly-dvd-1} by auto
  fix \( b \) assume \( b \ 	ext{dvd} \ 	ext{cf-pos-poly} \ p \)
  also note \( \text{cf-pos-poly-dvd} \)
  finally have \( b \ 	ext{dvd} \ p \).
  with \( \text{irr} \) [unfolded irreducible-altdef] have \( \_p \ 	ext{dvd } 1 \) unfolding \text{poly-dvd-1} by auto
  then show \( \_p \ 	ext{dvd } b \lor b \ 	ext{dvd } 1 \) by (auto dest: dvd-trans[OF cf-pos-poly-dvd])

qed

locale \text{dvd-preserving-hom} = \text{comm-semiring-1-hom} +
  assumes \( \text{hom-eq-mult-hom-imp} \): \( \text{hom } x = \text{hom } y \ast \text{hz} \Rightarrow \exists z. \text{hz} = \text{hom } z \land x = y \ast z \)
begin

lemma \( \text{hom-dvd-hom-iff} \) [simp]: \( \text{hom } x \ 	ext{dvd} \ 	ext{hom} \ y \iff x \ \text{dvd} \ y \)

proof
  assume \( \text{hom } x \ 	ext{dvd} \ 	ext{hom} \ y \)
  then obtain \( \text{hz} \) where \( \text{hom } y = \text{hom } x \ast \text{hz} \) by (elim dvdE)
  from \( \text{hom-eq-mult-hom-imp} \) [OF this] obtain \( z \)
  where \( \text{hz} = \text{hom } z \) and \( \text{mult}: y = x \ast z \) by auto
  then show \( x \ 	ext{dvd} \ y \) by auto

qed auto

sublocale unit-preserving-hom

proof unfold-locales
  fix \( a \) assume \( \text{hom } a = 0 \)
  then have \( 0 \ 	ext{dvd} \ a \) by auto
  then have \( a = 0 \) by auto

qed

sublocale zero-hom-0
-proof unfold-locales
  fix \( a :: 'a \)
  assume \( \text{hom } a = 0 \)
  then have \( \text{hom } 0 \ 	ext{dvd} \ a \) by auto
  then have \( a = 0 \) by auto

qed
end

lemma smult-inverse-monom: \( p \neq 0 \implies \text{smult}(\text{inverse } c) (p::\text{rat poly}) = 1 \iff p = [: c :] \)
proof (cases \( c=0 \))
case True thus \( p \neq 0 \implies \) ?thesis by auto
next
case False ?thesis by (metis left-inverse right-inverse smult-1 smult-1-left smult-smult)
qed

lemma of-int-monom: \( \text{of-int-poly } p = [: \text{rat-of-int } c :] \iff p = [: c :] \) by (induct p, auto)

lemma degree-0-content:
fixes \( p :: \text{int poly} \)
assumes \( \text{deg}: \text{degree } p = 0 \) shows \( \text{content } p = \text{abs}(\text{coeff } p 0) \)
proof-
from \( \text{deg} \) obtain \( a \) where \( p = [: a :] \) by (auto dest: \text{degree0-coeffs})
show ?thesis by (auto simp: \( p \))
qed

lemma prime-elem-imp-gcd-eq:
fixes \( x :: \text{\'a::ring-gcd} \)
shows \( \text{prime-elem } x \implies \gcd x y = \text{normalize } x \lor \gcd x y = 1 \)
using prime-elem-imp-coprime [of \( x y \)]
by (auto simp add: \text{gcd-proj1-iff intro: coprime-imp-gcd-eq-1})

lemma irreducible-pos-gcd:
fixes \( p q :: \text{int poly} \)
assumes \( \text{ir}: \text{irreducible } p \) and \( \text{pos}: \text{lead-coeff } p > 0 \) shows \( \gcd p q \in \{1, p\} \)
proof-
from \( \text{pos} \) have \([: \text{sgn } (\text{lead-coeff } p):] = 1 \) by auto
with prime-elem-imp-gcd-eq[of \( p \), unfolded prime-elem-iff-irreducible, OF \( \text{ir}, of \( q \)]
show ?thesis by (auto simp: \text{normalize-poly-def})
qed

lemma irreducible-pos-gcd-twice:
fixes \( p q :: \text{int poly} \)
assumes \( \text{p}: \text{irreducible } p \) lead-coeff \( p > 0 \) and \( \text{q}: \text{irreducible q lead-coeff q > 0} \)
shows \( \gcd p q = 1 \lor p = q \)
proof (cases \( \gcd p q = 1 \))
case False note \( pq \) = this
have \( p = \gcd p q \) using irreducible-pos-gcd [OF \( p, of q \)] \( pq \)
  by auto
also have \( \ldots = q \) using irreducible-pos-gcd [OF \( q, of p \)] \( pq \)
by (auto simp add: ac-simps)
finally show ?thesis by auto
qed simp

interpretation of-rat-hom: field-hom-0' of-rat..

lemma poly-zero-imp-not-unit:
assumes poly p x = 0 shows \( \neg p \) dvd 1
proof (rule notI)
assume p dvd 1
from poly-hom.hom-dvd-I[OF this] have poly p x dvd 1 by auto
with assms show False by auto
qed

lemma poly-prod-mset-zero-iff:
fixes x :: 'a :: idom
shows poly (prod-mset F) x = 0 \iff \( \exists f \in \# F. \) poly f x = 0
by (induct F, auto simp: poly-mult-zero-iff)

lemma algebraic-imp-represents-irreducible:
fixes x :: 'a :: field-char-0
assumes algebraic x
shows \( \exists p. p \) represents x \land irreducible p
proof -
from assms obtain p
where px: ipoly p x = 0 and p0: p \neq 0 unfolding algebraic-altdef-ipoly by auto
from poly-zero-imp-not-unit[OF px0]
have \( \neg p \) dvd 1 by (auto dest: of-int-poly-hom.hom-dvd-I[where 'a = 'a])
from mset-factors-exist[OF p0 this]
obtain F where F: mset-factors F p by auto
then have p = prod-mset F by auto
also have (af-int-poly ... :: 'a poly) = prod-mset (image-mset of-int-poly F) by simp
finally have poly ... x = 0 using px0 by auto
from this[unfolded poly-prod-mset-zero-iff]
obtain f where f \in \# F and fx0: ipoly f x = 0 by auto
with F have irreducible f by auto
with fx0 show ?thesis by auto
qed

lemma algebraic-imp-represents-irreducible-cf-pos:
assumes algebraic (x::'a::field-char-0)
shows \( \exists p. p \) represents x \land irreducible p \land lead-coeff p > 0 \land primitive p
proof -
from algebraic-imp-represents-irreducible[OF assms(1)]
obtain p where px: p represents x and irr: irreducible p by auto
let ?p = cf-pos-poly p
from px irr represents-imp-degree
have 1: \( ?p \) represents \( x \) and 2: irreducible \( ?p \) and 3: \( \text{cf-pos} \) \( ?p \)
by (auto intro: irreducible-cf-pos-poly)
then show \( \text{thesis} \) by (auto intro: \( \text{ex1[of - ?p]} \) simp: \( \text{cf-pos-def} \))
qed

lemma \( \text{gcd-of-int-poly} \):
\( \text{gcd} (\text{of-int-poly} \ f \  ::  \ \{\text{field-char-0, field-gcd}\} \ \text{poly}) = \text{smult} (\text{inverse} (\text{of-int} (\text{lead-coeff} (\text{gcd} \ f \ g)))) (\text{of-int-poly} (\text{gcd} \ f \ g)) \)
proof
let \( ?ia = \text{of-int-poly} :: - \Rightarrow 'a \ \text{poly} \)
let \( ?ir = \text{of-int-poly} :: - \Rightarrow \text{rat \ poly} \)
let \( ?ra = \text{map-poly} \ \text{of-rat} :: - \Rightarrow 'a \ \text{poly} \)
have id: \( ?ia \ x = ?ra (\ ?ir \ x ) \) for \( x \) by (subst map-poly-map-poly, auto)
show \( \text{thesis} \)
unfolding id
unfolding \( \text{of-rat-hom.map-poly-gcd[symmetric]} \)
unfolding \( \text{gcd-rat-to-gcd-int} \) by (auto simp: \( \text{hom-distribs} \))
qed

lemma \( \text{algebraic-imp-represents-unique} \):
fixes \( x :: 'a :: \{\text{field-char-0, field-gcd}\} \)
assumes algebraic \( x \)
shows \( \exists ! \ p. \ p \) represents \( x \wedge \) irreducible \( p \wedge \) lead-coeff \( p \) \( > \) 0 (is \( \text{Ex1 ?p} \))
proof
from assms obtain \( p \)
where \( p: \ ?p \) and cf\( p: \ \text{cf-pos} \ p \)
by (auto simp: \( \text{cf-pos-def dest: algebraic-imp-represents-irreducible-cf-pos} \))
show \( \text{thesis} \)
proof (rule \( \text{ex1I} \))
show \( ?p \ p \) by fact
fix \( q \)
assume \( q: \ ?p \ q \)
then have \( q \) represents \( x \) by auto
from represents-imp-degree[OF this] \( q \) irreducible-content[of \( q \)]
have cf\( q: \ \text{cf-pos} \ q \) by (auto simp: \( \text{cf-pos-def} \))
show \( q = p \)
proof (rule ccontr)
let \( ?ia = \text{map-poly} \ \text{of-int} :: \ \text{int \ poly} \Rightarrow 'a \ \text{poly} \)
assume \( q \neq p \)
with irreducible-pos-gcd-twice[of \( p \ q \) \( p \ q \) cf\( p \) cf\( q \) have gcd: \( \text{gcd} \ p \ q \) = \( 1 \) by auto
from \( p \ q \) have \( rt: \ \text{ipoly} \ p \ x = 0 \ \text{ipoly} \ q \ x = 0 \) unfolding \( \text{represents-def} \) by auto
define \( c :: 'a \) where \( c = \text{inverse} (\text{of-int} (\text{lead-coeff} (\text{gcd} \ p \ q))) \)
have \( rt: \ \text{poly} \ (\ ?ia \ p ) \ x = 0 \ \text{poly} \ (\ ?ia \ q ) \ x = 0 \) using \( rt \) by auto
hence \( [: -x, f:] \) ded ?ia \( p \ [: -x, f:] \) ded ?ia \( q \)
unfolding \( \text{poly-eq-0-iff-ded} \) by auto
hence \( [: -x, f:] \) ded \( \text{gcd} \ (\ ?ia \ p ) \ (\ ?ia \ q ) \) by (rule gcd-greatest)
also have \( \ldots = \text{smult} \ c \ (\ ?ia \ (\text{gcd} \ p \ q)) \) unfolding \( \text{gcd-of-int-poly} \ c \)-def ..
also have \( \text{ia \ (gcd \ p \ q) = 1} \) by (simp add: gcd)
also have \( \text{smult \ c \ 1 = [: c :]} \) by simp
finally show \( \text{False using c-def \ gcd \ by \ (simp \ add: \ dvd-iff-poly-eq-0)} \)
qed
qed
qed
lemma ipoly-poly-compose:
fixes \( x :: 'a :: \text{idom} \)
shows \( \text{ipoly \ (p \circ p q) x = ipoly \ p \ \text{ipoly} q \ x} \)
proof (induct p)
case (pCons a p)
have \( \text{ipoly \ ((pCons a p) \circ p q) x = of-int \ a + \text{ipoly} \ (q \ast p \circ p q) \ x} \) by (simp add: hom-distribs)
also have \( \text{ipoly} \ (q \ast p \circ p q) \ x = \text{ipoly} q \ x \ast \text{ipoly} \ (p \circ p q) \ x \) by (simp add: hom-distribs)
also have \( \text{ipoly} \ (p \circ p q) \ x = \text{ipoly} p \ \text{ipoly} q \ x \) unfolding pCons
also have \( \text{of-int} \ a + \text{ipoly} q \ x \ast \ldots \ast = \text{ipoly} \ (pCons a p) \ (\text{ipoly} q \ x) \)
unfolding map-poly-pCons[OF pCons(1)] by simp
finally show \( \text{case}\).
qed simp

Polynomial for unary minus.

definition poly-uminus :: 'a :: \text{ring-1} \poly \Rightarrow \ 'a \\poly
where \( \text{poly-uminus} \ p \equiv \sum i \leq \text{degree} \ p. \ \text{monom} \ (((-1)^i \ast \text{coeff} \ p \ i) \ i) \)

lemma poly-uminus-pCons-pCons[simp]:
poly-uminus \ (pCons a \ (pCons b p)) = pCons a \ (pCons (-b) \ (poly-uminus \ p)) \ (\is \ ?l = ?r)
proof(cases \ p = 0)
case False
then have \( \text{deg: \ degree} \ (pCons a \ (pCons b p)) = \text{Suc} \ (\text{Suc} \ \text{degree} \ p) \) by simp
show \( \text{thesis} \)
by (unfold poly-uminus-def deg sum.atMost-Suc-shift monom-Suc monom-0 sum-pCons-0-commute, simp)
next
case True
then show \( \text{thesis} \) by (auto simp add: poly-uminus-def monom-0 monom-Suc)
qed

fun poly-uminus-inner :: 'a :: \text{ring-1} \\text{list} \Rightarrow \ 'a \\poly
where \( \text{poly-uminus-inner} \ [] = \emptyset \)
| \( \text{poly-uminus-inner} \ [a] = [:-a:] \)
| \( \text{poly-uminus-inner} \ (a\#b\#cs) = pCons a \ (pCons \ (-b) \ (\text{poly-uminus-inner} \ cs)) \)

lemma poly-uminus-code[code,simp]: \( \text{poly-uminus} \ p = \text{poly-uminus-inner} \ \text{coeffs} \ p \)
proof-
have \( \text{poly-uminus} \ \text{(Poly \ as) = poly-uminus-inner \ as \ as :: 'a \list} \)
proof (induct length as arbitrary;as rule: less-induct)

20
case less
show ?case
proof (cases as)
  case Nil
  then show ?thesis by (simp add: poly-uminus-def)
next
  case [simp]: (Cons a bs)
  show ?thesis
  proof (cases bs)
    case Nil
    then show ?thesis by (simp add: poly-uminus-def monom-0)
  next
    case [simp]: (Cons b cs)
    show ?thesis by (simp add: less)
  qed
qed
qed

from this[of coeffs p]
show ?thesis by simp
qed

lemma poly-uminus-inner-0[simp]: poly-uminus-inner as = 0 ←→ Poly as = 0
  by (induct as rule: poly-uminus-inner.induct, auto)

lemma degree-poly-uminus-inner[simp]: degree (poly-uminus-inner as) = degree (Poly as)
  by (induct as rule: poly-uminus-inner.induct, auto)

lemma ipoly-poly-uminus-inner[simp]:
  ipoly (poly-uminus-inner as) (x::'a::comm-ring-1) = ipoly (Poly as) (−x)
  by (induct as rule: poly-uminus-inner.induct, auto simp: hom-distribs ring-distribs)

lemma represents-uminus: assumes alg: p represents x
  shows (poly-uminus p) represents (−x)
proof –
  from representsD[OF alg] have p ≠ 0 and rp: ipoly p x = 0 by auto
  hence 0: poly-uminus p ≠ 0 by simp
  show ?thesis
    by (rule representsI[OF - 0], insert rp, auto)
qed

lemma content-poly-uminus-inner[simp]:
  fixes as :: 'a :: ring-gcd list
  shows content (poly-uminus-inner as) = content (Poly as)
  by (induct as rule: poly-uminus-inner.induct, auto)

  Multiplicative inverse is represented by reflect-poly.

lemma inverse-pow-minus: assumes x ≠ (0 :: 'a :: field)
and \( i \leq n \)
shows \( \text{inverse } x^n \cdot x^i = \text{inverse } x^{n-i} \)
using assms by (simp add: field-class.field-divide-inverse power-diff power-inverse)

lemma (in inj-idom-hom) reflect-poly-hom:
reflect-poly (map-poly hom p) = map-poly hom (reflect-poly p)
proof –
obtain xs where xs: rev (coeffs p) = xs by auto
show ?thesis unfolding reflect-poly-def coeffs-map-poly-hom rev-map
xs by (induct xs, auto simp: hom-distribs)
qed

lemma ipoly-reflect-poly: assumes x: (x :: 'a :: field-char-0) \( \neq \) 0
shows ipoly (reflect-poly p) x = x ^ (degree p) * ipoly p (inverse x) (is \( \langle l = \rangle r \)
proof –
let \( \text{?or} = \text{of-int :: int } \Rightarrow 'a \)
have hom: inj-idom-hom \( \text{?or} .. \).
show ?thesis
using poly-reflect-poly-nz[OF x, of map-poly \( \text{?or} \) p] by (simp add: inj-idom-hom.reflect-poly-hom[OF hom])
qed

lemma represents-inverse: assumes x: x \( \neq \) 0
and alg: p represents x
shows (reflect-poly p) represents (inverse x)
proof (intro representsI)
from representsD[OF alg] have p \( \neq \) 0 and rp: ipoly p x = 0 by auto
then show reflect-poly p \( \neq \) 0 by (metis reflect-poly-0 reflect-poly-at-0-eq-0_iff)
show ipoly (reflect-poly p) (inverse x) = 0 by (subst ipoly-reflect-poly, insert x, auto simp: rp)
qed

lemma inverse-roots: assumes x: (x :: 'a :: field-char-0) \( \neq \) 0
shows ipoly (reflect-poly p) x = 0 \( \longleftrightarrow \) ipoly p (inverse x) = 0
using x by (auto simp: ipoly-reflect-poly)

context
fixes n :: nat
begin

Polynomial for n-th root.

definition poly-nth-root :: 'a :: idom poly \( \Rightarrow \) 'a poly where
poly-nth-root p = p \( \circ \)_p monom 1 n

lemma ipoly-nth-root:
fixes x :: 'a :: idom
shows ipoly (poly-nth-root p) x = ipoly p (x ^ n)
unfolding poly-nth-root-def ipoly-poly-compose by (simp add: map-poly-monom poly-monom)

22
context
  assumes \( n \): \( n \neq 0 \)
begin
lemma poly-nth-root-0 [simp]: poly-nth-root \( p = 0 \) \( \iff \) \( p = 0 \)
  unfolding poly-nth-root-def
  by (rule pcompose_eq_0, insert \( n \), auto simp: degree_monom_eq)

lemma represents-nth-root:
  assumes \( y \): \( y^\langle n \rangle = x \) and \( alg \): \( p \) represents \( x \)
  shows \( (poly-nth-root \ p) \) represents \( y \)
proof –
  from \( \text{representsD[OF alg]} \) have \( p \neq 0 \) and \( rp \): \( \text{ipoly} \ p \ x = 0 \) by auto
  hence \( 0 \): poly-nth-root \( p \neq 0 \) by simp
  show \( \text{thesis} \)
    by (rule representsI[OF -0], unfold \( \text{ipoly-nth-root} \ y \ \text{rp} \), simp)
qed

lemma represents-nth-root-odd-real:
  assumes \( alg \): \( p \) represents \( x \) and \( odd \): \( odd \ n \)
  shows \( (poly-nth-root \ p) \) represents \( (\text{root} \ n \ x) \)
by (rule represents-nth-root[OF \( \text{odd-real-root-pow}[OF \text{odd}] \ alg \])

lemma represents-nth-root-pos-real:
  assumes \( alg \): \( p \) represents \( x \) and \( pos \): \( x > 0 \)
  shows \( (poly-nth-root \ p) \) represents \( (\text{root} \ n \ x) \)
proof –
  from \( n \) have \( \text{id} \): \( \text{Suc} \ (n - 1) = n \) by auto
  show \( \text{thesis} \)
    proof (rule represents-nth-root[OF \( \text{OF} - alg \])
      show \( \text{root} \ n \ x ^ n = x \) using \( \text{id} \) \( \text{pos} \) by auto
    qed
  qed

lemma represents-nth-root-neg-real:
  assumes \( alg \): \( p \) represents \( x \) and \( neg \): \( x < 0 \)
  shows \( (\text{poly-uminus (poly-nth-root (poly-uminus} p))) \) represents \( (\text{root} \ n \ x) \)
proof –
  have \( rt \): \( \text{root} \ n \ x = - \text{root} \ n \ (-x) \) unfolding \( \text{real-root-minus} \) by simp
  show \( \text{thesis} \) unfolding \( rt \)
    by (rule \( \text{represents-uminus}[OF \text{represents-nth-root-pos-real}[OF \text{represents-uminus}\ [OF} \ \text{alg}]], \text{insert neg, auto})
qed
end

lemma represents-csqr:
  assumes \( alg \): \( p \) represents \( x \) shows \( (poly-nth-root 2 \ p) \) represents \( (\text{csqrt} \ x) \)
by (rule represents-nth-root[OF \( \text{OF} - alg \], auto)
lemma \texttt{represents-sqrt}:
\begin{itemize}
  \item assumes \texttt{alg}: \texttt{p} represents \texttt{x} and \texttt{pos}: \texttt{x} \geq 0
  \item shows (\texttt{poly-nth-root} 2 \texttt{p}) represents (\texttt{sqrt} \texttt{x})
\end{itemize}
\begin{itemize}
  \item by (\texttt{rule represents-nth-root}\{\texttt{OF |- alg}\}, \texttt{insert pos}, \texttt{auto})
\end{itemize}

lemma \texttt{represents-degree}:
\begin{itemize}
  \item assumes \texttt{p} represents \texttt{x}
  \item shows degree \texttt{p} \neq 0
\end{itemize}
\begin{itemize}
  \item proof
  \begin{itemize}
    \item assume degree \texttt{p} = 0
    \item from degree0-coeffs\{\texttt{OF this}\} obtain \texttt{c} where \texttt{p} = [:c:]
    \item by \texttt{auto}
    \item from \texttt{assms}\{\texttt{unfolded represents-def} \texttt{p}\}
    \item show False by \texttt{auto}
  \end{itemize}
\end{itemize}
\begin{itemize}
  \item qed
\end{itemize}

Polynomial for multiplying a rational number with an algebraic number.

\begin{itemize}
  \item definition \texttt{poly-mult-rat-main} where
    \texttt{poly-mult-rat-main} \texttt{n d} \texttt{(f :: 'a :: idom poly)} = (let \texttt{fs} = \texttt{coeffs f}; \texttt{k} = length \texttt{fs} in
    \texttt{poly-of-list} \texttt{(map (\lambda (fi, i). fi * d ^ i * n ^ (k - Suc i)) (zip \texttt{fs} [0 ..< k]))})
\end{itemize}

\begin{itemize}
  \item definition \texttt{poly-mult-rat} :: \texttt{rat} \Rightarrow \texttt{int poly} \Rightarrow \texttt{int poly}
    \texttt{poly-mult-rat} \texttt{r p} \equiv \texttt{case quotient-of} \texttt{r} of
    \texttt{(n, d)} \Rightarrow \texttt{poly-mult-rat-main} \texttt{n d p}
\end{itemize}

\begin{itemize}
  \item lemma \texttt{coeff-poly-mult-rat-main}:
    \texttt{coeff} \texttt{(poly-mult-rat-main} \texttt{n d f}) \texttt{i} = \texttt{coeff} \texttt{f i} * \texttt{n} ^ \texttt{(degree f - i) * d ^ i}
  \item proof --
    \begin{itemize}
      \item have \texttt{id}: \texttt{coeff} \texttt{(poly-mult-rat-main} \texttt{n d f}) \texttt{i} = (\texttt{coeff} \texttt{f i} * \texttt{d ^ i}) * \texttt{n ^ (length (coeffs f) - Suc i)}
      \item unfolding \texttt{poly-mult-rat-main-def} \texttt{Let-def poly-of-list-def coeff-Poly}
      \item unfolding \texttt{nth-default-coeffs-eq}\{\texttt{symmetric}\}
      \item unfolding \texttt{nth-default-def} by \texttt{auto}
      \item show \texttt{thesis} unfolding \texttt{id} by \texttt{(simp add: degree-eq-length-coeffs)}
    \end{itemize}
  \item qed
\end{itemize}

\begin{itemize}
  \item lemma \texttt{degree-poly-mult-rat-main}:
    \texttt{n} \neq 0 \Rightarrow \texttt{degree} \texttt{(poly-mult-rat-main} \texttt{n d f)} =\texttt{(if d = 0 then 0 else degree f)}
  \item proof (\texttt{cases d = 0})
  \begin{itemize}
    \item case True
    \item thus \texttt{thesis} unfolding \texttt{degree-def} \texttt{unfolding} \texttt{coeff-poly-mult-rat-main} \texttt{by simp}
    \item next
    \begin{itemize}
      \item case False
      \item hence \texttt{id}: (d = 0) = False by \texttt{simp}
      \item show \texttt{n} \neq 0 \Rightarrow \texttt{thesis} unfolding \texttt{degree-def coeff-poly-mult-rat-main} \texttt{id}
        \texttt{by (simp add: id)}
    \end{itemize}
  \end{itemize}
  \item qed
\end{itemize}

\begin{itemize}
  \item lemma \texttt{ipoly-mult-rat-main}:
    \texttt{fixes x :: 'a :: \{field, ring-char-0\}}
  \item assumes \texttt{d} \neq 0 and \texttt{n} \neq 0
\end{itemize}
shows ipoly (poly-mult-rat-main n d p) x = of-int n ^ degree p * ipoly p (x * of-int d / of-int n)

proof –
  from assms have d: (if d = 0 then t else f) = f for t f :: 'b by simp
show ?thesis
  unfolding poly-altdel of-int-hom coeff-map-poly-hom mult_assoc[symmetric]
  of-int-mult[symmetric]
  sum-distrib-left
  unfolding of-int-hom.degree-map-poly-hom degree-poly-mult-rat-main[OF assms(2)]
  d
proof (rule sum.cong[OF refl])
  fix i
  assume i ∈ {...degree p}
  hence i: i ≤ degree p by auto
  hence id: of-int n ^ (degree p - i) = (of-int n ^ degree p / of-int n ^ i :: 'a)
    by (simp add: assms(2) power-diff)
  thus of-int (coeff (poly-mult-rat-main n d p) i) * x ^ i = of-int n ^ degree p * of-int (coeff p i) * (x * of-int d / of-int n) ^ i
    unfolding coeff-poly-mult-rat-main
    by (simp add: field-simps)
qed

lemma degree-poly-mult-rat[simp]: assumes r ≠ 0 shows degree (poly-mult-rat r p) = degree p
proof –
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 by auto
  with assms r have n0: n ≠ 0 by simp
  from quot have id: poly-mult-rat r p = poly-mult-rat-main n d p unfolding poly-mult-rat-def by simp
  show ?thesis unfolding id degree-poly-mult-rat-main[OF n0] using d by simp
qed

lemma ipoly-mult-rat:
  assumes r0: r ≠ 0
  shows ipoly (poly-mult-rat r p) x = of-int (fst (quotient-of r)) ^ degree p * ipoly p (x * inverse (of-rat r))
proof –
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 by auto
  from r r0 have n: n ≠ 0 by simp
  from r d n have inv: of-int d / of-int n = inverse r by simp
  from quot have id: poly-mult-rat r p = poly-mult-rat-main n d p unfolding poly-mult-rat-def by simp
  inv[symmetric]
by (simp add: of-rat-divide)

qed

lemma poly-mult-rat-main-0[simp]:
assumes n ≠ 0 d ≠ 0 shows poly-mult-rat-main n d p = 0 ↔ p = 0

proof
assume p = 0 thus poly-mult-rat-main n d p = 0
  by (simp add: poly-mult-rat-main-def)
next
assume 0: poly-mult-rat-main n d p = 0

{ fix i
  from 0 have coeff (poly-mult-rat-main n d p) i = 0 by simp
  hence coeff p i = 0 unfolding coeff-poly-mult-rat-main using assms by simp
}
thus p = 0 by (intro poly-eqI, auto)

qed

lemma poly-mult-rat-0[simp]: assumes r0: r ≠ 0 shows poly-mult-rat r p = 0 ↔ p = 0

proof
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 by auto
  from r r0 have n: n ≠ 0 by simp

  from quot have id: poly-mult-rat r p = poly-mult-rat-main n d p unfolding poly-mult-rat-def by simp
  show ?thesis unfolding id using n d by simp

qed

lemma represents-mult-rat:
assumes r: r ≠ 0 and p represents x shows (poly-mult-rat r p) represents (of-rat r * x)
using assms

unfolding represents-def ipoly-mult-rat[OF r] by (simp add: field-simps)

Polynomial for adding a rational number on an algebraic number. Again, we do not have to factor afterwards.

definition poly-add-rat :: rat ⇒ int poly ⇒ int poly where
poly-add-rat r p ≡ case quotient-of r of (n,d) ⇒
(poly-mult-rat-main d 1 p ◦ p[−n,d:])

lemma poly-add-rat-code[code]: poly-add-rat r p ≡ case quotient-of r of (n,d) ⇒
let p' = (let fs = coeffs p; k = length fs in poly-of-list (map (λ(f, i). f * d ^ (k - Suc i)) (zip fs [0..<k])));
    p'' = p' ◦ p[−n,d:]
in p''

unfolding poly-add-rat-def poly-mult-rat-main-def Let-def by simp
lemma degree-poly-add-rat[simp]: degree (poly-add-rat r p) = degree p
proof
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 d > 0 by auto
  show ?thesis unfolding poly-add-rat-def quot split
    by (simp add: degree-poly-mult-rat-main d)
qed

lemma ipoly-add-rat: ipoly (poly-add-rat r p) x = (of-int (snd (quotient-of r))) ^
  degree p) * ipoly p (x − of-rat r)
proof
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 d > 0 by auto
  have id: ipoly [- n, 1:] (x / of-int d :: 'a) = − of-int n + x / of-int d by simp
  show ?thesis unfolding poly-add-rat-def quot split
    by (simp add: ipoly-mult-rat-main ipoly-poly-compose d r degree-poly-mult-rat-main
       field-simps id of-rat-divide)
qed

lemma poly-add-rat-0[simp]: poly-add-rat r p = 0 ←→ p = 0
proof
  obtain n d where quot: quotient-of r = (n,d) by force
  from quotient-of-div[OF quot] have r: r = of-int n / of-int d by auto
  from quotient-of-denom-pos[OF quot] have d: d ≠ 0 d > 0 by auto
  show ?thesis unfolding poly-add-rat-def quot split
    by (simp add: d pcompose-eq-0)
qed

lemma add-rat-roots: ipoly (poly-add-rat r p) x = 0 ←→ ipoly p (x − of-rat r) = 0
  unfolding ipoly-add-rat using quotient-of-nonzero by auto

lemma represents-add-rat:
  assumes p represents x shows (poly-add-rat r p) represents (of-rat r + x)
  using assms unfolding represents-def ipoly-add-rat by simp

lemmas pos-mult[simplified,simp] = mult-less-cancel-left-pos[of - 0] mult-less-cancel-left-pos[of - - 0]

lemma ipoly-add-rat-pos-neg:
  ipoly (poly-add-rat r p) (x::'a::linordered-field) < 0 ←→ ipoly p (x − of-rat r) < 0
  ipoly (poly-add-rat r p) (x::'a::linordered-field) > 0 ←→ ipoly p (x − of-rat r) > 0
  using quotient-of-nonzero unfolding ipoly-add-rat by auto
lemma $\text{sgn-ipoly-add-rat}[\text{simp}]:$

$\text{sgn} \ (\text{ipoly} \ (\text{poly-add-rat} \ r \ p) \ (x::'a::\text{linordered-field})) = \text{sgn} \ (\text{ipoly} \ p \ (x - \text{of-rat} \ r)) \ (\text{is} \ \text{sgn} \ ?l = \text{sgn} \ ?r)$

using $\text{ipoly-add-rat-pos-neg}[\text{of rat x}]$

by (cases $r$ 0: 'a rule: \text{linorder-cases}, auto simp: $\text{sgn-1-pos}$ $\text{sgn-1-neg}$ $\text{sgn-eq-0-iff}$)

lemma $\text{deg-nonzero-represents}$:

assumes $\text{deg}: \text{degree} \ p \neq 0$

shows $\exists \ x :: \text{complex}. \ p \ \text{represents} \ x$

proof –

let $?p = \text{of-int-poly} \ p :: \text{complex poly}$

from fundamental-theorem-algebra-factorized[of $?p$]

obtain as c where $\text{id} :: \text{smult} \ c \ (\prod a\leftarrow \text{as}. \ [-a, 1]) = ?p$

and $\text{len} :: \text{length} \ as = \text{degree} \ ?p$ by blast

have $\text{degree} \ ?p = \text{degree} \ p$ by simp

with $\text{deg len obtain} \ b \ \text{bs where as; as = b \# bs}$ by (cases as, auto)

have $p \ \text{represents} \ b$ unfolding $\text{represents-def}$ id[symmetric] as using $\text{deg}$ by auto

thus $\text{?thesis}$ by blast

qed

end

4 Resultants

We need some results on resultants to show that a suitable prime for Berlekamp’s algorithm always exists if the input is square free. Most of this theory has been developed for algebraic numbers, though. We moved this theory here, so that algebraic numbers can already use the factorization algorithm of this entry.

4.1 Bivariate Polynomials

theory Bivariate-Polynomials

imports

Polynomial-Interpolation.\text{Ring-Hom-Poly}

Subresultants.\text{More-Homomorphisms}

Berlekamp-Zassenhaus.\text{Unique-Factorization-Poly}

begin

4.1.1 Evaluation of Bivariate Polynomials

definition poly2 :: 'a::\text{comm-semiring-1} \ poly \ poly \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a

where poly2 p x y = poly (poly p :: y ::) x

lemma poly2-by-map: poly2 p x = poly (map-poly (\c. poly c x) p)

apply (rule ext) unfolding poly2-def by (induct p; simp)
lemma poly2-const[simp]: poly2 [:[:a:]:] x y = a by (simp add: poly2-def)
lemma poly2-smult[simp,hom-distribs]: poly2 (smult a p) x y = poly a x * poly2 p x y by (simp add: poly2-def)

interpretation poly2-hom: comm-semiring-hom \(\lambda p.\) poly2 p x y by (unfold-locales; simp add: poly2-def)
interpretation poly2-hom: comm-ring-hom \(\lambda p.\) poly2 p x y..
interpretation poly2-hom: idom-hom \(\lambda p.\) poly2 p x y..

lemma poly2-pCons[simp,hom-distribs]: poly2 (pCons a p) x y = poly a x + y * poly2 p x y by (simp add: poly2-def)
lemma poly2-monom: poly2 (monom a n) x y = poly a x * y ^ n by (auto simp: poly-monom poly2-def)

The following lemma is an extension rule for bivariate polynomials.

lemma poly2-ext: fixes \(p\) \(q\) :: 'a :: \{ring-char-0,idom\} poly poly assumes \(\forall x y.\) poly2 p x y = poly2 q x y shows \(p = q\)
proof (intro poly-ext)
  fix \(r\) \(x\)
  show poly (poly p r) x = poly (poly q r) x
    unfolding poly-poly-as-poly2[symmetric] using assms by auto
qed

abbreviation (input) coeff-lift2 === \(\lambda a.\) [:[:a:]:]

lemma coeff-lift2-lift: coeff-lift2 = coeff-lift o coeff-lift by auto

definition poly-lift = map-poly coeff-lift
definition poly-lift2 = map-poly coeff-lift2

lemma degree-poly-lift[simp]: degree (poly-lift p) = degree p
  unfolding poly-lift-def by (rule degree-map-poly; auto)

lemma poly-lift-0[simp]: poly-lift 0 = 0 unfolding poly-lift-def by simp

lemma poly-lift-0-iff[simp]: poly-lift p = 0 \iff p = 0
  unfolding poly-lift-def by (induct p; simp)

lemma poly-lift-pCons[simp]:
  poly-lift (pCons a p) = pCons [:[:a:]:] (poly-lift p)
  unfolding poly-lift-def map-poly-simps by simp

lemma coeff-poly-lift[simp]:
  fixes \(p::\) 'a :: comm-monoid-add poly
shows \( \text{coeff} (\text{poly-lift } p) \ i = \text{coeff-lift} (\text{coeff } p \ i) \)

unfolding poly-lift-def by simp

lemma pcompose-conv-poly: \( \text{pcompose } p \ q = \text{poly} (\text{poly-lift } p) \ q \)
   by (induction \( p \)) auto

interpretation poly-lift-hom: inj-comm-monomoid-add-hom poly-lift
proof-
  interpret map-poly-inj-comm-monomoid-add-hom coeff-lift..
  show inj-comm-monomoid-add-hom poly-lift by (unfold-locales, auto simp: poly-lift-def hom-distrib)
qed
interpretation poly-lift-hom: inj-comm-semiring-hom poly-lift
proof-
  interpret map-poly-inj-comm-semiring-hom coeff-lift..
  show inj-comm-semiring-hom poly-lift by (unfold-locales, auto simp add: poly-lift-def hom-distrib)
qed
interpretation poly-lift-hom: inj-comm-ring-hom poly-lift..
interpretation poly-lift-hom: inj-idom-hom poly-lift..

lemma (in comm-monomoid-add-hom) map-poly-hom-coeff-lift[simp, hom-distrib]:
  \( \text{map-poly } \text{hom} (\text{coeff-lift } a) = \text{coeff-lift} (\text{hom } a) \) by (cases \( a = \theta \); simp)

lemma (in comm-ring-hom) map-poly-coeff-lift-hom:
  \( \text{map-poly} (\text{coeff-lift } \circ \text{hom}) \ p = \text{map-poly} (\text{map-poly } \text{hom}) \ (\text{map-poly coeff-lift } p) \)
proof (induct \( p \))
case (pCons \( a \) \( p \)) show \(?case\)
  proof (cases \( a = \theta \))
  case True
   hence poly-lift \( p \) \( \neq \theta \) using pCons(1) by simp
   thus \(?thesis\)
    unfolding map-poly-pCons[OF pCons(1)]
    unfolding pCons(2) True by simp
  next case False
   hence coeff-lift \( a \) \( \neq \theta \) by simp
   thus \(?thesis\)
    unfolding map-poly-pCons[OF pCons(1)]
    unfolding pCons(2) by simp
  qed
qed auto

lemma poly-poly-lift[simp]:
  fixes \( p :: 'a :: \text{comm-semiring-0 poly} \)
  shows \( \text{poly} (\text{poly-lift } p) [x:] = [\text{poly } p \ x ] \)
proof (induct \( p \))
case \( \theta \) show \(?case\) by simp
next case (pCons \( a \) \( p \)) show \(?case\)
  unfolding poly-lift-pCons

30
unfolding poly-pCons
unfolding pCons apply (subst mult.commute) by auto
qed

lemma degree-poly-lift2[simp]:
  degree (poly-lift2 p) = degree p unfolding poly-lift2-def by (induct p; auto)

lemma poly-lift2-0[simp]: poly-lift2 0 = 0 unfolding poly-lift2-def by simp

lemma poly-lift2-0-iff[simp]: poly-lift2 p = 0 ↔ p = 0
  unfolding poly-lift2-def by (induct p; simp)

lemma poly-lift2-pCons[simp]: poly-lift2 (pCons a p) = pCons [:a:]: (poly-lift2 p)
  unfolding poly-lift2-def map-poly-simps by simp

lemma poly-lift2-lift:
  poly-lift2 = poly-lift ∘ poly-lift (is ?l = ?r)
proof
  fix p show ?l p = ?r p
    unfolding poly-lift2-def coeff-lift2-lift poly-lift-def by (induct p; auto)
qed

lemma poly2-poly-lift[simp]: poly2 (poly-lift p) x y = poly p y by (induct p; simp)

lemma poly-lift2-nonzero:
  assumes p ≠ 0 shows poly-lift2 p ≠ 0
  unfolding poly-lift2-def
  apply (subst map-poly-zero)
  using assms by auto

4.1.2 Swapping the Order of Variables

definition
  poly-y-x p ≡ ∑ i°≤ degree p. ∑ j°≤ degree (coeff p i) . monom (monom (coeff (coeff p i) j) i) j

lemma poly-y-x-fix-y-deg:
  assumes gdeg: ∀ i°≤ degree p. degree (coeff p i) ≤ d
  shows poly-y-x p = (∑ i°≤ degree p. ∑ j°≤ d. monom (coeff (coeff p i) j) i) j
    (is - = sum (λi. sum (if i) -) -)
  unfolding poly-y-x-def
  apply (rule sum.cong, simp)
  unfolding atMost-iff
proof
  fix i assume i°≤ degree p
  let ?d = degree (coeff p i)
  have {...} = {... ?d} ∪ {Suc ?d .. d} using gdeg[rule-format, OF i] by auto
  also have sum (if i) ... = sum (if i) {... ?d} + sum (if i) {Suc ?d .. d}
by (rule sum.union-disjoint, auto)
also { fix j
  assume \( j \in \{ \ Suc \ ?d \ldots d \} \)
  have \( \text{coeff} \ (\text{coeff} \ p \ i) \ j = 0 \) apply (rule coeff-eq-0) using \( j \) by auto
  hence \( \text{if } i \ j = 0 \) by auto
} hence sum ( \( \text{if } i \) \{ \ Suc \ ?d \ldots d \}) = 0 by auto
finally show sum ( \( \text{if } i \) \{ \ldots \?d \}) = sum ( \( \text{if } i \) \{ \ldots \d \}) by auto
qed

lemma poly-y-x-fixed-deg:
  fixes \( p :: \cdot a :: \text{comm-monoid-add} \cdot \text{poly} \cdot \text{poly} \)
  defines \( d \equiv \text{Max} \{ \text{degree} \ (\text{coeff} \ p \ i) \mid i, i \leq \text{degree} \ p \} \)
  shows poly-y-x \( p = (\sum i \leq \text{degree} \ p. \ \sum j \leq d. \ \text{monom} \ (\text{coeff} \ (\text{coeff} \ p \ i) \ j) \ i) \ j) \)
    apply (rule poly-y-x-fix-y-deg, intro allI impI)
    unfolding d-def
    by (subst Max-ge, auto)

lemma poly-y-x-swapped:
  fixes \( p :: \cdot a :: \text{comm-monoid-add} \cdot \text{poly} \cdot \text{poly} \)
  defines \( d \equiv \text{Max} \{ \text{degree} \ (\text{coeff} \ p \ i) \mid i, i \leq \text{degree} \ p \} \)
  shows poly-y-x \( p = (\sum j \leq d. \ \sum i \leq \text{degree} \ p. \ \text{monom} \ (\text{coeff} \ (\text{coeff} \ p \ i) \ j) \ i) \ j) \)
    using poly-y-x-fixed-deg[of \( p \), folded d-def]
    sum.swap by auto

lemma poly2-poly-y-x[simp]: poly2 (poly-y-x \( p \)) \( x \ y = \text{poly2} \ p \ y \ x \)
  using [[unfold-abs-def = false]]
    apply (subst(3) poly-as-sum-of-monoms[symmetric])
    apply (subst poly-as-sum-of-monoms[symmetric, of coeff p -])
    unfolding poly-y-x-def
    unfolding coeff-sum monom-sum
    unfolding poly2-hom.hom-sum
    apply (rule sum.cong, simp)
    apply (rule sum.cong, simp)
    unfolding poly2-monom poly-monom
    unfolding mult.assoc
    unfolding mult.commute..

context begin
private lemma poly-monom-mult:
  fixes \( p :: \cdot a :: \text{comm-semiring-1} \)
  shows poly (monom \( p \ i * q \ ~' j \)) \( y = \text{poly} \ (\text{monom} \ p \ j * [:y:] \ ~' i) \ (\text{poly} \ q \ y) \)
  unfolding poly-hom.hom-mult
  unfolding poly-monom
  apply (subst mult.assoc)
  apply (subst(2) mult.commute)
  by (auto simp: mult.assoc)

lemma poly-poly-y-x:
fixes \( p :: 'a :: \text{comm-semiring-1} \) poly poly
shows poly (poly (poly-y-x p) q) y = poly (poly p [y;]) (poly q y)
apply(subst(5) poly-as-sum-of-monomms[symmetric])
apply(subst poly-as-sum-of-monomms[symmetric,of coeff p -])
unfolding poly-y-x-def
unfolding coeff-sum monom-sum
unfolding poly-hom.hom-sum
apply(rule sum.cong, simp)
unfolding atMost-iff
unfolding poly2-monom poly-monom
apply(subst poly-monom-mult).

end

interpretation poly-y-x-hom: zero-hom poly-y-x by (unfold-locales, auto simp: poly-y-x-def)
interpretation poly-y-x-hom: one-hom poly-y-x by (unfold-locales, auto simp: poly-y-x-def monom-0)

lemma map-poly-sum-commute:
assumes \( h \ 0 = 0 \ \forall p q, h (p + q) = h p + h q \)
shows \( \lambda i. \text{map-poly } h (f i) S = \text{map-poly } h (\text{sum } f S) \)
using map-poly-add[OF assms] by auto

lemma poly-y-x-const: poly-y-x [:p:] = poly-lift p (is ?l = ?r)
proof –
have \( ?l = (\sum j \leq \text{degree } p. \text{monom [coeff } p j:] j) \)
unfolding poly-y-x-def by (simp add: monom-0)
also have \( ... = \text{poly-lift } (\sum x \leq \text{degree } p. \text{monom } (\text{coeff } p x) x) \)
unfolding poly-lift-hom.hom-sum unfolding poly-lift-def by simp
also have \( ... = \text{poly-lift } p \) unfolding poly-as-sum-of-monomms..
finally show \( \text{thesis}. \)
qed

lemma poly-y-x-pCons:
shows poly-y-x (pCons a p) = poly-lift a + map-poly (pCons 0) (poly-y-x p)
proof(cases p = 0)
interpret ml: map-poly-comm-monoid-add-hom coeff-lift..
interpret mc: map-poly-comm-monoid-add-hom pCons 0..
interpret mm: map-poly-comm-monoid-add-hom \( \lambda x. \text{monom } x i \) for i..
{ case False show \( \text{thesis} \)
apply(subst(1) poly-y-x-fixed-deg)
apply(unfold degree-pCons-eq[OF False])
apply(subst(2) atLeast0AtMost[symmetric])
apply(subst atLeastAtMost-insertL[OF le0,symmetric])
apply(subst sum.insert,simp,simp)
}
apply\(\text{unfold coeff-pCons-0}\)
apply\(\text{unfold monom-0}\)
apply\(\text{fold poly-lift-hom.map-poly-hom-monom poly-lift-def}\)
apply\(\text{fold poly-lift-hom.hom-sum}\)
apply\(\text{subst poly-as-sum-of-monom’s, subst Max-ge,simp,simp,force,simp}\)
apply\(\text{rule cong[of }\lambda x.\text{ poly-lift } a + x,\text{ OF refl]}\)
apply\(\text{simp only: image-Suc-atLeastAtMost \{symmetric\}}\)
apply\(\text{unfold atLeast0AtMost}\)
apply\(\text{unfold o-def}\)
apply\(\text{unfold coeff-pCons-Suc}\)
apply\(\text{unfold monom-Suc}\)
apply\(\text{subst poly-y-x-fix-y-deg\{of - Max \{degree (coeff (pCons a p) i) | i. i \leq Suc (degree p)\}\} of - Suc -}\)
apply\(\text{intro allI impI}\)
apply\(\text{rule Max.coboundedI}\)
by \(\text{auto simp: hom-distribs intro: exI of - Suc -}\)
}\)
case True show \(\text{thesis by (simp add: True poly-y-x-const)}\)
qed

\textbf{lemma} poly-y-x-pCons-0: poly-y-x \((\text{pCons 0 p}) = \text{map-poly (pCons 0) (poly-y-x p)}\)
\textbf{proof}(cases \(p=0\))
\text{case } False
interpret mc: \text{map-poly-comm-monoid-add-hom pCons 0}..
interpret mm: \text{map-poly-comm-monoid-add-hom }\lambda x.\text{ monom x i i..}
from False show \(\text{thesis}\)
apply \(\text{unfold poly-y-x-def degree-pCons-eq}\)
apply \(\text{unfold sum.atMost-Suc-shift}\)
by \(\text{simp add: hom-distribs monom-Suc}\)
qed simp

\textbf{lemma} poly-y-x-map-poly-pCons-0: poly-y-x \((\text{map-poly (pCons 0 p)} = \text{pCons 0 (poly-y-x p)}\)
\textbf{proof–}
\text{let } ?l = \lambda i j.\text{ monom (monom (coeff (pCons 0 (coeff p i)) j) i) j}\n\text{let } ?r = \lambda i j.\text{ pCons 0 (monom (monom (coeff (coeff p i) j) i) j) j}\n\text{have *: }\text{ (}\sum j\leq\text{degree (pCons 0 (coeff p i))}. \text{?l i j) = (}\sum j\leq\text{degree (coeff p i)\}. \text{?r i j) for i}\n\text{proof(cases coeff p i = 0)}
\text{case True then show }\text{thesis by simp}\nnext
\text{case False}
show \(\text{thesis}\)
apply \(\text{unfold degree-pCons-eq[OF False]}\)
apply \(\text{unfold sum.atMost-Suc-shift simp}\)
apply \(\text{fold monom-Suc}\)..
qed
show \(\text{thesis}\)
apply (unfold poly-y-x-def)
apply (unfold hom-distrib pCons-0-hom.degree-map-poly-hom pCons-0-hom.coeff-map-poly-hom)
unfolding *...

qed

interpretation poly-y-x-hom: comm-monoid-add-hom poly-y-x :: 'a :: comm-monoid-add
proof (unfold-locales)
fix p q :: 'a poly poly
show poly-y-x (p + q) = poly-y-x p + poly-y-x q
proof (induct p arbitrary; q)
  case 0 show ?case by simp
next
  case (pCons a p)
  show ?case
  proof (induct q)
    case q: (pCons b q)
    show ?case
    apply (unfold add-pCons)
    apply (unfold poly-y-x-pCons)
    apply (unfold p)
    by (simp add: poly-y-x-const ac-simps hom-distrib)
  qed
  qed

poly-y-x is bijective.

lemma poly-y-x-poly-lift:
fixes p :: 'a :: comm-monoid-add poly
shows poly-y-x (poly-lift p) = [:p:]
apply (subst poly-y-x-fix-y-deg[of - 0], force)
apply (subst (10) poly-as-sum-of-monomoms[symmetric])
by (auto simp add: monom-sum monom-0 hom-distrib)

lemma poly-y-x-id[simp]:
fixes p :: 'a :: comm-monoid-add poly poly
shows poly-y-x (poly-y-x p) = p
proof (induct p)
  case 0
  then show ?case by simp
next
  case (pCons a p)
  interpret mm: map-poly-comm-monoid-add-hom λx. monom x i for i..
  interpret mc: map-poly-comm-monoid-add-hom pCons 0 ..
  have pCons-as-add: pCons a p = [:a:] + pCons 0 p by simp
  from pCons show ?case
  apply (unfold pCons-as-add)
  by (simp add: poly-y-x-pCons poly-y-x-poly-lift poly-y-x-map-poly-pCons-0 hom-distrib)
qed
interpretation poly-y-x-hom:
  bijective poly-y-x :: 'a :: comm-monoid-add poly poly ⇒ -
  by (unfold bijective-eq-bij, auto intro: o-bij [of poly-y-x])

lemma inv-poly-y-x [simp]: Hilbert-Choice.inv poly-y-x = poly-y-x by auto

interpretation poly-y-x-hom: comm-monoid-add-isom poly-y-x
  by (unfold-locales, auto)

lemma pCons-as-add:
  fixes p :: 'a :: comm-semiring-1 poly
  shows pCons a p = [a:] + monom 1 1 * p by (auto simp: monom-Suc)

lemma mult-pCons-0: (∗) (pCons 0 1) = pCons 0 by auto

lemma pCons-0-as-mult:
  shows pCons (0 :: 'a :: comm-semiring-1) = (λp. pCons 0 1 * p) by auto

lemma map-poly-pCons-0-as-mult:
  fixes p :: 'a :: comm-semiring-1 poly
  shows map-poly (pCons 0) p = [pCons 0 1] * p by (simp, subst)

lemma poly-y-x-monom:
  fixes a :: 'a :: comm-semiring-1 poly
  shows poly-y-x (monom a n) = smult (monom 1 n) (poly-lift a)
  proof (cases a = 0)
    case True then show ?thesis by simp
  next
    case False
    interpret map-poly-comm-monoid-add-hom λx. c * x for c :: 'a poly..
    from False show ?thesis
      apply (unfold poly-y-x-def)
      apply (unfold degree-monom-eq)
      apply (subst(2) lessThan-Suc-atMost[symmetric])
      apply (unfold sum.lessThan-Suc)
      apply (subst sum.neutral_force)
      apply (subst(14) poly-as-sum-of-monom[antisymmetric])
      apply (unfold smult-as-map-poly)
      by (auto simp: monom-altdef unfolded x-as-monom x-pow-n, antisymmetric] hom-distrib)
  qed

lemma poly-y-x-smult:
  fixes c :: 'a :: comm-semiring-1 poly
  shows poly-y-x (smult c p) = poly-lift c * poly-y-x p (is ?l = ?r)
  proof
    have smult c p = (∑ i≤degree p. monom (coeff (smult c p) i) i)

by (metis (no-types, lifting) degree-smult-le poly-as-sum-of-monom's sum.cong)
also have ... = (∑ i≤degree p. monom (c * coeff p i) i)
  by auto
also have poly-y-x ... = poly-lift (c * (∑ i≤degree p. smult (monom 1 i) (poly-lift (coeff p i))))
  by (simp add: poly-y-x-monom hom-distrib)
also have ... = poly-lift c * poly-y-x (∑ i≤degree p. monom (coeff p i) i)
  by (simp add: poly-y-x-monom hom-distrib)
finally show ?thesis by (simp add: poly-as-sum-of-monom's)
qed

interpretation poly-y-x-hom:
  comm-semiring-isom poly-y-x :: 'a :: comm-semiring-1 poly poly ⇒ -
proof
  fix p q :: 'a poly
  show poly-y-x (p * q) = poly-y-x p * poly-y-x q
  proof (induct p)
    case (pCons a p)
    show ?case
      apply (unfold mult-pCons-left)
      apply (unfold hom-distrib)
      apply (unfold poly-y-x-smult)
      apply (unfold poly-y-x-pCons-0)
      apply (unfold pCons)
      by (simp add: poly-y-x-pCons map-poly-pCons-0-as-mult field-simps)
  qed simp
qed

interpretation poly-y-x-hom: comm-ring-isom poly-y-x..
interpretation poly-y-x-hom: idom-isom poly-y-x..

lemma Max-degree-coef-pCons:
  Max { (degree (coeff (pCons a p) i) | i. i ≤ degree (pCons a p)) | (degree a) (Max {degree (coeff p x) | x. x ≤ degree p})}
proof (cases p = 0)
case False show ?thesis
  unfolding degree-pCons-eq[OF False]
  unfolding image-Collect[symmetric]
  unfolding atMost-def[symmetric]
  apply (subst(1) atLeast0AtMost[symmetric])
  unfolding atLeastAtMost-insertL[OF le0,symmetric]
  unfolding image-insert
  apply (subst Max-insert,simp,simp)
  unfolding image-Suc-atLeastAtMost [symmetric]
  unfolding image-image
  unfolding atLeast0AtMost by simp
qed simp
lemma degree-poly-y-x:
  fixes p :: 'a :: comm_ring_1 poly poly
  assumes p ≠ 0
  shows degree (poly-y-x p) = Max { degree (coeff p i) | i. i ≤ degree p }
  (is s = ?d p)
  using asms
proof (induct p)
  interpret rhm: map-poly-comm-ring-hom coeff-lift ..
  let ?f = λp i j. monom (monom (coeff (coeff p i) j) i) j
  case (pCons a p)
  show ?case
  proof (cases rule: linorder_cases[of degree ?a degree ?p])
    case less
    have dle: degree a ≤ degree (poly-y-x p)
      apply (rule le_trans[OF less_imp_le[OF less[simplified]]])
    using degree-map-poly_le by auto
    show ?thesis
      unfolding poly-y-x-pCons
      unfolding degree-add_eq_right[OF less]
      unfolding Max_degree_coeff-pCons
      unfolding IH[symmetric]
      unfolding max_absorb2[OF dle]
      apply (rule degree_map_poly) by auto
    next case equal
    have dege: degree ?a = degree a by auto
    have degp: degree (poly-y-x p) = degree a
      using equal[unfolded dege]
    using degree_map_poly[of pCons 0 poly-y-x p] by auto
    have ∗: degree (?a + ?p) = degree a
    proof (cases a = 0)
      case True
      show ?thesis using equal unfolding True by auto
    next case False
      show ?thesis
        apply (rule antisym)
        apply (rule degree_add_le, simp, fold equal, simp)
        apply (rule le_degree)
        unfolding coeff_add
        using False
        by auto
    qed
  show ?thesis unfolding poly-y-x-pCons
    unfolding *
    unfolding Max_degree_coeff-pCons
  qed
unfolding IH[symmetric]
unfolding degp by auto
next case greater
have dge: degree a ≥ degree (poly-y-x p)
apply (rule le-trans[OF - less-imp-le[OF greater[simplified]]])
by auto
show ?thesis
unfolding poly-y-x-pCons
unfolding degree-add-eq-left[OF greater]
unfolding Max-degree-coeff-pCons
unfolding IH[symmetric]
unfolding max-absorb1[OF dge] by simp
qed
qed
qed auto
end

4.2 Resultant

This theory contains facts about resultants which are required for addition and multiplication of algebraic numbers.

The results are taken from the textbook [2, pages 227ff and 235ff].

theory Resultant
imports
  HOL-Computational-Algebra.Fundamental-Theorem-Algebra
  Subresultants.Resultant-Prelim
  Berlekamp-Zassenhaus.Unique-Factorization-Poly
  Bivariate-Polynomials
begin

4.2.1 Sylvester matrices and vector representation of polynomials

definition vec-of-poly-rev-shifted where
vec-of-poly-rev-shifted p n j ≡
vec n (λi. if i ≤ j ∧ j ≤ degree p + i then coeff p (degree p + i - j) else 0)

lemma vec-of-poly-rev-shifted-dim[simp]: dim-vec (vec-of-poly-rev-shifted p n j) = n
unfolding vec-of-poly-rev-shifted-def by auto

lemma col-sylvester:
  fixes p q
  defines m ≡ degree p and n ≡ degree q
  assumes j: j < m+n
  shows col (sylvester-mat p q) j =
   vec-of-poly-rev-shifted p n j @<v vec-of-poly-rev-shifted q m j (is ?l = ?r)
proof
  note [simp] = m-def[symmetric] n-def[symmetric]
show \( \dimvec \ ?l = \dimvec \ ?r \) by simp
fix \( i \) assume \( i < \dimvec \ ?r \) hence \( i < m+n \) by auto
show \$l \$ \$r \$
  unfolding vec-of-poly-rev-shifted-def
  apply \( \text{subst index-col} \) using \( i \) apply simp using \( j \) apply simp
  apply \( \text{subst sylvester-index-mat} \) using \( i \) apply simp using \( j \) apply simp
  apply \( \text{cases} \ i < n \) apply force using \( i \) by simp
qed

lemma inj-on-diff-nat2: \( \inj-on (\lambda i. (n::nat) - i) \{..n\} \) by \( \text{rule inj-on}/1, \text{auto} \)

lemma image-diff-atMost: \( (\lambda i. (n::nat) - i) \}' \{..n\} = \{..n\} \) (is \( \sum \) \?l \?r )
unfolding set-eq-iff
proof \( (\text{intro allI iffI} \)
  fix \( x \) assume \( x \in \ ?r \)
  thus \( x \in \ ?l \) unfolding image-def mem-Collect-eq
  by \( (\text{intro bexI} [of - n - x], \text{auto} \)
qed \text{auto}

lemma sylvester-sum-mat-upper:
fixes \( p \) \( q \) :: 'a :: comm-semiring-1 poly
defines \( m \equiv \text{degree} \ p \) and \( n \equiv \text{degree} \ q \)
assumes \( i: i < n \)
shows \( \sum j < m+n. \ \text{monom} (\text{sylvester-mat} \ p \ q \ (\text{Suc} \ j)) \) \( (m + n - \text{Suc} \ j)) = \) \( \text{monom} \ 1 \ (n - \text{Suc} \ i) \) \( * \ p \) (is \( \sum \) \?f \?r )
proof 
  have \( n1: n \geq 1 \) using \( i \) by \text{auto}
  define \( ni1 \) where \( ni1 = n - \text{Suc} \ i \)
  hence \( ni1: n-i = \text{Suc} \ ni1 \) using \( i \) by \text{auto}
  define \( l \) where \( l = m+n-1 \)
  hence \( \text{Suc} \ l = m+n \) using \( ni1 \) by \text{auto}
  let \( ?g = \lambda j. \ \text{monom} (\text{coeff} (\text{monom} \ 1 \ (n - \text{Suc} \ i) \ * \ p \ j) \ j) \)
  let \( ?p = \lambda j. \ l-j \)
  have \( \text{sum} \ \?f \{..<m+n\} = \text{sum} \ \?f \{..l\} \)
    unfolding \{\text{symmetric}\} unfolding lessThan-Suc-atMost..
  also \{
    fix \( j \) assume \( j \leq l \)
    have \( \text{if} \ j = ((\lambda j. \ \text{monom} (\text{coeff} (\text{monom} \ 1 \ (n-i) \ * \ p) \ (\text{Suc} \ j)) \ j) \circ \ ?p) \ j \)
      apply \( \text{subst sylvester-index-mat2} \)
      using \( i \) unfolding \( l\)-def \( m\)-def \[\text{symmetric}\] \( n\)-def \[\text{symmetric}\]
      by \( (\text{auto simp add: Suc-diff-Suc}) \)
    also have \( \ldots = (?g \circ \ ?p) \ j \)
      unfolding \( ni1 \)
      unfolding \( \text{coeff-monom-Suc} \)
      unfolding \( ni1\)-def
      using \( i \) by \text{auto}
    finally have \( \text{if} \ j = (?g \circ \ ?p) \ j. \)
  \}
  hence \( \sum j \leq l. \ ?f \ j = (\sum j \leq l. \ (?g \circ \ ?p) \ j) \) using \( l \) by \text{auto}

40
also have ... = \( \sum_{j \leq l} \ ?g_j \)

unfolding l-def
using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost].
also have degree \(?r \leq l\)
using degree-mult-le[of monom 1 \( (n-Suc\ i)\ p \)]
unfolding l-def m-def
unfolding degree-monom-eq[OF one-neq-zero] using i by auto
from poly-as-sum-of-monomes[OF this]
have \( \sum_{j \leq l} \ ?g_j = \ ?r. \)
finally show \(?thesis\).

qed

lemma sylvester-sum-mat-lower:
fixes \( p\ q\ ::\ ′a\ ::\ comm-semiring-1\ poly \)
defines \( m \equiv \text{degree}\ p\) and \( n \equiv \text{degree}\ q\)
assumes \( ni\ :\ n \leq i\) and \( imn\ :\ i < m+n\)
shows \( \sum_{j < m+n.} \text{monom} \ (\text{sylvester-mat} \ p\ q\ (i,j))\ (m+n-Suc\ i) \ast q \) (is sum \(?f\ -\ ?r\))
proof –
define \( l\ where\ l = m+n-1\)
hence \( l\ :\ Suc\ l = m+n\) using imn by auto
define \( mni1\ where\ mni1 = m + n - Suc\ i\)
hence \( mni1\ :\ m+n-i = Suc\ mni1\) using imn by auto
let \(?g = \lambda\ j.\ \text{monom} \ (\text{coeff} \ (\text{monom} 1 \ (m + n - Suc\ i) \ast q))\ j\)\ j
let \(?p = \lambda\ j.\ l-j\)
have \( \text{sum} \ {?f \{..<m+n\}} = \text{sum} \ {?f \{..l\}}\)
unfolding \(?f\ \{..\}\) unfolding lessThan-Suc-atMost..
also \{
fix \( j\) assume \( j : j \leq l\)
have \(?f\ j = ((\lambda\ j.\ \text{monom} \ (\text{coeff} \ (\text{monom} 1 \ (m + n - i) \ast q))\ (Suc\ j))\ j) \circ \ ?p\)\ j
apply (subst sylvester-index-mat2)
using ni imn j unfolding l-def m-def[ symmetric] n-def[symmetric]
by (auto simp add: Suc-diff-Suc)
also have \( ... = (\ ?g \circ \ ?p)\ j\)
unfolding mni1
unfolding coeff-monom-Suc
unfolding mni1-def..
finally have \(?f\ j = \ldots\)\}
hence \( \sum_{j \leq l.} \ ?f\ j = (\sum_{j \leq l.} \ ?g\ j)\) by auto
also have \( ... = (\sum_{j \leq l.} \ ?g\ j)\)
using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost].
also have degree \(?r \leq l\)
using degree-mult-le[of monom 1 \( (m+n-1-i)\ q\)]
unfolding l-def n-def[ symmetric]
unfolding degree-monom-eq[OF one-neq-zero] using ni imn by auto
from poly-as-sum-of-monomes[OF this]
have \( \sum_{j \leq l.} \ ?g\ j = \ ?r. \)
finally show \(?thesis\).
definition vec-of-poly \( p \equiv \) let \( m = \text{degree } p \) in vec (Suc \( m \)) (\( \lambda i. \text{coeff } p \ (m-i) \))

definition poly-of-vec \( v \equiv \) let \( d = \text{dim-vec } v \) in \( \sum i<d. \text{monom} \ (v \ (d - \text{Suc } i)) \)

lemma poly-of-vec-of-poly[simp]:
fixes \( p :: \cdot a :: \text{comm-monoid-add poly} \)
shows poly-of-vec (vec-of-poly \( p \)) = \( p \)
unfolding poly-of-vec-def vec-of-poly-def Let-def
unfolding dim-vec
unfolding lessThan-Suc-atMost
using poly-as-sum-of-monom\(s\)[of \( p \)] by auto

lemma poly-of-vec-0[simp]: poly-of-vec (\( 0 \)) = \( 0 \)
unfolding poly-of-vec-def Let-def
by auto

lemma poly-of-vec-0-if\(f\)[simp]:
fixes \( v :: \cdot a :: \text{comm-monoid-add vec} \)
shows poly-of-vec \( v = 0 \) \( \iff \) \( v = 0 \) (dim-vec \( v \)) (is \( ?v = - \iff - = ?z \))
proof
assume \( ?v = 0 \)
hence \( \forall i \in \{..<\text{dim-vec } v\}. \ v \ (\text{dim-vec } v - \text{Suc } i) = 0 \)
unfolding poly-of-vec-def Let-def
by (subst sum-monom-0-iff[symmetric],auto)
{ fix \( i \) assume \( i < \text{dim-vec } v \)
hence \( \forall i \in \{..<\text{dim-vec } v\}. \ v \ (\text{dim-vec } v - \text{Suc } i) = 0 \) by auto
}
thus \( ?v = ?z \) by auto
next assume \( r: \ v = ?z \)
show \( ?v = 0 \) apply (subst \( r \)) by auto
qed

lemma degree-sum-smaller:
assumes \( n > 0 \) finite \( A \)
shows \( (\forall x. \ x \in A \Rightarrow \text{degree } (f \ x) < n) \Rightarrow \text{degree } (\sum x\in A. \ f \ x) < n \)
using (finite \( A \))
by(induct rule: finite-induct)
(simp-all add: degree-add-less assms)

lemma degree-poly-of-vec-less:
fixes \( v :: \cdot a :: \text{comm-monoid-add vec} \)
assumes \( \text{dim: dim-vec } v > 0 \)
shows degree (poly-of-vec \( v \)) < dim-vec \( v \)
unfolding poly-of-vec-def Let-def
apply(rule degree-sum-smaller)
using dim apply force
apply force
unfolding lessThan-iff
by (metis degree-0 degree-monom-eq dim monom-eq-0-iff)

lemma coeff-poly-of-vec:
coeff (poly-of-vec v) i = (if i < dim-vec v then v $ (dim-vec v - Suc i) else 0) (is ?l = ?r)
proof -
have ?l = (∑ x<dim-vec v. if x = i then v $ (dim-vec v - Suc x) else 0) (is - = ?m)
  unfolding poly-of-vec-def Let-def coeff-sum coeff-monom ..
also have ... = ?r
proof (cases i < dim-vec v)
case False
  show ?thesis
  by (subst sum.neutral, insert False, auto)
next
case True
  show ?thesis
  by (subst sum.remove[of - i], force, force simp: True, subst sum.neutral, insert True, auto)
qed
finally show ?thesis.
qed

lemma vec-of-poly-rev-shifted-scalar-prod:
fixes p v defines q ≡ poly-of-vec v assumes m[|simp|]: degree p = m and n: dim-vec v = n assumes j: j < m+n shows vec-of-poly-rev-shifted p n (n+m−Suc j) · v = coeff (p ∗ q) j (is ?l = ?r)
proof -
have id1: ∀ i. m + i - (n + m − Suc j) = i + Suc j − n
  using j by auto
let ?g = λ i. if i ≤ n + m − Suc j ∧ n − Suc j ≤ i then coeff p (i + Suc j − n) ∗ v $ i else 0
have ?thesis = ( (∑ i=0..<n. ?g i) = ∑ i≤j. coeff p i * (if j − i < n then v $(n − Suc (j − i)) else 0)) (is - = (?l = ?r))
  unfolding vec-of-poly-rev-shifted-def coeff-mult m scalar-prod-def n q-def coeff-poly-of-vec
  by (subst sum.cong, insert id1, auto)
also have ...
proof -
have ?r = (∑ i≤j. (if j − i < n then coeff p i ∗ v $(n − Suc (j − i)) else 0)) (is - = sum |if |-)
  by (rule sum.cong, auto)
also have sum ?f {..j} = sum ?f {{i. i ≤ j ∧ j − i < n} ∪ {i. i ≤ j ∧ ¬ j −
\[ i < n \}

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } \{ F, R \} \]

by (subst sum.union-disjoint, auto)

also have \[ \text{sum } \{ F, R \} + 0 = \text{sum } (\lambda i. \text{coeff } p i * v \} (i + n - \text{Suc } j)) \]

(by subst sum.cong, auto simp: ac_simps)

also have \[ \ldots = \text{sum } (\{ F, R \} \cup \{ F, R' \}) \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } (\{ F, R \} \cap \{ F, R' \}) \]

(by subst sum.cong, auto)

also have \[ \ldots = \text{sum } (\{ F, R \} \cup \{ F, R' \}) \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } (\{ F, R \} \cap \{ F, R' \}) \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } (\{ F, R \} \cup \{ F, R' \}) \]

(by (rule sum.cong, auto)

proof

{ fix \( x \)
  assume \( x > m \)
  from coeff-eq-0[OF this[folded m]]
  have \( \{ F, R \} = 0 \) by simp
}

thus \( \text{thesis} \)

by (subst sum.neutral, auto)

qed

finally have \( r = \text{sum } \{ F, R \} \) by simp

have \( l = \text{sum } \{ G, L \} \)

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by subst sum.union-disjoint, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by subst sum.cong, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by subst sum.union-disjoint, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by (rule sum.cong, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by subst sum.union-disjoint, auto)

also have \[ \ldots = \text{sum } \{ G, L \} \]

(by (rule sum.cong, auto)

proof

{ fix \( x \)
assume $x + \text{Suc } j - n > m$
from coeff-eq-0[OF this[folded m]]
have $?G x = 0$ by simp
}
thus $?\text{thesis}$
by (subst sum.neutral, auto)
qed
finally have $l$: $?l = \text{sum } ?G ?L$ by simp

let $?bij = \lambda i. \, i + n - \text{Suc } j$
{
fix $x$
assume $x: \, j < m + n \ \text{Suc } (x + j) - n \leq m \, x < n \, n - \text{Suc } j \leq x$
define $y$ where $y = x + \text{Suc } j - n$
from $x$ have $x + \text{Suc } j \geq n$ by auto
with $x$ have $xy: \, x = ?bij y$ unfolding $y$-def by auto
from $x$ have $y: \, y \in ?R$ unfolding $y$-def by auto
have $x \in ?bij$ : $\, y$ using $y$ by blast
} note tedious = this
show $?\text{thesis}$ unfolding $l r$
by (rule sum.reindex-cong[of $?bij$], insert $j$, auto simp: inj-on-def tedious)
qed
finally show $?\text{thesis}$ by simp
qed

lemma sylvester-vec-poly:
fixes $p \, q :: 'a :: \text{comm-semiring-0 poly}$
defines $m \equiv \text{degree } p$
and $n \equiv \text{degree } q$
assumes $v: \, v \in \text{carrier-vec } (m+n)$
shows $\text{poly-of-vec } (\text{transpose-mat } (\text{sylvester-mat } p \, q) \,* \, v) =$
$\text{poly-of-vec } (\text{vec-first } v \, n) \,* \, p + \text{poly-of-vec } (\text{vec-last } v \, m) \,* \, q$ (is $?l = ?r$)
proof (rule poly-eqI)
fix $i$

note $mn[simp] = m$-def[symmetric] $n$-def[symmetric]

let $?Tv = \text{transpose-mat } (\text{sylvester-mat } p \, q) \,* \, v$

have $\text{dim: } \text{dim-vec } (\text{vec-first } v \, n) = n \, \text{dim-vec } (\text{vec-last } v \, m) = m \, \text{dim-vec } ?Tv$
$= n + m$
using $v$ by auto

have if-distrib: $\land \, x \, y \, z. \, (if \, x \, then \, y \, else \, (0 :: 'a)) \,* \, z = (if \, x \, then \, y \,* \, z \, else \, 0)$
by auto

show coeff $?l \, i = \text{coeff } ?r \, i$
proof (cases $i < m+n$)

case False

hence $i$-mn: $i \geq m+n$
and $i$-n: $\land \, x. \, x \leq i \land \, x < n \leftrightarrow x < n$
and $i$-m: $\land \, x. \, x \leq i \land \, x < m \leftrightarrow x < m$ by auto

have coeff $?r \, i =$
$(\sum \, x < n. \, \text{vec-first } v \, n \, \$ \, (n - \text{Suc } x) \,* \, \text{coeff } p \, (i - x)) +$
\[
(\sum_{x < m} \text{vec-last } v \ m \ (m - \text{Suc } x) \ * \ \text{coeff } q \ (i - x))
\]
(is - \ sum ?f - + sum ?g -)
unfolding coeff-add coeff-mult Let-def
unfolding coeff-poly-of-vec dim if-distrib
unfolding atMost-def
apply(subst sum.inter-filter[symmetric],simp)
apply(subst sum.inter-filter[symmetric],simp)
unfolding mem-Collect-eq
unfolding i-n i-m
unfolding lessThan-def
by simp
also \{ fix \ x assume \ x: x < n
have coeff p \ (i - x) = 0
apply(rule coeff-eq-0) using \ i-mn \ x unfolding m-def by auto
hence \ ?f x = 0 by auto \}
\}
hence \ sum \ ?f \{..<n\} = 0 by auto
also \{ fix \ x assume \ x: x < m
have coeff q \ (i - x) = 0
apply(rule coeff-eq-0) using \ i-mn \ x unfolding n-def by auto
hence \ ?g x = 0 by auto \}
\}
hence \ sum \ ?g \{..<m\} = 0 by auto
finally have coeff \ ?r \ i = 0 by auto
also from False have \ 0 = coeff \ ?l \ i
unfolding coeff-poly-of-vec dim sum.distrib[symmetric] by auto
finally show \ ?thesis by auto
next case True
  hence coeff \ ?l \ i = (transpose-mat \ \text{sylvester-mat } p \ q) *_v \ v \ \ (n + m - \text{Suc } i)
unfolding coeff-poly-of-vec dim sum.distrib[symmetric] by auto
also have \ ... = coeff \ (p * poly-of-vec \ \text{vec-first } v \ n) + q * poly-of-vec \ (vec-last v m)) \ i
apply(subst index-mult-mat-vec) using True apply simp
apply(subst row-transpose) using True apply simp
apply(subst col-sylvester)
unfolding mn using True apply simp
apply(subst vec-first-last-append[of \ v \ n \ m,symmetric]) using v apply(simp
add: add.commute)
apply(subst scalar-prod-append)
apply (rule carrier-vecI,simp)+
apply (subt vec-of-poly-rev-shifted-scalar-prod,simp,simp) using True apply simp
by simp
also have \ ... =
(\sum_{x \leq i. \ (if \ x < n \ then \ vec-first \ v \ n \ \ (n - \text{Suc } x) \ else \ 0) \ * \ \text{coeff } p \ (i - x))
+ (\sum_{x \leq i. \ (if \ x < m \ then \ vec-last \ v \ m \ \ (m - \text{Suc } x) \ else \ 0) \ * \ \text{coeff } q \ (i - x))
unfolding coeff-poly-of-vec[of vec-first \ v \ n,unfolded dim-vec-first,symmetric]
unfolding coeff-poly-of-vec[of vec-last v m, unfolded dim-vec-last, symmetric]
unfolding coeff-mult[symmetric] by (simp add: mult.commute)
also have ... = coeff ?r i
unfolding coeff-add coeff-mult Let-def
unfolding coeff-poly-of-mult dim..
finally show ?thesis.
qed

4.2.2 Homomorphism and Resultant

Here we prove Lemma 7.3.1 of the textbook.

lemma (in comm-ring-hom) resultant-sub-map-poly:
  fixes p q :: 'a poly
  shows hom (resultant-sub m n p q) = resultant-sub m n (map-poly hom p)
                   (map-poly hom q)
             (is (?l = ?r'))
proof –
  let ?mh = map-poly hom
  have ?l = det (sylvester-mat-sub m n (?mh p) (?mh q))
              unfolding resultant-sub-def
  apply (subst sylvester-mat-sub-map[symmetric]) by auto
  thus ?thesis unfolding resultant-sub-def.
qed

4.2.3 Resultant as Polynomial Expression

context begin

This context provides notions for proving Lemma 7.2.1 of the textbook.

private fun mk-poly-sub where
  mk-poly-sub A l 0 = A
| mk-poly-sub A l (Suc j) = mat-addcol (monom 1 (Suc j)) l (l - Suc j) (mk-poly-sub A l j)

definition mk-poly A = mk-poly-sub (map-mat coeff-lift A) (dim-col A - 1)
                      (dim-col A - 1)

private lemma mk-poly-sub-dim[simp]:
  dim-row (mk-poly-sub A l j) = dim-row A
  dim-col (mk-poly-sub A l j) = dim-col A
by (induct j,auto)

private lemma mk-poly-sub-carrier:
  assumes A ∈ carrier-mat nr nc shows mk-poly-sub A l j ∈ carrier-mat nr nc
  apply (rule carrier-matI) using assms by auto

private lemma mk-poly-dim[simp]:
  dim-col (mk-poly A) = dim-col A

47
\textit{dim-row} (\textit{mk-poly} \(A\)) = \textit{dim-row} \(A\)

\textbf{unfolding} \textit{mk-poly-def} \textbf{by} auto

\textbf{private lemma} \textit{mk-poly-sub-others}[simp]:
\begin{itemize}
  \item \textbf{assumes} \(l \neq j'\) \textbf{and} \(i < \text{dim-row} \(A\)\) \textbf{and} \(j' < \text{dim-col} \(A\)\)
  \item \textbf{shows} \textit{mk-poly-sub} \(A \ l \ j \ A \ (i,j') = \ A \ (i,j')\)
  \item \textbf{using} \textit{assms} \textbf{by} (\textit{induct} \(j\); simp)
\end{itemize}

\textbf{private lemma} \textit{mk-poly-others}[simp]:
\begin{itemize}
  \item \textbf{assumes} \(i : i < \text{dim-row} \(A\)\) \textbf{and} \(j : j < \text{dim-col} \(A\) - 1\)
  \item \textbf{shows} \textit{mk-poly} \(A \ (i,j) = \\ [\text{:} \ A \ (i,j) :] \)
  \item \textbf{unfolding} \textit{mk-poly-def}
  \item \textbf{apply}(\textit{subst} \textit{mk-poly-sub-others})
  \item \textbf{using} \(i\) \(j\) \textbf{by} auto
\end{itemize}

\textbf{private lemma} \textit{mk-poly-delete}[simp]:
\begin{itemize}
  \item \textbf{assumes} \(i : i < \text{dim-row} \(A\)\)
  \item \textbf{shows} \textit{mat-delete} \(\textit{mk-poly} \(A\) \ i (\text{dim-col} \(A\) - 1) = \text{map-mat coeff-lift} \ (\textit{mat-delete} \ A \ i (\text{dim-col} \(A\) - 1))\)
  \item \textbf{apply}(\textit{rule} eq-matI) \textbf{unfolding} \textit{mat-delete-def} \textbf{by} auto
\end{itemize}

\textbf{private lemma} \textit{col-mk-poly-sub}[simp]:
\begin{itemize}
  \item \textbf{assumes} \(l \neq j'\) \textbf{and} \(j' < \text{dim-col} \(A\)\)
  \item \textbf{shows} \textit{col} \(\textit{mk-poly-sub} \(A \ l \ j\) \(j'\) = \textit{col} \(A\) \(j'\)\)
  \item \textbf{by}(\textit{rule} eq-vecI; \textit{insert} \textit{assms}; simp)
\end{itemize}

\textbf{private lemma} \textit{det-mk-poly-sub}:
\begin{itemize}
  \item \textbf{assumes} \(A : (A :: 'a :: comm-ring-1 poly mat) \in \text{carrier-mat} \(n\ n\)\) \textbf{and} \(i : i < n\)
  \item \textbf{shows} \textit{det} \(\textit{mk-poly-sub} \(A \ (n-1) \ i\) = \textit{det} \(A\)\)
  \item \textbf{using} \(i\)
  \item \textbf{proof} (\textit{induct} \(i\))
  \item \textbf{case} \(Suc\ i\)
  \item \textbf{show} \(\forall \text{\textit{case}}\ \textbf{unfolding} \textit{mk-poly-sub.simps}\)
  \item \textbf{apply}(\textit{subst} \textit{det-addcol[of - n]}\)
  \item \textbf{using} \(Suc\ \text{apply}\ \text{simp}\)
  \item \textbf{using} \(Suc\ \text{apply}\ \text{simp}\)
  \item \textbf{apply}(\textit{rule} \textit{mk-poly-sub-carrier[OF \(A\)}])
  \item \textbf{using} \(Suc\ \text{by}\ \text{auto}\)
\end{itemize}

\textbf{qed} \textit{simp}

\textbf{private lemma} \textit{det-mk-poly}:
\begin{itemize}
  \item \textbf{fixes} \(A : 'a :: comm-ring-1 mat\)
  \item \textbf{shows} \textit{det} \(\textit{mk-poly} \(A\) = [: \textit{det} \(A\) :]\)
  \item \textbf{proof} (\textit{cases} \textit{dim-row} \(A\) = \textit{dim-col} \(A\))
  \item \textbf{case} \(True\)
  \item \textbf{define} \(n\ \textbf{where}\ \(n = \text{dim-col} \(A\)\)
  \item \textbf{have} \textit{map-mat coeff-lift} \(A \in \text{carrier-mat} \(\text{dim-row} \(A\)\) \(\text{dim-col} \(A\)\) \textbf{by} simp}
  \item \textbf{hence} \(sq: \textit{map-mat coeff-lift} \(A \in \text{carrier-mat} \(\text{dim-col} \(A\)\) \(\text{dim-col} \(A\)\) \textbf{unfolding} True.\)
\end{itemize}

48
show \( ?\text{thesis} \)

proof (cases \( \text{dim-col} \ A = 0 \))

  case True thus \( ?\text{thesis} \) unfolding det-def by simp

  next case False thus \( ?\text{thesis} \)
   unfolding mk-poly-def
   by (subst det-mk-poly-sub[OF sq]; simp)

qed

next case False

  hence \( f_2: \text{dim-row} \ A = \text{dim-col} \ A \leftrightarrow \text{False} \) by simp

  hence \( f_3: \text{dim-row} (\text{mk-poly} \ A) = \text{dim-col} (\text{mk-poly} \ A) \leftrightarrow \text{False} \)
   unfolding mk-poly-dim by auto

  show \( ?\text{thesis} \) unfolding det-def unfolding \( f_2 \) \( f_3 \) if-False by simp

qed

private fun \( \text{mk-poly2-row} \) where

\[
\text{mk-poly2-row} \ A \ d \ j \ pv \ 0 = pv \\
| \ \text{mk-poly2-row} \ A \ d \ j \ pv \ (\text{Suc} \ n) = \\
\text{mk-poly2-row} \ A \ d \ j \ pv \ n \ |_{v \ n \mapsto pv \ n} = pv \ $ n + \text{monom} (A $$ (n, j)) \ d
\]

private fun \( \text{mk-poly2-col} \) where

\[
\text{mk-poly2-col} \ A \ pv \ 0 = pv \\
| \ \text{mk-poly2-col} \ A \ pv \ (\text{Suc} \ m) = \\
\text{mk-poly2-row} \ A \ m \ (\text{dim-col} \ A - \text{Suc} \ m) \ (\text{mk-poly2-col} \ A \ pv \ m) \ (\text{dim-row} \ A)
\]

private definition \( \text{mk-poly2} \ A \equiv \text{mk-poly2-col} \ A \ (0_v \ (\text{dim-row} \ A)) \ (\text{dim-col} \ A) \)

private lemma \( \text{mk-poly2-row-dim} \) [simp]:

\[
\text{dim-vec} (\text{mk-poly2-row} \ A \ d \ j \ pv \ i) = \text{dim-vec} \ pv \ \text{by (induct} \ i \ \text{arbitrary:} \ pv, \ \text{auto)}
\]

private lemma \( \text{mk-poly2-col-dim} \) [simp]:

\[
\text{dim-vec} (\text{mk-poly2-col} \ A \ pv \ j) = \text{dim-vec} \ pv \ \text{by (induct} \ j \ \text{arbitrary:} \ pv, \ \text{auto)}
\]

private lemma \( \text{mk-poly2-row} \):

assumes \( n: n \leq \text{dim-vec} \ pv \)

shows \( \text{mk-poly2-row} \ A \ d \ j \ pv \ n \ |_{v \ n \mapsto pv \ n} = \) 

\[
(\text{if} \ i < n \ \text{then} \ pv \ $ i + \text{monom} (A $$ (i, j)) \ d \ \text{else} \ pv \ $ i)
\]

using \( n \)

proof (induct \( n \) arbitrary: \( pv \))

  case (Suc \( n \)) thus \( ?\text{case} \)
   unfolding \( \text{mk-poly2-row} \).simps \( \text{by (cases rule: linorder-cases[of} \ i \ n\),auto) \)

qed simp

private lemma \( \text{mk-poly2-row科尔} \):

assumes \( \text{dim}[\text{simp}]: \text{dim-vec} \ pv = n \ \text{dim-row} \ A = n \ \text{and} \ j: j < \text{dim-col} \ A \)

shows \( \text{mk-poly2-row科尔} \ A \ d \ j \ pv \ n = pv + \text{map-vec} (\lambda a. \ \text{monom} \ a \ d) \ (\text{col} \ A \ j) \)

apply rule using \( \text{mk-poly2科尔[of} \ pv\] \ j \ \text{by auto) \)
private lemma mk-poly2-col:
  fixes pv :: 'a :: comm-semiring-1 poly vec and A :: 'a mat
  assumes i: i < dim-row A and dim: dim-row A = dim-vec pv
  shows mk-poly2-col A pv j $ i = pv $ i + (∑ j'<j. monom (A $$ (i, dim-col A − Suc j')) j')
  using dim
proof (induct j arbitrary: pv)
case (Suc j)
  show ?case
    unfolding mk-poly2-col.
simps
    apply (subst mk-poly2-row)
    using Suc
    apply simp
    unfolding Suc(1)[OF Suc(2)]
    using i by (simp add: add.assoc)
qed simp

private lemma mk-poly2-pre:
  fixes A :: 'a :: comm-semiring-1 mat
  assumes i: i < dim-row A
  shows mk-poly2 A $ i = (∑ j'<dim-col A. monom (A $$ (i, Suc j'))) j'
  unfolding mk-poly2-def
  apply (subst mk-poly2-col)
  using i by auto

private lemma mk-poly2:
  fixes A :: 'a :: comm-semiring-1 mat
  assumes i: i < dim-row A
  and c: dim-col A > 0
  shows mk-poly2 A $ i = (∑ j'<dim-col A. monom (A $$ (i,j')) (dim-col A − Suc j'))
  (is ?l = sum ?f ?S)
proof
  define l where l = dim-col A − 1
  have dim: dim-col A = Suc l unfolding l-def using i c by auto
  let ?q = λj. l − j
  have ?l = sum (?if o ?q) ?S unfolding l-def mk-poly2-pre[OF i] by auto
  also have ... = sum ?f ?S
  unfolding dim
  unfolding lessThan-Suc-atMost
  using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost],
  finally show ?thesis.
qed

private lemma mk-poly2-sylvester-upper:
  fixes p q :: 'a :: comm-semiring-1 poly
  assumes i: i < degree q
  shows mk-poly2 (sylvester-mat p q) $ i = monom 1 (degree q − Suc i) * p
  apply (subst mk-poly2)
  using i apply simp using i apply simp
  apply (subst sylvester-sum-mat-upper[OF i,symmetric])
apply (rule sum.cong)
unfolding sylvester-mat-dim lessThan-Suc-atMost apply simp
by auto

private lemma mk-poly2-sylvester-lower:
  fixes p q :: 'a :: comm-semiring-1 poly
  assumes mi: i ≥ degree q and imn: i < degree p + degree q
  shows mk-poly2 (sylvester-mat p q) $ i = monom 1 (degree p + degree q − Suc i) * q
  apply (subst mk-poly2)
  using imn apply simp using mi imn apply simp
  unfolding sylvester-mat-dim
  using sylvester-sum-mat-lower[OF mi imn]
  apply (subst sylvester-sum-mat-lower) using mi imn by auto

private lemma foo:
  fixes v :: 'a :: comm-semiring-1 vec
  shows monom 1 d · v map-vec coeff-lift v = map-vec (λa. monom a d) v
  apply (rule eq-vecI)
  unfolding index-map-vec index-col
  by (auto simp add: Polynomial.smult-monom)

private lemma mk-poly-sub-corresp:
  assumes dimA[simp]: dim-col A = Suc l and dimpv[simp]: dim-vec pv = dim-row A
  and j: j < dim-col A
  shows pv + col (mk-poly-sub (map-mat coeff-lift A) l j) l =
  mk-poly2-col A pv (Suc j)
proof(insert j, induct j)
  have le: dim-row A ≤ dim-vec pv using dimpe by simp
  have l: l < dim-col A using dimA by simp
  { case 0 show ?case
    apply (rule eq-vecI)
    using mk-poly2-row[OF le]
    by (auto simp add: monom-0)
  }
  { case (Suc j)
    hence j: j < dim-col A by simp
    show ?case
      unfolding mk-poly-sub.simps
      apply(subst col-addcol)
      apply simp
      apply simp
      apply(subst(2) comm-add-vec)
      apply(rule carrier-vecI, simp)
      apply(rule carrier-vecI, simp)
      apply(subst assoc-add-vec[symmetric])
      apply(rule carrier-vecI, rule refl)
      apply(rule carrier-vecI, simp)
  }
apply (rule carrier-vecI, simp)
unfolding Suc(1)[OF j]
apply (subst (2) mk-poly2-col.simps)
apply (subst mk-poly2-row-col)
  apply simp
  apply simp
using Suc apply simp
apply (subst col-mk-poly-sub)
using Suc apply simp
using Suc apply simp
apply (subst col-map-mat)
using dimA apply simp
unfolding foo dimA by simp
}
qed

private lemma col-mk-poly-mk-poly2:
  fixes A :: 'a :: comm-semiring-1 mat
  assumes dim: dim-col A > 0
  shows col (mk-poly A) (dim-col A - 1) = mk-poly2 A
proof –
  define l where l = dim-col A - 1
  have dim: dim-col A = Suc l unfolding l-def using dim by auto
  show ?thesis
    unfolding mk-poly-def mk-poly2-def dim
    apply (subst mk-poly-sub-corresp[symmetric])
    apply (rule dim)
    apply simp
    using dim apply simp
    apply (subst left-zero-vec)
    apply (rule carrier-vecI) using dim apply simp
    apply simp
    done
qed

private lemma mk-poly-mk-poly2:
  fixes A :: 'a :: comm-semiring-1 mat
  assumes dim: dim-col A > 0 and i: i < dim-row A
  shows mk-poly A $(i, dim-col A - 1) = mk-poly2 A $ i
proof –
  have mk-poly A $(i, dim-col A - 1) = col (mk-poly A) (dim-col A - 1) $ i
    apply (subst index-col(1)) using dim i by auto
  also note col-mk-poly-mk-poly2[OF dim]
  finally show ?thesis.
qed

lemma mk-poly-sylvester-upper:
  fixes p q :: 'a :: comm-ring-1 poly
  defines m ≡ degree p and n ≡ degree q
assumes \(i : i < n\)
sshows \(\text{mk-poly} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q) \ \text{as} \ (i, m + n - 1) = \text{monom} \ 1 \ (n - \text{Suc} \ i)\)

\(* \ p \ (\text{is} \ ?l = ?r)\)

**proof**

- let \(?S = \text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q\)
  - have \(c : m + n = \text{dim-col} \ ?S\) and \(r : m + n = \text{dim-row} \ ?S\) unfolding \(m\)-\text{def} \(n\)-\text{def}
  - hence \(\text{dim-col} \ ?S > 0\) i < \(\text{dim-row} \ ?S\) using \(i\) by auto
  - from \(\text{mk-poly-}\text{mk-poly2}[OF \ this]\)
    - have \(?l = \text{mk-poly2} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q) \ $\ i\) unfolding \(m\)-\text{def} \(n\)-\text{def} by auto
    - also have \(... = ?r\)
      - apply\((\text{subst \ mk-poly2-sy}v\text{l}e\text{st}e\text{r-upper})\)
        - using \(i\) unfolding \(n\)-\text{def} \(m\)-\text{def} by auto
  - finally show \(?\text{thesis}\).

qed

**Lemma** \(\text{mk-poly-sy}v\text{l}e\text{st}e\text{r-lower}:\)

**fixes** \(p \ q :: \ 'a :: \text{comm-ring-1 \ poly}\)

**defines** \(m \equiv \text{degree} \ p\) and \(n \equiv \text{degree} \ q\)

**assumes** \(ni : n \leq i\) and \(imn : i < m + n\)

**shows** \(\text{mk-poly} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q) \ \text{as} \ (i, m + n - 1) = \text{monom} \ 1 \ (m + n - \text{Suc} \ i) \ (i) \ * \ p \ (\text{is} \ ?l = ?r)\)

**proof**

- let \(?S = \text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q\)
  - have \(c : m + n = \text{dim-col} \ ?S\) and \(r : m + n = \text{dim-row} \ ?S\) unfolding \(m\)-\text{def} \(n\)-\text{def}
  - hence \(\text{dim-col} \ ?S > 0\) i < \(\text{dim-row} \ ?S\) using \(i\) by auto
  - from \(\text{mk-poly-}\text{mk-poly2}[OF \ this]\)
    - have \(?l = \text{mk-poly2} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q) \ $\ i\) unfolding \(m\)-\text{def} \(n\)-\text{def} by auto
    - also have \(... = ?r\)
      - apply\((\text{subst \ mk-poly2-sy}v\text{l}e\text{st}e\text{r-lower})\)
        - using \(i\) unfolding \(n\)-\text{def} \(m\)-\text{def} by auto
  - finally show \(?\text{thesis}\).

qed

The next lemma corresponds to Lemma 7.2.1.

**Lemma** \(\text{resultant-as-poly}:\)

**fixes** \(p \ q :: \ 'a :: \text{comm-ring-1 \ poly}\)

**assumes** \(\text{deg}p : \text{degree} \ p > 0\) and \(\text{deg}q : \text{degree} \ q > 0\)

**shows** \(\exists \ p' \ q'. \ \text{degree} \ p' < \text{degree} \ q \ \land \ \text{degree} \ q' < \text{degree} \ p \ \land \ [\text{resultant} \ p \ q ] = p' \ * \ p + q' \ * \ q\)

**proof** \((\text{intro \ exI \ conjI})\)

- define \(m \ where \ m = \text{degree} \ p\)
- define \(n \ where \ n = \text{degree} \ q\)
- define \(d \ where \ d = \text{dim-row} \ (\text{mk-poly} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q))\)
- define \(c \ where \ c = (\lambda i. \ \text{co}ff\text{e}f-lift \ (\text{co}f\text{actor} \ (\text{sy}v\text{l}e\text{st}e\text{r-mat} \ p \ q) \ i \ (m + n - 1)))\)
- define \(p' \ where \ p' = (\sum i < n. \ \text{monom} \ 1 \ (n - \text{Suc} \ i) \ * \ c \ i)\)
- define \(q' \ where \ q' = (\sum i < m. \ \text{monom} \ 1 \ (m - \text{Suc} \ i) \ * \ c \ (n + i))\)
have **degc**: \( \forall i. \text{degree} (c_i) = 0 \) unfolding c-def by auto

have **dmn**: \( d = m + n \) and **mdn**: \( m + n = d \) unfolding d-def m-def n-def by auto

have \([: \text{resultant} \; p \; q :] = \)
\( (\sum i < d. \text{mk-poly} (\text{sylvestermat} \; p \; q) \; \$$ (i, m+n-1) * \\
\text{cofactor} (\text{mk-poly} (\text{sylvestermat} \; p \; q)) \; i \; (m+n-1)) \)
unfolding resultant-def
unfolding det-mk-poly[symmetric]
unfolding m-def n-def d-def
apply(rule laplace-expansion-column[of - - degree p + degree q - i])
apply(rule carrier-matI)
also \{ fix i assume i: i < d \}

have **d2**: \( d = \text{dim-row} (\text{sylvestermat} \; p \; q) \) unfolding d-def by auto
have **cofactor** (\( \text{mk-poly} (\text{sylvestermat} \; p \; q) \)) \( i \; (m+n-1) = \)
\( (−1) \; ^{\big(} i + (m+n-1) \big) \ast \text{det} (\text{mat-delete} (\text{mk-poly} (\text{sylvestermat} \; p \; q)) \; i \; (m+n-1)) \)
using cofactor-def.
also have ... = 
\( (−1) \; ^{\big(} i + m+n-1 \big) \ast \text{coeff-lift} (\text{det} (\text{mat-delete} (\text{mk-poly} (\text{sylvestermat} \; p \; q)) \; i \; (m+n-1))) \)
using mk-poly-delete[\( \text{OF} \; i \ast \text{unfolded} \; \text{d2} \)]
unfolding m-def n-def by (auto simp add: add.assoc)
also have \( i + m + n - 1 = i + (m+n-1) \) using \( \text{folded mnd} \) by auto
finally have **cofactor** (\( \text{mk-poly} (\text{sylvestermat} \; p \; q) \)) \( i \; (m+n-1) = c_i \)
unfolding c-def cofactor-def hom-distrib by simp

{ hence ... = \( (\sum i < d. \text{mk-poly} (\text{sylvestermat} \; p \; q) \; \$$ (i, m+n-1) * c_i \)
\( \text{is } - = \text{sum } \text{if } \text{- - by auto} \)
also have ... = \text{sum } \text{if } (\{..<n\} \cup \{n..<d\}) \text{ unfolding dmn apply(subst ivl-disj-un(8)) by auto} \)
also have ... = \text{sum } \text{if } (\{..<n\} \cup \{n..<d\}) \text{ apply(subst sum.union-disjoint) by auto} \)
also \{ fix i assume i: i < n \}

have \( \text{if } i = \text{monom } 1 (n - \text{Suc } i) \ast c_i \ast p \)
unfolding m-def n-def
apply(subst mk-poly-sylvesterm-upper)
using i unfolding n-def by auto

{ hence sum \( \text{if } (\{..<n\} = p' \ast p \text{ unfolding p'-def sum-distrib-right by auto} \)
also \{ fix i assume i: i \in \{..<d\} \}

have \( \text{if } i = \text{monom } 1 (m + n - \text{Suc } i) \ast c_i \ast q \)
unfolding m-def n-def
apply(subst mk-poly-sylvesterm-lower)
using i unfolding dmn n-def m-def by auto

{ hence sum \( \text{if } (\{..<d\} = \sum i=\text{n..<d}. \text{monom } 1 (m + n - \text{Suc } i) \ast c_i \ast q \)
\( \text{is } - = \text{sum } \text{if } h - \ast - \text{ unfolding sum-distrib-right by auto} \)
also have \( \{n..<d\} = (\lambda i. i+n) \cdot \{0..<m\} \)
by (simp add: dmn)
also have sum ?h ... = sum (?h o (λi. i+n)) {0..<m}
  apply (subst sum.reindex[symmetric])
  apply (rule inj-onI) by auto
also have ... = q' unfolding q'-def apply (rule sum.cong) by (auto simp add: add.commute)
finally show main: [:resultant p q:] = p' * p + q' * q.
show degree p' < n
  unfolding p'-def
  apply (rule degree-sum-smaller)
  using degq[folded n-def] apply force+
proof –
  fix i assume i: i ∈ {..<n}
  show degree (monom 1 (n − Suc i) * c i) < n
    apply (rule order.strict-trans1)
    apply (rule degree-mult-le)
    unfolding add.right-neutral degc
    apply (rule order.strict-trans1)
    apply (rule degree-monom-le) using i by auto
qed
show degree q' < m
  unfolding q'-def
  apply (rule degree-sum-smaller)
  using degp[folded m-def] apply force+
proof –
  fix i assume i: i ∈ {..<m}
  show degree (monom 1 (m−Suc i) * (n+i)) < m
    apply (rule order.strict-trans1)
    apply (rule degree-mult-le)
    unfolding add.right-neutral degc
    apply (rule order.strict-trans1)
    apply (rule degree-monom-le) using i by auto
qed
qed

end

4.2.4 Resultant as Nonzero Polynomial Expression

lemma resultant-zero:
  fixes p q :: 'a :: comm-ring-1 poly
  assumes deg: degree p > 0 ∨ degree q > 0
    and xp: poly p x = 0 and xq: poly q x = 0
  shows resultant p q = 0
proof –
  { assume degp: degree p > 0 and degq: degree q > 0
    obtain p' q' where [: resultant p q :] = p' * p + q' * q
      using resultant-as-poly[OF degp degq] by force
    hence resultant p q = poly (p' * p + q' * q) x

  55
using `mpoly-base-conv(2)` of resultant \( p \, q \) \ by \ auto
also have \( \ldots = \text{poly} \, p \, x \, \ast \, \text{poly} \, p' \, x + \text{poly} \, q \, x \, \ast \, \text{poly} \, q' \, x \)
unfolding \( \text{poly2-def} \) \ by \ simp
finally have \( \text{?thesis using} \ \text{xp \, xq} \) \ by \ simp
}

moreover
\{ assume \( \text{degp: degree} \, p = 0 \)
have \( p: p = [0:] \ \text{using} \ \text{xp \ degree-0-id[OF \, degp, \ \text{symmetric}]} \) \ by \ (metis `mpoly-base-conv(2)`)
have \( \text{?thesis unfolding} \ p \ \text{using} \ \text{degp \ deg} \ \text{by} \ \text{simp} \)
\}

moreover
\{ assume \( \text{degq: degree} \, q = 0 \)
have \( q: q = [0:] \ \text{using} \ \text{xq \ degree-0-id[OF \, degq, \ \text{symmetric}]} \) \ by \ (metis `mpoly-base-conv(2)`)
have \( \text{?thesis unfolding} \ q \ \text{using} \ \text{degq \ deg} \ \text{by} \ \text{simp} \)
\}
ultimately show \( \text{?thesis by} \ \text{auto} \)
qed

\begin{description}
\item[\text{lemma \ poly-resultant-zero:}]\end{description}

\text{fixes} \( p \, q \, :: \, 'a \, :: \, \text{comm-ring-1} \ \text{poly} \ \text{poly} \)
\text{assumes} \( \text{deg: degree} \, p > 0 \ \lor \ \text{degree} \, q > 0 \)
\text{assumes} \( \text{p0: poly2 \, p \, x \, y = 0} \ \text{and} \ \text{q0: poly2 \, q \, x \, y = 0} \)
\text{shows} \( \text{poly} \, (\text{resultant} \, p \, q) \, x = 0 \)
\text{proof} –
\{ assume \( \text{degree} \, p > 0 \ \text{degree} \, q > 0 \)
from \( \text{resultant-as-poly[OF \, this]} \)
obtain \( p' \, q' \) \ where \( [\, \text{resultant} \, p \, q \, :] = p' \, \ast \, p + q' \, \ast \, q \) \ by \ force
hence \( \text{resultant} \, p \, q = \text{poly} \, (p' \, \ast \, p + q' \, \ast \, q) \, [:y:] \)
\text{using `mpoly-base-conv(2)` of resultant \( p \, q \) \ by \ auto}
also have \( \text{poly} \, \ldots \, x = \text{poly2} \, p \, x \, y \, \ast \, \text{poly2} \, p' \, x \, y + \text{poly2} \, q \, x \, y \ast \, \text{poly2} \, q' \, x \, y \)
unfolding \( \text{poly2-def} \) \ by \ simp
finally have \( \text{?thesis unfolding} \ \text{p0 \ q0 by simp} \)
\}
moreover \{ assume \( \text{degp: degree} \, p = 0 \)
\text{hence} \( p: p = [\, \text{coeff} \, p \, 0:] \) by (subst \( \text{degree-0-id[OF \, degp, \ \text{symmetric}]}, \text{simp})
\text{hence} \( \text{resultant} \, p \, q = \text{coeff} \, p \, 0 \, \sim \, \text{degree} \, q \ \text{using} \ \text{resultant-const(1)} \) by \ metis
also have \( \text{poly} \, \ldots \, x = \text{poly} \, (\text{coeff} \, p \, 0) \, x \, \sim \, \text{degree} \, q \) \ by \ auto
also have \( \ldots = \text{poly2} \, p \, x \, y \, \sim \, \text{degree} \, q \ \text{unfolding} \ \text{poly2-def by subst p, auto} \)
finally have \( \text{?thesis unfolding} \ \text{p0 using} \ \text{degp \ degp \ zero-power by auto} \)
\}
moreover \{ assume \( \text{degq: degree} \, q = 0 \)
\text{hence} \( q: q = [\, \text{coeff} \, q \, 0:] \) by (subst \( \text{degree-0-id[OF \, degq, \ \text{symmetric}]}, \text{simp})
\text{hence} \( \text{resultant} \, p \, q = \text{coeff} \, q \, 0 \, \sim \, \text{degree} \, p \ \text{using} \ \text{resultant-const(2)} \) by \ metis
also have \( \text{poly} \, \ldots \, x = \text{poly} \, (\text{coeff} \, q \, 0) \, x \, \sim \, \text{degree} \, p \) \ by \ auto
also have \( \ldots = \text{poly2} \, q \, x \, y \, \sim \, \text{degree} \, p \ \text{unfolding} \ \text{poly2-def by subst q, auto} \)
finally have \( \text{?thesis unfolding} \ \text{q0 using} \ \text{degq \ degp \ zero-power by auto} \)
\}
ultimately show \( \text{?thesis by} \ \text{auto} \)
qed

\begin{description}
\item[\text{lemma \ resultant-as-nonzero-poly-weak:}]\end{description}
fixes \( p \), \( q :: 'a :: \text{idom poly} \)
assumes \( \text{degp: degree } p > 0 \) and \( \text{degq: degree } q > 0 \)
and \( r0; \text{resultant } p \neq 0 \)
shows \( \exists p' \), \( q' \). degree \( p' < \text{degree } q \) \land degree \( q' < \text{degree } p \) \land 
\( [: \text{resultant } p \ q : ] = p' \ast p + q' \ast q \) \land \( p' \neq 0 \) \land \( q' \neq 0 \)
proof
obtain \( p', q' \)
where \( \text{deg: degree } p' < \text{degree } q \) degree \( q' < \text{degree } p \) 
and main: 
\( [: \text{resultant } p \ q : ] = p' \ast p + q' \ast q \)
using resultant-as-poly[OF degp degq] by auto
have \( p0: p \neq 0 \) using degp by auto
have \( q0: q \neq 0 \) using degq by auto
show \( ?\text{thesis} \)
proof
(cases resultant \( p \ q \) = 0)
case False
thus \( ?\text{thesis} \)
using resultant-as-nonzero-poly-weak degp degq
unfolding m-def n-def by auto
next
assume \( q' = 0 \)

hence 
\( [: \text{resultant } p \ q : ] = p' \ast p \) using main by auto
also hence \( d0: 0 = \text{degree } (p' \ast q) \) by (metis degree-pCons-0)
\{ assume \( q' \neq 0 \)
  hence 
  \( \text{degree } (q' \ast q) = \text{degree } q' + \text{degree } q \)
  apply(rule degree-mult-eq) using q0 by auto
  hence False using d0 degq by auto
\} hence \( q' = 0 \) by auto
finally show \( False \) using r0 by auto
qed fact+
qed

Next lemma corresponds to Lemma 7.2.2 of the textbook

lemma resultant-as-nonzero-poly:

fixes \( p \), \( q :: 'a :: \text{idom poly} \)
defines \( m \equiv \text{degree } p \) and \( n \equiv \text{degree } q \)
assumes \( \text{degp: } m > 0 \) and \( \text{degq: } n > 0 \)
shows \( \exists p' \), \( q' \). degree \( p' < n \) \land degree \( q' < m \) \land 
\( [: \text{resultant } p \ q : ] = p' \ast p + q' \ast q \) \land \( p' \neq 0 \) \land \( q' \neq 0 \)
proof (cases resultant \( p \ q \) = 0)
case False
thus \( ?\text{thesis} \)
using resultant-as-nonzero-poly-weak degp degq
unfolding m-def n-def by auto
next case True
define S where S = transpose-mat (sylvester-mat p q)
have S: S ∈ carrier-mat (m+n) (m+n) unfolding S-def m-def n-def by auto
have det S = 0 using True
  unfolding resultant-def S-def apply (subst det-transpose) by auto
then obtain v
  where v: v ∈ carrier-vec (m+n) and v0: v ≠ 0v (m+n) and S *v v = 0v (m+n)
    using det-0-iff-vec-prod-zero[OF S] by auto
hence poly-of-vec (S *v v) = 0 by auto
hence main: poly-of-vec (vec-first v n) * p + poly-of-vec (vec-last v m) * q = 0
  (is ?p * - + ?q * - = -)
    using sylvester-vec-poly[OF v[unfolded m-def n-def], folded m-def n-def S-def] by auto
have split: vec-first v n @v vec-last v m = v
    using vec-first-last-append[simplified add.commute] v by auto
show ?thesis
proof
  assume p': ?p = 0
  hence ?q * q = 0 using main by auto
  hence ?q = 0 using degq n-def by auto
  hence vec-last v m = 0v m unfolding poly-of-vec-0-iff by auto
  also have vec-first v n @v ... = 0v (m+n) using p'0 unfolding poly-of-vec-0-iff by auto
  finally have v = 0v (m+n) using split by auto
  thus False using v0 by auto
qed
show ?q ≠ 0
proof
  assume q'0: ?q = 0
  hence ?p * p = 0 using main by auto
  hence ?p = 0 using degp n-def by auto
  hence vec-first v n = 0v n unfolding poly-of-vec-0-iff by auto
  also have ... @v vec-last v m = 0v (m+n) using q'0 unfolding poly-of-vec-0-iff by auto
  finally have v = 0v (m+n) using split by auto
  thus False using v0 by auto
qed
show degree ?p < n using degree-poly-of-vec-less[of vec-first v n] using degq by auto
show degree ?q < m using degree-poly-of-vec-less[of vec-last v m] using degp by auto
qed
def \textit{Corresponds to Lemma 7.2.3 of the textbook}
lemma resultant-zero-imp-common-factor:
  fixes p q :: 'a :: ufd poly
  assumes deg: degree p > 0 ∨ degree q > 0 and r0: resultant p q = 0
  shows ¬ coprime p q
  unfolding neq0_conv[symmetric]
proof –
  { assume degp: degree p > 0 and degq: degree q > 0
    assume cop: coprime p q
    obtain p' q' where p' * p + q' * q = 0
      and p': degree p' < degree q and q': degree q' < degree p
      and p'0: p' ≠ 0 and q'0: q' ≠ 0
      using resultant-as-nonzero-poly[OF degp degq] r0 by auto
    hence p' * p = - q' * q by (simp add: eq-neg-iff-add-eq-0)
    from some-gcd.coprime-mult-cross-dvd[OF cop this]
    have p dvd q' by auto
    from dvd-imp-degree-le[OF this q'0] have degree p ≤ degree q' by auto
    hence False using q' by auto
  }
  moreover
  { assume degp: degree p = 0
    then obtain x where p = [:x:] by (elim degree-eq-zeroE)
    moreover hence resultant p q = x ^ degree q using resultant-const by auto
    hence x = 0 using r0 by auto
    ultimately have p = 0 by auto
    hence ?thesis unfolding not-coprime-iff-common-factor
      by (metis deg degp dvd-0-right dvd-refl less-numeral-extra(3) poly-dvd-1)
  }
  moreover
  { assume degq: degree q = 0
    then obtain x where q = [:x:] by (elim degree-eq-zeroE)
    moreover hence resultant p q = x ^ degree p using resultant-const by auto
    hence x = 0 using r0 by auto
    ultimately have q = 0 by auto
    hence ?thesis unfolding not-coprime-iff-common-factor
      by (metis deg degq dvd-0-right dvd-refl less-numeral-extra(3) poly-dvd-1)
  }
  ultimately show ?thesis by auto
qed

lemma resultant-non-zero-imp-coprime:
  assumes nz: resultant (f :: 'a :: field poly) g ≠ 0
  and nz': f ≠ 0 ∨ g ≠ 0
  shows coprime f g
proof (cases degree f = 0 ∨ degree g = 0)
  case False
  define r where r = [:resultant f g:]
  from nz have r: r ≠ 0 unfolding r-def by auto

59
from False have degree $f > 0$ degree $g > 0$ by auto
from resultant-as-nonzero-poly-weak[of this nz]
obtain $p$ $q$ where degree $p <$ degree $g$ degree $q <$ degree $f$
  and id: $r = p * f + q * g$
  and $p \neq 0$ $q \neq 0$ unfolding r-def by auto
define $h$ where $h = \text{some-gcd } f g$
have $h$ dvd $f$ $h$ dvd $g$ unfolding h-def by auto
then obtain $j$ $k$ where $f$: $f = h * j$ and $g$: $g = h * k$ unfolding dvd-def by auto
from id[unfolded $f$ $g$]
have id: $h * (p * j + q * k) = r$ by (auto simp: field-simps)
from arg-cong[of degree]
have degree $(h * (p * j + q * k)) = 0$
  unfolding r-def by auto
also have degree $(h * (p * j + q * k)) = \text{degree } h + \text{degree } (p * j + q * k)$
  by (subst degree-mult-eq, insert id $r$, auto)
finally have $h$: degree $h = 0$ $h \neq 0$ using $r$ id by auto
thus ?thesis unfolding h-def using is-unit-iff-degree some-gcd.gcd-dvd-1 by blast
next
case True
thus ?thesis
proof
  assume deg-\$g$: degree $g = 0$
  show ?thesis
  proof (cases $g = 0$)
    case False
    then show ?thesis using divides-degree[of - $g$, unfolded deg-\$g$]
      by (simp add: is-unit-right-imp-coprime)
  next
    case $g$: $g = [:0:]$ by auto
    from nz[unfolded this resultant-const]
have degree $f = 0$ by auto
    with nz' show ?thesis unfolding $g$ by auto
  qed
next
assume deg-\$f$: degree $f = 0$
show ?thesis
proof (cases $f = 0$)
  case False
  then show ?thesis using divides-degree[of - $f$, unfolded deg-\$f$]
    by (simp add: is-unit-left-imp-coprime)
next
  case $f$: $f = [:0:]$ by auto
  from nz[unfolded this resultant-const]
have degree $g = 0$ by auto
  with nz' show ?thesis unfolding $f$ by auto
  qed
qed
qed

end
5 Algebraic Numbers: Addition and Multiplication

This theory contains the remaining field operations for algebraic numbers, namely addition and multiplication.

theory Algebraic-Numbers
imports Algebraic-Numbers-Prelim Resultant Polynomial-Factorization Polynomial-Divisibility
begin

interpretation coeff-hom: monoid-add-hom λp. coeff p i by (unfold-locales, auto)
interpretation coeff-hom: comm-monoid-add-hom λp. coeff p i..
interpretation coeff-hom: group-add-hom λp. coeff p i..
interpretation coeff-hom: ab-group-add-hom λp. coeff p i..
interpretation coeff-0-hom: monoid-mult-hom λp. coeff p 0 by (unfold-locales, auto simp: coeff-mult)
interpretation coeff-0-hom: semiring-hom λp. coeff p 0..
interpretation coeff-0-hom: comm-monoid-mult-hom λp. coeff p 0..
interpretation coeff-0-hom: comm-semiring-hom λp. coeff p 0..

5.1 Addition of Algebraic Numbers

definition x-y ≡ [: [0, 1 :], −1 :]
definition poly-x-minus-y p = poly-lift p ◦ p x-y

lemma coeff-xy-power:
  assumes k ≤ n
  shows coeff (x-y ^ n :: 'a :: comm-ring-1 poly poly) k =
    monom (of-nat (n choose (n − k)) * (−1) ^ k) (n − k)
proof −
define X :: 'a poly poly where X = monom (monom 1 1) 0
define Y :: 'a poly poly where Y = monom (−1) 1

have [simp]: monom 1 b * (−1) ^ k = monom ((−1) ^ k :: 'a) b for b k
  by (auto simp: monom-altdel minus-one-power-iff)

have (X + Y) ^ n = (∑ i≤n. of-nat (n choose i) * X ^ i * Y ^ (n − i))
  by (subst binomial-ring) auto
also have ... = (∑ i≤n. of-nat (n choose i) * monom (monom ((−1) ^ (n − i)) i) (n − i))
  by (simp add: X-def Y-def monom-power mult-monom mult.assoc)
also have ... = (∑ i≤n. monom (monom (of-nat (n choose i) * (−1) ^ (n − i)) i) (n − i))
  by (simp add: of-nat-poly smult-monom)
also have coeff ... k =

61
(\sum_{i \leq n. \text{ if } n - i = k \text{ then } \text{monom (of-nat (n choose i)) * (-1) ^ (n - i)} \text{ i else 0) by (simp add: of-nat-poly coeff-sum})

also have \ldots = (\sum_{i \in \{n-k\}} \text{monom (of-nat (n choose i)) * (-1) ^ (n - i)})

using \langle k \leq n \rangle by (intro sum.mono-neutral-cong-right) auto

also have \ldots = (\sum_{i \in \{n-k\}} \text{monom (of-nat (n choose i)) * (-1) ^ (n - i)})

by (simp add: x-y-def x-y-def monom-altdef)

finally show \ldots

using \langle k \leq n \rangle by simp

qed

The following polynomial represents the sum of two algebraic numbers.

definition poly-add :: 'a :: comm-ring-1 poly ⇒ 'a poly ⇒ 'a poly where
poly-add p q = resultant (poly-x-minus-y p) (poly-lift q)

5.1.1 poly-add has desired root

interpretation poly-x-minus-y-hom:
comm-ring-hom poly-x-minus-y by (unfold-locales; simp add: poly-x-minus-y-def hom-distribs)

lemma poly2-x-y[simp]:
fixes x :: 'a :: comm-ring-1
shows poly2 x y x y = x - y unfolding poly2-def by (simp add: x-y-def)

lemma degree-poly-x-minus-y[simp]:
fixes p :: 'a :: idom poly
shows degree (poly-x-minus-y p y) = degree p unfolding poly-x-minus-y-def x-y-def
by auto

lemma poly-x-minus-y-pCons[simp]:
poly-x-minus-y (pCons a p) = [a \vdash x - y

unfolding poly-x-minus-y-def x-y-def by simp

lemma poly-poly-poly-x-minus-y[simp]:
fixes p :: 'a :: comm-ring-1 poly
shows poly (poly (poly-x-minus-y p) q) x = poly p (x - poly q x)
by (induct p; simp add: ring-distribs x-y-def)

lemma poly2-poly-x-minus-y[simp]:
fixes p :: 'a :: comm-ring-1 poly
shows poly2 (poly-x-minus-y p) x y = poly p (x - y) unfolding poly2-def by simp

interpretation x-y-mult-hom: zero-hom-0 \lambda p :: 'a :: comm-ring-1 poly poly. x-y * p

proof (unfold-locales)
fix p :: 'a poly poly
assume x-y * p = 0
then show p = 0 apply (simp add: x-y-def)
by (metis eq-neq-iff-add-eq-0 minus-equation-iff minus-pCons synthetic-div-unique-lemma)
lemma x-y-nonzero[simp]: x-y ≠ 0 by (simp add: x-y-def)

lemma degree-x-y[simp]: degree x-y = 1 by (simp add: x-y-def)

interpretation x-y-mult-hom: inj-comm-monoid-add-hom λp :: 'a :: idom poly poly. x-y * p
proof (unfold-locales)
show x-y * p = x-y * q ⟷ p = q for p q :: 'a poly poly
proof (induct p arbitrary: q)
case 0
then show ?case by simp
next
case p: (pCons a p)
from p(3)[unfolded mult-pCons-right]
have x-y * (monom a 0 + pCons 0 1 * p) = x-y * q
apply (subst(asm) pCons-0-as-mult)
apply (subst(asm) smult-prod) by (simp only: field-simps distrib-left)
then have monom a 0 + pCons 0 1 * p = q by simp
then show pCons a p = q using pCons-as-add by (simp add: monom-0 monom-Suc)
qed
qed

interpretation poly-x-minus-y-hom: inj-idom-hom poly-x-minus-y
proof
fix p :: 'a poly
assume 0: poly-x-minus-y p = 0
then have poly-lift p o_p x-y = 0 by (simp add: poly-x-minus-y-def)
then show p = 0
proof (induct p)
case 0
then show ?case by simp
next
case (pCons a p)
note p = this[unfolded poly-lift-pCons pcompose-pCons]
show ?case
proof (cases a=0)
case a0: True
with p have x-y * poly-lift p o_p x-y = 0 by simp
then have poly-lift p o_p x-y = 0 by simp
then show ?thesis using p by simp
next
case a0: False
with p have p0: p ≠ 0 by auto
from p have ::'a:: = − x-y * poly-lift p o_p x-y by (simp only: eq-neq-iff-add-eq-0)
then have degree ::'a:: = degree (x-y * poly-lift p o_p x-y) by simp
also have ... = degree (x-y:'a poly poly) + degree (poly-lift p o_p x-y)
apply (subst degree-mult-eq)
apply simp
apply (subst pcompose-eq-0)
apply (simp add: x-y-def)
apply (simp add: p0)
done
finally have False by simp
then show ?thesis..
qed
qed
qed

lemma poly-add:
  fixes p q :: 'a :: comm-ring_1 poly
  assumes q0: q ≠ 0 and x: poly p x = 0 and y: poly q y = 0
  shows poly (poly-add p q) (x+y) = 0
proof (unfold poly-add-def, rule poly-resultant-zero[OF disjI2])
  have degree q > 0 using poly-zero q0 y by auto
  thus degq: degree (poly-lift q) > 0 by auto
qed (insert x y, simp-all)

5.1.2 poly-add is nonzero

We first prove that poly-lift preserves factorization. The result will be essential also in the next section for division of algebraic numbers.

interpretation poly-lift-hom:
  unit-preserving-hom poly-lift :: 'a :: {comm-semiring_1, semiring-no-zero-divisors} poly ⇒ -
proof
  fix x :: 'a poly
  assume poly-lift x dvd 1
  then have poly-y-x (poly-lift x) dvd poly-y-x 1
    by simp
  then show x dvd 1
    by (auto simp add: poly-y-x-poly-lift)
qed

interpretation poly-lift-hom:
  factor-preserving-hom poly-lift::'a::idom poly ⇒ 'a poly poly
proof unfold-locales
  fix p :: 'a poly
  assume p: irreducible p
  show irreducible (poly-lift p)
    proof (rule ccontr)
      from p have p0: p ≠ 0 and ¬ p dvd 1 by (auto dest: irreducible-not-unit)
      with poly-lift-hom.hom-dvd[of p 1] have p1: ¬ poly-lift p dvd 1 by auto
      assume ¬ irreducible (poly-lift p)
      from this[unfolded irreducible-altdef,simplified] p0 p1
obtain $q$ where $q \text{ dvd } \text{ poly-lift } p$ and $pq : \neg \text{ poly-lift } p \text{ dvd } q$ and $q : \neg q \text{ dvd } 1$

by auto

then obtain $r$ where $q * r = \text{ poly-lift } p$ by (elim dvdE, auto)
then have $\text{ poly-y-x } (q * r) = \text{ poly-y-x } (\text{ poly-lift } p)$ by auto
also have ... = $[:p:]$ by (auto simp: poly-y-x-poly-lift monom-0)
also have $\text{ poly-y-x } (q * r) = \text{ poly-y-x } q * \text{ poly-y-x } r$ by (auto simp: hom-distribs)
finally have ... = $[:q:]$ by auto

then have $qp : \text{ poly-y-x } q \text{ dvd } [:q:]$ by auto

from dvd-const[OF this] have degree ($\text{ poly-y-x } q$) = 0 by auto

also have ... = $[:p:]$ by (auto)

finally have ... = $[:s:]$ by (auto simp: poly-y-x-poly-lift monom-0)

also have ... = $[:q:]$ by (auto simp: qs[symmetric])

finally have $sq : \text{ poly-lift } s = q$ by auto

from irreducibleD[OF this] have $sp : s \text{ dvd } p$ by (auto simp: const-poly-dvd)

from irreducibleD[OF this] sq pq show False by auto

qed

We now show that $\text{ poly-x-minus-y }$ is a factor-preserving homomorphism. This is essential for this section. This is easy since $\text{ poly-x-minus-y }$ can be represented as the composition of two factor-preserving homomorphisms.

**lemma** $\text{ poly-x-minus-y-as-comp } : \text{ poly-x-minus-y } = (\lambda p . p \circ p \text{x-y}) \circ \text{ poly-lift}$

by (intro ext, unfold poly-x-minus-y-def, auto)

**context** idom-isom begin
sublocale comm-semiring-isom.. end

interpretation poly-x-minus-y-hom:
factor-preserving-hom $\text{ poly-x-minus-y } :: \check{\text{ a :: idom poly } \Rightarrow \check{\text{ a poly poly}}}$

**proof**
interpret $x-y-hom : \check{\text{ bijective } \lambda p :: \check{\text{ a poly poly } . p \circ p \text{x-y}}}$

**proof** (unfold bijective-eq-bij, rule id-imp-bij)

fix $p :: \check{\text{ a poly poly show } p \circ p \text{x-y} \circ p \text{x-y} = p}$

apply (induct $p$, simp)

apply (unfold x-y-def hom-distribs pcompose-pCons) by (simp)

qed

interpret $x-y-hom : \check{\text{ idom-isom } \lambda p :: \check{\text{ a poly poly } . p \circ p \text{x-y}}}$ by (unfold-locales, auto)

show $\text{ factor-preserving-hom } (\text{ poly-x-minus-y } :: \check{\text{ a poly } \Rightarrow -})$

by (unfold poly-x-minus-y-as-comp, rule factor-preserving-hom-comp, unfold-locales)

qed

Now we show that results of $\text{ poly-x-minus-y }$ and $\text{ poly-lift }$ are coprime.

**lemma** $\text{ poly-y-x-const[simp]} : \text{ poly-y-x } [:a:] = [:a:]$ by (simp add: poly-y-x-def monom-0)

**context** begin
private abbreviation \( y-x = [: [: 0, -1 :], 1 :] \)

lemma poly-y-x-y[simp]: poly-y x y y-x by (simp add: x-y-def poly-y-x-def monom-Suc monom-0)

private lemma y-x[simp]: fixes x :: 'a :: comm-ring-1 shows poly2 y-x x y = y
- x
  unfolding poly2-def by simp

private definition poly-y-minus-x p ≡ poly-lift p ◦ p

private lemma poly-y-minus-x-0[simp]: poly-y-minus-x 0 = 0 by (simp add: poly-y-minus-x-def)

private lemma poly-y-minus-x-pCons[simp]:
  poly-y-minus-x (pCons a p) = [:[: a :]:] + poly-y-minus-x p ∗ y-x by (simp add: poly-y-minus-x-def)

private lemma poly-y-x-poly-x-minus-y:
  fixes p :: 'a :: idom poly
  shows poly-y-x (poly-x-minus-y p) = poly-y-minus-x p
  apply (induct p, simp)
  apply (unfold poly-x-minus-y-pCons hom-distribs) by simp

lemma degree-poly-y-minus-x[simp]:
  fixes p :: 'a :: idom poly
  shows degree (poly-y-x (poly-x-minus-y p)) = degree p
  by (simp add: poly-y-minus-x-def poly-y-x-poly-x-minus-y)

end

lemma dvd-all-coeffs-iff:
  fixes x :: 'a :: comm-semiring-1
  shows (\( \forall i \in \text{set} (\text{coeffs} p). \ x \ \text{dvd} \ p_i \) \iff \( \forall i. \ x \ \text{dvd} \ \text{coeff} p i \)) (is \( ?l = ?r \))
  proof-
  have \( ?r = (\forall i\in{\degree p}\cup\{\text{Suc} (\degree p)\}. \ x \ \text{dvd} \ \text{coeff} p i \) by auto
  also have \( ?l = (\forall i\leq\degree p. \ x \ \text{dvd} \ \text{coeff} p i \) by (auto simp add: ball-Un coeff-eq-0)
  finally show \( ?l \iff ?r \) by (auto simp: coeffs-def)
  qed

lemma primitive-imp-no-constant-factor:
  fixes p :: 'a :: {comm-semiring-1, semiring-no-zero-divisors} poly
  assumes pr: \( \text{primitive} \ p \) and F: \( \text{mset-factors} \ F \ p \) and fF: \( f \in\# \ F \)
  shows degree f ≠ 0
  proof
  from F fF have irr: \( \text{irreducible} f \) and fp: \( f \ \text{dvd} \ p \) by (auto dest: mset-factors-imp-dvd)
  assume deg: \( \text{degree} f = 0 \)

  66
then obtain \( f_0 \) where \( f_0 : f = [\cdot f_0 \cdot] \) by (auto dest: degree0-coeffs)

with \( fp \) have \([\cdot f_0 \cdot] \) dvd \( p \) by simp

then have \( f_0 \) dvd \( \text{coeff} \ p \) for \( i \) by (simp add: const-poly-dvd-iff)

with primitiveD \([\text{OF} \ p] \) dvd-all-coeffs-iff have \( f_0 \) dvd \( 1 \) by (auto simp: coeffs-def)

with \( f_0 \) irr show False by auto

qed

lemma coprime-poly-x-minus-y-poly-lift:

fixes \( p \ q :: 'a :: ufd \text{ poly} \)

assumes \( \text{degp} : \text{degree} \ p > 0 \) and \( \text{degq} : \text{degree} \ q > 0 \)

and \( \text{pr} : \text{primitive} \ p \)

shows coprime \((\text{poly-x-minus-y} \ p) \ (\text{poly-lift} \ q)\)

proof (rule ccontr)

from \( \text{degp} \) have \( \neg \ p \) dvd \( 1 \) by (auto simp: dvd-const)

from \( \text{degp} \) have \( p_0 : \ p \neq 0 \) by auto

from mset-factors-exist[OF \( p \ p_0 \)] obtain \( F \) where \( F : \text{mset-factors} \ F \ p \) by auto

with poly-x-minus-y-hom.hom-mset-factors have \( pF : \text{mset-factors} \ (\text{image-mset poly-x-minus-y} \ F) \ (\text{poly-x-minus-y} \ p) \) by auto

from \( \text{degq} \) have \( \neg \ q \) dvd \( 1 \) by (auto simp: dvd-const)

from \( \text{degq} \) have \( q_0 : q \neq 0 \) by auto

from mset-factors-exist[OF \( q_0 \ q \)] obtain \( G \) where \( G : \text{mset-factors} \ G \ q \) by auto

with poly-lift-hom.hom-mset-factors have \( pG : \text{mset-factors} \ (\text{image-mset poly-lift} \ G) \ (\text{poly-lift} \ q) \) by auto

assume \( \neg \) coprime \((\text{poly-x-minus-y} \ p) \ (\text{poly-lift} \ q)\)

from this unfolded not-coprime-iff-common-factor obtain \( r \)

where \( rp : r \) dvd \( (\text{poly-x-minus-y} \ p) \)

and \( rq : r \) dvd \( (\text{poly-lift} \ q) \)

and \( rU : \neg r \) dvd \( 1 \) by auto

note poly-lift-hom.hom-dvd

from \( rp \ p_0 \) have \( r_0 : r \neq 0 \) by auto

from mset-factors-exist[OF \( \text{OF} \ r_0 \ rU \)] obtain \( H \) where \( H : \text{mset-factors} \ H \ r \) by auto

then have \( H \neq \{\} \) by auto

then obtain \( h \) where \( hH : h \in\# \ H \) by fastforce

with \( H \) mset-factors-imp-dvd have \( hr : h \) dvd \( r \) and \( h : \text{irreducible} \ h \) by auto

from irreducible-not-unit[OF \( \text{OF} \ h \)] have \( hU : \neg h \) dvd \( 1 \) by auto

from \( hr \ rp \) have \( h \) dvd \( (\text{poly-x-minus-y} \ p) \) by (rule dvd-trans)

from irreducible-dvd-imp-factor[OF this \( h \ pF \)] \( p_0 \)

obtain \( f \) where \( f : f \in\# \ F \) and \( fh : \text{poly-x-minus-y} \ f \) ddvd \( h \) by auto

from \( hr \ rq \) have \( h \) dvd \( (\text{poly-lift} \ g) \) by (rule dvd-trans)

from irreducible-dvd-imp-factor[OF this \( h \ pG \)] \( q_0 \)

obtain \( g \) where \( g : g \in\# \ G \) and \( gh : \text{poly-lift} \ g \) ddvd \( h \) by auto

from \( fh \ gh \) have \( \text{poly-x-minus-y} \ f \) ddvd \( \text{poly-lift} \ g \) using ddvd-trans by auto

then have \( \text{poly-y-x} \ (\text{poly-x-minus-y} \ f) \) ddvd \( \text{poly-y-x} \ (\text{poly-lift} \ g) \) by simp

also have \( \text{poly-y-x} \ (\text{poly-lift} \ g) = [\cdot g\cdot] \) unfolding poly-y-x-poly-lift monom-0 by
auto

**Finally have** `dvd: poly-y-x (poly-x-minus-y f) dvd [g:]` **by** `auto`

**Then have** `degree (poly-y-x (poly-x-minus-y f)) = 0` **by** `(metis degree-pCons-0
dvd-0-left-iff ded-const)`

**Then have** `degree f = 0` **by** `simp`

**With** `primitive-imp-no-constant-factor[OF pr F f] show False` **by** `auto`

**Qed**

**Lemma** `poly-add-nonzero`:

**Fixes** `p q :: 'a :: ufd poly`

**Assumes** `p0: p \neq 0` **and** `q0: q \neq 0` **and** `x: poly p x = 0` **and** `y: poly q y = 0`

**And** `pr: primitive p`

**Shows** `poly-add p q \neq 0`

**Proof**

**Have** `degp: degree p > 0` **using** `le-0-eq order-degree order-root p0 x` **by** `(metis gr0I)`

**Have** `degg: degree q > 0` **using** `le-0-eq order-degree order-root q0 y` **by** `(metis gr0I)`

**Assume** `0: poly-add p q = 0`

**From** `resultant-zero-imp-common-factor[OF - this[unfolded poly-add-def]]` `degp`

**And** `coprime-poly-x-minus-y-poly-lift[OF degp degq pr]`

**Show** `False` **by** `auto`

**Qed**

**5.1.3 Summary for addition**

Now we lift the results to one that uses `ipoly`, by showing some homomorphism lemmas.

**Lemma** `(in comm-ring-hom) map-poly-x-minus-y`:

`map-poly (map-poly hom) (poly-x-minus-y p) = poly-x-minus-y (map-poly hom p)`

**Proof**

**Interpret** `mp: map-poly-comm-ring-hom hom..`

**Interpret** `mpmp: map-poly-comm-ring-hom map-poly hom..`

**Show** `?thesis`

**Apply** `(induct p, simp)`

**Apply** `(unfold x-y-def hom-distrib poly-x-minus-y-pCons, simp) done`

**Qed**

**Lemma** `(in comm-ring-hom) hom-poly-lift[simp]`:

`map-poly (map-poly hom) (poly-lift q) = poly-lift (map-poly hom q)`

**Proof**

**Show** `?thesis`

**Unfolding** `poly-lift-def`

**Unfolding** `[of coeff-lift, OF coeff-lift-hom.hom-zero]`

**Unfolding** `map-poly-coeff-lift-hom by simp`

**Qed**

**Lemma** `lead-coeff-poly-x-minus-y`:
fixes $p :: 'a::idom poly$
shows $\text{lead-coeff} (\text{poly-x-minus-y} p) = [(\text{lead-coeff} p \times ((-1) ^ \text{degree} p)) : ] (\text{is} ?l = {?r})$
proof
have $?l = \text{Polynomial.smult} (\text{lead-coeff} p) ((-1) ^ \text{degree} p)$
  by (unfold poly-x-minus-y-def, subst lead-coeff-comp; simp add: x-y-def)
also have $... = {?r}$ by (unfold hom-distrib, simp add: smult-as-map-poly[symmetric])
finally show $?\text{thesis}$. qed

lemma $\text{degree-coeff-poly-x-minus-y}$:
fixes $p q :: 'a :: \{\text{idom, semiring-char-0}\} poly$
shows $\text{degree} (\text{coeff} (\text{poly-x-minus-y} p) i) = \text{degree} p - i$
proof
consider $i = \text{degree} p \mid i > \text{degree} p \mid i < \text{degree} p$
  by force
thus $?\text{thesis}$
proof cases
  assume $i > \text{degree} p$
  thus $?\text{thesis}$ by (subst coeff-eq-0) auto
next
  assume $i = \text{degree} p$
  thus $?\text{thesis}$ using lead-coeff-poly-x-minus-y[of p]
    by (simp add: lead-coeff-poly-x-minus-y)
next
  assume $i < \text{degree} p$
  define $n$ where $n = \text{degree} p$
  have $\text{degree} (\text{coeff} (\text{poly-x-minus-y} p) i) =$
    $\text{degree} (\sum^\text{n} \text{coeff} p j) * \text{coeff} (x-y ^ j) i)$
    (is $= \text{degree} (\sum ?f)$)
    by (simp add: poly-x-minus-y-def pcompose-conv-poly poly-altdef coeff-sum)
  also have $\{..n\} = \text{insert} n \{..<n\}$
    by auto
  also have $\text{sum} ?f \ldots = ?f n + \text{sum} ?f \{..<n\}$
    by (subst sum.insert) auto
  also have $\text{degree} \ldots = n - i$
  proof
    have $\text{degree} (\text{if} n) = n - i$
      using $i < \text{degree} p$
      by (simp add: n-def coeff-xy-power degree-monom-eq)
    moreover have $\text{degree} (\text{sum} ?f \{..<n\}) < n - i$
    proof (intro degree-sum-smaller)
      fix $j$
      assume $j \in \{..<n\}$
      have $\text{degree} ([\text{coeff} p j] * \text{coeff} (x-y ^ j) i) \leq j - i$
        proof (cases $i \leq j$
          case True
          thus $?\text{thesis}$
          by (auto simp: n-def coeff-xy-power degree-monom-eq)
        next
        case False
  qed

69
hence $\text{coeff } (x - y \cdot j :: 'a \text{ poly poly}) \cdot i = 0$
by (subst coeff-eq-0) (auto simp: degree-power-eq)
thus $?\text{thesis}$ by simp
qed
also have $\ldots < n - i$
using $j \in \{..<n\} :: i < \text{degree } p$ by (auto simp: n-def)
finally show $\text{degree } ([\text{coeff } p \cdot j] \ast \text{coeff } (x - y \cdot j) \cdot i) < n - i$.
qed (use $i < \text{degree } p$ in (\text{auto simp: n-def}))
ultimately show $?\text{thesis}$ by simp
qed

lemma coeff-0-poly-x-minus-y [simp]: $\text{coeff } (\text{poly-x-minus-y } p) \cdot 0 = p$
by (induction p) (auto simp: poly-x-minus-y-def x-y-def)

lemma (in idom-hom) poly-add-hom:
assumes $p0: \text{hom } (\text{lead-coeff } p) \neq 0$ and $q0: \text{hom } (\text{lead-coeff } q) \neq 0$
shows $\text{map-poly } \text{hom } (\text{poly-add } p \cdot q) = \text{poly-add } (\text{map-poly } \text{hom } p) \cdot (\text{map-poly } \text{hom } q)$
proof -
interpret mh: map-poly-idom-hom..
show $?\text{thesis}$ unfolding poly-add-def
apply (subst mh.resultant-map-poly(1)[symmetric])
apply (subst degree-map-poly-2)
apply (unfold lead-coeff-poly-x-minus-y, unfold hom-distrib, simp add: p0)
apply simp
apply (subst degree-map-poly-2)
apply (simp-all add: q0 map-poly-x-minus-y)
done
qed

lemma (in zero-hom) hom-lead-coeff-nonzero-imp-map-poly-hom:
assumes $\text{hom } (\text{lead-coeff } p) \neq 0$
shows $\text{map-poly } \text{hom } p \neq 0$
proof
assume $\text{map-poly } \text{hom } p = 0$
then have $\text{coeff } (\text{map-poly } \text{hom } p) \cdot (\text{degree } p) = 0$ by simp
with assms show False by simp
qed

lemma ipoly-poly-add:
fixes $x \cdot y :: 'a :: \text{idom}$
assumes $p0: \text{(of-int } (\text{lead-coeff } p) :: 'a) \neq 0$ and $q0: \text{(of-int } (\text{lead-coeff } q) :: 'a) \neq 0$
and $x: \text{ipoly } p \cdot x = 0$ and $y: \text{ipoly } q \cdot y = 0$
shows \( \text{ipoly (poly-add } p \ q ) (x+y) = 0 \)
using assms of-int-hom.hom-lead-coeff-nonzero-imp-map-poly-hom[OF q0]
by (auto intro: poly-add simp: of-int-hom.poly-add-hom[OF p0 q0])

lemma (in comm-monoid-gcd) gcd-list-eq-0-iff[simp]: listgcd \( xs = 0 \) \( \iff \) (\( \forall x \in \text{set } xs. x = 0 \))
by (induct xs, auto)

lemma primitive-field-poly[simp]:
\( \text{primitive } (p :: ’a :: field poly) \iff p \neq 0 \)
by (unfold primitive-iff-some-content-dvd-1,auto simp: dvd-field-iff coeffs-def)

lemma ipoly-poly-add-nonzero:
fixes \( x \ y :: ’a :: field \)
assumes \( p \neq 0 \) and \( q \neq 0 \) and \( \text{ipoly } p \ x = 0 \) and \( \text{ipoly } q \ y = 0 \)
and (of-int (lead-coeff p) :: ’a) \( \neq 0 \) and (of-int (lead-coeff q) :: ’a) \( \neq 0 \)
shows poly-add \( p \ q \neq 0 \)
proof-
from assms have (of-int-poly (poly-add p q) :: ’a poly) \( \neq 0 \)
apply (subst of-int-hom.poly-add-hom,simp)
by (rule poly-add-nonzero, auto dest: of-int-hom.hom-lead-coeff-nonzero-imp-map-poly-hom)
then show \( ?\text{thesis} \) by auto
qed

lemma represents-add:
assumes \( x \ y :: p \text{ represents } x \) and \( y :: q \text{ represents } y \)
shows (poly-add \( p \ q \)) \text{represents } (\( x + y \))
using assms by (intro representsI ipoly-poly-add ipoly-poly-add-nonzero, auto)

5.2 Division of Algebraic Numbers

definition poly-x-mult-y where
[code del]: poly-x-mult-y \( p \equiv (\sum \{i \leq \text{degree } p. \text{monom } (\text{monom } (\text{coeff } p \ i) \ i) \ i\}) \)

lemma coeff-poly-x-mult-y:
shows coeff (poly-x-mult-y \( p \)) \( i = \text{monom } (\text{coeff } p \ i) \ i \) \( (?l = ?r) \)
proof(cases degree \( p < i \))
case False
have \( ?l = \text{sum } (\lambda j. \text{if } j = i \then (\text{monom } (\text{coeff } p \ j) \ j) \text{ else } 0) \) \{..degree \( p \}\)
(is - = \text{sum } ?f ?A) by (simp add: poly-x-mult-y-def coeff-sum)
also have \( ... = \text{sum } ?f \{i\} \) using \( i \) by (intro sum.mono-neutral-right, auto)
also have \( ... = ?f \ i \) by simp
also have \( ... = ?r \) by auto
finally show \( ?\text{thesis} \).
next
case True then show \( ?\text{thesis} \) by (auto simp: poly-x-mult-y-def coeff-eq-0 coeff-sum)
qed

lemma poly-x-mult-y-code[code]: poly-x-mult-y \( p = (\text{let } cs = \text{coeffs } p\)
in poly-of-list (map (λ (i, ai). monom ai i) (zip [0..< length cs] cs)))

proof (rule poly-eqI, unfold coeff-poly-x-mult-y)
  fix n
  let ?xs = zip [0..<length (coeffs p)] (coeffs p)
  let ?f = (λ(i, ai). monom ai i)
  show monom (coeff p n) n = coeff (Poly (map ?f ?xs)) n
  proof (cases n < length (coeffs p))
    case True
      hence n: n < length (map ?f ?xs) and nn: n < length ?xs
      unfolding degree-eq-length-coeffs by auto
        using True by (simp add: nth-coeffs-coeff)
  next
    case False
      hence id: coeff (Poly (map ?f ?xs)) n = 0 unfolding coeff-Poly
      by (subst nth-default-beyond, auto)
      from False have n > degree p ∨ p = 0 unfolding degree-eq-length-coeffs by (cases n, auto)
      hence monom (coeff p n) n = 0 using coeff-eq-0[of p n] by auto
      thus ?thesis unfolding id by simp
  qed

qed

definition poly-div :: 'a :: comm-ring-1 poly ⇒ 'a poly ⇒ 'a poly where
  poly-div p q = resultant (poly-x-mult-y p) (poly-lift q)

  poly-div has desired roots.

lemma poly2-poly-x-mult-y:
  fixes p :: 'a :: comm-ring-1 poly
  shows poly2 (poly-x-mult-y p) x y = poly p (x * y)
  apply (subst(3) poly-as-sum-of-monoms[symmetric])
  apply (unfold poly-x-mult-y-def hom-distribs)
  by (auto simp: poly2-monom poly-monom power-mult-distrib ac-simps)

lemma poly-div:
  fixes p q :: 'a :: field poly
  assumes q0: q ≠ 0 and x: poly p x = 0 and y: poly q y = 0 and y0: y ≠ 0
  shows poly (poly-div p q) (x/y) = 0
  proof (unfold poly-div-def, rule poly-resultant-zero[OF disjI2])
    have degree q > 0 using poly-zero q0 y by auto
    thus degq: degree (poly-lift q) > 0 by auto
  qed (insert x y y0, simp-all add: poly2-poly-x-mult-y)

  poly-div is nonzero.

interpretation poly-x-mult-y-hom: ring-hom poly-x-mult-y :: 'a :: {idom,ring-char-0}
  poly ⇒ 
  by (unfold-locales, auto intro: poly2-ext simp: poly2-poly-x-mult-y hom-distribs)
interpretation poly-x-mult-y-hom: inj-ring-hom poly-x-mult-y :: 'a :: {idom, ring-char-0} poly ⇒ -
proof
let ?h = poly-x-mult-y
fix f :: 'a poly
assume ?h f = 0
then have poly2 (?h f) x 1 = 0 for x by simp
from this[unfolded poly2-poly-x-mult-y]
show f = 0 by auto
qed

lemma degree-poly-x-mult-y[simp]:
fixes p :: 'a :: {idom, ring-char-0} poly
shows degree (poly-x-mult-y p) = degree p (is ?l = ?r)
proof(rule antisym)
show ?r ≤ ?l by (cases p=0, auto intro: le-degree simp: coeff-poly-x-mult-y)
show ?l ≤ ?r unfolding poly-x-mult-y-def
by (auto intro: degree-sum-le le-trans[OF degree-monom-le])
qed

interpretation poly-x-mult-y-hom: unit-preserving-hom poly-x-mult-y :: 'a :: field-char-0 poly ⇒ -
proof(unfold-locales)
let ?h = poly-x-mult-y :: 'a poly ⇒ -
fix f :: 'a poly
assume unit: ?h f dvd 1
then have degree (?h f) = 0 and coeff (?h f) 0 dvd 1 unfolding poly-dvd-1 by auto
then have deg: degree f = 0 by (auto simp add: degree-monom-eq)
with unit show f dvd 1 by(cases f = 0, auto)
qed

lemmas poly-y-x-o-poly-lift = o-def[of poly-y-x poly-lift, unfolded poly-y-x-poly-lift]

lemma irreducible-dvd-degree: assumes (f::'a::field poly) dvd g
irreducible g
degree f > 0
shows degree f = degree g
using assms
by (metis irreducible-altdef degree-0 dvd-refl is-unit-field-poly linorder-neqE-nat
poly-divides-conv0)

lemma coprime-poly-x-mult-y-poly-lift:
fixes p q :: 'a :: field-char-0 poly
assumes degp: degree p > 0 and degq: degree q > 0
and nz: poly p 0 ≠ 0 ∨ poly q 0 ≠ 0
shows coprime (poly-x-mult-y p) (poly-lift q)
proof(rule ccontr)
from degp have p: ¬ p dvd 1 by (auto simp: dvd-const)
from \( \text{degp} \) have \( p0 \): \( p \neq 0 \) by auto
from \( \text{mset-factors-exist[of } p, \text{ OF } p0 \ p] \)
obtain \( F \) where \( F: \text{mset-factors } F \ p \) by auto
then have \( pF\): \( \text{prod-mset } (\text{image-mset } \text{poly-x-mult-y } F) = \text{poly-x-mult-y } p \)
    by (auto simp: \text{hom-distribs})

from \( \text{degq} \) have \( q\): \( \neg \text{is-unit } q \) by (auto simp: \text{dvd-const})
from \( \text{degq} \) have \( q0\): \( q \neq 0 \) by auto
from \( \text{mset-factors-exist[of } q0 \ q] \)
obtain \( G \) where \( G: \text{mset-factors } G \ q \) by auto
with \( \text{poly-lift-hom.hom-mset-factors} \)
have \( pG\): \( \text{mset-factors } (\text{image-mset } \text{poly-lift } G) \) (\text{poly-lift } q) by auto
from \( \text{poly-y-x-hom.hom-mset-factors[of this]} \)
have \( pG\): \( \text{mset-factors } (\text{image-mset } \text{coeff-lift } G) \) [:\( q\)]
    by (auto simp: \text{poly-y-x-poly-lift monom-0 image-mset.compositionality poly-y-x-o-poly-lift})

assume \( \neg \text{coprime } (\text{poly-x-mult-y } p) \) (\text{poly-lift } q)
then have \( \neg \text{coprime } (\text{poly-y-x } (\text{poly-x-mult-y } p)) \) (\text{poly-y-x } (\text{poly-lift } q))
    by (simp del: \text{coprime-iff-coprime})
from this[unfolded \text{not-coprime-iff-common-factor}]
obtain \( r \)
where \( rp\): \( r \text{ dvd } \text{poly-y-x } (\text{poly-x-mult-y } p) \)
    and \( rq\): \( r \text{ dvd } \text{poly-y-x } (\text{poly-lift } q) \)
    and \( rU\): \( \neg r \text{ dvd } 1 \) by auto
from \( rp \ p0 \) have \( r0\): \( r \neq 0 \) by auto
from \( \text{mset-factors-exist[of } r0 \ rU] \)
obtain \( H \) where \( H: \text{mset-factors } H \ r \) by auto
then have \( H \neq \{\text{\#}\} \) by auto
then obtain \( h \) where \( hH\): \( h \in \# \text{ H by fastforce} \)
with \( H \text{ mset-factors-imp-dvd have hr: } h \text{ dvd } r \) and \( h: \text{irreducible } h \) by auto
from \( \text{irreducible-not-unit[of } H \] have \( hU\): \( \neg h \text{ dvd } 1 \) by auto
from \( hr \ rp \) have \( h \text{ poly-y-x } (\text{poly-x-mult-y } p) \) by (rule \text{dvd-trans})

note this[\text{folded } pF, \text{unfolded } \text{poly-y-x-hom.hom-prod-mset image-mset.compositionality}]    
from \( \text{prime-elem-dvd-prod-mset[of } H[\text{folded } \text{prime-elem-iff-irreducible} \] this]} \)
obtain \( f \) where \( f \in \# \text{ F and hf: } h \text{ dvd } \text{poly-y-x } (\text{poly-x-mult-y } f) \) by auto
have \( \text{irrF: irreducible } f \) using \( F \) by blast
from \( \text{dvd-trans[of } hr \ q0] \) have \( h \text{ dvd } [:\text{q}] \) by (simp add: \text{poly-y-x-poly-lift monom-0})
    from \( \text{irreducible-dvd-imp-factor[of } \text{this } h \ pG \ q0} \)
    obtain \( g \) where \( g \in \# \text{ G and gh: } [:\text{g}] \) \text{ dvd } h \) by auto
    from \( \text{dvd-trans[of } gh \ hf] \) have \( [:\text{g}] \) \text{ dvd } \text{poly-y-x } (\text{poly-x-mult-y } f) \) using \text{dvd-trans by auto}
show \( \text{False} \)
proof (cases poly \( f \ 0 = 0 \))
case \( f0\): \( \text{False} \)
    from \( \text{poly-hom.hom-dvd[of } *] \)
    have \( g \text{ dvd poly } (\text{poly-y-x } (\text{poly-x-mult-y } f)) [:0] \) by simp
    also have \( \text{...} = [:\text{poly } f 0:] \) by (intro \text{poly-ext, fold poly2-def, simp add: poly2-poly-x-mult-y})
also have ... dvd 1 using f-0 by auto
finally have g dvd 1.
with g G show False by (auto elim!: mset-factorsE dest!: irreducible-not-unit)
next
case True
hence [:0,1:] dvd f by (unfold dvd-iff-poly-eq-0, simp)
from irreducible-dvd-degree[OF this irrF]
have degree f = 1 by auto
from degree1-coeffs[OF this] True obtain c where c: c ≠ 0 and f: f = [:0,c:]
by auto
from g G have irrG: irreducible g by auto
from poly-hom.hom-dvd[OF *]
have g dvd poly (poly-y-x (poly-x-mult-y f)) 1 by simp
also have ... = f by (auto simp: f poly-x-mult-y-code Let-def c poly-y-x-pCons
map-poly-monom poly-monom poly-lift-def)
also have ... dvd [:0,1:] unfolding f dvd-def using c
by (intro exI[of - c], auto)
finally have g01: g dvd [:0,1:]
from divides-degree[OF this irrG] have degree g = 1 by auto
from degree1-coeffs[OF this] obtain a b where g: g = [:b,a:] and a: a ≠ 0 by auto
from g01[unfolded dvd-def] g obtain k where id [:0,1:] = g * k by auto
from id have 0: g ≠ 0 k ≠ 0 by auto
from arg-cong[OF id, of degree] have degree k = 0 unfolding degree-mult-eq[OF 0]
unfolding g using a by auto
from degree0-coeffs[OF this] obtain kk where k: k = [:kk:] by auto
from id[unfolded g k] a have b = 0 by auto
hence poly g 0 = 0 by (auto simp: g)
from True this nz f ∈ # F: g ∈ # G: F G
show False by (auto dest!: mset-factors-imp-dvd elim:dvdE)
qed
qed

lemma poly-div-nonzero:
fixes p q :: 'a :: field-char-0 poly
assumes p0: p ≠ 0 and q0: q ≠ 0 and x: poly p x = 0 and y: poly q y = 0
and p-0: poly p 0 ≠ 0 ∨ poly q 0 ≠ 0
shows poly-div p q ≠ 0
proof
have degp: degree p > 0 using le-0-eq order-degree order-root p0 x by (metis gr0I)
    have degq: degree q > 0 using le-0-eq order-degree order-root q0 y by (metis gr0I)
    assume 0: poly-div p q = 0
from resultant-zero-imp-common-factor[OF - this[unfolded poly-div-def]] degp
    and coprime-poly-x-mult-y-poly-lift[OF degp degq] p-0
show False by auto
qed

75
5.2.1 Summary for division

Now we lift the results to one that uses ipoly, by showing some homomorphism lemmas.

**lemma (in inj-comm-ring-hom) poly-x-mult-y-hom:**

\[ \text{poly-x-mult-y (map-poly p) = map-poly (map-poly hom (poly-x-mult-y p))} \]

**proof** –

- interpret mh: map-poly-inj-comm-ring-hom
- interpret mmh: map-poly-inj-comm-ring-hom map-poly hom
- show ?thesis unfolding poly-x-mult-y-def by (simp add: hom-distribs)

qed

**lemma (in inj-comm-ring-hom) poly-div-hom:**

\[ \text{map-poly hom (poly-div p q) = poly-div (map-poly hom p) (map-poly hom q)} \]

**proof** –

- have zero: \( \forall x. \text{hom x} = 0 \rightarrow x = 0 \) by simp
- interpret mh: map-poly-inj-comm-ring-hom
- show ?thesis unfolding poly-div-def mh resultant-hom [symmetric]
  - by (simp add: poly-x-mult-y-hom)

qed

**lemma ipoly-poly-div:**

fixes \( x, y :: 'a :: \text{field-char-0} \)
assumes \( q \neq 0 \) and ipoly p x = 0 and ipoly q y = 0 and y \neq 0
shows ipoly (poly-div p q) (x/y) = 0
by (unfold of-int-hom poly-div-hom, rule poly-div, insert assms, auto)

**lemma ipoly-poly-div-nonzero:**

fixes \( x, y :: 'a :: \text{field-char-0} \)
assumes \( p \neq 0 \) and \( q \neq 0 \) and ipoly p x = 0 and ipoly q y = 0 and \( \text{poly p} \neq 0 \)
\( \neq 0 \lor \text{poly q} \neq 0 \)
shows poly-div p q \neq 0
**proof** –

- from assms have (of-int-poly (poly-div p q) :: 'a poly) \( \neq 0 \) using of-int-hom poly-map-poly[of p]
  - by (subst of-int-hom poly-div-hom, subst poly-div-nonzero, auto)
  then show ?thesis by auto

qed

**lemma represents-div:**

fixes \( x, y :: 'a :: \text{field-char-0} \)
assumes \( p \) represents \( x \) and \( q \) represents \( y \) and \( \text{poly q} \neq 0 \)
shows (poly-div p q) represents \( (x / y) \)
using assms by (intro representsI ipoly-poly-div ipoly-poly-div-nonzero, auto)

5.3 Multiplication of Algebraic Numbers

**definition poly-mult where** poly-mult p q \( \equiv \) poly-div p (reflect-poly q)
lemma represents-mult:
  assumes px: p represents x and qy: q represents y and q-0: poly q 0 ≠ 0
  shows (poly-mult p q) represents (x * y)
proof-
  from q-0 qy have y0: y ≠ 0 by auto
  from represents-inverse[OF y0 qy] y0 px q-0
  have poly-mult p q represents x / (inverse y)
  unfolding poly-mult-def by (intro represents-div, auto)
  with y0 show ?thesis by (simp add: field-simps)
qed

5.4 Summary: Closure Properties of Algebraic Numbers

lemma algebraic-representsI: p represents x ⇒ algebraic x
  unfolding represents-def algebraic-altdef-ipoly by auto

lemma algebraic-of-rat: algebraic (of-rat x)
  by (rule algebraic-representsI[OF poly-rat-represents-of-rat])

lemma algebraic-uminus: algebraic x ⇒ algebraic (−x)
  by (auto dest: algebraic-imp-represents-irreducible intro: algebraic-representsI represents-uminus)

lemma algebraic-inverse: algebraic x ⇒ algebraic (inverse x)
  using algebraic-of-rat[of 0]
  by (cases x = 0, auto dest: algebraic-imp-represents-irreducible intro: algebraic-representsI represents-inverse)

lemma algebraic-plus: algebraic x ⇒ algebraic y ⇒ algebraic (x + y)
  by (auto dest!: algebraic-imp-represents-irreducible-cf-pos intro!: algebraic-representsI[OF represents-add])

lemma algebraic-div:
  assumes x: algebraic x and y: algebraic y shows algebraic (x/y)
proof(cases y = 0 ∨ x = 0)
  case True
  then show ?thesis using algebraic-of-rat[of 0] by auto
next
case False
  then have x0: x ≠ 0 and y0: y ≠ 0 by auto
  from x y obtain p q
  where px: p represents x and irr: irreducible q and qy: q represents y
  by (auto dest!: algebraic-imp-represents-irreducible)
  show ?thesis
  using False px represents-irr-non-0[OF irr qy]
  by (auto intro!: algebraic-representsI[OF represents-div] qy)
qed

lemma algebraic-times: algebraic x ⇒ algebraic y ⇒ algebraic (x * y)
using algebraic-die[OF - algebraic-inverse, of x y] by (simp add: field-simps)

lemma algebraic-root: algebraic x ⟹ algebraic (root n x)
proof
assume algebraic x
then obtain p where p: p represents x by (auto dest: algebraic-imp-represents-irreducible-cf-pos)
from algebraic-representsI[OF represents-nth-root-neg-real[OF - this, of n]]
algebraic-representsI[OF represents-nth-root-pos-real[OF - this, of n]]
algebraic-of-rat[of 0]
show ?thesis by (cases n = 0, force, cases n > 0, force, cases n < 0, auto)
qed

lemma algebraic-nth-root: n ≠ 0 ⟹ algebraic x ⟹ y^n = x ⟹ algebraic y
by (auto dest: algebraic-imp-represents-irreducible-cf-pos intro: algebraic-representsI)

5.5 More on algebraic integers

definition poly-add-sign :: nat ⇒ nat ⇒ 'a :: comm-ring-1 where
poly-add-sign m n = signof (λ i. if i < n then m + i else if i < m + n then i − n else i)

lemma lead-coeff-poly-add:
fixes p q :: 'a :: {idom, semiring-char-0} poly
defines m ≡ degree p and n ≡ degree q
assumes lead-coeff p = 1 lead-coeff q = 1 m > 0 n > 0
shows lead-coeff (poly-add p q :: 'a poly) = poly-add-sign m n
proof
from assms have [simp]: p ≠ 0 q ≠ 0
by auto
define M where M = sylvester-mat (poly-x-minus-y p) (poly-lift q)
define π :: nat ⇒ nat where
π = (λ i. if i < n then m + i else if i < m + n then i − n else i)
have π: π permutes {0..<m+n}
by (rule inj-on-nat-permutes) (auto simp: π-def inj-on-def)
have nz: M $$( i, π i) ≠ 0 if i < m + n for i
using that by (auto simp: M-def π-def sylvester-index-mat m-def n-def)

have indices-eq: {0..<m+n} = {..<n} ∪ (+) n · {..<m}
by (auto simp flip: atLeast0LessThan)
define f where f = (λ σ. signof σ * (∏ i=0..<m+n. M $$ (i, σ i)))
have degree (f π) = degree (∏ i=0..<m+n. M $$ (i, π i))
using nz by (auto simp: f-def degree-mult-eq signof-def)
also have ... = (∑ i=0..<m+n. degree (M $$ (i, π i)))
using nz by (subst degree-prod-eq-sum-degree) auto
also have \( \sum i \prec n. \deg (M \times i, \pi i)) + (\sum i \prec m. \deg (M \times (n + i, \pi (n + i)))) \)
\( \) by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
also have \((\sum i \prec n. \deg (M \times (i, \pi i))) = (\sum i \prec n. m) \)
\( \) by (intro sum.cong) (auto simp: M-def sylvestery-index-mat π-def m-def n-def)
also have \((\sum i \prec m. \deg (M \times (n + i, \pi (n + i)))) = (\sum i \prec m. 0) \)
\( \) by (intro sum.cong) (auto simp: M-def sylvestery-index-mat π-def m-def n-def)
finally have deg-fl: \( \deg (f \pi) = m \times n \)
\( \) by simp

have deg-2: \( \deg (f \sigma) < m \times n \) if \( \sigma \) permutes \( \{0..<m+n\} \) \( \sigma \neq \pi \) for \( \sigma \)
proof (cases \( \exists i \in \{0..<m+n\}. M \times i, \sigma i = 0 \))
case True
hence \( \ast \): \((\prod i = 0..<m + n. M \times (i, \sigma i)) = 0 \)
\( \) by auto
show \( \th \) using \( m > 0 \) \( n > 0 \)
\( \) by (simp add: f-def \ast )
next
case False
note nz = this
from that have \( \sigma \)-less: \( \sigma \ i < m + n \) if \( i < m + n \) for \( i \)
using permutes-in-image[OF \( \sigma \) permutes -] that by auto
have degree \( (f \sigma) = \deg ((\prod i = 0..<m + n. M \times (i, \sigma i)) \)
\( \) by (auto simp: f-def degree-mult-eq sigmoid-def)
also have \( \ldots = (\sum i = 0..<m+n. \deg (M \times (i, \sigma i))) \)
\( \) by (subst degree-prod-eq-sum-degree) auto
also have \( \ldots = (\sum i \prec n. \deg (M \times (i, \sigma i))) + (\sum i \prec m. \deg (M \times (n + i, \sigma (n + i)))) \)
\( \) by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
also have \((\sum i \prec m. \deg (M \times (n + i, \sigma (n + i)))) = (\sum i \prec m. 0) \)
\( \) using \( \sigma \)-less by (intro sum.cong) (auto simp: M-def sylvestery-index-mat π-def m-def n-def)
also have \((\sum i \prec n. \deg (M \times (i, \sigma i))) < (\sum i \prec n. m) \)
\( \) proof (rule sum-strict-mono-ex1)
show \( \forall x \epsilon \{1..n\}. \deg (M \times (x, \sigma x)) \leq m \) using \( \sigma \)-less
\( \) by (auto simp: M-def sylvestery-index-mat π-def m-def n-def degree-coeff-poly-x-minus-y)
next

have \( \exists i \prec n. \sigma i \neq \pi i \)
proof (rule ccontr)
assume \( \neg (\exists i \prec n. \sigma i \neq \pi i) \)
have \( \forall i \geq m + n - k. \sigma i = \pi i \) if \( k \leq m \) for \( k \)
\( \) using that
proof (induction \( k \))
case 0
thus \( \forall i \prec m. \deg (\pi \sigma) \leq m \) using \( \pi \sigma \)-permutes- ⊨ \( \sigma \) permutes- ⊨
by (fastforce simp: permutes-def)
next
case (Suc \( k \))
have IH: \( \sigma \ i = \pi \ i \) if \( i \geq m+n-k \) for \( i \)
using Suc.prems Suc.IH that by auto
from nz have \( M \$$ (m + n – Suc k, \sigma (m + n – Suc k)) \neq 0 \)
using Suc.prems by auto
moreover have \( m + n – Suc k \geq n \)
using Suc.prems by auto
ultimately have \( \sigma (m+n–Suc k) \geq m–Suc k \)
using assms \( \sigma\)-less[of \( m+n–Suc k \)] Suc.prems
by (auto simp: M-def sylvestre-index-mat \( m\)-def \( n\)-def split: if-splits)
have \( \neg(\sigma (m+n–Suc k) > m–Suc k) \)
proof
assume \(*\): \( \sigma (m+n–Suc k) > m–Suc k \)
have less: \( (m+n–Suc k) < m \)
proof (rule ccontr)
  assume \(*\): \( \neg(\sigma (m+n–Suc k) < m) \)
  define \( j \) where \( j = \sigma (m+n–Suc k) – m \)
  have \( \sigma (m+n–Suc k) = m+j \)
    using \(*\) by (simp add: j-def)
  moreover {
    have \( j < n \)
      using \( \sigma\)-less[of \( m+n–Suc k \)] \( \langle m > 0 \rangle \ \langle n > 0 \rangle \)
      by (simp add: j-def)
    hence \( \sigma j = \pi j \)
      using nex by auto
    with \( j < n \) have \( \sigma j = m+j \)
      by (auto simp: \( \pi\)-def)
  }
ultimately have \( \sigma (m+n–Suc k) = \sigma j \)
  by simp
hence \( m+n–Suc k = j \)
  using permutes-inj[OF \( \sigma \) permutes -] unfolding inj-def by blast
thus False using \( \langle n \leq m+n–Suc k \rangle \ \sigma\)-less[of \( m+n–Suc k \)] \( \langle n > 0 \rangle \)
  unfolding \( \pi\)-def by linarith
qed

define \( j \) where \( j = \sigma (m+n–Suc k) – (m–Suc k) \)
from \(*\) have \( j: \sigma (m+n–Suc k) = m–Suc k + j \ j > 0 \)
by (auto simp: j-def)
have \( \sigma (m+n–Suc k + j) = \pi (m+n–Suc k + j) \)
  using \(*\) by (intro IH) (auto simp: j-def)
also {
  have \( j < Suc k \)
    using less by (auto simp: j-def algebra-simps)
  hence \( m+n–Suc k + j < m+n \)
    using \( m > 0 \) \( n > 0 \) Suc.prems by linarith
  hence \( \pi (m+n–Suc k + j) = m–Suc k + j \)
    unfolding \( \pi\)-def using Suc.prems by (simp add: \( \pi\)-def)
}
finally have \( \sigma (m+n–Suc k + j) = \sigma (m+n–Suc k) \)
  using \( j \) by simp

80
hence $m + n - \text{Suc } k + j = m + n - \text{Suc } k$
using permutes-inj[OF $\sigma$ permutes -] unfolding inj-def by blast
thus False using $j > 0$ by simp
qed
with $\sigma \ (m+n-\text{Suc } k) \geq m-\text{Suc } k$ have eq: $\sigma \ (m+n-\text{Suc } k) = m - \text{Suc } k$
by linarith

show ?case
proof safe
fix $i :: \text{nat}$
assume $i: i \geq m + n - \text{Suc } k$
show $\sigma \ i = \pi \ i$
using eqSuc.prems $(m > 0); \text{IH } i$
proof (cases $i = m + n - \text{Suc } k$)
case True
thus ?thesis using eqSuc.prems $(m > 0)$
by (auto simp: $\pi$-def)
qed (use $\text{IH } i$ in auto)
qed

from this[of m] and $\text{nex}$ have $\sigma \ i = \pi \ i$ for $i$
by (cases $i \geq n$) auto
hence $\sigma = \pi$ by force
thus False using $\langle \sigma \neq \pi \rangle$ by contradiction
qed

then obtain $i$ where $i: i < n \ \sigma \ i \neq \pi \ i$
by auto
have $\sigma \ i < m + n$
using $i$ by (intro $\sigma$-less) auto
moreover have $\pi \ i = m + i$
using $i$ by (auto simp: $\pi$-def)
ultimately have degree $(M \ (i, \sigma \ i)) < m$ using $i$ $(m > 0)$
by (auto simp: $\text{M-def} \ m$-def $n$-def $\text{syvlerster-index-mat} \ \text{degree-coeff-poly-x-minus-y}$)
thus $\exists i \in \langle..<n\rangle$. degree $(M \ (i, \sigma \ i)) < m$
using $i$ by blast
qed auto

finally show degree $(f \ \sigma) < m * n$
by (simp add: $\text{mult-ac}$)
qed

have lead-coeff $(f \ \pi) = \text{poly-add-sign} \ m \ n$
proof
have lead-coeff $(f \ \pi) = \text{signof} \ \pi * (\prod_{i=0..<m+n}. \text{lead-coeff} \ (M \ (i, \pi \ i)))$
by (simp add: $\text{f-def} \ \text{signof-def} \ \text{lead-coeff-prod}$)
also have $(\prod_{i=0..<m+n}. \text{lead-coeff} \ (M \ (i, \pi \ i))) =$
$(\prod_{i<n}. \text{lead-coeff} \ (M \ (i, \pi \ i)) \ast (\prod_{i<m}. \text{lead-coeff} \ (M \ (n + i, \pi \ (n + i)))))$
by (subst indices-eq, subst prod.union-disjoint) (auto simp: prod.reindex)
also have $(\prod_{i<n. \text{lead-coeff} \ (M \# (i, \pi \ i))} = \prod_{i<n. \text{lead-coeff} \ p)}$
by (intro prod.cong) (auto simp: M-def m-def n-def π-def sylvester-index-mat)
also have $(\prod_{i<m. \text{lead-coeff} \ (M \# (n+i, \pi \ (n+i)))} = \prod_{i<m. \text{lead-coeff} \ q})$
by (intro prod.cong) (auto simp: prod.reindex)
also have $\text{signof} \ \pi = \text{poly-add-sign} \ m \ n$
by (simp add: π-def poly-add-sign-def m-def n-def cong)
finally show \text{idem} using \text{assms} by simp
qed

have $\text{lead-coeff} \ (\text{poly-add} \ p \ q) =$
lead-coeff (det (sylvester-mat (poly-x-minus-y p) (poly-lift q)))
by (simp add: poly-add-def resultant-def)
also have $\text{det} \ (\text{sylvester-mat} \ (\text{poly-x-minus-y} \ p) \ (\text{poly-lift} \ q)) =$
$(\sum_{\pi \mid \pi \ \text{permutes} \ \{0..<m+n\}}. f \ \pi)$
by (simp add: det-def M-def n-def π-def sylvester-index-mat)
also have $f \ (\pi. \pi \ \text{permutes} \ \{0..<m+n\}) = \text{insert} \ \pi \ \{\pi. \pi \ \text{permutes} \ 0..<m+n\}$
= (\pi)
using π by auto
also have $\sum_{\sigma \in \ldots f \ \sigma} = (\sum_{\sigma \in \sigma. \sigma \ \text{permutes} \ \{0..<m+n\}} - \{\pi\}, f \ \sigma) + f \ \pi$
by (subst sum.insert) (auto simp: finite-permutations)
also have $\text{lead-coeff} \ . \ldots = \text{lead-coeff} \ (f \ \pi)$
proof
have $\text{degree} \ (\sum_{\sigma \in \sigma. \sigma \ \text{permutes} \ \{0..<m+n\}} - \{\pi\}, f \ \sigma) < m * n$ using \text{assms}
by (intro degree-sum-smaller deg-f2) (auto simp: m-def n-def finite-permutations)
with \text{deg-f1} show \text{idem}
by (subst lead-coeff-add-le) auto
qed
finally show \text{idem}
using (lead-coeff (f \ \pi) = \pi) by simp
qed

lemma lead-coeff-poly-mult:
fixes \ p \ q :: \ 'a :: \ (idom, ring-char-0) \ poly
defines \ m :: \ \text{degree} \ p \ and \ n :: \ \text{degree} \ q
assumes \ lead-coeff \ p = 1 \ \text{lead-coeff} \ q = 1 \ m > 0 \ n > 0
assumes \ \text{coeff} \ q \ 0 \neq 0
shows $\text{lead-coeff} \ (\text{poly-mult} \ p \ q :: \ 'a \ \text{poly}) = 1$
proof
from \text{assms} have [simp]: \ p \neq 0 \ q \neq 0
by auto
have [simp]: \text{degree} \ (\text{reflect-poly} \ q) = n
using \text{assms} by (subst degree-reflect-poly-eq) (auto simp: n-def)
define \ M \ where \ M = \text{sylvester-mat} \ (\text{poly-x-mult-y} \ p) \ (\text{poly-lift} \ (\text{reflect-poly} \ q))
have nz: $M \not\equiv (i, i) \neq 0$ if $i < m + n$ for $i$
using that by (auto simp: M-def sylvester-index-mat m-def n-def coeff-poly-x-mult-y)

have indices-eq: $\{0..<m+n\} = \{<n\} \cup \{+\} n \cup \{..<m\}$
  by (auto simp flip: atLeast0LessThan)

define $f$ where $f = (\lambda \sigma. \text{signof } \sigma \ast (\prod i=0..<m+n. M \equiv (i, \sigma \ i)))$

have degree $(f id) = \text{degree} (\prod i=0..<m+n. M \equiv (i, i))$
  using nz by (auto simp: f-def degree-mult-eq signof-def)
also have \ldots $(\sum i=0..<m+n. \text{degree} (M \equiv (i, i)))$
  using nz by (auto simp: subst degree-prod-eq-sum-degree) auto

also have \ldots $(\sum i<n. \text{degree} (M \equiv (i, i))) + (\sum i<m. \text{degree} (M \equiv (n+i, n+i)))$
  by subst indices-eq, subst sum.union-disjoint (auto simp: sum.reindex)
also have $(\sum i<n. \text{degree} (M \equiv (i, i))) = (\sum i<n. m)$
  by (intro sum.cong)
  (auto simp: M-def sylvester-index-mat m-def n-def coeff-poly-x-mult-y degree-monom-eq)

also have $(\sum i<m. \text{degree} (M \equiv (n+i, n+i))) = (\sum i<m. 0)$
  by (intro sum.cong) (auto simp: M-def sylvester-index-mat m-def n-def)

finally have deg-f1: degree $(f id) = m * n$
  by (simp add: mult-ac id-def)

have deg-f2: degree $(f \sigma) < m * n$ if $\sigma$ permutes $\{0..<m+n\}$ $\sigma \neq id$ for $\sigma$
proof (cases $\exists i \in \{0..<m+n\}. M \equiv (i, \sigma \ i) = 0$)
case True
  hence \*: $(\prod i = 0..<m+n. M \equiv (i, \sigma \ i)) = 0$
  by auto

show \?thesis using \:$m > 0$ \:$n > 0$.
  by (simp add: f-def \*)
next

false

note nz = this

from that have $\sigma$-less: $\sigma < m + n$ if $i < m + n$ for $i$
  using permutes-in-image[OF $\sigma$ permutes -] that by auto
have degree $(f \sigma) = \text{degree} (\prod i=0..<m+n. M \equiv (i, \sigma \ i))$
  using nz by (auto simp: f-def degree-mult-eq signof-def)
also have \ldots $(\sum i=0..<m+n. \text{degree} (M \equiv (i, \sigma \ i)))$
  using nz by (auto simp: subst degree-prod-eq-sum-degree) auto

also have \ldots $(\sum i<n. \text{degree} (M \equiv (i, \sigma \ i))) + (\sum i<m. \text{degree} (M \equiv (n+i, n+i)))$
  by subst indices-eq, subst sum.union-disjoint (auto simp: sum.reindex)

also have $(\sum i<m. \text{degree} (M \equiv (n+i, \sigma \ i))) = (\sum i<m. 0)$
  using $\sigma$-less by (intro sum.cong) (auto simp: M-def sylvester-index-mat m-def n-def)

also have $(\sum i<n. \text{degree} (M \equiv (i, \sigma \ i))) < (\sum i<n. m)$
proof (rule sum-strict-mono-ex1)
  show $\forall x \in \{..<n\}. \text{degree} (M \equiv (x, \sigma \ x)) \leq m$ using $\sigma$-less
  by (auto simp: M-def sylvester-index-mat m-def n-def degree-coeff-poly-x-minus-y}
next
have \( \exists i < n. \sigma i \neq i \)
proof (rule ccontr)
assume nex: \( \neg (\exists i < n. \sigma i \neq i) \)
have \( \sigma i = i \) for \( i \)
using that
proof (induction \( i \) rule: less-induct)
case (less \( i \))
consider \( i < n \mid i \in \{n..<m+n\} \mid i \geq m + n \)
by force
thus ?case
proof cases
assume \( i < n \)
thus ?thesis using nex by auto
next
assume \( i \geq m + n \)
thus ?thesis using \( \sigma \) permutes \( \Rightarrow \)
by (auto simp: permutes-def)
next
assume \( i : i \in \{n..<m+n\} \)
have IH: \( \sigma j = j \) if \( j < i \) for \( j \)
using that less.prems by (intro less.IH) auto
from nz have \( M \) \( (i, \sigma i) \neq 0 \)
using \( i \) by auto
hence \( i \leq i \)
using \( i \) \( \sigma \)-less[of \( i \)] by (auto simp: M-def sylwester-index-mat m-def)
moreover have \( \sigma i \geq i \)
proof (rule ccontr)
assume *: \( \neg \sigma i \geq i \)
from * have \( \sigma (\sigma i) = \sigma i \)
by (subst IH) auto
hence \( \sigma i = i \)
using permutes-inj[OF \( \sigma \) permutes \( \Rightarrow \)] unfolding inj-def by blast
with * show False by simp
qed
ultimately show ?case by simp
qed
qed
hence \( \sigma = id \)
by force
with \( \sigma \neq id \) show False
by contradiction
qed
then obtain \( i \) where \( i : i < n \ \sigma i \neq i \)
by auto
have $\sigma \cdot i < m + n$
  using $\sigma$ by (intro $\sigma$-less) auto
hence $\deg (M \sum_{i=0}^{n} (i, \sigma \cdot i)) < m$ using $i \cdot m > 0$
by (auto simp: M-def m-def n-def sylvester-index-mat degree-coeff-poly-x-minus-y
  coeff-poly-x-mult-y intro le-less-trans[OF degree-monom-le])
thus $\exists i \in \{..<n\}. \deg (M \sum_{i=0}^{n} (i, \sigma \cdot i)) < m$
  using $\sigma$ by blast
qed auto
finally show $\deg (f \cdot \sigma) < m \cdot n$
  by (simp add: mult-ac)
qed

have $\leadcoeff (f \cdot id) = 1$
proof –
  have $\leadcoeff (f \cdot id) = (\prod_{i=0}^{m + n} \leadcoeff (M \sum_{i=0}^{n} (i, \sigma \cdot i)))$
  by (simp add: f-def signof-def lead-coeff-prod sign-id)
also have $(\prod_{i=0}^{m + n} \leadcoeff (M \sum_{i=0}^{n} (i, \sigma \cdot i))) =$
  $(\prod_{i=n}^{n+i} \leadcoeff (M \sum_{i=0}^{n} (i, \sigma \cdot i))) \cdot (\prod_{i=m}^{n+i} \leadcoeff (M \sum_{i=0}^{n} (n + i, \sigma \cdot i)))$
  by (subst indices-eq, subst prod.union_disjoint) (auto simp: prod.reindex)
also have $(\prod_{i=n}^{n+i} \leadcoeff (M \sum_{i=0}^{n} (i, \sigma \cdot i))) = (\prod_{i=n}^{n+i} \leadcoeff p) \cdot (\prod_{i=m}^{n+i} \leadcoeff (M \sum_{i=0}^{n} (n + i, \sigma \cdot i)))$
  using assms by (simp add: id-def)
finally show $\some{\deg}$
  using assms by (simp add: id-def)
qed

have $\leadcoeff (\poly_mult p q) = \leadcoeff (\det M)$
  by (simp add: poly-mult_def resultant_def M-def poly-div-def)
also have $\det M = \sum_{\pi} (\pi \permutes (0..<m+n), f \cdot \pi)$
  by (simp add: det-def m-def n-def M-def f-def)
also have $\{\pi. \pi \permutes (0..<m+n)\} \cdot \insert id \{\{\pi. \pi \permutes (0..<m+n)\} - \{id\}. f \cdot \sigma\} + f \cdot id$
  by (auto simp: permutes-id)
also have $(\sum_{\sigma} \cdots f \sigma) = (\sum_{\sigma} \{\sigma. \sigma \permutes (0..<m+n)\} - \{id\}. f \cdot \sigma\) + f \cdot id$
  by (auto simp: sum.insert) (auto simp: finite-permutations)
also have $\leadcoeff \cdots = \leadcoeff (f \cdot id)$
proof –
  have $\deg (\sum_{\sigma} \{\sigma. \sigma \permutes (0..<m+n)\} - \{id\}. f \cdot \sigma) < m \cdot n$ using assms
  by (intro degree-sum-smaller deg-f2) (auto simp: m-def n-def finite-permutations
    with deg-f1 show $\some{\deg}$
    by (subst lead-coeff-add-le) auto
qed
finally show ?thesis
  using ⟨lead-coeff (f id) = 1⟩ by simp
qed

lemma algebraic-int-plus [intro]:
  fixes x y :: 'a :: field-char-0
  assumes algebraic-int x algebraic-int y
  shows algebraic-int (x + y)
proof
  from assms(1) obtain p where p: lead-coeff p = 1 ipoly p x = 0
    by (auto simp: algebraic-int-altdef-ipoly)
  from assms(2) obtain q where q: lead-coeff q = 1 ipoly q y = 0
    by (auto simp: algebraic-int-altdef-ipoly)
  have deg-pos: degree p > 0 degree q > 0
    using p q by (auto intro!: Nat.gr0I elim!: degree-eq-zeroE)
  define r where r = poly-add-sign (degree p) (degree q) * poly-add p q
  have lead-coeff r = 1 using p q deg-pos
    by (simp add: r-def lead-coeff-mult poly-add-sign-def signof-def lead-coeff-poly-add)
  moreover have ipoly r (x + y) = 0
    using p q by (simp add: ipoly-poly-add r-def of_int_poly_hom.hom-mult)
  ultimately show ?thesis
    by (auto simp: algebraic-int-altdef-ipoly)
qed

lemma algebraic-int-times [intro]:
  fixes x y :: 'a :: field-char-0
  assumes algebraic-int x algebraic-int y
  shows algebraic-int (x * y)
proof (cases y = 0)
  case simp: False
  from assms(1) obtain p where p: lead-coeff p = 1 ipoly p x = 0
    by (auto simp: algebraic-int-altdef-ipoly)
  from assms(2) obtain q where q: lead-coeff q = 1 ipoly q y = 0
    by (auto simp: algebraic-int-altdef-ipoly)
  have deg-pos: degree p > 0 degree q > 0
    using p q by (auto intro!: Nat.gr0I elim!: degree-eq-zeroE)
  have [simp]: q ≠ 0
    using q by auto
  define n where n = Polynomial.order 0 q
  have monom 1 n dvd q
    by (simp add: n-def monom-1-dvd-iff)
  then obtain q' where q-split: q = q' * monom 1 n
    by auto
  have Polynomial.order 0 q = Polynomial.order 0 q' + n
    using q ≠ 0 unfolding q-split by (subst order-mult) auto
  hence poly q' 0 ≠ 0
    unfolding n-def using :q ≠ 0 by (simp add: q-split order-root)
have \( q' \): \( \text{ipoly } q' \ y = 0 \) lead-coeff \( q' = 1 \) using \( q' \)-split \( q \)
by (auto simp: of-int-poly-hom.hom-mult poly-monom lead-coeff-mult degree-monom-eq)

from this have \( \text{deg-pos}' \): degree \( q' > 0 \)
by (intro Nat.gr0I) (auto elim!: degree-eq-zeroE)

from \( q' \ 0 \neq 0 \) have [simp]: \( \text{coeff } q' \ 0 \neq 0 \)
by (auto simp: monom-1-dvd-iff poly-0-coeff-0)

have \( p \) represents \( x \) \( q' \) represents \( y \)
using \( p \ q' \) by (auto simp: represents-def)

hence \( \text{poly-mult } p \ q' \) represents \( x \ast y \)
by (rule represents-mult) (simp add: poly-0-coeff-0)

moreover have lead-coeff \( (\text{poly-mult } p \ q') = 1 \) using \( p \ \text{deg-pos } q \ \text{deg-pos}' \)

ultimately show \?thesis
by (auto simp: algebraic-int-altdef-ipoly represents-def)

qed auto

lemma algebraic-int-power [intro]:
\( \text{algebraic-int } (x :: 'a :: field-char-0) \implies \text{algebraic-int } (x ^ n) \)
by (induction \( n \)) auto

lemma algebraic-int-diff [intro]:
fixes \( x \ y :: 'a :: field-char-0 \)
assumes \( \text{algebraic-int } x \ \text{algebraic-int } y \)
shows \( \text{algebraic-int } (x - y) \)
using algebraic-int-plus[OF assms(1)] algebraic-int-minus[OF assms(2)] by simp

lemma algebraic-int-sum [intro]:
(\( \forall x. x \in A \implies \text{algebraic-int } (f x :: 'a :: field-char-0) \))
\implies \( \text{algebraic-int } (\text{sum } f \ A) \)
by (induction \( A \) rule: infinite-finite-induct) auto

lemma algebraic-int-prod [intro]:
(\( \forall x. x \in A \implies \text{algebraic-int } (f x :: 'a :: field-char-0) \))
\implies \( \text{algebraic-int } (\text{prod } f \ A) \)
by (induction \( A \) rule: infinite-finite-induct) auto

lemma algebraic-int-nth-root-real-iff:
\( \text{algebraic-int } (\text{root } n \ x) \iff n = 0 \lor \text{algebraic-int } x \)

proof

have algebraic-int \( x \) if algebraic-int (root \( n \ x) \ n \neq 0 \)
proof

from that(1) have algebraic-int (root \( n \ x \ ^ n) \)
by auto

also have root \( n \ x \ ^ n = (\text{if even } n \ \text{then } \vert x \vert \ \text{else } x) \)
using \( \text{sgn-power-root[of n } x\vert \vert \) by (auto simp: sgn-if split: if-splits)

finally show \?thesis
by (auto split: if-splits)
thus \( ?\text{thesis} \) by auto

\textbf{lemma} \ \texttt{algebraic-int-power-iff}: 
\[ \text{algebraic-int} (x \uparrow n :: 'a :: field-char-0) \iff n = 0 \lor \text{algebraic-int} \ x \]

\textbf{proof} –
\begin{enumerate}
  \item have \( \text{algebraic-int} \ x \) if \( \text{algebraic-int} (x \uparrow n) \ n > 0 \)
  \item show \( \text{poly} (\text{monom} 1 \ n) \ x = x \uparrow n \)
    \begin{enumerate}
      \item by (auto simp: \text{poly-monom})
    \end{enumerate}
\end{enumerate}

\textbf{qed} (use that in \langle auto simp: degree-monom-eq \rangle)

thus \( ?\text{thesis} \) by auto

\textbf{lemma} \ \texttt{algebraic-int-power-iff\'} [simp]: 
\[ n > 0 =\Rightarrow \text{algebraic-int} (x \uparrow n :: 'a :: field-char-0) \iff \text{algebraic-int} \ x \]

\textbf{by} (subst \text{algebraic-int-power-iff}) auto

\textbf{lemma} \ \texttt{algebraic-int-sqrt-iff} [simp]: 
\[ \text{algebraic-int} (\sqrt{x}) \iff \text{algebraic-int} \ x \]

\textbf{by} (simp add: sqrt-def algebraic-int-nth-root-real-iff)

\textbf{lemma} \ \texttt{algebraic-int-csqrt-iff} [simp]: 
\[ \text{algebraic-int} (\text{csqrt} \ x) \iff \text{algebraic-int} \ x \]

\textbf{proof} 
\begin{enumerate}
  \item assume \( \text{algebraic-int} (\text{csqrt} \ x) \)
  \item hence \( \text{algebraic-int} (\text{csqrt} \ x \uparrow 2) \)
    \begin{enumerate}
      \item by (rule \text{algebraic-int-power})
    \end{enumerate}
  \end{enumerate}

\textbf{thus} \( \text{algebraic-int} \ x \)

\begin{enumerate}
  \item by \text{simp}
\end{enumerate}

\textbf{qed auto}

\textbf{lemma} \ \texttt{algebraic-int-norm-complex} [intro]:
\begin{enumerate}
  \item assumes \( \text{algebraic-int} (z :: \text{complex}) \)
  \item shows \( \text{algebraic-int} (\text{norm} \ z) \)
\end{enumerate}

\textbf{proof} –
\begin{enumerate}
  \item from \text{assms} have \( \text{algebraic-int} (z \ast \text{cnj} \ z) \)
    \begin{enumerate}
      \item by \text{auto}
    \end{enumerate}
  \end{enumerate}

\textbf{also have} \( z \ast \text{cnj} \ z = \text{of-real} (\text{norm} \ z \uparrow 2) \)

\begin{enumerate}
  \item by (rule \text{complex-norm-square} \[ \text{symmetric} \])
\end{enumerate}

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

\begin{enumerate}
  \item by \text{simp}
\end{enumerate}

\textbf{qed}

\begin{enumerate}
  \item hide-const (open) \texttt{x-y}
\end{enumerate}

\textbf{end}
6 Separation of Roots: Sturm

We adapt the existing theory on Sturm’s theorem to work on rational numbers instead of real numbers. The reason is that we want to implement real numbers as real algebraic numbers with the help of Sturm’s theorem to separate the roots. To this end, we just copy the definitions of of the algorithms w.r.t. Sturm and let them be executed on rational numbers. We then prove that corresponds to a homomorphism and therefore can transfer the existing soundness results.

theory Sturm-Rat

imports
  Sturm-Sequences,Sturm-Theorem
  Algebraic-Numbers-Prelim
  Berlekamp-Zassenhaus,Square-Free-Int-To-Square-Free-GFp

begin

hide-const (open) UnivPoly.coeff

lemma root-primitive-part [simp]:
  fixes p :: 'a :: {semiring-gcd, semiring-no-zero-divisors} poly
  shows poly (primitive-part p) x = 0 ↔ poly p x = 0
proof(cases p = 0)
  case True
  then show ?thesis by auto
next
  case False
  have poly p x = content p * poly (primitive-part p) x
    by (metis content-times-primitive-part poly-smult)
  also have . . . = 0 ↔ poly (primitive-part p) x = 0 by (simp add: False)
  finally show ?thesis by auto
qed

lemma irreducible-primitive-part:
  assumes irreducible p and degree p > 0
  shows primitive-part p = p
using irreducible-content[OF assms(1), unfolded primitive-iff-content-eq-1] assms(2)
by (auto simp: primitive-part-def abs-poly-def)

6.1 Interface for Separating Roots

For a given rational polynomial, we need to know how many real roots are in a given closed interval, and how many real roots are in an interval \((-\infty, r]\).

datatype root-info = Root-Info (l-r: rat ⇒ rat ⇒ nat) (number-root: rat ⇒ nat)
hide-const (open) l-r
hide-const (open) number-root
definition count-roots-interval-sf :: real poly ⇒ (real ⇒ real ⇒ nat) × (real ⇒ nat) where
count-roots-interval-sf p = (let ps = sturm-squarefree p
in ((\(a \ b). \ \text{sign-changes} \ ps \ a - \ \text{sign-changes} \ ps \ b + \ \text{(if poly} \ p \ a = 0 \ \text{then} \ 1 \ \text{else} \ 0)),
(\a. \ \text{sign-changes-neq-inf} \ ps - \ \text{sign-changes} \ ps \ a)))

definition count-roots-interval :: real poly ⇒ (real ⇒ real ⇒ nat) × (real ⇒ nat) where
count-roots-interval p = (let ps = sturm p
in ((\(a \ b). \ \text{sign-changes} \ ps \ a - \ \text{sign-changes} \ ps \ b + \ \text{(if poly} \ p \ a = 0 \ \text{then} \ 1 \ \text{else} \ 0)),
(\a. \ \text{sign-changes-neq-inf} \ ps - \ \text{sign-changes} \ ps \ a)))

lemma count-roots-interval-iff: square-free p ⇒ count-roots-interval p = count-roots-interval-sf p
unfolding count-roots-interval-iff
square-free-iff-separable separable-def by (cases p = 0, auto)

lemma count-roots-interval-sf: assumes p: p ≠ 0
and cr: count-roots-interval-sf p = (cr,nr)
shows a ≤ b ⇒ cr a b = (card \{x. a ≤ x ∧ x ≤ b ∧ poly p x = 0\})
  nr a = card \{x. x ≤ a ∧ poly p x = 0\}
proof —
  have id: a ≤ b ⇒ \{x. a ≤ x ∧ x ≤ b ∧ poly p x = 0\} =
  \{x. a < x ∧ x ≤ b ∧ poly p x = 0\} ∪ (if poly p a = 0 then \{a\} else \{\})
  (is < → = ?R ∪ ?S) using not-less by force
  show a ≤ b ⇒ cr a b = (card \{x. a ≤ x ∧ x ≤ b ∧ poly p x = 0\})
  nr a = card \{x. x ≤ a ∧ poly p x = 0\} using cr unfolding arg-cong[OF id,
of card] card-Un-disjoint[OF RS]
  count-roots-interval-sf-def count-roots-between-correct[symmetric]
  count-roots-below-correct[symmetric] count-roots-below-def
  count-roots-between-def Let-def using p by auto
qed

lemma count-roots-interval: assumes cr: count-roots-interval p = (cr,nr)
and sf: square-free p
shows a ≤ b ⇒ cr a b = (card \{x. a ≤ x ∧ x ≤ b ∧ poly p x = 0\})
  nr a = card \{x. x ≤ a ∧ poly p x = 0\}
using count-roots-interval-sf[OF ∼ cr[unfolded count-roots-interval-iff[OF sf]]]
sf[unfolded square-free-def] by blast+

definition root-cond :: int poly × rat × rat ⇒ real ⇒ bool where
root-cond plr x = (case plr of (p,l,r) ⇒ of-rat l ≤ x ∧ x ≤ of-rat r ∧ ipoly p x = 0)

definition root-info-cond :: root-info ⇒ int poly ⇒ bool where
\[
\text{root-info-cond } ri \ p \equiv (\forall \ a. \ a \leq b \rightarrow \text{root-info.l-r } ri \ a \ b = \text{card } \{x. \ \text{root-cond } (p,a,b) \ x\})
\]
\[
\wedge (\forall \ a. \ \text{root-info.number-root } ri \ a = \text{card } \{x. \ x \leq \text{real-of-rat } a \land \text{ipoly } p \ x = 0\})
\]

**Lemma** root-info-condD: \(\text{root-info-cond } ri \ p \implies a \leq b \implies \text{root-info.l-r } ri \ a \ b = \text{card } \{x. \ \text{root-cond } (p,a,b) \ x\}\)

unfolding root-info-cond-def by auto

**Definition** count-roots-interval-sf-rat :: int poly \(\Rightarrow\) root-info where

\[
\text{count-roots-interval-sf-rat } p = (\text{let } pp = \text{real-of-int-poly } p; \ (cr,nr) = \text{count-roots-interval-sf } pp \in \text{Root-Info } (\lambda \ a b. \ cr (\text{of-rat } a) (\text{of-rat } b)) (\lambda \ a. \ nr (\text{of-rat } a)))
\]

**Definition** count-roots-interval-rat :: int poly \(\Rightarrow\) root-info where

\[
\text{count-roots-interval-rat } p = (\text{let } pp = \text{real-of-int-poly } p; \ (cr,nr) = \text{count-roots-interval } pp \in \text{Root-Info } (\lambda \ a b. \ cr (\text{of-rat } a) (\text{of-rat } b)) (\lambda \ a. \ nr (\text{of-rat } a)))
\]

**Definition** count-roots-rat :: int poly \(\Rightarrow\) nat where

\[
\text{count-roots-rat } p = (\text{count-roots } (\text{real-of-int-poly } p))
\]

**Lemma** count-roots-interval-sf-rat: assumes \(p: p \neq 0\) shows \(\text{root-info-cond } (\text{count-roots-interval-sf-rat } p)\)

proof –

let \(?p = \text{real-of-int-poly } p\)

let \(?r = \text{real-of-rat}\)

let \(?ri = \text{count-roots-interval-sf-rat } p\)

from \(p\) have \(?p \neq 0\) by auto

obtain \(cr \ nr\) where \(cr: \text{count-roots-interval-sf } ?p = (cr,nr)\) by force

have \(?ri = \text{Root-Info } (\lambda a b. \ cr (?r a) (?r b)) (\lambda a. \ nr (?r a))\)

unfolding count-roots-interval-sf-rat-def Let-def cr by auto

hence \(id: \text{root-info.l-r } ?ri = (\lambda a b. \ cr (?r a) (?r b)) \text{root-info.number-root } ?ri = (\lambda a. \ nr (?r a))\)

by auto

note \(cr = \text{count-roots-interval-sf}[OF } p \ cr\]

show \(?\text{thesis unfolding } \text{root-info-cond-def id}\)

proof (intro conjI impI allI)

fix \(a\)

show \(\text{nr } (?r a) = \text{card } \{x. \ x \leq (?r a) \land \text{ipoly } p \ x = 0\}\)

using cr(2)[of \(?r a\)] by simp

next

fix \(a b:: \text{rat}\)

assume \(ab: a \leq b\)

from \(ab\) have \(ab: ?r a \leq ?r b\) by (simp add: of-rat-less-eq)

from cr(1)[OF this] show \(cr (?r a) (?r b) = \text{card } (\text{Collect } \text{root-cond } (p, a, a, b))\)
lemma of-rat-of-int-poly:  
\text{map-poly of-rat (of-int-poly } p) = of-int-poly p 
by (subst map-poly-map-poly, auto simp: o-def)

lemma square-free-of-int-poly:  
\text{assumes square-free } p 
\text{shows square-free (of-int-poly } p :: 'a :: \{ \text{field-gcd, field-char-0} \} \text{ poly}) 
proof – 
\text{have square-free (map-poly of-rat (of-int-poly } p :: 'a poly) 
unfolding of-rat-hom.square-free-map-poly by (rule square-free-int-rat[OF assms]) 
thus \ ?thesis unfolding of-rat-of-int-poly .
qed

lemma count-roots-interval-rat:  
\text{assumes sf: square-free } p 
\text{shows root-info-cond (count-roots-interval-rat } p) p 
proof – 
\text{from sf have sf: square-free (real-of-int-poly } p) by (rule square-free-of-int-poly) 
\text{from sf have } p \neq 0 \text{ unfolding square-free-def by auto} 
\text{show \ ?thesis using count-roots-interval-sf-rat[OF sf] .}
qed

lemma count-roots-rat:  
\text{count-roots-rat } p = \text{card \{ } x. \text{ipoly } p x = (0 :: real)\} 
unfolding count-roots-rat-def count-roots-correct . .

6.2 Implementing Sturm on Rational Polynomials

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

lemma sturm-aux-rat:  
\text{sturm-aux (real-of-rat-poly } p) \text{ (real-of-rat-poly } q) = 
\text{map real-of-rat-poly (sturm-aux-rat } p \text{ q)}
proof (induct p q rule: sturm-aux-rat.induct) 
case (1 p q) 
interpret map-poly-inj-idom-hom of-rat.. 
note deg = af-int-hom.degree-map-poly-hom 
show \ ?case 
unfolding sturm-aux.simps[of real-of-rat-poly } p\] sturm-aux-rat.simps[of p]
using 1 by (cases degree q = 0; simp add: hom-distrib)
qed

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

lemma sturm-rat: sturm (real-of-rat-poly p) = map real-of-rat-poly (sturm-rat p)
  unfolding sturm-rat-def sturm-def
  apply (fold af-rat-hom.map-poly-pderiv)
  unfolding sturm-aux-rat..,

definition poly-number-rootat :: rat poly ⇒ rat where
  poly-number-rootat p ≡ sgn (coeff p (degree p))

definition poly-neg-number-rootat :: rat poly ⇒ rat where
  poly-neg-number-rootat p ≡ if even (degree p) then sgn (coeff p (degree p))
  else − sgn (coeff p (degree p))

lemma poly-number-rootat: poly-inf (real-of-rat-poly p) = real-of-rat (poly-number-rootat p)
  unfolding poly-inf-def poly-number-rootat-def of-int-hom.degree-map-poly-hom
  of-rat-hom.coeff-map-poly-hom
  real-of-rat-sgn by simp

lemma poly-neg-number-rootat: poly-neg-inf (real-of-rat-poly p) = real-of-rat (poly-number-rootat p)
  unfolding poly-neg-inf-def poly-number-rootat-def of-int-hom.degree-map-poly-hom
  of-rat-hom.coeff-map-poly-hom
  real-of-rat-sgn by (simp add: hom-distrib)

definition sign-changes-rat where
  sign-changes-rat ps (x::rat) =
    length (remdups-adj (filter (λx. x ≠ 0) (map (λp. sgn (poly p x)) ps))) − 1

definition sign-changes-number-rootat where
  sign-changes-number-rootat ps =
    length (remdups-adj (filter (λx. x ≠ 0) (map poly-number-rootat ps))) − 1

definition sign-changes-neg-number-rootat where
  sign-changes-neg-number-rootat ps =
    length (remdups-adj (filter (λx. x ≠ 0) (map poly-neg-number-rootat ps))) − 1

lemma real-of-rat-list-neq: list-neq (map real-of-rat xs) 0
  = map real-of-rat (list-neq xs 0)
  by (induct xs, auto)

lemma real-of-rat-remdups-adj: remdups-adj (map real-of-rat xs) = map real-of-rat
  (remdups-adj xs)
  by (induct xs rule: remdups-adj.induct, auto)
lemma sign-changes-rat: sign-changes (map real-of-rat-poly ps) (real-of-rat x) = sign-changes-rat ps x (is \( ?l = ?r \))
proof
  define xs where xs = list-neq (map (\( \lambda p. \ sgn (poly p x) \)) ps) 0
  have \( ?l = \) length (remdups-adj (list-neq (map real-of-rat (map (\( \lambda xa. \ (sgn (poly xa x)))) ps)) 0)) \( = \) 1
    by (simp add: sign-changes-def real-of-rat-sgn o-def)
  also have \( \ldots = ?r \) unfolding sign-changes-rat-def real-of-rat-list-neq
    unfolding real-of-rat-remdups-adj by simp
finally show \( \)thesis .
qed

lemma sign-changes-neg-number-rootat: sign-changes-neg-inf (map real-of-rat-poly ps) = sign-changes-neg-number-rootat ps (is \( ?l = ?r \))
proof
  have \( ?l = \) length (remdups-adj (list-neq (map real-of-rat (map poly-neg-number-rootat ps)) 0)) \( = \) 1
    unfolding sign-changes-neg-inf-def o-def real-of-rat-sgn poly-neg-number-rootat
  also have \( \ldots = ?r \) unfolding sign-changes-neg-number-rootat-def real-of-rat-list-neq
    unfolding real-of-rat-remdups-adj by simp
finally show \( \)thesis .
qed

lemma sign-changes-number-rootat: sign-changes-inf (map real-of-rat-poly ps) = sign-changes-number-rootat ps (is \( ?l = ?r \))
proof
  have \( ?l = \) length (remdups-adj (list-neq (map real-of-rat (map poly-number-rootat ps)) 0)) \( = \) 1
    unfolding sign-changes-inf-def
    unfolding map-map o-def real-of-rat-sgn poly-number-rootat ..
  also have \( \ldots = ?r \) unfolding sign-changes-number-rootat-def real-of-rat-list-neq
    unfolding real-of-rat-remdups-adj by simp
finally show \( \)thesis .
qed

lemma count-roots-interval-rat-code[code]:
count-roots-interval-rat p = (let rp = map-poly rat-of-int p; ps = sturm-rat rp
  in Root-Info
    (\( \lambda a b. \ \) sign-changes-rat ps a - sign-changes-rat ps b + (if poly rp a = 0 then
      1 else 0))
    (\( \lambda a. \ \) sign-changes-neg-number-rootat ps - sign-changes-rat ps a))
  unfolding count-roots-interval-rat-def Let-def count-roots-interval-def split of-rat-of-int-poly[symmetric,
where \( 'a = \) real]
  sturm-rat sign-changes-rat
  by (simp add: sign-changes-neg-number-rootat)
lemma count-roots-rat-code[code]:
\[\text{count-roots-rat } p = (\text{let } rp = \text{map-poly rat-of-int } p \text{ in if } p = 0 \text{ then 0 else let } ps = \text{sturm-rat } rp \text{ in if } p = 0 \text{ then 0 else let } ps = \text{sturm-rat } rp \text{ in sign-changes-neg-number-rootat } ps - \text{sign-changes-number-rootat } ps)\]

unfolding count-roots-rat-def Let-def sturm-rat count-roots-code of-rat-of-int-poly[symmetric, where \( a = \text{real} \)]

sign-changes-neg-number-rootat sign-changes-number-rootat
by simp

hide-const (open) count-roots-interval-sf-rat

Finally we provide an even more efficient implementation which avoids the "poly \( p \times = 0 \)" test, but it is restricted to irreducible polynomials.

definition root-info :: int poly ⇒ root-info where
\[\text{root-info } p = (\text{if } \text{degree } p = 1 \text{ then } (\text{let } x = \text{Rat.Fract} (- \text{coeff } p 0) (\text{coeff } p 1) \text{ in Root-Info } (\lambda l r. \text{if } l \leq x \land x \leq r \text{ then 1 else 0}) (\lambda b. \text{if } x \leq b \text{ then 1 else 0})) \text{ else } (\text{let } rp = \text{map-poly rat-of-int } p \text{; } ps = \text{sturm-rat } rp \text{ in Root-Info } (\lambda a b. \text{sign-changes-rat } ps a - \text{sign-changes-rat } ps b) (\lambda a. \text{sign-changes-neg-number-rootat } ps - \text{sign-changes-rat } ps a)))\]

lemma root-info:
assumes irr: irreducible \( p \) and deg: degree \( p > 0 \)
shows root-info-cond (root-info \( p \)) \( p \)
proof (cases degree \( p \) = 1)
  case True
  from degree1-coeffs[of this] obtain a b where \( p = [b,a] \) and \( a \neq 0 \) by auto
  from deg have degree (real-of-int-poly \( p \)) = 1 by simp
  from roots1[of this, unfolded roots1-def] \( p \)
  have id: (opol \( p \times = 0 \)) = ((\( \times \times \). real) = - b / a) for \( \times \) by auto
  have idd: \( \{ x. real-of-rat aa \leq x \land x \leq real-of-rat ba \land x = real-of-int (- b) / real-of-int a \} = (if real-of-rat aa \leq real-of-int (- b) / real-of-int a \land real-of-int (- b) / real-of-int a \leq real-of-int ba then \{ real-of-int (- b) / real-of-int a \} else {}) \}
  for aa ba by auto
  have iddd: \( \{ x. x \leq real-of-rat aa \land x = real-of-int (- b) / real-of-int a \} = (if real-of-int (- b) / real-of-int a \leq real-of-rat aa then \{ real-of-int (- b) / real-of-int a \} else {}) \}
  for aa by auto
  have id4: real-of-int \( x \times = real-of-rat (rat-of-int \( x \)) \) for \( \times \) by simp
    unfolding p Fract-of-int-quotient Let-def idd iddd
    unfolding idd4 of-rat-divide symmetric of-rat-less-eq by auto
next
  case False
  have irr-d: irreducible \( d \) \( p \) by (simp add: deg irr irreducible-connect-rev)
from irreducible_d-int-rat [OF this]
have irreducible (of-int-poly p :: rat poly) by auto
from irreducible-root-free [OF this]
have idd: (poly (of-int-poly p) a = 0) = False for a :: rat
  unfolding root-free-def using False by auto
have id: root-info p = count-roots-interval-rat p
  unfolding root-info-def if-False count-roots-interval-rat-code Let-def idd using False by auto
show ?thesis unfolding id by (rule count-roots-interval-rat [OF irreducible_d-square-free [OF irr-d]])
qed

7 Getting Small Representative Polynomials via Factorization

In this theory we import a factorization algorithm for integer polynomials to turn a representing polynomial of some algebraic number into a list of irreducible polynomials where exactly one list element represents the same number. Moreover, we prove that the certain polynomial operations preserve irreducibility, so that no factorization is required.

theory Factors-of-Int-Poly
  imports
    Berlekamp-Zassenhaus.Factorize-Int-Poly
    Algebraic-Numbers-Prelim
begin

lemma degree-of-gcd: degree (gcd q r) ≠ 0 ←→
degree (gcd (of-int-poly q :: 'a :: {field-char-0, field-gcd} poly) (of-int-poly r)) ≠ 0
proof –
  let ?r = of-rat :: rat ⇒ 'a
  interpret rpoly: field-hom' ?r
  by (unfold-locales, auto simp: of-rat-add of-rat-mult)
{  
  fix p
  have of-int-poly p = map-poly (?r o of-int) p unfolding o-def
  by auto
  also have . . . = map-poly ?r (map-poly of-int p)
  by (subst map-poly-map-poly, auto)
  finally have of-int-poly p = map-poly ?r (map-poly of-int p).
} note id = this
show ?thesis unfolding id by (fold hom-distrib, simp add: gcd-rat-to-gcd-int)
qed

definition factors-of-int-poly :: int poly ⇒ int poly list where
factors-of-int-poly p = map (abs-int-poly o fst) (snd (factorize-int-poly p))
lemma factors-of-int-poly-const: assumes degree p = 0
shows factors-of-int-poly p = []
proof –
  from degree0-coeffs[OF assms] obtain a where: p: p = [a : ] by auto
  show ?thesis unfolding p factors-of-int-poly_def
  factorize-int-poly-generic-def x-split-def
  by (cases a = 0, auto simp add: Let_def factorize-int-last-nz-poly-def)
qed

lemma factors-of-int-poly:
defines rp ≡ ipoly :: int poly ⇒ 'a :: {field-gcd, field-char-0} ⇒ 'a
assumes factors-of-int-poly p = qs
shows ∨ q. q ∈ set qs ⇒ irreducible q ∧ lead-coeff q > 0 ∧ degree q ≤ degree
p ∧ degree q ≠ 0
p ≠ 0 ⇒ rp p x = 0 (⇒ (∃ q ∈ set qs. rp q x = 0))
p ≠ 0 ⇒ rp p x = 0 ⇒ ∃! q ∈ set qs. rp q x = 0
distinct qs
proof –
  obtain c qis where facttl: factorize-int-poly p = (c, qis) by force
  from assms[unfolded factors-of-int-poly-def facttl]
  have qs: qs = map (abs-int-poly o fst) (snd (c, qis)) by auto
  note fact = factorize-int-poly(1)(OF facttl)
  note fact-mem = factorize-int-poly(2,3)(OF facttl)
  have sff: square-free-factorization p (c, qis) by (rule fact(1))
  note sff = square-free-factorizationD[OF sff]
  have sff': p = Polynomial.smult c (∏ (a, i)← qis. a ^ Suc i)
    unfolding sff(1) prod.distinct-set-conv-list[OF sff(5)] ..
    { fix q
      assume q: q ∈ set qs
      then obtain r i where qi: (r, i) ∈ set qis and qr: q = abs-int-poly r unfolding qis by auto
      from split-list[OF qr] obtain qis1 qis2 where qis: qis = qis1 @ (r, i) # qis2
        by auto
      have dvd: r dvd p unfolding sff' qis dvd-def
        by (intro exI[OF - smult c (r ^ i * (∏ (a, i)← qis1 @ qis2. a ^ Suc i))], auto)
      from fact-mem[OF q] have r0: r ≠ 0 by auto
      from qi factlt have p: p ≠ 0 by (cases p, auto)
      with dvd have deg: degree r ≤ degree p by (metis dvd_imp_degree_le)
      with fact-mem[OF q] r0
      show irreducible q ∧ lead-coeff q > 0 ∧ degree q ≤ degree p ∧ degree q ≠ 0
      unfolding qr lead-coeff-abs-int-poly by auto
    } note * = this
  show distinct qs unfolding distinct-conv-nth
  proof (intro allIImpl)
    fix i j
    assume i < length qs j < length qs and diff: i ≠ j
    hence ij: i < length qs j < length qs

97
and id: qs ! i = abs-int-poly (fst (qs ! i)) qs ! j = abs-int-poly (fst (qs ! j))

unfolding qs by auto
obtain qi I where qi: qs ! i = (qi, I) by force
obtain qj J where qj: qs ! j = (qj, J) by force
from sff(5)[unfolded distinct-cone-nth, rule-format, OF ij diff] qi qj
have diff: (qi, I) ≠ (qj, J) by auto
from qi qj qi qj have (qi, I) ∈ set qs (qj, J) ∈ set qs unfolding set-cone-nth by
force+
from sff(3)[OF this diff] sff(2) this
have cop: coprime qi qj degree qi ≠ 0 degree qj ≠ 0 by auto
note i = cf-pos-poly-main[of qi, unfolded smult-prod monom-0]
note j = cf-pos-poly-main[of qj, unfolded smult-prod monom-0]
from cop(2) i have deg: degree (qs ! i) ≠ 0 by (auto simp: id qi)
have cop: coprime (qs ! i) (qs ! j)
unfolding id qj qj fat-conv
apply (rule coprime-prod[of [:sgn (lead-coeff qi):] [:sgn (lead-coeff qj):]])
using cop
unfolding i j by (auto simp: sgn-eq-0-iff)
show qs ! i ≠ qs ! j
proof
assume id: qs ! i = qs ! j
have deg: (gcd (qs ! i) (qs ! j)) = deg (qs ! i) unfolding id by simp
also have ... ≠ 0 using deg by simp
finally show False using cop by simp
qed

qed

assume p: p ≠ 0
from fact(1) p have c: c ≠ 0 using sff(1) by auto
let ∀r = of-int :: int ⇒ 'a
let ∀rp = map-poly ∀r
have rp: A x. rp p x = 0 i→→ poly (?rp p) x = 0 unfolding rp-def ...
have rp p x = 0 i→→ rp (prod x y. x ^ Suc y) x = 0 unfolding sff(1)
unfolding rp hom-distribs using c by simp
also have ... = (∃ q,i) ∈ set qs. poly (?rp (q ^ Suc i)) x = 0
unfolding qs rp of-int-poly-hom.hom-prod-list poly-prod-list-zero-iff set-map by
fastforce
also have ... = (∃ q,i) ∈ set qs. poly (?rp q) x = 0
unfolding of-int-poly-hom.hom-power poly-power-zero-iff by auto
also have ... = (∃ q ∈ set qs. rp q x = 0) unfolding rp qs snd-conv o-def
bex-simps set-map by simp
finally show iff: rp p x = 0 i→→ (∃ q ∈ set qs. rp q x = 0) by auto
assume rp p x = 0
with iff obtain q where q: q ∈ set qs and rtq: rp q x = 0 by auto
then obtain i q' where qi: (q', i) ∈ set qis and qq': q = abs-int-poly q' unfolding
qs by auto
show ∃! q ∈ set qs. rp q x = 0
proof (intro ex1I, intro conj1, rule q, rule rtq, clarify)
fix r
assume r ∈ set qs and rt′: rp r x = 0
then obtain j r′ where rj: (r′, j) ∈ set qs is and rr′: r = abs-int-poly r′
unfolding qs by auto
from rt′ rj have rt′: rp r′ x = 0 and rtq: rp q′ x = 0
unfolding rp rr′ qq′ by auto
from rt′ rj have [−x, 1:] dvd ?rp q′ [−x, 1:] dvd ?rp r′ unfolding rp
by (auto simp: poly-eq-0-iff-dvd)
hence [−x, 1:] dvd gcd (?rp q′) (?rp r′) by simp
hence gcd (?rp q′) (?rp r′) = 0 ∨ degree (gcd (?rp q′) (?rp r′)) ≠ 0
by (metis is-unit-gcd-iff is-unit-iff-degree is-unit-pCons-iff one-poly-eq-simps(1))
hence gcd q′ r′ = 0 ∨ degree (gcd q′ r′) ≠ 0
unfolding gcd-eq-0-iff degree-of-gcd[of q′ r′, symmetric] by auto
hence ¬ coprime q′ r′ by auto
with sff(3)[OF qi rj] have q′ = r′ by auto
thus r = q unfolding rr′ qq′ by simp
qed
qed

lemma factors-int-poly-represents:
fixes x :: 'a :: {field-char-0, field-gcd}
assumes p: p represents x
shows ∃ q ∈ set (factors-of-int-poly p).
  q represents x ∧ irreducible q ∧ lead-coeff q > 0 ∧ degree q ≤ degree p
proof —
from representsD[OF p] have p: p ≠ 0 and rt: ipoly p x = 0 by auto
note fact = factors-of-int-poly[OF refl]
from fact(2)[OF p, of x] rt obtain q where q: q ∈ set (factors-of-int-poly p)
and
  rt: ipoly q x = 0 by auto
from fact(1)[OF q] rt show ?thesis
  by (intro bexI[OF - q], auto simp: represents-def irreducible-def)
qed

corollary irreducible-represents-imp-degree:
fixes x :: 'a :: {field-char-0, field-gcd}
assumes irreducible f and f represents x and g represents x
shows degree f ≤ degree g
proof —
from factors-of-int-poly(1)[OF refl, of - g] factors-of-int-poly(3)[OF refl, of g x]
  assms(3) obtain h where h: h represents x degree h ≤ degree g irreducible h
by blast
let ?af = abs-int-poly f
let ?ah = abs-int-poly h
from assms have af: irreducible ?af ?af represents x lead-coeff ?af > 0 by fastforce+
from * have ah: irreducible ?ah ?ah represents x lead-coeff ?ah > 0 by fastforce+
from algebraic-imp-represents-unique[of x] af ah have id: ?af = ?ah
unfolding algebraic-iff-represents by blast

99
show thesis using arg-cong[OF id, of degree] (degree h \leq degree g) by simp
qed

lemma irreducible-preservation:
fixes x :: 'a :: {field-char-0,field-gcd}
assumes irr: irreducible p
and x: p represents x
and y: q represents y
and deg: degree p \geq degree q
and f: \\forall q. q represents y \implies (f q) represents x \wedge degree (f q) \leq degree q
and pr: primitive q
shows irreducible q
proof (rule ccontr)
define pp where pp = abs-int-poly p
have dp: degree p \neq 0 using x by (rule represents-degree)
have dq: degree q \neq 0 using y by (rule represents-degree)
from dp have p0: p \neq 0 by auto
from x deg irr p0
have irr: irreducible pp and x: pp represents x and
deg: degree pp \geq degree q and cf-pos: lead-coeff pp > 0

unfolding pp-def lead-coeff-abs-int-poly by (auto intro: representsI)
from x have ax: algebraic x unfolding algebraic-altdef-ipoly represents-def by blast
assume \neg thesis
from this irreducible-connect-int[of q] pr have \neg irreducible q by auto
from this dq obtain r where
  r: degree r \neq 0 degree r < degree q and r dvd q by auto
then obtain rr where q: q = r \ast rr unfolding dvd-def by auto
have degree q = degree r + degree rr using dq unfolding q
  by (subst degree-mult-eq, auto)
with r have rr: degree rr \neq 0 degree rr < degree q by auto
from representsD(2)(OF y, unfolded q hom-distrib]
have ipoly r y = 0 \lor ipoly rr y = 0 by auto
with r rr have r represents y \lor rr represents y unfolding represents-def by auto
with r rr obtain r where r: r represents y degree r < degree q by blast
from f[OF r(1)] deg r(2) obtain r where r: r represents x degree r < degree pp
  by auto
  from factors-int-poly-represents[OF r(1)] r(2) obtain r where
    r: r represents x irreducible r lead-coeff r > 0 and deg: degree r < degree pp
    by force
  from algebraic-imp-represents-unique[OF ax] r irr cf-pos x have r = pp by auto
with deg show False by auto
qed

declare irreducible-const-poly-iff [simp]

lemma poly-uminus-irreducible:
assumes p: irreducible (p :: int poly) and deg: degree p \neq 0
shows irreducible (poly-uminus p)
proof-
from deg-nonzero-represents[OF deg] obtain x :: complex where x: p represents x by auto
from represents-uminus[OF x]
have y: poly-uminus p represents (- x).
show ?thesis
proof (rule irreducible-preservation[OF p x y], force)
from deg irreducible-imp-primitive[OF p] have primitive p by auto
then show primitive (poly-uminus p) by simp
fix q
assume q represents (- x)
from represents-uminus[OF this] have (poly-uminus q) represents x by simp
thus (poly-uminus q) represents x ∧ degree (poly-uminus q) ≤ degree q by auto
qed
qed

lemma reflect-poly-irreducible:
fixes x :: 'a :: {field-char-0,field-gcd}
assumes p: irreducible p and x: p represents x and x0: x ≠ 0
shows irreducible (reflect-poly p)
proof –
from represents-inverse[OF x0 x]
have y: (reflect-poly p) represents (inverse x) by simp
from x0 have ix0: inverse x ≠ 0 by auto
show ?thesis
proof (rule irreducible-preservation[OF p x y])
from x irreducible-imp-primitive[OF p]
show primitive (reflect-poly p) by (auto simp: content-reflect-poly)
fix q
assume q represents (inverse x)
from represents-inverse[OF ix0 this] have (reflect-poly q) represents x by simp
with degree-reflect-poly-le
show (reflect-poly q) represents x ∧ degree (reflect-poly q) ≤ degree q by auto
qed (insert p, auto simp: degree-reflect-poly-le)
qed

lemma poly-add-rat-irreducible:
assumes p: irreducible p and deg: degree p ≠ 0
shows irreducible (cf-pos-poly (poly-add-rat r p))
proof –
from deg-nonzero-represents[OF deg] obtain x :: complex where x: p represents x by auto
from represents-add-rat[OF x]
have y: cf-pos-poly (poly-add-rat r p) represents (of-rat r + x) by simp
show ?thesis
proof (rule irreducible-preservation[OF p x y], force)
fix q
assume q represents (of-rat r + x)
from represents-add-rat[OF this, of \(- r\)] have (poly-add-rat \(- r\) q) represents x by (simp add: of-rat-minus)
thus (poly-add-rat \(- r\) q) represents x ∧ degree (poly-add-rat \(- r\) q) ≤ degree q by auto
qed (insert p, auto)
qed

lemma poly-mult-rat-irreducible:
  assumes p: irreducible p and deg: degree p ≠ 0 and r: r ≠ 0
  shows irreducible (cf-pos-poly (poly-mult-rat r p))
proof –
  from deg-nonzero-represents[OF deg] obtain x :: complex where x: p represents x by auto
  from represents-mult-rat[OF r x] have y: cf-pos-poly (poly-mult-rat r p) represents (of-rat r ∗ x) by simp
  show ?thesis
  proof (rule irreducible-preservation[OF p x y], force simp: r)
    fix q
    from r have r’: inverse r ≠ 0 by simp
    assume q represents (of-rat r ∗ x)
    from represents-mult-rat[OF r’ this] have (poly-mult-rat (inverse r) q) represents x using r
      by (simp add: of-rat-divide field-simps)
    thus (poly-mult-rat (inverse r) q) represents x ∧ degree (poly-mult-rat (inverse r) q) ≤ degree q
      using r by auto
  qed (insert p r, auto)
qed

interpretation coeff-lift-hom:
  factor-preserving-hom coeff-lift :: 'a :: {comm-semiring-1,semiring-no-zero-divisors} ⇒ -
  by (unfold-locales, auto)

end

8 Real Algebraic Numbers

Whereas we previously only proved the closure properties of algebraic numbers, this theory adds the numeric computations that are required to separate the roots, and to pick unique representatives of algebraic numbers.

The development is split into three major parts. First, an ambiguous representation of algebraic numbers is used, afterwards another layer is used with special treatment of rational numbers which still does not admit unique representatives, and finally, a quotient type is created modulo the equivalence.
The theory also contains a code-setup to implement real numbers via real algebraic numbers.

The results are taken from the textbook [2, pages 329ff].

```
theory Real-Algebraic-Numbers
imports
  Abstract-Rewriting.SN-Order-Carrier
  Deriving.Compare-Rat
  Deriving.Compare-Real
  Jordan-Normal-Form.Gauss-Jordan-IArray-Impl
  Algebraic-Numbers
  Sturm-Rat
  Factors-of-Int-Poly
begin

For algebraic numbers, it turned out that \texttt{gcd-int-poly} is not preferable to the default implementation of \texttt{gcd}, which just implements Collin’s primitive remainder sequence.

\texttt{declare gcd-int-poly-code[code-unfold del]}

\texttt{lemma ex1-imp-Collect-singleton: (\exists! x. \textit{P} x) \land P x \longleftrightarrow \textit{Collect} P = \{x\}}
\texttt{proof (intro iffI conjI, unfold conj-imp-eq-imp-imp)}
\texttt{  assume Ex1 P P x then show \textit{Collect} P = \{x\} by blast}
\texttt{next}
\texttt{  assume Px: \textit{Collect} P = \{x\}}
\texttt{  then have \textit{P} y \longleftrightarrow x = y for y by auto}
\texttt{  then show Ex1 P by auto}
\texttt{from Px show \textit{P} x by auto}
\texttt{qed}

\texttt{lemma ex1-Collect-singleton[consumes 2]:}
\texttt{  assumes \exists! x. \textit{P} x and \textit{P} x and \textit{Collect} P = \{x\} \Longrightarrow thesis shows thesis}
\texttt{by (rule assms(3), subst ex1-imp-Collect-singleton[symmetric], insert assms(1,2), auto)}

\texttt{lemma ex1-iff-Collect-singleton: P x \Longrightarrow (\exists! x. \textit{P} x) \longleftrightarrow \textit{Collect} P = \{x\}}
\texttt{by (subst ex1-imp-Collect-singleton[symmetric], auto)}

context
  fixes \textit{f}
  assumes \textit{bij}: \textit{bij} \textit{f}
begin

\texttt{lemma bij-imp-ex1-iff: (\exists! x. \textit{P} (f x)) \longleftrightarrow (\exists! y. \textit{P} y) (is \ ?l = \ ?r)}
\texttt{proof (intro iffI)}
\texttt{  assume l: \ ?l}
\texttt{  then obtain x where \textit{P} (f x) by auto}
\texttt{  with l have \*: \{x\} = \textit{Collect} (P o f) by auto}
\texttt{also have f ‘… = \{y. \textit{P} (f (Hilbert-Choice.inv f y))\}} using \texttt{bij-image-Collect-eq[OF}
```

103
bij] by auto
also have . . . = {y. P y}
proof-
  have f (Hilbert-Choice.inv f y) = y for y by (meson bij bij-inv-eq-iff)
  then show ?thesis by simp
qed
finally have Collect P = {x. P x} by auto
then show ?r by (fold ex1-imp-Collect-singleton, auto)
next
assume r: ?r
then obtain y where P y by auto
with r have {y} = Collect P by auto
also have Hilbert-Choice.inv f . . . = Collect (P o f)
  using bij-image-Collect-eq[OF bij-imp-bij-inv[OF bij]] bij by (auto simp: inv-inv-eq)
finally have Collect (P o f) = {Hilbert-Choice.inv f y} by (simp add: o-def)
then show ?l by (fold ex1-imp-Collect-singleton, auto)
qed

lemma bij-ex1-imp-the-shift:
  assumes ex1: \exists! y. P y shows (THE x. P (f x)) = Hilbert-Choice.inv f (THE y. P y) (is \(?l = ?r\))
proof-
  from ex1 have P (THE y. P y) by (rule the1I2)
  moreover from ex1[folded bij-imp-ex1-iff] have P (f (THE x. P (f x))) by (rule the1I2)
  ultimately have (THE y. P y) = f (THE x. P (f x)) using ex1 by auto
  also have Hilbert-Choice.inv f . . . = (THE x. P (f x)) using bij by (simp add: bij-is-inj)
  finally show ?l = ?r by auto
qed

lemma bij-imp-Collect-image: {x. P (f x)} = Hilbert-Choice.inv f \{y. P y\} (is \(?l = \equiv \{y. P y\})
proof-
  have \(?l = \equiv \{y. P y\} \equiv \{x. P x\}\) by (simp add: image-comp inv-o-cancel[OF bij-is-inj[OF bij]])
  also have f \equiv \{x. P x\}\) by auto
  also have . . . = \{y. P y\} by (metis bij bij-iff)
  finally show ?thesis.
qed

lemma bij-imp-card-image: card (f \equiv \{x. P x\} = card \{x. P x\}
  by (metis bij bij-iff card.card.finite_imageD inj_onI inj_on_iff_eq_card)
end

lemma bij-imp-card: assumes bij: bij f shows card {x. P (f x)} = card {x. P x}
lemma bij-add: bij (λx. x + y :: 'a :: group-add) (is ?g1)
  and bij-minus: bij (λx. x - y :: 'a) (is ?g2)
  and inv-add[simp]: Hilbert-Choice.inv (λx. x + y) = (λx. x - y) (is ?g3)
  and inv-minus[simp]: Hilbert-Choice.inv (λx. x - y) = (λx. x + y) (is ?g4)
proof
  have 1: (λx. x - y) o (λx. x + y) = id and 2: (λx. x + y) o (λx. x - y) = id
  by auto
  from o-bij[OF 1 2] show ?g1.
  from o-bij[OF 2 1] show ?g2.
  from inv-unique-comp[OF 2 1] show ?g3.
  from inv-unique-comp[OF 1 2] show ?g4.
qed


lemma ex1-the-shift:
  assumes ex1: ∃y :: 'a :: group-add. P y
  shows (THE x. P (x + d)) = (THE y. P y) - d
  and (THE x. P (x - d)) = (THE y. P y) + d

lemma card-shift-image[simp]:
  shows card ((λx :: 'a :: group-add. x + d) ' X) = card X
  and card ((λx :: 'a :: group-add. x - d) ' X) = card X
  by (auto simp: bij-imp-card-image[OF bij-add] bij-imp-card-image[OF bij-minus])

lemma irreducible-root-free:
  fixes p :: 'a :: {idom, comm-ring-1} poly
  assumes irr: irreducible p shows root-free p
proof (cases degree p 1::nat rule: linorder-cases)
  case greater
  { fix x
    assume poly p x = 0
    hence [:-x,1:] dvd p using poly-eq-0-iff-dvd by blast
    then obtain r where p: p = r * [:-x,1:] by (elim dvdE, auto)
    have deg: degree [:-x,1:] = 1 by simp
    have dvd: ¬ [:-x,1:] dvd 1 by (auto simp: poly-dvd-1)
    from greater have degree r ≠ 0 using degree-mult-le[of r [:-x,1:], unfolded
    deg, folded p] by auto
    then have ¬ r dvd 1 by (auto simp: poly-dvd-1)
    with p irr irreducibleD[OF irr p] dvd False by auto
  }
  thus ?thesis unfolding root-free-def by auto
next
  case less then have deg: degree p = 0 by auto
from deg obtain p0 where p: p = [:p0:] using degree0-coeffs by auto
with irr have p \neq 0 by auto
with p have poly p x \neq 0 for x by auto
thus \{thesis \} by (auto simp: root-free-def)
qed (auto simp: root-free-def)

8.1 Real Algebraic Numbers – Innermost Layer

We represent a real algebraic number \( \alpha \) by a tuple \((p,l,r)\): \( \alpha \) is the unique root in the interval \([l,r]\) and \(l\) and \(r\) have the same sign. We always assume that \(p\) is normalized, i.e., \(p\) is the unique irreducible and positive content-free polynomial which represents the algebraic number.

This representation clearly admits duplicate representations for the same number, e.g. \((...,x-3,3,3)\) is equivalent to \((...,x-3,2,10)\).

8.1.1 Basic Definitions

\textbf{type-synonym} real-alg-1 = int poly \times rat \times rat

\textbf{fun} poly-real-alg-1 :: real-alg-1 \Rightarrow int poly where poly-real-alg-1 \((p,\cdot,\cdot)\) = p

\textbf{fun} rai-ub :: real-alg-1 \Rightarrow rat where rai-ub \((\cdot,\cdot,r)\) = r

\textbf{fun} rai-lb :: real-alg-1 \Rightarrow rat where rai-lb \((\cdot,l,\cdot)\) = l

\textbf{abbreviation} roots-below p x \equiv \{ y :: real. y \leq x \land ipoly p y = 0 \}

\textbf{abbreviation} (input) unique-root :: real-alg-1 \Rightarrow bool where
unique-root plr \equiv (\exists! x. root-cond plr x)

\textbf{abbreviation} the-unique-root :: real-alg-1 \Rightarrow real where
the-unique-root plr \equiv (THE x. root-cond plr x)

\textbf{abbreviation} real-of-1 where real-of-1 \equiv the-unique-root

\textbf{lemma} root-condI[intro]:
assumes of-rat (rai-lb plr) \leq x and x \leq of-rat (rai-ub plr) and ipoly (poly-real-alg-1 plr) x = 0
shows root-cond plr x
using assms by (auto simp: root-cond-def)

\textbf{lemma} root-condE[elim]:
assumes root-cond plr x
and of-rat (rai-lb plr) \leq x \Longrightarrow x \leq of-rat (rai-ub plr) \Longrightarrow ipoly (poly-real-alg-1 plr) x = 0 \Longrightarrow thesis
shows thesis
using assms by (auto simp: root-cond-def)

\textbf{lemma}
assumes ur: unique-root plr
defines \( x \equiv \text{the-unique-root plr} \) and \( p \equiv \text{poly-real-alg-1 plr} \) and \( l \equiv \text{rai-lb plr} \) and \( r \equiv \text{rai-ub plr} \)

shows unique-rootD: \( \text{of-rat } l \leq x \leq \text{of-rat } r \implies \text{ipoly } p \ x = 0 \implies \text{root-cond plr } x \)

and the-unique-root-eqI: \( \text{root-cond plr } y \implies y = x \implies \text{root-cond plr } y \implies x = y \)

proof –
from ur show \( x \): root-cond plr \( x \)
unfolding \( x \)-def by (rule theI’)
have \( \text{plr} = (p, l, r) \) by (cases plr, auto simp: \( p \)-def \( l \)-def \( r \)-def)
from \( x[\text{unfolded this}] \) show \( \text{of-rat } l \leq x \leq \text{of-rat } r \implies \text{ipoly } p \ x = 0 \) by auto
from \( x \) ur
show root-cond plr \( y \implies y = x \) and root-cond plr \( y \implies x = y \)
and \( x = y \Leftarrow \text{root-cond plr } y \) and \( y = x \Leftarrow \text{root-cond plr } y \) by auto
qed

lemma unique-rootE:
assumes \( \text{ur} \): unique-root plr
defines \( x \equiv \text{the-unique-root plr} \) and \( p \equiv \text{poly-real-alg-1 plr} \) and \( l \equiv \text{rai-lb plr} \) and \( r \equiv \text{rai-ub plr} \)
assumes main: \( \text{of-rat } l \leq x \implies x \leq \text{of-rat } r \implies \text{ipoly } p \ x = 0 \implies \text{root-cond plr } x \implies \)
\( (\forall y. \ x = y \Leftarrow \text{root-cond plr } y) \implies (\forall y. \ y = x \Leftarrow \text{root-cond plr } y) \implies \)
thesis
shows \( \text{thesis} \) by (rule main, unfold \( x \)-def \( p \)-def \( l \)-def \( r \)-def; rule unique-rootD[OF \( \text{ur} \)])

lemma unique-rootI:
assumes \( \forall y. \ \text{root-cond plr } y \implies y = x \) root-cond plr \( x \)
shows unique-root plr using assms by blast

definition poly-cond :: int poly \Rightarrow bool where
poly-cond \( p \) = (lead-coeff \( p \) > 0 \land \text{irreducible } p)

lemma poly-condI[intro]:
assumes lead-coeff \( p \) > 0 and irreducible \( p \) shows poly-cond \( p \) using assms by
(auto simp: poly-cond-def)

lemma poly-condD:
assumes poly-cond \( p \)
shows irreducible \( p \) and lead-coeff \( p \) > 0 and root-free \( p \) and square-free \( p \) and \( p \neq 0 \)
using assms unfolding poly-cond-def using irreducible-root-free irreducible-imp-square-free
cf-pos-def by auto

lemma poly-condE[elim]:
assumes poly-cond \( p \)
and irreducible \( p \) \(\Rightarrow\) lead-coeff \( p \) > 0 \(\Rightarrow\) root-free \( p \) \(\Rightarrow\) square-free \( p \) \(\Rightarrow\)
\( p \neq 0 \) \(\Rightarrow\) thesis
shows thesis
using assms by (auto dest:poly-condD)
definition invariant-1 :: real-alg-1 ⇒ bool where
  invariant-1 tup ≡ case tup of (p,l,r) ⇒
    unique-root (p,l,r) ∧ sgn l = sgn r ∧ poly-cond p

lemma invariant-1I:
  assumes unique-root plr and sgn (rai-lb plr) = sgn (rai-ub plr) and poly-cond (poly-real-alg-1 plr)
  shows invariant-1 plr
  using assms by (auto simp: invariant-1-def)

lemma
  assumes invariant-1 plr defines x ≡ the-unique-root plr and p ≡ poly-real-alg-1 plr and l ≡ rai-lb plr
  and r ≡ rai-ub plr
  shows invariant-1-root-cond: ∀ y. root-cond plr y ←→ y = x
  proof –
    let ᵃl = of-rat l :: real
    let ᵃr = of-rat r :: real
    have plr: plr = (p,l,r) by (cases plr, auto simp: p-def l-def r-def)
    from assms
    show ur: unique-root plr and sgn: sgn l = sgn r and pc: poly-cond p by (auto simp: invariant-1-def)
    from ur show rc: root-cond plr x by (auto simp add: x-def plr intro: theI′)
    from this[unfolded plr] have x: ipoly p x = 0 and bnd: ᵃl ≤ x ≤ ᵃr by auto
    show sgn x = of-rat (sgn r)
    proof (cases 0::real x rule:linorder-cases)
      case less
      with bnd[2] have 0 < ᵃr by arith
      thus ?thesis using less by simp
    next
      case equal
      with bnd have ᵃl ≤ 0 ᵃr ≥ 0 by auto
      hence l ≤ 0 r ≥ 0 by auto
      with ⟨sgn l = sgn r⟩ have l = 0 r = 0 unfolding sgn-rat-def by (auto split: if-splits)
      with rc[unfolded plr]
      show ?thesis by auto
    next
      case greater
      with bnd[1] have ᵃl < 0 by arith
      thus ?thesis unfolding ⟨sgn l = sgn r⟩[symmetric] using greater by simp
    qed
    from the-unique-root-eqI[OF ur] rc
    show ∀ y. root-cond plr y ←→ y = x by metis
{ assume \( \deg p = 0 \) with poly-zero[OF x, simplified] sgn bnd have \( p = 0 \) by auto with pc have False by auto }
then show \( \deg p > 0 \) by auto with pc show primitive \( p \) by (intro irreducible-imp-primitive, auto)

defined by

lemma invariant-1E[elim]:
assumes invariant-1 plr
defines \( x \equiv \text{the-unique-root} \ plr \) and \( p \equiv \text{poly-real-alg-1} \ plr \) and \( l \equiv \text{rai-lb} \ plr \)
and \( r \equiv \text{rai-ub} \ plr \)
assumes main: root-cond \( x \) and sgn \( l = \text{sgn} \ r \) \( \Rightarrow \) unique-root \( plr \) \( \Rightarrow \) poly-cond \( p \)
\( \Rightarrow \) degree \( p > 0 \) \( \Rightarrow \)
primitive \( p \) \( \Rightarrow \) thesis
shows thesis apply (rule main)
using assms(1) unfolding x-def p-def l-def r-def by (auto dest: invariant-1D)

lemma invariant-1-realI:
fixes plr :: real-alg-1
defines \( p \equiv \text{poly-real-alg-1} \ plr \) and \( l \equiv \text{rai-lb} \ plr \) and \( r \equiv \text{rai-ub} \ plr \)
assumes x: root-cond \( plr \) \( x \) and sgn \( l = \text{sgn} \ r \)
and ur: unique-root \( plr \)
and poly-cond \( p \)
shows invariant-1 \( plr \) \( \land \) real-of-1 \( plr = x \)
using the-unique-root-eqI[OF ur x] assms by (cases plr, auto intro: invariant-1I)

lemma real-of-1-0:
assumes invariant-1 \( (p,l,r) \)
shows \( \text{simp}: \text{the-unique-root} \ (p,l,r) = 0 \iff r = 0 \)
and \( \text{dest}: l = 0 \Rightarrow r = 0 \)
and \( \text{intro}: r = 0 \Rightarrow l = 0 \)
using assms by (auto simp: sgn-0-0)

lemma invariant-1-pos: assumes rc: invariant-1 \( (p,l,r) \)
shows \( \text{simp}: \text{the-unique-root} \ (p,l,r) > 0 \iff r > 0 \) (is \(?x > 0 \iff -\)
and \( \text{simp}: \text{the-unique-root} \ (p,l,r) < 0 \iff r < 0 \)
and \( \text{simp}: \text{the-unique-root} \ (p,l,r) \leq 0 \iff r \leq 0 \)
and \( \text{simp}: \text{the-unique-root} \ (p,l,r) \geq 0 \iff r \geq 0 \)
and \( \text{intro}: r > 0 \Rightarrow l > 0 \)
and \( \text{dest}: l > 0 \Rightarrow r > 0 \)
and \( \text{intro}: r < 0 \Rightarrow l < 0 \)
and \( \text{dest}: l < 0 \Rightarrow r < 0 \)
proof(atomize(full),goal-cases)
case 1
let \(?r = \text{real-of-rat} \)
from assms[unfolded invariant-1-def]
have ur: unique-root (p,l,r) and sgn: sgn l = sgn r by auto
from unique-rootD(1−2)[OF ur] have le: ?r l ≤ ?x ?x ≤ ?r r by auto
from rc show ?case
proof (cases r 0::rat rule:linorder-cases)
  case greater
  with sgn have sgn l = 1 by simp
  hence l0: l > 0 by (auto simp: sgn-1-pos)
  hence ?r l > 0 by auto
  hence ?x > 0 using le(l) by arith
  with greater l0 show ?thesis by auto
next
  case equal
  with real-of-1-0[OF rc] show ?thesis by auto
next
  case less
  hence ?r r < 0 by auto
  with le(2) have ?x < 0 by arith
  with less sgn show ?thesis by (auto simp: sgn-1-neg)
next
lemma poly-cond2I[intro!]: poly-cond p \Longrightarrow degree p > 1 \Longrightarrow poly-cond2 p by (simp add: poly-cond2-def)
lemma poly-cond2D:
  assumes poly-cond2 p
  shows poly-cond p and degree p > 1 using assms by (auto simp: poly-cond2-def)
lemma poly-cond2E[elim!]:
  assumes poly-cond2 p and poly-cond p \Longrightarrow degree p > 1 \Longrightarrow thesis shows thesis using assms by (auto simp: poly-cond2-def)
lemma invariant-1-2-poly-cond2: invariant-1-2 rai \Longrightarrow poly-cond2 (poly-real-alg-1 rai)
  unfolding invariant-1-1-def invariant-1-2-def poly-cond2-def by auto
lemma invariant-1-2I[introl!]:
  assumes invariant-1 rai and degree (poly-real-alg-1 rai) > 1 shows invariant-1-2 rai
  using assms by (auto simp: invariant-1-2-def)
lemma invariant-1-2E[elim!]:
  assumes invariant-1-2 rai

110
and invariant-1 rai \Rightarrow \text{degree} (\text{poly-real-alg-1 rai}) > 1 \Rightarrow \text{thesis}

shows thesis using assms[unfolded invariant-1-2-def] by auto

lemma invariant-1-2-realI:
  fixes plr :: real-alg-1
  defines p ≡ poly-real-alg-1 plr and l ≡ rai-lb plr and r ≡ rai-ub plr
  assumes x: root-cond plr x and sgn: sgn l = sgn r and ur: unique-root plr and p: poly-cond2 p
  shows invariant-1-2 plr ∧ real-of-1 plr = x
  using invariant-1-realI[OF x] p sgn ur unfolding p-def l-def r-def by auto

8.2 Real Algebraic Numbers = Rational + Irrational Real Algebraic Numbers

In the next representation of real algebraic numbers, we distinguish between rational and irrational numbers. The advantage is that whenever we only work on rational numbers, there is not much overhead involved in comparison to the existing implementation of real numbers which just supports the rational numbers. For irrational numbers we additionally store the number of the root, counting from left to right. For instance \( -\sqrt{2} \) and \( \sqrt{2} \) would be root number 1 and 2 of \( x^2 - 2 \).

8.2.1 Definitions and Algorithms on Raw Type

datatype real-alg-2 = Rational rat | Irrational nat real-alg-1

fun invariant-2 :: real-alg-2 ⇒ bool where
  invariant-2 (Irrational n rai) = (invariant-1-2 rai ∧ n = card(roots-below (poly-real-alg-1 rai) (real-of-1 rai)))
  | invariant-2 (Rational r) = True

fun real-of-2 :: real-alg-2 ⇒ real where
  real-of-2 (Rational r) = of-rat r
  | real-of-2 (Irrational n rai) = real-of-1 rai

definition of-rat-2 :: rat ⇒ real-alg-2 where
  [code-unfold]: of-rat-2 = Rational

  by (auto simp: of-rat-2-def)

typedef real-alg-3 = Collect invariant-2
morphism rep-real-alg-3 Real-Alg-Invariant
  by (rule exI[of - Rational 0], auto)
setup-lifting type-definition-real-alg-3

lift-definition real-of-3 :: real-alg-3 => real is real-of-2.

8.2.2 Definitions and Algorithms on Quotient Type

quotient-type real-alg = real-alg-3 / λ x y. real-of-3 x = real-of-3 y

morphisms rep-real-alg Real-Alg-Quotient

by (auto simp: equivp-def) metis

lift-definition real-of :: real-alg => real is real-of-3.

8.2.3 Sign

definition sgn-1 :: real-alg-1 => rat where

sgn-1 x = sgn (rai-ub x)

lemma sgn-1: invariant-1 x ==> real-of-rat (sgn-1 x) = sgn (real-of-1 x)

unfolding sgn-1-def by auto

lemma sgn-1-inj: invariant-1 x ==> invariant-1 y ==> real-of-1 x = real-of-1 y ==> sgn-1 x = sgn-1 y

by (auto simp: sgn-1-def elim!: invariant-1E)

8.2.4 Normalization: Bounds Close Together

lemma unique-root-lr: assumes ur: unique-root plr shows rai-lb plr \leq rai-ub plr
(is \?l \leq \?r)

proof -

let \?p = poly-real-alg-1 plr

from ur[unfolded root-cond-def]

have ex1: \exists x :: real. of-rat ?l \leq x \land x \leq of-rat ?r \land ipoly \?p x = 0 by (cases plr, simp)

then obtain x :: real where bnd: of-rat ?l \leq x \leq of-rat ?r and rt: ipoly \?p x = 0 by auto

from bnd have of-rat ?l \leq of-rat ?r by linarith

thus \?l \leq ?r by (simp add: of-rat-less-eq)

qed

locale map-poly-zero-hom-0 = base: zero-hom-0

begin

sublocale zero-hom-0 map-poly hom by (unfold-locales,auto)

end

interpretation of-int-poly-hom:

map-poly-zero-hom-0 of-int :: int => 'a :: {ring-1, ring-char-0} ..
lemma ipoly-roots-finite: \( p \neq 0 \implies \text{finite} \{ x :: 'a :: \{\text{idom, ring-char-0}\}. \text{ipoly} p \ x = 0 \}

by (rule poly-roots-finite, simp)

lemma roots-below-the-unique-root:
assumes ur: unique-root \((p,l,r)\)
shows roots-below \(p\) (the-unique-root \((p,l,r)\)) = roots-below \(p\) (of-rat \(r\)) (is roots-below \(p\) ?\(x\) = -)

proof-
from ur have rc: root-cond \((p,l,r)\) ?\(x\) by (auto dest!: unique-rootD)
with ur have \(x\): \{ \text{root-cond} \((p,l,r)\) \(x\) \} = \{ ?\(x\) \} by (auto intro: the-unique-root-equivI)
from rc have ?\(x\) \\(\in\) \{ \(y\) \. \(y\) \(\leq\) \(y\) \(\wedge\) \(y\) \(\leq\) of-rat \(r\) \(\wedge\) \(\text{ipoly} p\) \(y\) = 0 \} by auto

with rc have \(lx\): \ldots = \{ ?\(x\) \} by (intro equalityI, fold x(1), force, simp add: \(x\))

have \(rb\): roots-below \((p, \text{of-rat} \(r\)) = \text{roots-below} \(p\) ?\(x\) \(\cup\) \{ \(y\). \(y\) \(\leq\) \(y\) \(\wedge\) \(y\) \(\leq\) of-rat \(r\) \(\wedge\) \(\text{ipoly} p\) \(y\) = 0 \}

using rc by auto

have \(emp\): \(\exists x\). the-unique-root \((p, l, r)\) \(<\) \(x\) \implies \(x\) \(\notin\) \{ \(ra\). \(\exists x\). \(ra\) \(\leq\) \(ra\) \(\wedge\) \(ra\) \(\leq\) \(\text{real-of-rat} r\) \(\wedge\) \(\text{ipoly} p\) \(ra\) = 0 \}

using \(lx\) by auto

with \(rb\) show \(?\text{thesis}\) by auto

qed

lemma unique-root-sub-interval:
assumes ur: unique-root \((p,l,r)\)
and rc: root-cond \((p,l',r')\) (the-unique-root \((p,l,r)\))
and between: \(l \leq l'\ r' \leq r\)
shows unique-root \((p,l',r')\)
and the-unique-root \((p,l',r')\) = the-unique-root \((p,l,r)\)

proof –
from between have ord: real-of-rat \(l\) \(\leq\) real-of-rat \(l'\) real-of-rat \(r'\) \(\leq\) real-of-rat \(r\) by (auto simp: of-rat-less-eq)
from rc have \(lr\): real-of-rat \(l'\) \(\leq\) real-of-rat \(r'\) by auto
with ord have \(lr\): real-of-rat \(l\) \(\leq\) real-of-rat \(r\) by auto

show \(\exists x\). root-cond \((p, l', r')\) \(x\)

proof (rule, rule rc)
fix \(y\)

assume root-cond \((p,l',r')\) \(y\)

with ord have root-cond \((p,l,r)\) \(y\) by (auto intro!: root-condI)

from the-unique-root-equivI[OF ur this] show \(y\) = the-unique-root \((p,l,r)\) by simp

qed

from the-unique-root-equivI[OF this rc]
show the-unique-root \((p,l',r')\) = the-unique-root \((p,l,r)\) by simp

qed

lemma invariant-1-sub-interval:
assumes rc: invariant-1 \((p,l,r)\)
and sub: root-cond \((p,l',r')\) (the-unique-root \((p,l,r)\))
and between: $l \leq l' r' \leq r$

shows invariant-1 $(p,l',r')$ and real-of-1 $(p,l',r') = real-of-1 (p,l)$

proof --

let $r = real-of-rat$

note rcD = invariant-1D[OF rc]

from rc

have ur: unique-root $(p, l', r')$
    and id: the-unique-root $(p, l', r') = the-unique-root (p, l, r)$
    by (atomize(full), intro conj1 unique-root-sub-interval[OF - sub between], auto)

show real-of-1 $(p,l',r') = real-of-1 (p,l)$
    using id by simp

from rcD[1][unfolded split] have $\forall r \leq \forall r$ by auto

hence $l : l \leq r$ by (auto simp: of-rat-less-eq)

from unique-rootD[OF ur] have $\forall r \leq \forall r$ by auto

hence $l : l \leq r$ by (auto simp: of-rat-less-eq)

have $sgn l = sgn r'$

proof (cases r 0: rat rule: linorder-cases)

  case less
  with $l : l'$ between have $l < 0 l' < 0 r' < 0 r < 0$ by auto
  thus $\forall r l \leq \forall l r'$ by auto

next

  case equal with rcD(2) have $l = 0$ using sgn-0 by auto

  with equal between $l r'$ have $l = 0 r' = 0$ by auto then show $\forall r l \leq \forall l r'$ by auto

next

  case greater

  with rcD(4) have $sgn r = 1$ unfolding sgn-rat-def by (cases r = 0, auto)

  with rcD(2) have $sgn l = 1$ by simp

  hence $l : l > 0$ unfolding sgn-rat-def by (cases l = 0; cases l < 0; auto)

  with $l : l'$ between have $l > 0 l' > 0 r' > 0 r > 0$ by auto

  thus $\forall r l \leq \forall l r'$ by auto

qed

with between ur rc show invariant-1 $(p,l',r')$ by (auto simp add: invariant-1-def id)

qed

lemma root-sign-change: assumes
    p0: $poly (p::real poly) \ x = 0$ and
    pd-ne0: $poly (pderiv p) \ x \neq 0$

obtains $d$ where
    $0 < d$
    $sgn (poly p (x - d)) \neq sgn (poly p (x + d))$
    $sgn (poly p (x - d)) \neq 0$
    $0 \neq sgn (poly p (x + d))$
    $\forall d' > 0. \ d' \leq d \Longrightarrow sgn (poly p (x + d')) = sgn (poly p (x + d)) \land sgn (poly p (x - d')) = sgn (poly p (x - d))$

proof --

  assume $a: \bigwedge d, 0 < d \Longrightarrow$
    $sgn (poly p (x - d)) \neq sgn (poly p (x + d)) \Longrightarrow$
    $sgn (poly p (x - d)) \neq 0 \Longrightarrow$
0 ≠ \text{sgn} (\text{poly} \ p \ (x + d)) \implies \\
\forall d' > 0. \ d' \leq d \implies \\
\text{sgn} (\text{poly} \ p \ (x + d')) = \text{sgn} (\text{poly} \ p \ (x + d)) \land \text{sgn} (\text{poly} \ p \ (x - d'))
= \text{sgn} (\text{poly} \ p \ (x - d)) \implies \\
\text{thesis}

\text{from} \ \text{pd-ne0 consider} \ \text{poly} \ (\text{pderiv} \ p) \ x > 0 \ \mid \ \text{poly} \ (\text{pderiv} \ p) \ x < 0 \ \text{by linarith}
\text{thus} \ \text{?thesis proof(cases)}

\text{case 1}
\begin{align*}
\text{obtain} \ d1 \ \text{where} \ d1 \land h. \ 0 < h \implies h < d1 \implies \text{poly} \ p \ (x - h) < 0 \ 0 d1 > 0 \\
\text{using} \ \text{DERIV-pos-inc-left}[\text{OF} \ \text{poly-DERIV} \ 1] \ p0 \ \text{by} \ \text{auto}
\end{align*}
\text{obtain} \ d2 \ \text{where} \ d2 \land h. \ 0 < h \implies h < d2 \implies \text{poly} \ p \ (x + h) > 0 \ 0 d2 > 0 \\
\text{using} \ \text{DERIV-pos-inc-right}[\text{OF} \ \text{poly-DERIV} \ 1] \ p0 \ \text{by} \ \text{auto}
\text{have} \ g0:0 < (\min d1 d2) / 2 \ \text{using} \ d1 \ d2 \ \text{by} \ \text{auto}
\text{hence} \ m1: \min d1 d2 / 2 < d1 \ \text{and} \ m2: \min d1 d2 / 2 < d2 \ \text{by auto}
\{ \ \text{fix} \ d \ \\
\text{assume} \ a1:0 < d \ \text{and} \ a2: d < \min d1 d2 \ \\
\text{have} \ \text{sgn} (\text{poly} \ p \ (x - d)) = -1 \ \text{sgn} (\text{poly} \ p \ (x + d)) = 1 \\
\text{using} \ d1(1)[\text{OF} \ a1] \ d2(1)[\text{OF} \ a1] \ a2 \ \text{by} \ \text{auto} \ \\
\text{note} \ d\text{-this} \ \\
\text{show} \ \text{?thesis by}(\text{rule} \ \text{a}[\text{OF} \ g0]; \text{insert} \ d0 \ m1 \ m2, \ \text{simp})
\}
\text{next}
\text{case 2}
\begin{align*}
\text{obtain} \ d1 \ \text{where} \ d1 \land h. \ 0 < h \implies h < d1 \implies \text{poly} \ p \ (x - h) > 0 \ 0 d1 > 0 \\
\text{using} \ \text{DERIV-neg-dec-left}[\text{OF} \ \text{poly-DERIV} \ 2] \ p0 \ \text{by} \ \text{auto}
\end{align*}
\text{obtain} \ d2 \ \text{where} \ d2 \land h. \ 0 < h \implies h < d2 \implies \text{poly} \ p \ (x + h) < 0 \ 0 d2 > 0 \\
\text{using} \ \text{DERIV-neg-dec-right}[\text{OF} \ \text{poly-DERIV} \ 2] \ p0 \ \text{by} \ \text{auto}
\text{have} \ g0:0 < (\min d1 d2) / 2 \ \text{using} \ d1 \ d2 \ \text{by} \ \text{auto}
\text{hence} \ m1: \min d1 d2 / 2 < d1 \ \text{and} \ m2: \min d1 d2 / 2 < d2 \ \text{by auto}
\{ \ \text{fix} \ d \ \\
\text{assume} \ a1:0 < d \ \text{and} \ a2: d < \min d1 d2 \ \\
\text{have} \ \text{sgn} (\text{poly} \ p \ (x - d)) = 1 \ \text{sgn} (\text{poly} \ p \ (x + d)) = -1 \\
\text{using} \ d1(1)[\text{OF} \ a1] \ d2(1)[\text{OF} \ a1] \ a2 \ \text{by} \ \text{auto} \ \\
\text{note} \ d\text{-this} \ \\
\text{show} \ \text{?thesis by}(\text{rule} \ \text{a}[\text{OF} \ g0]; \text{insert} \ d0 \ m1 \ m2, \ \text{simp})
\}
\text{qed}
\text{qed}

\text{lemma rational-root-free-degree-iff: assumes} \ \text{rf: root-free} \ (\text{map-poly rat-of-int} \ p) \\
\text{and} \ \text{rt: ipoly} \ p \ x = 0 \\
\text{shows} \ (x \in \mathbb{Q}) = (\text{degree} \ p = 1)
\text{proof}
\text{assume} \ x \in \mathbb{Q} \\
\text{then obtain} \ y \ \text{where} \ x : x = \text{of-rat} \ y \ (\text{is} \ - \ ?x) \ \text{unfolding} \ \text{Rats-def by} \ \text{blast}
\text{from} \ \text{rt[unfolded x]} \ \text{have} \ \text{poly} \ (\text{map-poly rat-of-int} \ p) \ y = 0 \ \text{by} \ \text{simp}
\text{with} \ \text{rf show} \ \text{degree} \ p = 1 \ \text{unfolding} \ \text{root-free-def by} \ \text{auto}
\text{next}
\text{assume} \ \text{degree} \ p = 1 \\
\text{from} \ \text{degree1-coeffs}[\text{OF} \ \text{this}]
\text{obtain} \ a \ b \ \text{where} \ p : p = [:a,b:] \ \text{and} \ b : b \neq 0 \ \text{by} \ \text{auto}
\text{next}

lemma rational-poly-cond-iff: assumes poly-cond p and ipoly p x = 0 and degree p > 1 shows \( (x \in \mathbb{Q}) = (\text{degree } p = 1) \)
proof (rule rational-root-free-degree-iff[OF assms(2)])
  from poly-condD[OF assms(1)] irreducible-connect-rev[of p] assms(3)
  have \( p: \text{irreducible} \) by auto
  from irreducible-a-int-rat[OF this]
  have irreducible (map-poly rat-of-int p) by simp
  thus root-free (map-poly rat-of-int p) by (rule irreducible-root-free)
qed

lemma poly-cond-degree-gt-1: assumes poly-cond p degree p > 1 ipoly p x = 0 shows \( x \notin \mathbb{Q} \) using rational-poly-cond-iff[OF assms(1,3)] assms(2) by simp

lemma poly-cond2-no-rat-root: assumes poly-cond2 p shows ipoly p (real-of-rat x) \( \neq 0 \) using poly-cond-degree-gt-1[of p real-of-rat x] assms by auto

context
defines p :: int poly
and x :: rat
begin

lemma gt-rat-sign-change:
  assumes ur: unique-root plr
defines p \equiv poly-real-alg-1 plr and l \equiv rai-lb plr and r \equiv rai-ub plr
assumes p: poly-cond2 p and in-interval:: l \leq y y \leq r
shows (\( \text{sgn} (\text{ipoly } p \ y) = \text{sgn} (\text{ipoly } p \ r) \)) = (\( \text{of-rat } y > \text{the-unique-root plr} \)) (is \( ?gt = - \))
proof (rule ccontr)
  have plr[simp]: plr = (p,l,r) by (cases plr, auto simp: p-def l-def r-def)
  assume \( ?gt \neq (\text{of-rat } y > \text{the-unique-root plr}) \)
  note a = this[unfolded plr]
  from p have irreducible p by auto
  note nz = poly-cond2-no-rat-root[OF p]
  hence p \( \neq 0 \) unfolding irreducible-def by auto
  hence p0-real: real-of-int-poly p \( \neq (0::\text{real poly}) \) by auto
  let \( ?p = \text{real-of-int-poly } p \)
  note urD = unique-rootD[OF ur, simplified]
  let \( ?ur = \text{the-unique-root} \ (p, l, r) \)
let \( ?r = \text{real-of-rat} \)

from \( \text{poly-cond2-no-rat-root} \ p \)

have \( \text{rc:poly } p \ y \neq 0 \) by auto

from \( \text{in-interval} \) have \( \text{in:} ?r \ l \leq ?r \ y \ ?y \ y \leq ?r \ r \) unfolding \( \text{of-rat-less-eq} \) by auto

from \( p \) square-free-of-int-poly[of \( p \)] square-free-rsquarefree

have \( \text{rsf:rsquarefree } p \) by auto

have \( \text{ur3:poly } \ ?p \ ?ur = 0 \) using \( \text{urD(3)} \) by simp

from \( ur \) have \( \text{?ur} \leq \text{of-rat } r \) by (auto elim!: unique-rootE)

moreover

from \( nz \) have \( \text{ipoly } p \ (\text{real-of-rat } r) \neq 0 \) by auto

with \( \text{ur3} \) have \( \text{real-of-rat } r \neq \text{real-of-1} \ (p,l,r) \) by force

ultimately have \( ?ur < ?r \ r \) by auto

hence \( \text{ur2: 0} < ?r \ r - ?ur \) by linarith

from \( \text{rsquarefree-roots } rsf \ ur3 \)

have \( \text{pd-nonz:poly (pderiv } p) \ ?ur \neq 0 \) by auto

obtain \( d \) where \( d: \wedge d', \ d' > 0 \Rightarrow d' \leq d \Rightarrow \)

\[ \text{sgn (poly } ?p \ (\text{?ur} + d')) = \text{sgn (poly } ?p \ (\text{?ur} + d)) \land \]

\[ \text{sgn (poly } ?p \ (\text{?ur} - d')) = \text{sgn (poly } ?p \ (\text{?ur} - d)) \land \]

\[ \text{sgn (poly } ?p \ (\text{?ur} - d)) \neq \text{sgn (poly } ?p \ (\text{?ur} + d)) \land \]

\[ \text{sgn (poly } ?p \ (\text{?ur} + d)) \neq 0 \land \]

\[ \text{d-ge-0: } d > 0 \]

by (metis root-sign-change[OF \( \text{ur3 } \text{pd-nonz} \])

have \( \text{sr:sgn (poly } ?p \ (\text{?ur} + d)) = \text{sgn (poly } ?p \ (\text{?ur } r)) \)

proof (cases \( ?r \ r - ?ur \leq d \))

  case \( \text{True} \)

  show \( ?\text{thesis using } d'(l)[OF ur2 True] \) by auto

next

  case \( \text{False} \)

  hence less: \( ?ur + d < ?r \ r \) by auto

  show \( ?\text{thesis} \)

  proof (rule no-roots-inbetween-imp-same-sign[OF less,rule-format!,goal-cases])

    case \( (1 \ x) \)

    from \( \text{ur1 } d\text{-ge-0} \) have \( \text{ran: real-of-rat } l \leq x x \leq \text{real-of-rat } r \) by (auto elim!: unique-rootE)

    from \( 1 \ d\text{-ge-0} \) have \( \text{the-unique-root } (p, l, r) \neq x \) by auto

    with \( ur \) have \( \text{\neg root-cond } (p,l,r) x \) by auto

    with \( \text{ran} \) show \( ?\text{case} \) by auto

qed

qed

consider \( ?r \ l < ?ur - d \ ?r \ l < ?ur \mid 0 < ?ur - ?r \ l ?ur - ?r \ l \leq d \mid ?ur = ?r \ l \)

using \( \text{urD} \) by argo

hence \( sl: \text{sgn (poly } ?p \ (\text{?ur} - d)) = \text{sgn (poly } ?p \ (\text{?ur } l)) \lor 0 = \text{sgn (poly } ?p \ (\text{?r } l)) \)

proof (cases)

  case \( 1 \)

  have \( \text{sgn (poly } ?p \ (\text{?r } l)) = \text{sgn (poly } ?p \ (\text{?ur} - d)) \)

  proof (rule no-roots-inbetween-imp-same-sign[OF \( 1(l) \),rule-format!,goal-cases])

    case \( (1 \ x) \)

    from \( \text{ur1 } d\text{-ge-0} \) have \( \text{ran: real-of-rat } l \leq x x \leq \text{real-of-rat } r \) by (auto elim!: unique-rootE)
from 1 d-ge-0 have the-unique-root (p, l, r) ≠ x by auto
with ur have ¬ root-cond (p,l,r) x by auto
with ran show ?case by auto
qed
thus ?thesis by auto
next case 2 show ?thesis using d'(1)|OF 2| by simp
qed (insert ur3,simp)
have diff-sign: sgn (ipoly p l) ≠ sgn (ipoly p r)
using d'(2−) sr sl real-of-rat-sgn by auto
have wr':∀ x. real-of-rat l ≤ x ∧ x ≤ real-of-rat y → ipoly p x = 0 → ¬ (?r y ≤ the-unique-root (p,l,r))
proof(standard+ goal-cases)
  case (1 x)
  {
    assume id: the-unique-root (p,l,r) = ?r y
    from nz[of y] id ur have False by (auto elim!: unique-rootE)
  }
  note neq = this
  have root-cond (p, l, r) x unfolding root-cond-def
  using 1 ur by (auto elim!: unique-rootE)
  with conjunct2[OF 1(1)] 1(2−) the-unique-root-eqI[OF ur]
  show ?case by (auto intro!: neq)
qed
hence ur''∀ x. real-of-rat y ≤ x ∧ x ≤ real-of-rat r → poly (real-of-int-poly p) x ≠ 0 → ¬ (?r y ≤ the-unique-root (p,l,r))
using urD(2,3) by auto
have (sgn (ipoly p y) = sgn (ipoly p r)) = (?r y > the-unique-root (p,l,r))
proof(cases sgn (ipoly p r) = sgn (ipoly p y))
  case True
  have sgn:sgn (poly ?p (real-of-rat l)) ≠ sgn (poly ?p (real-of-rat y)) using True diff-sign
  by (simp add: real-of-rat-sgn)
  have ly:of-rat l < (of-rat y::real) using in-interval True diff-sign less-eq-rat-def
  of-rat-less by auto
  show ?thesis by force
next
  case False
  hence neqsgn (ipoly p (real-of-rat y)) ≠ sgn (ipoly p (real-of-rat r)) by (simp add: real-of-rat-sgn)
  have ry:of-rat y < (of-rat r::real) using in-interval False diff-sign less-eq-rat-def
  of-rat-less by auto
  obtain x where x:real-of-rat y ≤ x x ≤ real-of-rat r ipoly p x = 0
  hence lx:real-of-rat l ≤ x using in-interval
  using False a wrD by auto
  have ?ur = x using lx zr ur by (intro the-unique-root-eqI, auto)
  then show ?thesis using False x by auto
qed
thus False using diff-sign(1) a ru by(cases ipoly p r = 0;auto simp:sgn-0-0)
qed

definition tighten-poly-bounds :: rat ⇒ rat ⇒ rat × rat × rat
where
  tighten-poly-bounds l r sr = (let m = (l + r) / 2; sm = sgn (ipoly p m) in
    if sm = sr
      then (l,m,sm) else (m,r,sr))

lemma tighten-poly-bounds; assumes res: tighten-poly-bounds l r sr = (l',r',sr')
and ur: unique-root (p,l,r)
and p: poly-cond2 p
and sr: sr = sgn (ipoly p r)
shows root-cond (p,l',r') (the-unique-root (p,l,r)) l ≤ l' l' ≤ r' r' ≤ r
  (r' − l') = (r − l) / 2 sr' = sgn (ipoly p r')
proof –
  let ?x = the-unique-root (p,l,r)
  let ?x' = the-unique-root (p,l',r')
  let ?m = (l + r) / 2
  note d = tighten-poly-bounds-def Let-def
from unique-root-br[OF ur] have lr: l ≤ r by auto
thus l ≤ l' l' ≤ r' r' ≤ r (r' − l') = (r − l) / 2 sr' = sgn (ipoly p r')
  using res sr unfolding d by (auto split: if-splits)
hence l ≤ ?m ?m ≤ r by auto
note le = gt-rat-sign-change[OF ur,simplified,OF p this]
note urD = unique-rootD[OF ur]
show root-cond (p,l',r') ?x
proof (cases sgn (ipoly p ?m) = sgn (ipoly p r))
  case #: False
    with res sr have id: l' = ?m r' = r unfolding d by auto
    from unfolding le urD show ?thesis unfolding id by auto
next
  case #: True
    with res sr have id: l' = l r' = ?m unfolding d by auto
    from unfolding le urD show ?thesis unfolding id by auto
qed
qed

partial-function (tailrec) tighten-poly-bounds-epsilon :: rat ⇒ rat ⇒ rat ⇒ rat ×
rat × rat where
  [code]: tighten-poly-bounds-epsilon l r sr = (if r - l ≤ x then (l,r,sr) else
    (case tighten-poly-bounds l r sr of (l',r',sr') ⇒ tighten-poly-bounds-epsilon l' r'
    sr'))

partial-function (tailrec) tighten-poly-bounds-for-x :: rat ⇒ rat ⇒ rat ⇒
rat × rat × rat where
  [code]: tighten-poly-bounds-for-x l r sr = (if x < l ∨ r < x then (l, r, sr) else
    (case tighten-poly-bounds l r sr of (l',r',sr') ⇒ tighten-poly-bounds-for-x l' r'
    sr'))

lemma tighten-poly-bounds-epsilon:
assumes ur: unique-root (p,l,r)
defines u: u = the-unique-root (p,l,r)
assumes p: poly-cond2 p
and res: tighten-poly-bounds-epsilon l r sr = (l',r',sr')
and sr: sr = sgn (ipoly p r)
and x: x > 0
shows l ≤ l' r' ≤ r root-cond (p,l',r') u r' - l' ≤ x sr' = sgn (ipoly p r')
proof –
let ?u = the-unique-root (p,l,r)
define delta where delta = x / 2
have delta: delta > 0 unfolding delta-def using x by auto
let ?dist = λ (l,r,sr). r - l
let ?rel = inv-image {(x, y). 0 ≤ y ∧ delta - gt delta x y} ?dist
note SN = SN-inv-image[OF delta - gt SN[OF delta], of ?dist]
note simps = res[unfolded tighten-poly-bounds-for-x,simps[of l r]]
let ?P = λ (l,r,sr). unique-root (p,l,r) → u = the-unique-root (p,l,r)
→ tighten-poly-bounds-epsilon l r sr = (l',r',sr')
→ sr = sgn (ipoly p r)
→ l ≤ l' ∧ r' ≤ r ∧ r' - l' ≤ x ∧ root-cond (p,l',r') u r' = sgn (ipoly p r')
have ?P (l,r,sr)
proof (induct rule: SN-induct[OF SN])
case (l r)
obtain l r sr where lr: lr = (l,r,sr) by (cases lr, auto)
show ?case unfolding lr split
proof (intro impI)
assume ur: unique-root (p, l, r)
and u: u = the-unique-root (p, l, r)
and res: tighten-poly-bounds-epsilon l r sr = (l', r', sr')
and sr: sr = sgn (ipoly p r)
note tur = unique-rootD[OF ur]
note simps = tighten-poly-bounds-epsilon.simps[of l r sr]
show l ≤ l' ∧ r' ≤ r ∧ r' - l' ≤ x ∧ root-cond (p, l', r') u ∧ sr' = sgn (ipoly p r')
proof (cases r - l ≤ x)
case True
with res[unfolded simps] urtur(4) u sr
show ?thesis by auto
next
case False
hence x: r - l > x by auto
let ?tight = tighten-poly-bounds l r sr
obtain L R SR where tight: ?tight = (L,R,SR) by (cases ?tight, auto)
note tighten = tighten-poly-bounds[OF tight[unfolded sr] ur p]
from unique-root-sub-interval[OF ur tighten[1 - 2,4]] p
have ur': unique-root (p,L,R) u = the-unique-root (p,L,R) unfolding u by auto
from res[unfolded simps tight] False sr have tighten-poly-bounds-epsilon L R SR = (l',r',sr') by auto
note IH = l[of (L,R,SR), unfolded tight split lr, rule-format, OF - ur' this]
have $L \leq l' \land r' \leq R \land r' - l' \leq x \land \text{root-cond} (p, l', r') \ u \land sr' = \text{sgn} (\text{ipoly p r}')$

by (rule IH, insert tighten False, auto simp: delta-def delta-def)
thus $\text{thesis using tighten by auto}$
qed
qed

defined by (cases lr, auto)

lemma tighten-poly-bounds-for-x:
assumes $ur: \text{unique-root} (p,l,r)$
defines $u: u \equiv \text{the-unique-root} (p,l,r)$
assumes $p: \text{poly-cond2} p$
and $\text{res}: \text{tighten-poly-bounds-for-x} l r sr = (l',r',sr')$
and $sr: sr = \text{sgn} (\text{ipoly p r})$

shows $l \leq l' \land l' \leq r \land r' \leq r \land \text{root-cond} (p,l',r') \ u \land (l' \leq x \land x \leq r') \ sr' = \text{sgn} (\text{ipoly p r'})$

proof –
let $?u = \text{the-unique-root} (p,l,r)$
let $?x = \text{real-of-rat} x$
define $\text{delta} \ where \ \text{delta} = \text{abs} ((u - ?x) / 2)$
let $?p = \text{real-of-int-poly} p$

note $ru = \text{unique-rootD}[OF ur]$.

assumes $u = ?x$

note $u = \text{this[unfolded u]}$

from $\text{poly-cond2-no-rat-root}[OF p]$ have $\text{False by (elim unique-rootE, auto simp: u)}$

hence $\text{delta}: \text{delta} > 0$ unfolding $\text{delta-def by auto}$

let $?\text{dist} = \lambda (l,r,\text{sr}). \ \text{real-of-rat} (r - l)$
let $?\text{rel} = \text{inv-image} \{(x, y). \ 0 \leq y \land \text{delta-gt delta x y}\}$ $?\text{dist}$

note $\text{SN} = \text{SN-inv-image}[OF \delta \text{def-SN}[OF \text{delta}], \ OF ?\text{dist}]$

note $\text{simps} = \text{res[unfolded tighten-poly-bounds-for-x.simp[of l r]]}$

let $?\text{P} = \lambda (l,r,\text{sr}), \text{unique-root} (p,l,r) \rightarrow u = \text{the-unique-root} (p,l,r)$

$\rightarrow \text{tighten-poly-bounds-for-x} l r sr = (l',r',sr')$

$\rightarrow sr = \text{sgn} (\text{ipoly p r})$

$\rightarrow l \leq l' \land r' \leq r \land \neg (l' \leq x \land x \leq r') \land \text{root-cond} (p,l',r') \ u \land sr' = \text{sgn} (\text{ipoly p r'})$

have $?\text{P} (l,r,\text{sr})$

proof (induct rule: $\text{SN-induct}[OF \text{SN}]$)
case $(l \ lr)$$obtain \ lr \ sr \ where \ lr = (l,r,\text{sr})$ by (cases lr, auto)

let $?l = \text{real-of-rat} l$
let $?r = \text{real-of-rat} r$

show $?\text{case unfolding lr split}$
proof (intro impl)
  assume ur: unique-root (p, l, r)
  and u: u = the-unique-root (p, l, r)
  and res: tighten-poly-bounds-for-x l r sr = (l', r', sr')
  and sr: sr = sgn (ipoly p r)
  note tur = unique-rootD[OF ur]
  note simps = tighten-poly-bounds-for-x.simps[of l r]
  show l ≤ l' ∧ r' ≤ r ∧ ¬ (l' ≤ x ∧ x ≤ r') ∧ root-cond (p, l', r') u ∧ sr' = sgn (ipoly p r')
  proof (cases x < l ∨ r < x)
    case True
    show ?thesis by auto
  next
  case False
  hence x: l ≤ x ∧ x ≤ r by (auto simp: of-rat-less-eq)
  let ?tight = tighten-poly-bounds l r sr
  obtain L R SR where tight: ?tight = (L,R.SR) by (cases ?tight, auto)
  note tighten = tighten-poly-bounds[OF tight ur p sr]
  from unique-root-sub-interval[OF ur tighten(1-2,4)] p
  have ur': unique-root (p,L,R) u = the-unique-root (p,L,R) unfolding u by auto
  from res[unfolded simps tight] False have tighten-poly-bounds-for-x L R SR = (l',r',sr') by auto
  note IH = [of ?tight, unfolded tight split lr, rule-format, OF - ur' this]
  let ?DIFF = real-of-rat (l' - l) let ?diff = real-of-rat (r - l)
  have diff0: 0 ≤ ?DIFF using tighten(3)
    by (metis cancel-comm-monoid-add-class.diff-cancel diff-right-mono of-rat-less-eq of-rat-hom.hom-zero)
  have *: r - l - (r - l) / 2 = (r - l) / 2 by (auto simp: field-simps)
  have delta-gt delta ?diff ?DIFF = (abs (u - of-rat x) ≤ real-of-rat (r - l))
  unfolding delta-gt-def tighten(5) delta-def of-rat-diff[symmetric] * by (simp add: hom-distribs)
  also have real-of-rat (r - l) * 1 = ?r - ?l
  unfolding of-rat-divide of-rat-mult of-rat-diff by auto
  also have abs (u - of-rat x) ≤ ?r - ?l using x ur by (elim unique-rootE, auto simp: u)
  finally have delta: delta-gt delta ?diff ?DIFF .
  have L ≤ l' ∧ r' ≤ R ∧ ¬ (l' ≤ x ∧ x ≤ r') ∧ root-cond (p, l', r') u ∧ sr' = sgn (ipoly p r')
    by (rule IH, insert delta diff0 tighten(6), auto)
  with *: l ≤ L: (?r ≤ r) show ?thesis by auto
  qed
qed

from this[unfolded split u, rule-format, OF ur refl res sr]
show *: l ≤ l' ∧ r' ≤ r root-cond (p,l',r') u ¬ (l' ≤ x ∧ x ≤ r') sr' = sgn (ipoly p r') unfolding u

122
by auto
from *(3)[unfolded split] have real-of-rat l' ≤ of-rat r' by auto
thus l' ≤ r' unfolding of-rat-less-eq.
show unique-root (p,l',r') using ur *(1−3) p poly-condD(5) u unique-root-sub-interval(1)
by blast
qed
end

definition real-alg-precision :: rat where
real-alg-precision ≡ Rat.Fract 1 2

lemma real-alg-precision: real-alg-precision > 0
by eval

definition normalize-bounds-1-main :: rat ⇒ real-alg-1 ⇒ real-alg-1 where
normalize-bounds-1-main eps rai = (case rai of (p,l,r) ⇒
let (l',r',sr') = tighten-poly-bounds-epsilon p eps l r (sgn (ipoly p r));
fr = rat-of-int (floor r');
(l''r''sr'') = tighten-poly-bounds-for-x p fr l' r' sr'
in (p,l'',r''))

definition normalize-bounds-1 :: real-alg-1 ⇒ real-alg-1 where
normalize-bounds-1 = (normalize-bounds-1-main real-alg-precision)

custom
fixes p q and l r :: rat
assumes cong: \( \forall x. \) real-of-rat l ≤ x \( \implies \) x ≤ of-rat r \( \implies \) (ipoly p x = (0 ::
real)) = (ipoly q x = 0)
begin
lemma root-cond-cong: root-cond (p,l,r) = root-cond (q,l,r)
by (intro ext, insert cong, auto simp: root-cond-def)

lemma the-unique-root-cong:
the-unique-root (p,l,r) = the-unique-root (q,l,r)
unfolding root-cond-cong ..

lemma unique-root-cong:
unique-root (p,l,r) = unique-root (q,l,r)
unfolding root-cond-cong ..

end

lemma normalize-bounds-1-main: assumes eps: eps > 0 and rc: invariant-1-2 x
defines y: y ≡ normalize-bounds-1-main eps x
shows invariant-1-2 y ∧ (real-of-1 y = real-of-1 x)
proof –
obtain p l r where x: x = (p,l,r) by (cases x) auto
note rc = rc[unfolded x]
obtain l' r' sr' where tb: tighten-poly-bounds-epsilon p eps l r (sgn (ipoly p r))
= (l',r',sr')
by (cases rule: prod-cases3, auto)
let ?fr = rat-of-int (floor r')
obtain \( l'' r'' sr'' \) where \( \text{tbx: tighten-poly-bounds-for-x p} \) ?fr l' r' sr' = (l'', r'', sr'')
  by (cases rule: prod-cases3, auto)
from y[unfolded normalize-bounds-1-main-def x] \( \text{tb tbx} \)
have y: \( y = (p, l'', r'') \)
  by (auto simp: Let-def)
from rc have unique-root \((p, l, r)\) and p2: poly-cond2 p by auto
from tighten-poly-bounds-epsilon[OF this refl eps]
have bnd: \( l \leq l' r' \leq r \) and rc': root-cond \((p, l', r')\) (the-unique-root \((p, l, r)\))
  and eps: \( r' - l' \leq eps \)
  and sr': sr'' = sgn (ipoly p r') by auto
from invariant-1-sub-interval[OF - rc' bnd] rc
have inv': invariant-1 \((p, l', r')\) and eq: real-of-1 \((p, l', r')\) = real-of-1 \((p, l, r)\)
by auto
have bnd: \( l' \leq l'' r'' \leq r' \) and rc': root-cond \((p, l'', r'')\) (the-unique-root \((p, l', r')\))
  by (rule tighten-poly-bounds-for-x[OF - p2 tbx sr'], fact invariant-1D[OF inv'])+
from invariant-1-sub-interval[OF inv' rc' bnd] p2 eq
show ?thesis unfolding y x by auto
qed

lemma normalize-bounds-1: assumes x: invariant-1-2 x
shows invariant-1-2 (normalize-bounds-1 x) \( \land \) (real-of-1 \((\text{normalize-bounds-1 x})\)) = real-of-1 x)
proof(cases x)
case xx:(fields p l r)
let ?res = (p,l,r)
have norm: normalize-bounds-1 x = (normalize-bounds-1-main real-alg-precision ?res)
  unfolding normalize-bounds-1-def by (simp add: xx)
from x have x: invariant-1-2 ?res real-of-1 ?res = real-of-1 x unfolding xx by auto
from normalize-bounds-1-main[OF real-alg-precision x(1)] x(2−)
show ?thesis unfolding normalize-bounds-1-def xx by auto
qed

lemma normalize-bound-1-poly: poly-real-alg-1 (normalize-bounds-1 rai) = poly-real-alg-1 rai
unfolding normalize-bounds-1-def normalize-bounds-1-main-def Let-def
by (auto split: prod.splits)

definition real-alg-2-main :: root-info ⇒ real-alg-1 ⇒ real-alg-2 where
real-alg-2-main ri rai ≡ let p = poly-real-alg-1 rai
  in (if degree p = 1 then Rational (Rat.Frac (- coeff p 0) (coeff p 1))
    else (case normalize-bounds-1 rai of (p',l,r) ⇒
      Irrational (root-info.number-root ri r) (p',l,r)))

definition real-alg-2 :: real-alg-1 ⇒ real-alg-2 where
real-alg-2 rai ≡ let p = poly-real-alg-1 rai
  in (if degree p = 1 then Rational (Rat.Fract (− coeff p 0) (coeff p 1))
    else (case normalize-bounds-1 rai of (p',l,r) ⇒
      Irrational (root-info.number-root (root-info p) r) (p',l,r)))

lemma degree-1-ipoly: assumes degree p = Suc 0
  shows ipoly p x = 0 ←→ (x = real-of-rat (Rat.Fract (− coeff p 0) (coeff p 1)))
proof −
  from roots1[of map-poly real-of-int p] assms
  have ipoly p x = 0 ←→ x ∈ {roots1 (real-of-int-poly p)} by auto
  also have . . . = (x = real-of-rat (Rat.Fract (− coeff p 0) (coeff p 1)))
  unfolding Fract-of-int-quotient roots1-def hom-distribs by auto
  finally show ?thesis .
qed

lemma invariant-1-degree-0:
  assumes inv: invariant-1 rai
  shows degree (poly-real-alg-1 rai) ≠ 0 (is degree ?p ≠ 0)
proof (rule notI)
  assume deg: degree ?p = 0
  from inv have ipoly ?p (real-of-1 rai) = 0 by auto
  with deg have ?p = 0 by (meson less-Suc0 representsI represents-degree)
  with inv show False by auto
qed

lemma real-alg-2-main:
  assumes inv: invariant-1 rai
  defines simp: p ≡ poly-real-alg-1 rai
  assumes ric: irreducible (poly-real-alg-1 rai) ⇒ root-info-cond ri (poly-real-alg-1 rai)
  shows invariant-2 (real-alg-2-main ri rai) real-of-2 (real-alg-2-main ri rai) = real-of-1 rai
proof (atomize(full))
  define l r where simp: l ≡ rai-lb rai and simp: r ≡ rai-ub rai
  show invariant-2 (real-alg-2-main ri rai) ∧ real-of-2 (real-alg-2-main ri rai) = real-of-1 rai
    unfolding id using invariant-1D
proof (cases degree p Suc 0 rule: linorder-cases)
  case deg: equal
    hence id: real-alg-2-main ri rai = Rational (Rat.Fract (− coeff p 0) (coeff p 1))
    unfolding real-alg-2-main-def Let-def by auto
    note rc = invariant-1D[OF inv]
  from degree-1-ipoly[OF deg, of the-unique-root rai] rc(1)
  show ?thesis unfolding id by auto
next
  case deg: greater
  with inv have inv: invariant-1-2 rai unfolding p-def by auto
define rai' where rai' = normalize-bounds-1 rai
have rai': real-of-1 rai = real-of-1 rai' and inv': invariant-1-2 rai'
  unfolding rai'-def using normalize-bounds-1[OF inv] by auto
obtain p' l' r' where rai' = (p', l', r') by (cases rai')
with arg-cong[OF rai'-def, of poly-real-alg-1, unfolded normalize-bound-1-poly]
  split
    have split: rai' = (p, l, r) by auto
    from inv'[unfolded split]
    have poly-cond p by auto
    from poly-condD[OF this] have irr: irreducible p by simp
    from ric irr have ric: root-info-cond ri p by auto
    unfolding real-alg-2-main-def Let-def using deg split rai'-def
    by (auto simp: rai'-def rai')
  showthesis unfolding id using rai' root-info-condD(2)[OF ric
    inv'[unfolded split]
    apply (elim invariant-1-2E invariant-1E) using inv'
    by (auto simp: split roots-below-the-unique-root)
  next
    case deg: less then have degree p = 0 by auto
    from this invariant-1-degree-0[OF inv] have p = 0 by simp
    with inv showthesis by auto
qed

lemma real-alg-2: assumes invariant-1 rai
  shows invariant-2 (real-alg-2 rai) real-of-2 (real-alg-2 rai) = real-of-1 rai
proof –
  have deg: 0 < degree (poly-real-alg-1 rai) using assms by auto
  have real-alg-2 rai = real-alg-2-main (root-info (poly-real-alg-1 rai)) rai
    unfolding real-alg-2-main-def real-alg-2-main-def Let-def by auto
  from real-alg-2-main[OF assms root-info, folded this, simp[folded] deg
    show invariant-2 (real-alg-2 rai) real-of-2 (real-alg-2 rai) = real-of-1 rai by auto
qed

lemma invariant-2-realI:
  fixes plr :: real-alg-1
  defines p ≡ poly-real-alg-1 plr and l ≡ rai-lb plr and r ≡ rai-ub plr
  assumes x: root-cond plr x and sgn: sgn l = sgn r
    and ur: unique-root plr
    and p: poly-cond p
  shows invariant-2 (real-alg-2 plr) ∧ real-of-2 (real-alg-2 plr) = x
  using invariant-1-realI[OF x, folded p-def l-def r-def] sgn ur p
    real-alg-2[of plr] by auto

8.2.5 Comparisons

fun compare-rat-1 :: rat ⇒ real-alg-1 ⇒ order where
  compare-rat-1 x (p, l, r) = (if x < l then Lt else if x > r then Gt else

126
if \( sgn (ipoly p x) = sgn(ipoly p r) \) then \( Gt \) else \( Lt \)

**lemma** compare-rat-1: **assumes** rai: invariant-1-2 y  
**shows** compare-rat-1 x y = compare (of-rat x) (real-of-1 y)  
**proof**  
define \( p \) \( l \) \( r \) where \( p \equiv poly-real-alg-1 y \) \( l \equiv rai-lb y \) \( r \equiv rai-ub y \)  
then have \( y \) \([simp]: \) \( y = (p, l, r) \) by (cases y, auto)  
from rai have \( ur \): unique-root y by auto  
show \(?thesis\)  
**proof** \((cases \( x < l \lor x > r \))\)  
\( case \) True  
\{  
\( assume \) \( xl \): \( x < l \)  
\( hence \) real-of-rat \( x < \) of-rat \( l \) \textbf{unfolding} of-rat-less by auto  
with rai have of-rat \( x \) \( < \) the-unique-root \( y \) by (auto elim!: invariant-1E)  
with \( xl \) rai have \(?thesis\) by (cases y, auto simp: compare-real-def comparator-of-def)  
\}  
moreover  
\{  
\( assume \) \( xr \): \( \neg x < l \) \( x > r \)  
\( hence \) real-of-rat \( x > \) of-rat \( r \) \textbf{unfolding} of-rat-less by auto  
with rai have of-rat \( x \) \( > \) the-unique-root \( y \) by (auto elim!: invariant-1E)  
with \( xr \) rai have \(?thesis\) by (cases y, auto simp: compare-real-def comparator-of-def)  
\}  
ultimately show \(?thesis\) using True by auto  
**next**  
**case** False  
\( have \) \( 0 \): \( ipoly p \) (real-of-rat \( x \)) \( \neq 0 \) by (rule poly-cond2-no-rat-root, insert rai, auto)  
with rai have \( \textbf{diff} \): real-of-1 \( y \) \( \neq \) of-rat \( x \) by (auto elim!: invariant-1E)  
\( have \) \( \( P \). \) \((1 < \text{degree} \( \text{poly-real-alg-1} \) \( y \)) \( \Rightarrow \exists !x. \text{root-cond} \) \( y \) \( x \) \( \Rightarrow \text{poly-cond} \) \( p \) \( \Rightarrow \) \( P \)) \( \Rightarrow \) \( P \) using poly-real-alg-1.simps rai invariant-1-2E invariant-1E by metis  
from this[OF gt-rat-sign-change] False  
\( have \) \( \textbf{left} \): compare-rat-1 \( x \) \( y \) = (if real-of-rat \( x \) \( \leq \) the-unique-root \( y \) then \( Lt \) else \( Gt \)) by (auto simp:poly-cond2-def)  
also have \( \ldots = \) compare (real-of-rat \( x \)) (real-of-1 \( y \)) using diff  
by (auto simp: compare-real-def comparator-of-def)  
**finally** show \(?thesis\) .  
qed  
qed  

**lemma** cf-pos-0[simp]: \( \neg \) cf-pos \( 0 \)  
unfolding cf-pos-def by auto
8.2.6 Negation

fun uminus-1 :: real-alg-1 ⇒ real-alg-1 where
uminus-1 (p,l,r) = (abs-int-poly (poly-uminus p), −r, −l)

lemma uminus-1: assumes x: invariant-1 x
defines y: y ≡ uminus-1 x
shows invariant-1 y ∧ (real-of-1 y = − real-of-1 x)

proof (cases x)
case plr: (fields p l r)
  from x plr have inv: invariant-1 (p,l,r) by auto
  note * = invariant-1D[OF this]
  from plr have x: x = (p,l,r) by simp
  let ?p = poly-uminus p
  let ?mp = abs-int-poly ?p
  have y: y = (?mp, −r, −l)
    unfolding y plr by (simp add: Let-def)
    { fix y
      assume root-cond (?mp, −r, −l) y
      hence unfolding root-cond-def by (auto simp: of-rat-minus)
        from mpy have id: ipoly (?mp) y = 0 and bnd: − of-rat r ≤ y y ≤ − of-rat l
        unfolding root-cond-def by (auto simp: of-rat-minus)
        from mpy have id: ipoly (?mp) y = 0 by auto
        from bnd have bnd: − of-rat l ≤ − y − y ≤ − of-rat r by auto
        from id bnd have root-cond (p, l, r) (−y) unfolding root-cond-def by auto
        with inv x have real-of-1 x = − y by (auto intro: the-unique-root-eqI)
        then have −real-of-1 x = y by auto
    } note inj = this
  have rc: root-cond (?mp, −r, −l) (− real-of-1 x)
    using unfolding root-cond-def y x by (auto simp: of-rat-minus sgn-minus-rat)
  from inj rc have ur': unique-root (?mp, −r, −l) by (auto intro: unique-rootI)
  with rc have thc: − real-of-1 x = the-unique-root (?mp, −r, −l) by (auto intro: the-unique-root-eqI)
  have xp: p represents (real-of-1 x) using unfolding root-cond-def split represents-def x by auto
  from * have mon: lead-coeff (?mp) > 0 by (unfold pos-poly-abs-poly, auto)
  from poly-uminus-irreducible * have mi: irreducible (?mp) by auto
  from mi mon have pc*: poly-cond (?mp) by (auto simp: cf-pos-def)
  from poly-condD[OF pc*] have irr: irreducible (?mp) by auto
  show ?thesis unfolding y apply (intro invariant-1-realI ur' rc) using pc' inv
  by auto
  qed

lemma uminus-1-2:
assumes x: invariant-1-2 x
defines y: y ≡ uminus-1 x
shows invariant-1-2 y ∧ (real-of-1 y = − real-of-1 x)

proof
  from x have invariant-1 x by auto
  from uminus-1[OF this] have #: real-of-1 y = − real-of-1 x

128
invariant-1 unfolding y by auto
obtain p l r where id: x = (p,l,r) by (cases x)
from x[unfolded id] have degree p > 1 by auto
moreover have poly-real-alg-1 y = abs-int-poly (poly-uminus p)
  unfolding y id uminus-1.simps split Let-def by auto
ultimately have degree (poly-real-alg-1 y) > 1 by simp
with * show thesis by auto
qed

fun uminus-2 :: real-alg-2 ⇒ real-alg-2 where
  uminus-2 (Rational r) = Rational (−r)
| uminus-2 (Irrational n x) = real-alg-2 (uminus-1 x)

lemma uminus-2: assumes invariant-2 x
  shows real-of-2 (uminus-2 x) = uminus (real-of-2 x)
  invariant-2 (uminus-2 x)
  using assms real-alg-2 uminus-1 by (atomize(full), cases x, auto simp: hom-distrib)

declare uminus-1.simps[simp del]

lift-definition uminus-3 :: real-alg-3 ⇒ real-alg-3 is uminus-2
  by (auto simp: uminus-2)

lemma uminus-3: real-of-3 (uminus-3 x) = − real-of-3 x
  by (transfer, auto simp: uminus-2)

instantiation real-alg :: uminus
begin
lift-definition uminus-real-alg :: real-alg ⇒ real-alg is uminus-3
  by (simp add: uminus-3)
instance ..
end

lemma uminus-real-alg: − (real-of x) = real-of (− x)
  by (transfer, rule uminus-3[symmetric])

8.2.7 Inverse

fun inverse-1 :: real-alg-1 ⇒ real-alg-2 where
  inverse-1 (p,l,r) = real-alg-2 (abs-int-poly (reflect-poly p), inverse r, inverse l)

lemma invariant-1-2-of-rat: assumes rc: invariant-1-2 rai
  shows real-of-1 rai ≠ of-rat x
proof −
  obtain p l r where rai: rai = (p, l, r) by (cases rai, auto)
  from r[unfolded rai]
  have poly-cond2 p ipoly p (the-unique-root (p, l, r)) = 0 by (auto elim!: invariant-1E)
lemma inverse-1:
assumes rcx: invariant-1-2 x
defines y: y ≜ inverse-1 x
shows invariant-2 y ∧ (real-of-2 y = inverse (real-of-1 x))
proof (cases x)
case x: (fields p l r)
from x rcx have rcx: invariant-1-2 (p,l,r) by auto
from invariant-1-2-poly-cond2[OF rcx] have pe2: poly-cond2 p by simp
have x0: real-of-1 (p,l,r) ≠ 0 using invariant-1-2-of-rat[OF rcx, of 0] x by auto
let ?mp = abs-int-poly (reflect-poly p)
from x0 rcx have br0: l ≠ 0 and r ≠ 0 by auto
from x0 rcx have y: y = real-alg-2 (?mp, inverse r, inverse l)
unfolding y x Let-def inverse-1
from rcx have mon: lead-coeff ?mp > 0 by (unfold lead-coeff-abs-int-poly, auto)
{
  fix y
  assume root-cond (?mp, inverse r, inverse l) y
  hence mpy: ipoly ?mp y = 0 and bnd: inverse (of-rat r) ≤ y y ≤ inverse (of-rat l)
  unfolding root-cond-def by (auto simp: of-rat-inverse)
  from sgn-real-mono[OF bnd(l)] sgn-real-mono[OF bnd(2)]
  have sgn (of-rat r) ≤ sgn y sgn y ≤ sgn (of-rat l)
  by (simp-all add: algebra-simps)
  with rcx have sgn: sgn (inverse (of-rat r)) = sgn y sgn y = sgn (inverse (of-rat l))
  unfolding sgn-inverse-of-rat
  by (auto simp add: real-alg-2 sgn-sgn intro: order-antisym)
  from sgn[simplified, unfolded real-of-rat-sgn] br0 have y ≠ 0 by (auto simp: sgn-0-0)
  with mpy have id: ipoly p (inverse y) = 0 by (auto simp: ipoly-reflect-poly)
  from inverse-le-sgn[OF sgn(1) bnd(1)] inverse-le-sgn[OF sgn(2) bnd(2)]
  have bnd: of-rat l ≤ inverse y inverse y ≤ of-rat r by auto
  from id bnd have root-cond (p,l,r) (inverse y) unfolding root-cond-def by auto
  from rcx this x0 have ?x = inverse y by auto
  then have inverse ?x = y by auto
} note inj = this
have rc: root-cond (?mp, inverse r, inverse l) (inverse ?x)
  using rcx x0 apply (elim invariant-1-2E invariant-1E)
  by (simp add: root-cond-def of-rat-inverse real-of-rat-sgn inverse-le-iff-sgn
    ipoly-reflect-poly)
  from inj rc have ur: unique-root (?mp, inverse r, inverse l) by (auto intro: unique-root)
  with rc have the: the-unique-root (?mp, inverse r, inverse l) = inverse ?x by (auto intro: the-unique-root-eql)
have xp: p represents \( ?x \) unfolding split represents-def using rcx by (auto elim: invariant-1E)
  from reflect-poly-irreducible[OF - xp x0] poly-condD rcx
have mi: irreducible \( ?mp \) by auto
from mi mon have an: poly-cond \( ?mp \) by (auto simp: poly-cond-def)
show \( ?\text{thesis} \) using rcx rc ur unfolding y
  by (intro invariant-2-realI, auto simp: x y un)

qed

fun inverse-2 :: real-alg-2 ⇒ real-alg-2 where
inverse-2 (Rational r) = Rational (inverse r)
| inverse-2 (Irrational n x) = inverse-1 x

lemma inverse-2: assumes invariant-2 x
shows real-of-2 (inverse-2 x) = inverse (real-of-2 x)
  using assms
  by (atomize(full), cases x, auto simp: real-alg-2 inverse-1 hom-distrib)

lift-definition inverse-3 :: real-alg-3 ⇒ real-alg-3 is inverse-2
  by (auto simp: inverse-2)

lemma inverse-3: real-of-3 (inverse-3 x) = inverse (real-of-3 x)
  by (transfer, auto simp: inverse-2)

8.2.8 Floor

fun floor-1 :: real-alg-1 ⇒ int where
floor-1 (p,l,r) = (let
  (l',r',sr') = tighten-poly-bounds-epsilon p (1/2) l r (sgn (ipoly p r));
  fr = floor r';
  fl = floor l';
  fr' = rat-of-int fr
  in if fr = fl then fr else
    let (l'',r'',sr'') = tighten-poly-bounds-for-x p fr' l' r' sr'
    in if fr' < l'' then fr else fl)

lemma floor-1: assumes invariant-1-2 x
shows floor (real-of-1 x) = floor-1 x
proof (cases x)
  case (fields p l r)
  obtain l' r' sr' where the: tighten-poly-bounds-epsilon p (1 / 2) l r (sgn (ipoly p r)) = (l',r',sr')
    by (cases rule: prod-cases3, auto)
  let \( \bar{fr} = \bar{fr} \)
  let \( \bar{fl} = \bar{fl} \)
  let \( \bar{fr}' = \bar{fr}' \)
  obtain l'' r'' sr'' where tba: tighten-poly-bounds-for-x p ?fr' l' r' sr' = (l'',r'',sr'')
by (cases rule: prod-cases3, auto)

note rc = assms[unfolded fields]

hence id1: invariant-1 (p,l,r) by auto

have id: floor-1 x = ((if ?fr = ?fl then ?fr
  else if ?fr' < l'' then ?fr else ?fl))
  unfolding fields floor-1.simps the Let-def split tbx by simp

let ?x = real-of-rat x

have x: ?x = the-unique-root (p,l,r) unfolding fields by simp

have bnd: l ≤ l' r' ≤ r r' − l' ≤ 1 / 2
  and rc': root-cond (p, l', r') (the-unique-root (p, l, r))
  and sr': sr' = sgn (ipoly p r')
  by (atomize(full), intro conjI tighten-poly-bounds-epsilon[OF - - the refl], insert
  rc,auto elim!: invariant-1E)

let ?r = real-of-rat

from rc'[folded x, unfolded split]

have ineq: ?r l' ≤ ?x ?x ≤ ?r r' ?r l' ≤ ?r r' by auto

hence br': l' ≤ r' unfolding of-rat-less-eq by simp

have frl: ?fl ≤ ?fr
  by (rule floor-monotone[OF br'])

from invariant-1-sub-interval[OF rc1 rc' bnd(1,2)]

have rc': invariant-1 (p, l', r')
  and id': the-unique-root (p, l', r') = the-unique-root (p, l, r) by auto

with rc have rc2': invariant-1-2 (p, l', r') by auto

have x: ?x = the-unique-root (p,l',r')
  unfolding fields using id' by simp

{ assume ?fr ≠ ?fl
  with frl have frl: ?fl ≤ ?fr − 1 by simp
  have ?fr' ≤ r' l' ≤ ?fr using frl bnd by linarith+
}

note fl-diff = this

show ?thesis

proof (cases ?fr = ?fl)

  case True

  hence id1: floor-1 x = ?fr unfolding id by auto

  from True have id: floor (?r l') = floor (?r r')
    by simp

  have floor ?x ≤ floor (?r r')
    by (rule floor-monotone[OF ineq(2)])

  moreover have floor (?r l') ≤ floor ?x
    by (rule floor-monotone[OF ineq(1)])

  ultimately have floor ?x = floor (?r r')

  unfolding id by (simp add: id)

  then show ?thesis by (simp add: id1)

next

  case False

  with id have id: floor-1 x = (if ?fr' < l'' then ?fr else ?fl) by simp

  from rc2' have unique-root (p,l',r') poly-cond2 p by auto

  from tighten-poly-bounds-for-x[OF this tbx sr']
  have ineq': l' ≤ l'' r'' ≤ r' and lr': l'' ≤ r'' and rc'': root-cond (p,l'',r'') ?x
and \( fr' : (l'' \leq r'') \) unfolding \( x \) by \( auto \)

from \( rec''[\text{unfolded split}] \)

have \( ineq''(2) \) \( r' \leq x \leq r'' \) by \( auto \)

from \( False \) have \( fr \neq fl \) by \( auto \)

note \( fr = fl-diff[\text{OF this}] \)

show \( ?thesis \)

proof (cases \( fr' < l'' \))

  case True
  with \( id \) have \( id: \text{floor-1} \) \( x = fr \) by \( simp \)
  have \( floor \) \( ?x \leq fr \) using \( \text{floor-mono}[\text{OF ineq(2)}] \) by \( simp \)
  moreover from \( True \) have \( \text{floor-rat-less} \) unfolding \( of-rat-less \).

  with \( ineq''(1) \) have \( ?r \) \( fr' \leq ?x \) by \( simp \)

  from \( \text{floor-mono}[\text{OF this}] \)
  have \( fr \leq \text{floor} \) \( ?x \) by \( simp \)

  ultimately show \( ?thesis \) unfolding \( id \) by \( auto \)

next

  case False
  with \( id \) have \( id: \text{floor-1} \) \( x = fl \) by \( simp \)
  from \( False \) have \( l'' \leq fr' \) by \( auto \)

  from \( \text{floor-mono}[\text{OF ineq(1)}] \)
  have \( fl \leq \text{floor} \) \( ?x \) by \( simp \)

  moreover have \( \text{floor} \) \( ?x \leq fl \)

proof (cases)

  from \( False \) \( fr' \) have \( fr' : r'' < fr' \) by \( auto \)

  hence \( \text{floor} \) \( r''' < fr \) by \( linarith \)

  with \( \text{floor-mono}[\text{OF ineq'(2)}] \)
  have \( \text{floor} \) \( ?x \leq fr - 1 \) by \( auto \)

  also have \( \text{floor} \) \( r' - 1 = \text{floor} \) \( (r' - 1) \) by \( simp \)

  also have \( \ldots \leq fl \)

  by (rule \( \text{floor-mono} \), insert \( \text{bnd} \), \( \text{auto} \))

  finally show \( ?thesis \).

qed

ultimately show \( ?thesis \) unfolding \( id \) by \( auto \)

qed

8.2.9 Generic Factorization and Bisection Framework

lemma card-1-Collect-ex1: assumes \( \text{card} (\text{Collect} \ P) = 1 \)

shows \( \exists! \) \( x \) \( \ P \) \( x \)

proof (cases)

  from \( \text{assms}[\text{unfolded card-eq-1-iff}] \) obtain \( x \) where \( \text{Collect} \ P = \{ x \} \) by \( auto \)

  thus \( ?thesis \)

  by (intro \( \text{ex1I}[\text{of - x}] \), \( \text{auto} \))

qed

fun sub-interval :: rat \times rat \Rightarrow rat \times rat \Rightarrow bool where

sub-interval \( (l,r) \) \( (l',r') \) = \( (l' \leq l \land r \leq r') \)

133
fun in-interval :: rat × rat ⇒ real ⇒ bool where
in-interval (l,r) x = (af-rat l ≤ x ∧ x ≤ af-rat r)

definition converges-to :: (nat ⇒ rat × rat) ⇒ real ⇒ bool where
converges-to f x ≡ (∀ n. in-interval (f n) x ∧ sub-interval (f (Suc n)) (f n)) ∧ (∀ (eps :: real) > 0. ∃ n l r. f n = (l,r) ∧ af-rat r − af-rat l ≤ eps)

custom
fixes bnd-update :: 'a ⇒ 'a
and bnd-get :: 'a ⇒ rat × rat

begin

definition at-step :: (nat ⇒ rat × rat) ⇒ nat ⇒ 'a ⇒ bool where
at-step f n a ≡ ∀ i. bnd-get (((bnd-update ^^ i) a) = f (n + i))

partial-function (tailrec) select-correct-factor-main
:: 'a ⇒ (int poly × root-info)list ⇒ (int poly × root-info)list
⇒ nat ⇒ nat ⇒ nat ⇒ (int poly × root-info) × rat × rat where
[code]: select-correct-factor-main bnd todo old l r n = (case todo of Nil ⇒ if n = 1 then (hd old, l, r) else let bnd’ = bnd-update bnd in (case bnd-get bnd’ of (l,r) ⇒
 select-correct-factor-main bnd’ old [] l r 0)
| Cons (p,ri) todo ⇒ let m = root-info.l-r ri l r in
if m = 0 then select-correct-factor-main bnd todo old l r n
else select-correct-factor-main bnd todo ((p,ri) # old) l r (n + m))

definition select-correct-factor :: 'a ⇒ (int poly × root-info)list ⇒
(int poly × root-info) × rat × rat where
select-correct-factor init polys = (case bnd-get init of (l,r) ⇒
select-correct-factor-main init polys [] l r 0)

lemma select-correct-factor-main: assumes conv: converges-to f x
and at: at-step f i a
and res: select-correct-factor-main a todo old l r n = ((q,ri-fin),(l-fin,r-fin))
and bnd: bnd-get a = (l,r)
and r0: ∃ q ri. (q,ri) ∈ set todo ∪ set old ⇒ root-info-cond ri q
and q0: ∃ q ri. (q,ri) ∈ set todo ∪ set old ⇒ q ≠ 0
and ex: ∃ q. q ∈ fst set todo ∪ fst set old ∧ ipoly q x = 0
and dist: distinct (map fst (todo @ old))
and old: ∃ q ri. (q,ri) ∈ set old ⇒ root-info.l-r ri l r ≠ 0
and old: ∃ q ri. (q,ri) ∈ set todo ⇒ root-info.l-r ri l r ≠ 0
and n: n = sum-list (map (λ (q,ri). root-info.l-r ri l r) old)
shows unique-root (q,l-fin,r-fin) ∧ (q,ri-fin) ∈ set todo ∪ set old ∧ x = the-unique-root (q,l-fin,r-fin)

proof –
define orig where orig = set todo ∪ set old
have orig: set todo ∪ set old ⊆ orig unfolding orig-def by auto
let \( \mathcal{rts} = \{ x :: \text{real}. \exists q \, r. (q,r) \in \text{orig} \land \text{ipoly} \, q \, x = 0 \} \)
define \( \text{rts} \) where \( \text{rts} = \mathcal{rts} \)
let \( \mathcal{h} = \lambda (x,y). \text{abs} \, (x - y) \)
let \( \mathcal{r} = \text{real-of-rat} \)
have \( \text{rts} : \mathcal{rts} = (\bigcup ((\lambda (q,r). \{ x. \text{ipoly} \, q \, x = 0 \}) \cdot \text{set} \, (\text{todo} @ \text{old}))) \) unfolding \( \text{orig-def} \) by auto
have finite \( \text{rts} \) unfolding \( \text{rts-def} \)
  using finite-ipoly-roots[\text{OF} \, q \theta] finite-set[\text{of} \, \text{todo} \, @ \, \text{old}] \) by auto
hence \( \text{fin} : \text{finite} \, (\text{rts} \times \text{rts} - \text{Id}) \) by auto
define \( \text{diffs} \) where \( \text{diffs} = \text{insert} \, 1 \{ \text{abs} \, (x - y) | x, y, x \in \text{rts} \land y \in \text{rts} \land x \neq y \} \)
have finite \( \{ \text{abs} \, (x - y) | x, y, x \in \text{rts} \land y \in \text{rts} \land x \neq y \} \)
  by (rule subst[\text{of} - \text{finite}, \text{OF} - \text{finite-imageI}][\text{OF} \, \text{fin}, \, \text{of} \, \mathcal{h}]\), auto
hence \( \text{diffs} : \text{finite} \, \text{diffs} \) \( \text{diffs} \neq \{ \} \) unfolding \( \text{diffs-def} \) by auto
define \( \text{eps} \) where \( \text{eps} = \text{Min} \, \text{diffs} \) / 2
have \( \land x, x \in \text{diffs} \implies x > 0 \) unfolding \( \text{diffs-def} \) by auto
with \( \text{Min-gr-iff}[\text{OF} \, \text{diffs}] \) have \( \text{eps} : \text{eps} > 0 \) unfolding \( \text{eps-def} \) by auto
note \( \text{conv} = \text{conv'[unfolded converges-to-def]} \)
from \( \text{conv} \, \text{eps} \) obtain \( N \, L \, R \) where
  \( N : \text{f} \, N = (L, R) \) \( \text{?r} \, R - \text{?r} \, L \leq \text{eps} \) by auto
obtain \( \text{pair where} \, \text{pair} = (\text{todo}, \text{i}) \) by auto
define \( \text{rel} \) where \( \text{rel} = \text{measures} \, [\lambda (t, i). \, N - i, \, \lambda (t :: (\text{int poly} \times \text{root-info}) \, \text{list}, i). \, \text{length} \, t] \)
have \( \text{wf} : \text{wf} \, \text{rel} \) unfolding \( \text{rel-def} \) by simp
show \( \mathcal{thesis} \)
  using \( \text{at} \, \text{res} \, \text{bnd} \, \text{ri} \, q \theta \) \( \text{ex} \) \( \text{dist} \) \( \text{old} \) \( \text{un} \) \( \text{n} \) \( \text{pair} \) \( \text{orig} \)
proof (induct \( \text{pair} \) \( \text{arbitrary} \); \( \text{todo} \) \( i \) \( \text{old} \) \( \text{a} \) \( \text{l} \) \( \text{r} \) \( \text{n} \) rule: \( \text{wf-induct}[\text{OF} \, \text{wf}] \))
case (\( \text{1} \) \( \text{pair} \) \( \text{todo} \) \( i \) \( \text{old} \) \( \text{a} \) \( \text{l} \) \( \text{r} \) \( \text{n} \))
  note \( \text{IH} = \text{i} (1) [\text{rule-format}] \)
  note \( \text{at} = \text{i} (2) \)
  note \( \text{res} = \text{i} (3) [\text{unfolded select-correct-factor-main} \, \text{simps}[\text{of} \, \text{-} \, \text{todo}]] \)
  note \( \text{bnd} = \text{i} (4) \)
  note \( \text{ri} = \text{i} (5) \)
  note \( \text{q} \theta = \text{i} (6) \)
  note \( \text{ex} = \text{i} (7) \)
  note \( \text{dist} = \text{i} (8) \)
  note \( \text{old} = \text{i} (9) \)
  note \( \text{un} = \text{i} (10) \)
  note \( \mathcal{n} = \text{i} (11) \)
  note \( \text{pair} = \text{i} (12) \)
  note \( \text{orig} = \text{i} (13) \)
from \( \text{at}[\text{unfolded at-step-def}, \text{rule-format}, \text{of} \, \theta] \) \( \text{bnd} \) have \( \text{fi} : \, \text{fi} = (\text{l}, \text{r}) \) by auto
with \( \text{conv} \) have \( \text{inx} : \text{in-interval} \, (\text{f} \, \text{i}) \, \text{x} \) by blast
hence \( \text{lkr} : \, \text{?r} \, \text{l} \leq \, \text{x} \leq \, ?r \, \text{r} \) unfolding \( \text{fi} \) by auto
from \( \text{order-trans}[\text{OF} \, \text{this}] \) have \( \text{lr} : \, \text{l} \leq \, \text{r} \) unfolding \( \text{of-rat-less-eq} \)
  show \( \text{?case} \)
proof (cases \( \text{todo} \))
case (\( \text{Cons} \, \text{rrr} \) \( \text{tod} \))
obtain \( s \, \text{ri} \) where \( \text{rrr} \) : \( \text{rrr} = (s, \text{ri}) \) by force
with Cons have todo: todo = (s, ri) # tod by simp
note res = res[unfolded todo list.simps split Let-def]
from root-info-condD(1)[OF ri[of s ri, unfolded todo] lr]
have ri': root-info.l-r ri l r = card \{x. root-cond (s, l, r) x\} by auto
from q0 have s0: s ≠ 0 unfolding todo by auto
from finite-ipoly-roots[OF s0] have fins: finite \{x. root-cond (s, l, r) x\}
  unfolding root-cond-def by auto
have rel: ((tod, i), pair) ∈ rel unfolding rel-def pair todo by simp
show ?thesis
proof (cases root-info.l-r ri l r = 0)
  case True
    with res have res: select-correct-factor-main a tod old l r n = ((q, ri-fin),
l-fin, r-fin) by auto
    from ri'[symmetric, unfolded True] fins have empty: \{x. root-cond (s, l, r) x\}
    x = {} by simp
    from ex lxr empty have ex': (∃ q, q ∈ fst ' set tod ∪ fst ' set old ∧ ipoly q x
    = 0)
    unfolding todo root-cond-def split by auto
    have unique-root (q, l-fin, r-fin) ∈ set tod ∪ set old ∧
x = the-unique-root (q, l-fin, r-fin)
    proof (rule IH[OF rel at res bnd ri - ex' - - n refl], goal-cases)
      case (5 y) thus ?thesis using un[of y] unfolding todo by auto
    next
      case 2 thus ?thesis using q0 unfolding todo by auto
    qed (insert dist old orig, auto simp: todo)
    thus ?thesis unfolding todo by auto
  next
  case False
    with res have res: select-correct-factor-main a tod ((s, ri) ≠ old) l r
    (n + root-info.l-r ri l r) = ((q, ri-fin), l-fin, r-fin) by auto
    from ex have ex': (∃ q, q ∈ fst ' set tod ∪ fst ' set ((s, ri) ≠ old) ∧ ipoly q
    x = 0)
    unfolding todo by auto
    from dist have dist: distinct (map fst (tod ⊕ (s, ri) ≠ old)) unfolding
todo by auto
    have id: set todo ∪ set old = set tod ∪ set ((s, ri) ≠ old) unfolding todo
    by simp
    show ?thesis unfolding id
    proof (rule IH[OF rel at res bnd ri - ex' dist], goal-cases)
      case 4 thus ?thesis using un unfolding todo by auto
    qed (insert old False orig, auto simp: q0 todo n)
    qed
  next
  case Nil
  note res = res[unfolded Nil list.simps Let-def]
  from ex[unfolded Nil] lxr obtain s where s ∈ fst ' set old ∧ root-cond (s,l,r)
x
  unfolding root-cond-def by auto
  then obtain q1 ri1 old' where old': old = (q1,ri1) ≠ old' using id by (cases
old, auto)
let \( \varphi_i = \text{root-info.l-r} \) ri1 l r
from \( \text{old[unfolded old]} \) have \( \theta \); \( \varphi_i \neq 0 \) by auto
from \( \text{n[unfolded old]} \) \( \theta \) have \( n; n \neq 0 \) by auto
from \( \text{ri[unfolded old]} \) have \( \varphi_i \); root-info-cond ri1 q1 by auto
show \( \theta \)thesis
proof (cases \( n = 1 \))
case False
with \( n;0 \) have \( n; n > 1 \) by auto
obtain \( l' r' \) where \( \text{bind'} \); bind-get (bind-update a) = \( (l',r') \) by force
with \( \text{res False} \) have \( \text{res: select-correct-factor-main} \) \( \text{(bind-update a)} \) \( \text{old} \) \( l' \)
proof (cases \( n \))
case \( 1 n \)
have \( \text{id: (bind-update} \; \text{Suc n}) \) \( a = \text{(bind-update} \; \text{Suc n}) \) by (induct \( n \), auto)
from \( \text{at[unfolded at-step-def, rule-format, of Suc n]} \) show \( \theta \)case unfolding \( \text{id} \) by simp
qed
from \( \text{0[unfolded root-info-condD(l)[OF ri' lr]]} \) obtain \( y1 \) where \( y1 \): root-cond (q1,l,r) y1
by (cases Collect (root-cond (q1,l,r)) = \{\}, auto)
from \( \text{n1[unfolded n old]} \)
have \( \varphi_i > 1 \) \( \lor \) sum-list (map (\( \lambda \text{(q,ri)}. \text{root-info.l-r} \) ri l r) \( \text{old'} \)) \( \neq 0 \)
by (cases sum-list (map (\( \lambda \text{(q,ri)}. \text{root-info.l-r} \) ri l r) \( \text{old'} \)), auto)
hence \( \exists q2 ri2 \) \( y2 \). (q2,ri2) \( \in \) set old \( \land \) root-cond (q2,l,r) y2 \( \land \) \( y1 \neq y2 \)
proof
assume \( \varphi_i > 1 \)
with \( \text{root-info-condD(l)[OF ri' lr]} \) have \( \text{card \{x. root-cond (q1,l,r) x}\} > 1 \) by simp
from \( \text{card-gt-ID[OF this]} \) \( y1 \) obtain \( y2 \) where \( \text{root-cond (q1,l,r) y2 and y1 \neq y2 by auto}\)
thus \( \theta \)thesis unfolding \( \text{old'} \) by auto
next
assume sum-list (map (\( \lambda \text{(q,ri)}. \text{root-info.l-r} \) ri l r) \( \text{old'} \)) \( \neq 0 \)
then obtain \( q2 ri2 \) \( \text{where} \) \( \text{mem:} \) (q2,ri2) \( \in \) set old' and \( \text{ri2: root-info.l-r} \)
\( \text{ri2 l r \neq 0} \) by auto
with \( q0 ri \) have \( \text{root-info-cond ri2 q2 unfolding} \) \( \text{old'} \) by auto
from \( \text{ri2[unfolded root-info-condD(l)[OF this lr]]} \) obtain \( y2 \) where \( y2 \): root-cond (q2,l,r) y2
by (cases Collect (root-cond (q2,l,r)) = \{\}, auto)
from \( \text{dist[unfolded old] split-list[OF mem]} \) have \( \text{diff:} \) q1 \( \neq q2 \) by auto
from \( \text{y1 have q1 : q1 \in fst'} \; \text{set todo} \cup \text{fst'} \; \text{set old} \; \land \; \text{ipoly q1 y1 = 0} \)
unfolding \( \text{old'} \) root-cond-def by auto
from \( \text{y2 have q2 : q2 \in fst'} \; \text{set todo} \cup \text{fst'} \; \text{set old} \; \land \; \text{ipoly q2 y2 = 0} \)
unfolding \( \text{old'} \) root-cond-def using \( \text{mem} \) by force
have \( y1 \neq y2 \)
proof
  assume id: \( y_1 = y_2 \)
  from q1 have \( \exists q_1. q_1 \in \text{fst } \text{set todo} \cup \text{fst } \text{set old} \land \text{ipoly q1 y1} = 0 \)
  by blast
  from un[\text{OF this}] q1 q2[folded id] have q1 = q2 by auto
  with diff show False by simp
  qed
  with mem y2 show ?thesis unfolding old' by auto
  qed
then obtain q2 ri2 y2 where
  mem2: \((q2,ri2) \in \text{set old} \and y2: \text{root-cond } (q2,l,r) \text{ y2 and diff: } y_1 \neq y_2\)
by auto
  from mem2 orig have \((q1,ri1) \in \text{orig } (q2,ri2) \in \text{orig } \text{unfolding old'} \by auto
  with y1 y2 diff have abs \((y_1 - y_2) \in \text{diffs unfolding diffs-def rts-def}\)
  root-cond-def by auto
  from Min-le[OF diffs(1) this] have abs \((y_1 - y_2) \geq 2 * \text{eps unfolding}\)
  eps-def by auto
  with eps have eps: \( \text{abs } (y_1 - y_2) > \text{eps by auto}\)
  from y1 y2 have r: \( \text{of-rat r} \geq \text{max y1 y2 unfolding root-cond-def by auto}\)
  from l r eps have eps: \( \text{of-rat r} - \text{of-rat l} > \text{eps by auto}\)
  have i < N
  proof (rule ccontr)
    assume \( \neg i < N\)
    hence \( \exists k. i = N + k \by presburger\)
then obtain k where i: \( i = N + k \by auto\)
  \{
  fix k l r
  assume f: \( \text{f} (N + k) = (l,r)\)
  hence of-rat r - of-rat l \leq eps
  proof (induct k arbitrary: l r)
    case 0
    with N show ?case by auto
  next
    case (Suc k l r)
    obtain l' r' where f: \( \text{f} (N + k) = (l',r') \by force\)
    from Suc(1)[\text{OF this}] have IH: \( \text{?r r' - ?r l' \leq eps by auto}\)
    from f Suc(2) conv[THEN conjunct1, rule-format, of N + k]
    have \( \text{?r l \geq ?r l' \and r \leq ?r r'}\)
    by (auto simp: of-rat-less-eq)
    thus ?case using IH by auto
  qed
  \}
ote * = this
  from at[unfolded at-step-def i, rule-format, of \emptyset] bnd have f \((N + k) = (l,r)\) by auto
  from *[\text{OF this}] eps show False by auto
  qed
hence rel: ((old, Suc i), pair) ∈ rel unfolding pair rel-def by auto
from dist have dist: distinct (map fst (old @ [])) unfolding Nil by auto
have id: set todo ∪ set old = set old ∪ set [] unfolding Nil by auto
show ?thesis unfolding id
proof (rule IH[OF rel at’ res bnd’ ri - - dist - - refl], goal-cases)
  case 2 thus ?case using q0 by auto
qed (insert ex un orig Nil, auto)
next
  case True
  with res old’ have id: q = q1 ri-fin = ri1 l-fin = l r-fin = r by auto
  from n[unfolded True old’] 0 have 1: ?ri = 1
    by (cases ?ri; cases ?ri - 1, auto)
  from root-info-condD(1)[OF ri’ lr’] 1 have card {x. root-cond (q1,l,r) x} = 1
  by auto
  from card-1-Collect-ex1[OF this]
  have unique: unique-root (q1,l,r) .
  from ex[unfolded Nil old’] consider (A) ipoly q1 x = 0
    | (B) q where q ∈ fst ‘ set old’ ipoly q x = 0 by auto
  hence x = the-unique-root (q1,l,r)
proof (cases)
  case A
  with br have root-cond (q1,l,r) x unfolding root-cond-def by auto
  from the-unique-root-eqI[OF unique this] show ?thesis by simp
next
  case (B q)
  with br have root-cond (q,l,r) x unfolding root-cond-def by auto
  hence empty: {x. root-cond (q,l,r) x} ≠ {} by auto
  from B(l) obtain ri’ where mem: (q,ri’) ∈ set old’ by force
  from q0[unfolded old’] mem have q0: q ≠ 0 by auto
  from finite-ipoly-roots[OF this] have finite {x. root-cond (q,l,r) x}
  unfolding root-cond-def by auto
  with empty have card: card {x. root-cond (q,l,r) x} ≠ 0 by simp
  from ri[unfolded old’] mem have root-info-cond ri’ q by auto
  from root-info-condD(1)[OF this lr’] card have root-info.l-r ri’ l r ≠ 0 by auto
  then with n[unfolded True old’] 1 split-list[OF mem] have False by auto
  thus ?thesis by simp
qed
thus ?thesis unfolding id using unique ri’ unfolding old’ by auto
qed
qed

lemma select-correct-factor: assumes
  res: select-correct-factor init polys = ((q,ri),(l,r))
  and ri: ∀ q. ri. (q,ri) ∈ set polys ⇒ root-info-cond ri q
  and q0: ∀ q. ri. (q,ri) ∈ set polys ⇒ q ≠ 0

139
and ex: \( \exists q. q \in \text{fst } \cdot \text{set polys } \land \text{ipoly } q x = 0 \)
and dist: distinct (map \text{fst } \text{polys})
and un: \( \forall x :: \text{real}. (\exists q. q \in \text{fst } \cdot \text{set polys } \land \text{ipoly } q x = 0) \implies \\
\exists q. q \in \text{fst } \cdot \text{set polys } \land \text{ipoly } q x = 0 \)
shows unique-root \((q,l,r) \land (q,r) \in \text{set polys } \land x = \text{the-unique-root } (q,l,r)\)

proof –
obtain \( l' \ r' \) where 
init: \( \text{bnd-get init } = (l',r') \) by force
from res[unfolded \text{select-correct-fact-o-main } \text{def split}]
have res: \( \text{select-correct-factor-main init polys } \land \text{th \text{e-bnd-update } ^{\text{-- i}} \text{init} } \) \( 0 \) init unfolding at-step-def by auto
have at: at-step (\( \lambda i. \text{bnd-get } (\text{bnd-update } ^{\text{-- i}} \text{init}) ) \) \( 0 \) init unfolding at-step-def by auto
have unique-root \((q,l,r) \land (q,r) \in \text{set polys } \cup \text{set } x = \text{the-unique-root } (q,l,r)\)
by (rule select-correct-factor-main[\( \text{OF conv at res init } ri \], insert dist un ex q0, auto)
thus \( \exists \text{thesis } \) by auto

qed

definition real-alg-2': \( \text{root-info } \Rightarrow \text{int poly } \Rightarrow \text{rat } \Rightarrow \text{rat } \Rightarrow \text{real-alg-2 } \text{where} \)
\[ \text{code def: real-alg-2'} \ \text{ri p l r } = ( \)
\[ \begin{align*}
\text{if degree } p = 1 \text{ then } \text{Rational } (\text{Rat.Frac } (\text{coeff } p 0) (\text{coeff } p 1)) \text{ else } \\
\text{real-alg-2-main } ri \text{ (case tighten-poly-bounds-for-x p 0 l r (sgn ipoly p r)) of } \\
(l',r',sr') \Rightarrow (p, l', r'))
\end{align*} \]

lemma real-alg-2'-code[\text{code}: real-alg-2' \text{ri p l r } = ( \)
\[ \begin{align*}
\text{if degree } p = 1 \text{ then } \text{Rational } (\text{Rat.Frac } (\text{coeff } p 0) (\text{coeff } p 1)) \text{ else case normalize-bounds-1 } \\
\text{case tighten-poly-bounds-for-x p 0 l r (sgn ipoly p r)) of } (l',r',sr') \Rightarrow (p, l', r'))
\end{align*} \]

unfolding real-alg-2'-def real-alg-2-main-def by (cases tighten-poly-bounds-for-x p 0 l r (sgn ipoly p r)), simp add: Let-def

definition real-alg-2'': \( \text{root-info } \Rightarrow \text{int poly } \Rightarrow \text{rat } \Rightarrow \text{rat } \Rightarrow \text{real-alg-2 } \text{where} \)
\[ \text{real-alg-2'} \ \text{ri p l r } = ( \text{case normalize-bounds-1 } \\
\text{case tighten-poly-bounds-for-x p 0 l r (sgn ipoly p r)) of } (l',r',sr') \Rightarrow (p, l', r'))
\]

of \((p', l, r) \Rightarrow \text{Irrational } (\text{root-info.number-root } ri r) (p', l, r)) \]

unfolding real-alg-2'-code real-alg-2''-def by auto

lemma poly-cond-degree-0-imp-no-root:

fixes \( x :: \text{'b } \cdot \{ \text{comm-ring-1, ring-char-0} \} \)
assumes pc: \( \text{poly-cond } p \text{ and } \text{deg } : \text{degree } p = 0 \) shows \( \text{ipoly } p x \neq 0 \)

proof
from pc have \( p \neq 0 \) by auto
moreover assume \( \text{ipoly } p x = 0 \)

note \( \text{poly-zero [OF this]} \)
ultimately show \( \text{False } \) using \( \text{deg } \) by auto

140
qed

**lemma** real-alg-2':

**assumes** ur: unique-root (q,l,r) and pc: poly-cond q and ri: root-info-cond ri q

**shows** invariant-2 (real-alg-2' ri q l r) ∧ real-of-2 (real-alg-2' ri q l r) = the-unique-root (q,l,r) (is - ∧ - = ?x)

**proof** (cases degree q Suc 0 rule: linorder-cases)

**case** deg: less

then have degree q = 0 by auto

from poly-cond-degree-0-imp-no-root[OF pc this] ur have False by force

then show ?thesis by auto

next

**case** deg: equal

hence id: real-alg-2' ri q l r = Rational (Rat.Fract (- coeff q 0) (coeff q 1))

unfolding real-alg-2'-def by auto

show ?thesis unfolding id using degree-1-ipoly[OF deg]

using unique-rootD(4)(OF ur) by auto

next

**case** deg: greater

with pc have pc2: poly-cond2 q by auto

let ?rai = real-alg-2' ri q l r

let ?r = real-of-rat

obtain l' r' sr' where tight: tighten-poly-bounds-for-x q 0 l r (sgn (ipoly r q)) = (l',r',sr')

by (cases rule: prod-cases3, auto)

let ?rai' = (q, l', r')

have rai': ?rai = real-alg-2-main ri ?rai'

unfolding real-alg-2'-def using deg tight by auto

hence rai': ?rai' = the-unique-root (q,l',r') by auto

note tight = tighten-poly-bounds-for-x[OF ur pc2 tight refl]

let ?x = the-unique-root (q, l, r)

from tight have tight: root-cond (q,l',r') ?x l l' l' ≤ l' l l' r' l' < r' l r > 0 ∨ r' < 0 by auto

from unique-root-sub-interval[OF ur tight(1) tight(2,4)] poly-condD[OF pc]

have ur': unique-root (q, l', r') and x: ?x = the-unique-root (q,l',r') by auto

from tight(2-) have sgn: sgn l' = sgn r' by auto


by (auto simp: tight(1) sgn pc ri ur')

qed

**definition** select-correct-factor-int-poly :: 'a ⇒ int poly ⇒ real-alg-2 where

select-correct-factor-int-poly init p ≡

let qs = factors-of-int-poly p;

polys = map (λ q, (q, root-info q)) qs;

((q,ri),(l,r)) = select-correct-factor init polys

in real-alg-2' ri q l r

**lemma** select-correct-factor-int-poly: assumes
conv: converges-to (λ i. bnd-get ((bnd-update ~ i) init)) x
and rai: select-correct-factor-int-poly init p = rai
and x: ipoly p x = 0
and p: p ≠ 0
shows invariant-2 rai ∧ real-of-2 rai = x

proof –

obtain qs where fact: factors-of-int-poly p = qs by auto
define polys where polys = map (λ q. (q, root-info q)) qs
obtain q r i l r where res: select-correct-factor init polys = ((q,ri),(l,r))

by (cases select-correct-factor init polys, auto)

have fps: map fps polys = qs fps ∧ set polys = set qs unfolding polys-def map-map o-def

by force+

note fact' = factors-of-int-poly[OF fact]

note rai = rai[unfolded select-correct-factor-int-poly-def Let-def fact,
folded polys-def, unfolded res split]
from fact' fps have dist: distinct (map fps polys) by auto
from fact'(2)[OF p, of x] x fps

have ex: ∃ q. q ∈ fps ∧ set polys ∧ ipoly q x = 0 by auto

{ fix q r
assume (q,ri) ∈ set polys

hence ri: ri = root-info q and q: q ∈ set qs unfolding polys-def by auto
from fact'(1)[OF q] have *: lead-coeff q > 0 irreducible q degree q > 0 by auto
from * have q0: q ≠ 0 by auto
from root-info[OF *1(2−3)] ri have ri: root-info-cond ri q by auto
note ri q0 *

} note polys = this

have unique-root (q, l, r) ∧ (q, ri) ∈ set polys ∧ x = the-unique-root (q, l, r)

by (rule select-correct-factor[OF polys res polys(1)] ex dist, unfolded fps, OF -
- fact'(3)[OF p]],
insert fact'(2)[OF p] polys(2) auto)

hence ur: unique-root (q,l,r) and mem: (q,ri) ∈ set polys and x: x = the-unique-root
(q,l,r) by auto

note polys = polys[OF mem]

from polys(3−4) have ty: poly-cond q by (simp add: poly-cond-def)

show ?thesis unfolding x rai[ symmetric] by (intro real-alg-2' ur ty polys(1))

qed

end

8.2.10 Addition

lemma ipoly-0-0[simp]: ipoly f (θ::'a::{comm-ring-1,ring-char-0}) = 0 ↔ poly f = 0

unfolding poly-0-coeff-0 by simp

lemma add-rat-roots-below[simp]: roots-below (poly-add-rat r p) x = (λy. y + of-rat r) * roots-below p (x - of-rat r)

proof (unfold add-rat-roots image-def; intro Collect-eqI, goal-cases)
case (1 y) then show ?case by (auto intro: exI[of - y - real-of-rat r])
qed

lemma add-rat-root-cond:
  shows root-cond (cf-pos-poly (poly-add-rat m p), l, r) x = root-cond (p, l - m, r - m) (x - of-rat m)
  by (unfold root-cond-def, auto simp add: add-rat-roots hom-distribs)

lemma add-rat-unique-root: unique-root (cf-pos-poly (poly-add-rat m p), l, r) = unique-root (p, l - m, r - m)
  by (auto simp: add-rat-root-cond)

fun add-rat-1 :: rat ⇒ real-alg-1 ⇒ real-alg-1 where
  add-rat-1 r1 (p2, l2, r2) = (let p = cf-pos-poly (poly-add-rat r1 p2);
                               (l, r, sr) = tighten-poly-bounds-for-x p 0 (l2 + r1) (r2 + r1) (sgn (ipoly p (r2 + r1))))
                              in (p, l, r))

lemma poly-real-alg-1-add-rat[simp]:
  poly-real-alg-1 (add-rat-1 r y) = cf-pos-poly (poly-add-rat r (poly-real-alg-1 y))
  by (cases y, auto simp: Let-def split prod.split)

lemma sgn-cf-pos:
  assumes lead-coeff p > 0 shows sgn (ipoly (cf-pos-poly p) (x::′a::linordered-field)) = sgn (ipoly p x)
  proof (cases p = 0)
    case True with assms show ?thesis by auto
  next
    case False from cf-pos-poly-main False obtain d :: Polynomial.smult d (cf-pos-poly p) = p by auto
    have d > 0
      proof (rule zero-less-mult-pos2)
        from False assms have 0 < lead-coeff p by (auto simp: cf-pos-def)
        also from p' have ... = d * lead-coeff (cf-pos-poly p) by (metis lead-coeff-smult)
        finally show 0 < .... using False by (unfold lead-coeff-cf-pos-poly)
      qed
    moreover from p' have ipoly p x = of-int d * ipoly (cf-pos-poly p) x
      by (fold poly-smult of-int-hom.map-poly-hom-smult, auto)
    ultimately show ?thesis by (auto simp: sgn-mult[where 'a='a])
  qed

lemma add-rat-1: fixes r1 :: rat assumes inv-y: invariant-1-2 y
  defines z ≡ add-rat-1 r1 y
  shows invariant-1-2 z ∧ (real-of-1 z = of-rat r1 + real-of-1 y)
  proof (cases y)
    case y-def: (fields p2 l2 r2)
\begin{verbatim}
define p where p ≡ cf-pos-poly (poly-add-rat r1 p2)

obtain l r sr where br: tighten-poly-bounds-for-x p θ (l2+r1) (r2+r1) (sgn (ipoly p (r2+r1))) = (l, r, sr)
  by (metis surj-pair)

from br have z: z = (p,l,r) by (auto simp: y-def z-def p-def Let-def)

from inv-y have ur: unique-root (p, l2 + r1, r2 + r1)
  by (auto simp: add-rat-unique-root)

from inv-y[unfolded y-def invariant-1-2-def, simplified] have pc2: poly-cond2 p

unfolding p-def
apply (intro poly-cond2I poly-add-rat-irreducible poly-condI, unfold lead-coeff cf-pos-poly)
apply (auto elim!: invariant-1E)
done

note main = tighten-poly-bounds-for-x[OF ur pc2 br refl, simplified]
then have sgn l = sgn r unfolding sgn-if apply simp apply linarith done
from invariant-1-2-reall[OF main(4) - main(7), simplified, OF this pc2] main(1-3)
ur
show ?thesis by (auto simp: z p-def y-def add-rat-root-cond ex1-the-shift)
qed

fun tighten-poly-bounds-binary :: int poly ⇒ int poly ⇒ (rat × rat × rat) × rat × rat ⇒
  (rat × rat × rat) × rat × rat × rat where
tighten-poly-bounds-binary cr1 cr2 ((l1, r1, sr1), (l2, r2, sr2)) =
  (tighten-poly-bounds cr1 l1 r1 sr1, tighten-poly-bounds cr2 l2 r2 sr2)

lemma tighten-poly-bounds-binary:
  assumes ur: unique-root (p1,l1,r1) unique-root (p2,l2,r2) and pt: poly-cond2 p1 poly-cond2 p2
  defines x ≡ the-unique-root (p1,l1,r1) and y ≡ the-unique-root (p2,l2,r2)
  assumes bnd: ∩ l1 r1 l2 r2 l v sr1 sr2. I l1 \l=\ r l2 \r=\ root-cond (p1,l1,r1) x
  \r=\ root-cond (p2,l2,r2) y
  → bnd ((l1,r1,sr1),(l2,r2,sr2)) = (l,r) \r=\ of-rat l ≤ f x y ∧ f x y ≤ of-rat r
  and approx: ∩ l1 r1 l2 r2 l' v l' r' s' r' s'. l v r s l v r s l \r=\ root-cond (p1,l1,r1) x
  \r=\ root-cond (p2,l2,r2) y
  \r=\ root-cond (p1,l1,r1) x
  \r=\ root-cond (p2,l2,r2) y
  \l=\ I l1 \l=\ I l2 \l=\ I l1 \l=\ I l2
  \l=\ l \v l \v r \v s \v l \v r \v s
  and: I l1 l2
  and sr: sr1 = sgn (ipoly p1 r1) sr2 = sgn (ipoly p2 r2)
  shows converges-to (λ i. bnd ((tighten-poly-bounds-binary p1 p2 ~ i) ((l1,r1,sr1),(l2,r2,sr2))))
  (f x y)

proof -
  let ?upd = tighten-poly-bounds-binary p1 p2
  define upd where upd = ?upd
  define init where init = ((l1, r1, sr1), l2, r2, sr2)
  let ?g = (λ i. bnd ((upd ~ i) init))

end
\end{verbatim}
obtain \(l r\) where \(\text{bnd-init: bnd init} = (l, r)\) by force

note \(\text{ur1 = unique-rootD}[\text{OF ur}(1)]\)

note \(\text{ur2 = unique-rootD}[\text{OF ur}(2)]\)

from \(\text{ur1}(4)\) \(\text{ur2}(4)\) x-def y-def

have \(\text{rc1: root-cond \ ((p1,l1,r1) \ x) \ and \ rc2: root-cond \ ((p2,l2,r2) \ y)\ by \ auto}\)

define \(g\) where \(g = ?y\)

\[
\begin{align*}
\text{fix } i & L1 R1 L2 R2 L R \ j SR1 SR2 \\
\text{assume } ((\text{upd } \sim i)) & \text{ init } = ((L1,R1,SR1),(L2,R2,SR2)) \ g \ i = (L,R) \\
\text{hence } I & L1 \land I L2 \land \text{root-cond } ((p1,l1,r1) \ x) \land \text{root-cond } ((p2,l2,r2) \ y) \land \\
\text{unique-root } & ((p1, l1, R1) \land \text{unique-root } (p2, l2, R2) \land \text{in-interval } (L, R) (f x) \\
& \land SR1 = \text{sgn } (\text{ipoly } p1 R1) \land SR2 = \text{sgn } (\text{ipoly } p2 R2) \\
\text{proof } & \text{(induct } i \text{ arbitrary: } L1 R1 L2 R2 L R \ j SR1 SR2) \\
\text{case } 0 \\
\text{thus } & \text{case using } I \text{ rc1 rc2 ur } \text{bnd[of l1 l2 r1 r2 sr1 sr2 L R] g-def sr unfolding} \\
\text{init-def by auto} \\
\text{next} \\
\text{case } & (\text{Suc } i) \\
\text{obtain } & l1 r1 l2 r2 sr1 sr2 where \text{updi( upd } \sim i) \text{ init } = ((l1, r1, sr1), (l2, r2, sr2)) \ by \ (\text{cases } (\text{upd } \sim i) \text{ init, auto}) \\
\text{obtain } & l r \text{ where } \text{bndi: bnd } ((l1, r1, sr1), (l2, r2, sr2)) = (l, r) \text{ by force} \\
\text{hence } & gi : g \ i = (l, r) \text{ using updi unfolding g-def by auto} \\
\text{have } & (\text{upd } \sim \text{Suc } i) \text{ init } = \text{upd } ((l1, r1, sr1), (l2, r2, sr2)) \text{ using updi by simp} \\
\text{from } & \text{Suc(2)}[\text{unfolded this}] \text{ have } \text{upd } ((l1, r1, sr1), (l2, r2, sr2)) = ((L1, R1, SR1), (L2, R2, SR2)) . \\
\text{from } & \text{upd updi Suc(3) have } \text{bndsi: bnd } ((l1, R1, SR1), (L2, R2, SR2)) = (L,R) \\
\text{by } & (\text{auto simp: g-def}) \\
\text{from } & \text{Suc(1)}[\text{OF upd gi}] \text{ have } I : I1 I1 I2 \\
\text{and } & rc: \text{root-cond } ((p1,l1,r1) \ x) \text{ root-cond } ((p2,l2,r2) \ y) \\
\text{and } & ur: \text{unique-root } ((p1, l1, r1) \text{ unique-root } (p2, l2, r2) \\
\text{and } & sr: lr1 = \text{sgn } (\text{ipoly } p1 r1) \ sr2 = \text{sgn } (\text{ipoly } p2 r2) \\
\text{by } & \text{auto} \\
\text{from } & \text{upd[unfolded upd-def]} \\
\text{have } & \text{tight: tighten-poly-bounds } p1 l1 r1 sr1 = ((L1, R1, SR1) \text{ tighten-poly-bounds} \ p2 l2 r2 sr2 = (L2, R2, SR2) \\
\text{by } & \text{auto} \\
\text{note } & \text{tight1 = tighten-poly-bounds[OF tight}(1) \text{ ur}(1) \text{ pt}(1) \text{ sr}(1))] \\
\text{note } & \text{tight2 = tighten-poly-bounds[OF tight}(2) \text{ ur}(2) \text{ pt}(2) \text{ sr}(2))] \\
\text{from } & \text{tight1 have } \text{ br1: l1 } \leq r1 \text{ by auto} \\
\text{from } & \text{tight2 have } \text{ br2: l2 } \leq r2 \text{ by auto} \\
\text{note } & \text{ur1 = unique-rootD}[\text{OF ur}(1)] \\
\text{note } & \text{ur2 = unique-rootD}[\text{OF ur}(2)] \\
\text{from } & \text{tight1 I-mono[OF I(1)] have } I : I1 I1 \text{ by auto} \\
\text{from } & \text{tight2 I-mono[OF I(2)] have } I : I2 I2 \text{ by auto} \\
\text{note } & \text{ur1 = unique-root-sub-interval[OF ur}(1) \text{ tight1}(1,2,4)] \\
\text{note } & \text{ur2 = unique-root-sub-interval[OF ur}(2) \text{ tight2}(1,2,4)]
from rc(1) ur ur1 have x: x = the-unique-root (p1,L1,R1) by (auto intro:the-unique-root-eqI)
from rc(2) ur ur2 have y: y = the-unique-root (p2,L2,R2) by (auto intro:the-unique-root-eqI)
from unique-root[OF ur1(l)] x have x: root-cond (p1,L1,R1) x by auto
from unique-root[OF ur2(l)] y have y: root-cond (p2,L2,R2) y by auto
from tight(1) have half1: (L1, RI) ∈ {(l1, (l1 + r1) / 2), ((l1 + r1) / 2, r1)} unfolding tighten-poly-bounds-def Let-def by (auto split: if-splits)
from tight(2) have half2: (L2, R2) ∈ {(l2, (l2 + r2) / 2), ((l2 + r2) / 2, r2)} unfolding tighten-poly-bounds-def Let-def by (auto split: if-splits)
from approx[OF I br1 br2 bndi[symmetric] bndsi[symmetric] half1 half2] have R - L ≤ 3 / 4 * (r - l) ∧ l ≤ L ∧ R ≤ r . hence sub-interval (g (Suc i)) (g i) R - L ≤ 3/4 * (snd (g i) - fst (g i)) unfolding g Suc(3) by auto
with bnd[OF I I I2 x y bndsi] show ?thesis using I1 I2 x y ur1 ur2 tight1(6) tight2(6) by auto qed
} note invariants = this
define L where L = (λ i. fst (g i))
define R where R = (λ i. snd (g i))
{
  fix i
  obtain l1 r1 l2 r2 sr1 sr2 where upd: (upd ^^ i) init = ((l1, r1, sr1), l2, r2, sr2) by (cases (upd ^^ i) init, auto)
  obtain l r where bnd': bnd ((l1, r1, sr1), l2, r2, sr2) = (l,r) by force
  have gi: g i = (l,r) unfolding g-def upd bnd' by auto
  hence id: l = L i r = R i unfolding L-def R-def by auto
  from invariants[OF upd gi[unfolded id]] have in-interval (L i, R i) (f x y) \ j. i = Suc j ⊢ suub-interval (g i) (g j) (g j) ∧ R i - L i ≤ 3 / 4 * (R j - L j) unfolding L-def R-def by auto
} note * = this
{
  fix i
  from *(I1)[of i] *(2)[of Suc i, OF refl] have in-interval (g i) (f x y) sub-interval (g (Suc i)) (g i) R (Suc i) - L (Suc i) ≤ 3 / 4 * (R i - L i) unfolding L-def R-def by auto
} note * = this
  fix eps :: real
  assume eps: 0 < eps
  let ?r = real-of-rat
define r where r = (λ n. ?r (R n))
define l where l = (λ n. ?r (L n))
define diff where diff = (λ n. r n - l n)

\textbf{fix }n\textbf{ from }\ast (3/4)\textbf{ of }n\textbf{ have }\ ?r \ (R \ (Suc \ n) - L \ (Suc \ n)) \leq \ ?r \ (3/4 \ast (R \ n - L \ n))\textbf{ unfolding of-rat-less-eq by simp}
\textbf{also have }\ ?r \ (R \ (Suc \ n) - L \ (Suc \ n)) = (r \ (Suc \ n) - l \ (Suc \ n))\textbf{ unfolding of-rat-diff r-def l-def by simp}
\textbf{finally have }\ ?r \ (3/4 \ast (R \ n - L \ n)) = 3/4 \ast (r \ n - l \ n)\textbf{ unfolding r-def l-def by (simp add: hom-distribs)}
\textbf{note }\ast = \textbf{ this }\{\textbf{fix }i\textbf{ have }\ ?r \ (R \ (Suc \ n) - L \ (Suc \ n)) \leq \ ?r \ (3/4)\textbf{ of }i\textbf{ show }\ ?\textbf{ case by auto }\textbf{ qed auto}\} \textbf{ then obtain }c \textbf{ where }\ast : \ ?r \ (R \ n - L \ n) \leq \ ?r \ (3/4)\textbf{ of }i\textbf{ show }\ ?\textbf{ thesis by (intro exI[of - 0], auto)}\textbf{ next case True with }\ast \textbf{ have }\ ?r \ (R \ n - L \ n) \leq \ ?r \ (3/4)\textbf{ of }i\textbf{ show }\ ?\textbf{ thesis by (intro exI[of - 0], auto)}\textbf{ qed }\textbf{ qed}
\textbf{fun add-1 :: real-alg-1 }\Rightarrow \textbf{ real-alg-1 }\Rightarrow \textbf{ real-alg-2 where add-1 }\ (p1,l1,r1) \ (p2,l2,r2) = \ (\textbf{select-correct-factor-int-poly (tighten-poly-bounds-binary }p1\ p2)\ (\lambda \ ((l1,r1,sgn ((ipoly \ p1 \ r1))) \ (l2,r2,sgn ((ipoly \ p2 \ r2)))) \ ((l1,r1,sgn ((ipoly \ p1 \ r1))) \ (l2,r2,sgn ((ipoly \ p2 \ r2)))) \ (poly-add 0 \ p1)\ p2)\)}
\textbf{lemma add-1: assumes }x: \textbf{ invariant-1-2 }x \textbf{ and }y: \textbf{ invariant-1-2 }y
defines \( z \equiv \text{add-1} \ x \ y \)
shows \( \text{invariant-2} \ z \land (\text{real-of-2} \ z = \text{real-of-1} \ x + \text{real-of-1} \ y) \)
proof (cases \( x \))
case \( xt \): (fields \( p1 \ l1 \ r1 \))
show \(?thesis\)
proof (cases \( y \))
case \( yt \): (fields \( p2 \ l2 \ r2 \))
let \(?x = \text{real-of-1} \ (p1, l1, r1)\)
let \(?y = \text{real-of-1} \ (p2, l2, r2)\)
let \(?p = \text{poly-add} \ p1 \ p2\)
note \( x = x[\text{unfolded} \ xt]\)
note \( y = y[\text{unfolded} \ yt]\)
from \( x \) have \( \text{ax}: p1 \text{ represents } ?x \)
unfolding \( \text{represents-def} \) by (auto elim!:\( \text{invariant-1E} \))
from \( y \) have \( \text{ay}: p2 \text{ represents } ?y \)
unfolding \( \text{represents-def} \) by (auto elim!:\( \text{invariant-1E} \))
let \(?bnd = (\lambda ((l1, r1, sr1 :: rat), l2 :: rat, r2 :: rat, sr2 :: rat). (l1 + l2, r1 + r2))\)
define \( \text{bnd where } \text{bnd} = ?bnd \)
have \( \text{invariant-2} \ z \land \text{real-of-2} \ z = ?x + ?y \)
proof (intro select-correct-factor-int-poly)
from \( \text{represents-add[OF ax ay]} \) show \(?p \neq 0 \text{ ipoly } ?p \ (\ ?x + ?y) = 0 \) by auto
from \( z[\text{unfolded} \ xt \ yt]\) show sel: select-correct-factor-int-poly
  ((tighten-poly-bounds-binary \( p1 \ p2 \)) \( \text{bnd} \)
   ((l1,r1,sgn (ipoly \ p1 \ r1)),l2,r2, sgn (ipoly \ p2 \ r2)))
   (poly-add \( p1 \ p2 \) = \( z \) by (auto simp: \text{bnd-def} ))
   have \( \text{ur1: unique-root} \ (p1,l1,r1) \text{ poly-cond2 } p1 \text{ using } x \) by auto
   have \( \text{ur2: unique-root} \ (p2,l2,r2) \text{ poly-cond2 } p2 \text{ using } y \) by auto
   show converges-to
   (\( \lambda i. \text{bnd } ((\text{tighten-poly-bounds-binary} \ p1 \ p2) \ i) \)
    ((l1,r1,sgn (ipoly \ p1 \ r1)),l2,r2, sgn (ipoly \ p2 \ r2)))\) \( (\ ?x + ?y) \)
   by (intro tighten-poly-bounds-binary \( \text{ur1 ur2; force simp} : \text{bnd-def hom-distribs} \))
qed
thus \(?thesis\) unfolding \( xt \ yt \).
qed
qed

declare \( \text{add-rat-1.simps\{simp del\}} \)
declare \( \text{add-1.simps\{simp del\}} \)

8.2.11 Multiplication

context
begin
private fun \( \text{mult-rat-1-pos} :: \text{rat} \Rightarrow \text{real-alg-1} \Rightarrow \text{real-alg-2} \) where
\( \text{mult-rat-1-pos} \ r1 \ (p2,l2,r2) = \text{real-alg-2} \ (\text{cf-pos-poly} \ (\text{poly-mult-rat} \ r1 \ p2), \ l2* r1)\),
private fun mult-1-pos :: real-alg-1 ⇒ real-alg-1 ⇒ real-alg-2 where
mult-1-pos (p1,l1,r1) (p2,l2,r2) =
  select-correct-factor-int-poly
  (tighten-poly-bounds-binary p1 p2)
  (λ ((l1,r1,l1),(l2,r2,r2)), (l1 * l2, r1 * r2))
  ((l1,r1,sgn (ipoly p1 r1)),(l2,r2, sgn (ipoly p2 r2)))
  (poly-mult p1 p2)

fun mult-rat-1 :: rat ⇒ real-alg-1 ⇒ real-alg-2 where
mult-rat-1 x y =
  if x < 0 then uminus-2 (mult-rat-1-pos (−x) y)
  else if x = 0 then Rational 0 else
  (mult-rat-1-pos x y)

fun mult-1 :: real-alg-1 ⇒ real-alg-1 ⇒ real-alg-2 where
mult-1 x y =
  case (x,y) of
    ((p1,l1,r1), (p2,l2,r2)) ⇒
      if r1 > 0 then
        if r2 > 0 then mult-1-pos x y
        else uminus-2 (mult-1-pos (uminus-1 x) y)
      else mult-1-pos (uminus-1 x) (uminus-1 y)

lemma mult-rat-1-pos: fixes r1 :: rat assumes r1: r1 > 0 and y: invariant-1 y
defines z: z ≡ mult-rat-1-pos r1 y
shows invariant-2 z ∧ (real-of-2 z = of-rat r1 * real-of-1 y)
proof −
obtain p2 l2 r2 where yD: y = (p2,l2,r2) by (cases y, auto)
let ?x = real-of-rat r1
let ?y = real-of-1 (p2, l2, r2)
let ?p = poly-mult-rat r1 p2
let ?mp = cf-pos-poly ?p
note y = y[unfolded yD]
note yD = invariant-1D[OF y]
from yD r1 have p: ?p ≠ 0 and r10: r1 ≠ 0 by auto
hence mp: ?mp ≠ 0 by simp
from yD(1) have rt: ipoly p2 (?y = 0 and bnd: of-rat l2 ≤ ?y ?y ≤ of-rat r2) by auto
from rt r1 have rt: ipoly ?mp (?x * ?y = 0 by (auto simp add: field-simps
ipoly-mult-rat[OF r10])
from yD(5) have irr: irreducible p2
  unfolding represents-def using y unfolding root-cond-def split by auto
from poly-mult-rat-irreducible[OF this - r10] yD
have irrv: irreducible ?mp by simp
from p have mon: cf-pos ?mp by auto
obtain l r where lr: l = l2 * r1 r = r2 * r1 by force
from bnd r1 have bnd: of-rat l ≤ ?x * ?y ?x * ?y ≤ of-rat r unfolding lr
of-rat-mult by auto
with rt have rc: root-cond (?mp,l,r) (?x * ?y) unfolding root-cond-def by auto
have ur: unique-root (?mp,l,r)
proof (rule exII, rule rc)
  fix z
  assume root-cond (?mp,l,r) z
  from this[unfolded root-cond-def split] have bndz: of-rat l ≤ z z ≤ of-rat r
  and r1: ipoly ?mp z = 0 by auto
  have fst (quotient-of r1) ≠ 0 using quotient-of-div[of r1] r10 by (cases quotient-of r1, auto)
  with r1 have rt: ipoly p2 (z * inverse ?x) = 0 by (auto simp: ipoly-mult-rat[OF r10])
  from bndz r1 have of-rat l2 ≤ z * inverse ?x z * inverse ?x ≤ of-rat r2
  unfolding br of-rat-mult
  by (auto simp: field-simps)
  with rt have root-cond (p2,l2,r2) (z * inverse ?x) unfolding root-cond-def by auto
  also note invariant-1-root-cond[OF y]
  finally have ?y = z * inverse ?x by auto
  thus z = ?x * ?y using r1 by auto
  qed

from r1 have sgn: sgn r = sgn r2 unfolding br
  by (cases r2 = 0; cases r2 < 0; auto simp: mult-neg-pos mult-less-0-iff)
from r1 have sgn: sgn l = sgn l2 unfolding br
  by (cases l2 = 0; cases l2 < 0; auto simp: mult-neg-pos mult-less-0-iff)
from the-unique-root-eq[OF ur rc] have xy: ?x * ?y = the-unique-root (?mp,l,r)
by auto
from z[unfolded yt, simplified, unfolded Let-def lr[symmetric] split]
  have z: z = real-alg-2 (?mp, l, r) by simp
  have yp2: p2 represents ?y using yD unfolding root-cond-def split represents-def
      by auto
with irr mon have pc: poly-cond ?mp by (auto simp: poly-cond-def cf-pos-def)
  have rc: invariant-1 (?mp, l, r) unfolding z using yD(2) pc ur
      by (auto simp add: invariant-1-def ur mp sgnr sgnl)
  show ?thesis unfolding z using real-alg-2[OF rc]
      unfolding yt xy unfolding z by simp
  qed

lemma mult-1-pos: assumes x: invariant-1-2 x and y: invariant-1-2 y
  defines z: z ≡ mult-1-pos x y
  assumes pos: real-of-1 x > 0 real-of-1 y > 0
  shows invariant-2 z ∧ (real-of-2 z = real-of-1 x * real-of-1 y)
proof –
  obtain p1 l1 r1 where xt: x = (p1,l1,r1) by (cases x, auto)
  obtain p2 l2 r2 where yt: y = (p2,l2,r2) by (cases y, auto)
  let ?x = real-of-1 (p1, l1, r1)
  let ?y = real-of-1 (p2, l2, r2)
  let ?r = real-of-rat
  let ?p = poly-mult p1 p2
  note x = x[unfolded xt]
  note y = y[unfolded yt]
from $x \ y$ have basic: unique-root $(p_1, l_1, r_1)$ poly-cond p1 unique-root $(p_2, l_2, r_2)$ poly-cond p2 by auto
from basic have irr1: irreducible $p_1$ and irr2: irreducible $p_2$ by auto
from $x$ have ax: $p_1$ represents $?x$ unfolding represents-def by (auto elim!: invariant-IE)
from $y$ have ay: $p_2$ represents $?y$ unfolding represents-def by (auto elim!: invariant-IE)
from ax ay pos[unfolded $xt \ yt$] have axy: $?p$ represents $(?x \ ?y)$
by (intro represents-mult represents-irr-non-0[OF irr2], auto)
from represents[OF this] have $p$: $?p \neq 0$ and $rt$: ipoly $?p$ $(?x \ ?y) = 0$ .
from $x$ pos[OF[unfolded $xt$]] have $?r \ r_1 > 0$ unfolding split by auto
hence $sgn \ r_1 = 1$ unfolding $sgn$-rat-def by (auto split: $if$-splits)
with $x$ have $sgn \ l_1 = 1$ by auto
hence $l_1$-pos: $l_1 > 0$ unfolding $sgn$-rat-def by (cases $l_1 = 0$; cases $l_1 < 0$; auto)
from $y$ pos[OF[unfolded $yt$]] have $?r \ r_2 > 0$ unfolding split by auto
hence $sgn \ r_2 = 1$ unfolding $sgn$-rat-def by (auto split: $if$-splits)
with $y$ have $sgn \ l_2 = 1$ by auto
hence $l_2$-pos: $l_2 > 0$ unfolding $sgn$-rat-def by (cases $l_2 = 0$; cases $l_2 < 0$; auto)
let $?bnd = (\lambda((l_1, r_1, s_1 :: rat), l_2 :: rat, r_2 :: rat). (l_1 \ l_2, r_1 \ r_2))$
define $bnd$ where $bnd = ?bnd$
obtain $z'$ where sel: select-correct-factor-int-poly
  (tightly-poly-bounds-binary $p_1 \ p_2$)
$bnd$
((l_1, r_1, $sgn$ (ipoly $p_1 \ r_1$)), (l_2, r_2, $sgn$ (ipoly $p_2 \ r_2$))
$?p = z'$ by auto
have main: invariant-2 $z' = ?x \ ?y$
proof (rule select-correct-factor-int-poly[OF sel $rt \ p$])
  \{ fix $l_1 \ r_1 \ l_2 \ r_2$ $l_1'$ $r_1'$ $l_2'$ $r_2'$ $l_1' \ r_1' :: rat$
define $d_1$ where $d_1 = r_1 - l_1$
define $d_2$ where $d_2 = r_2 - l_2$
let $?m_1 = (l_1+r_1)/2$ let $?m_2 = (l_2+r_2)/2$
assume le: $l_1 > 0 \ l_2 > 0 \ l_1 \leq \ r_1 \ l_2 \leq r_2$ and id: $(l, r) = (l_1 \ l_2, r_1 \ r_2)$
$(l_1', r_1') = (l_1' \ l_2', r_1' \ r_2')$
and mem: $(l_1', r_1') \in \{(l_1, ?m_1), (?m_1, r_1)\}$
$(l_2', r_2') \in \{(l_2, ?m_2), (?m_2, r_2)\}$
hence id: $l = l_1 \ l_2 = r = (l_1 + d_1) (*) (l_2 + d_2)$ $l' = l_1' \ l_2' = r_1' \ r_2'$
$r_1 = l_1 + d_1 \ r_2 = l_2 + d_2$ and id': $?m_1 = ?M_1 \ ?m_2 = ?M_2$
unfolding d1-def d2-def by (auto simp: field-simps)
define $ldd$ where $ldd = l_1 + d_1$
from le have ge0: $d_1 \geq 0 \ d_2 \geq 0 \ l_1 \geq 0 \ l_2 \geq 0$ unfolding d1-def d2-def by auto
have $4 \ast (r' - l') \leq 3 \ast (r - l)$
proof (cases $l_1' = l_1 \ r_1' = ?M_1 \ l_2' = l_2 \ r_2' = ?M_2$)
case True
  hence id2: $(l_1' = l_1 \ r_1' = ?M_1 \ l_2' = l_2 \ r_2' = ?M_2$ by auto
  show ?thesis unfolding id id2 unfolding ring-distrib using ge0 by simp
next
case False note 1 = this
show ?thesis
\}

151
proof (cases $l' = l1 \land r1' = ?M1 \land l2' = ?M2 \land r2' = r2$

  case True
  hence \( \text{id2: } l1' = l1 \land r1' = ?M1 \land l2' = ?M2 \land r2' = r2 \) by auto

  show \(?thesis unfolding \text{id id2 unfolding ring-distrib using ge0 by simp}\) next

  case False note 2 = this

  show \(?thesis unfolding \text{id id2 unfolding ring-distrib using ge0 by simp}\)

  qed

  qed

  qed

  hence \( r' - l' \leq 3 / 4 \cdot (r - l) \) by simp

  } note decr = this

  } show converges-to

  \((\lambda i . \text{bnd } ((\text{tighten-poly-bound-binary } p1 \ p2 \ ~ ~ i))
  ((l1, r1, \text{sgn (ipoly p1 r1)),(l2, r2, \text{sgn (ipoly p2 r2))})) \ (x \ast \ ?y))\)

proof (intro tighten-poly-bound-binary[where \( f = (\ast) \) and \( I = \lambda l . l > 0 \])

basic \( l1 \)-pos \( (\ast\ast\ast\; \text{goal-cases}) \)

 case \((1 \ L1 \ R1 \ L2 \ R2 \ L \ R)\)

 hence \( L = L1 \ast L2 \ R = R1 \ast R2 \) unfolding bnd-def by auto

 hence \( \text{id: } \forall r L = \forall r L1 \ast \forall r L2 \ ?r R = \forall r R1 \ast \forall r R2 \) by (auto simp: hom-distrib)

from \( I(3\sim4) \) have \( \text{le: } \forall r L1 < \forall x \ast \ ?x \ ?x \leq \ ?r R1 \ ?r R2 \leq \ ?y \ ?y \leq \ ?r R2 \)

unfolding root-cond-def by auto

from \( I(1\sim2) \) have \( \text{lt: } 0 < \forall r \ L1 \ ?r L1 \ ?r L2 \) by auto

from mult-mono[OF le'(1,3), folded id] lt le have \( \text{le: } \forall r L \leq \ ?x \ast \ ?y \) by linarith

have \( \text{R: } \forall r \ ?x \ ?y \leq \ ?r R \)

by (rule mult-mono[OF le'(2,4), folded id], insert le le, linarith+)

show \(?case using \( L \) \( R \) by blast\)

next

  case \((2 \ L1 \ R1 \ L2 \ R2 \ L' \ R' \ L2' \ R2' \ l' \ r')\)

from \( 2(5\sim6) \) have \( \text{lr: } l = l1 \ast l2 \ r = r1 \ast r2 \ l' = l1' \ast l2' \ r' = r1' \ast r2' \)

unfolding bnd-def by auto

from \( 2(1\sim4) \) have \( \text{le: } 0 < l1 \ ?l1 \ l2 \leq r1 \ l2 \leq r2 \) by auto

from \( 2(7\sim8) \) le have \( \text{le': } l1 \leq l1' \ r1' \leq r1 \ l2 \leq l2' \ r2' \leq r2 \ 0 < r2' \ 0 < r2 \)

by auto

  from mult-mono[OF le'(1,3), folded lr] le le' have \( \text{le: } l \leq l' \) by auto

  have \( \text{r: } r' \leq r \) by (rule mult-mono[OF le'(2,4), folded lr], insert le le', linarith+)

  have \( \text{r' - l' \leq 3 / 4 \ast (r - l)} \)

  by (rule decr[OF - - - - - - 2(7\sim8)], insert le le' lr, auto)

152
thus \( ?\text{case using } l \ r \text{ by } \text{blast} \)
\begin{verbatim}
  qed auto
\end{verbatim}
\begin{verbatim}
  qed
\end{verbatim}
\begin{verbatim}
  have \( z' : z' = z \text{ unfolding } z[\text{unfolded } xt \ yt, \text{ simplified}, \text{ unfolded } \text{bnd-def}[\text{symmetric}] \text{ sel}] \)
    by auto
\end{verbatim}
\begin{verbatim}
  from \text{main[unfolded this]} show \( ?\text{thesis unfolding } xt \ yt \text{ by } \text{simp} \)
\end{verbatim}
\begin{verbatim}
qed
\end{verbatim}

\textbf{lemma} \text{mult-1: assumes } x: \text{invariant-1-2 } x \text{ and } y: \text{invariant-1-2 } y
\begin{verbatim}
  defines \( z[\text{simp}] : z \equiv \text{mult-1 } x \ y \text{ shows } \text{invariant-2 } z \land (\text{real-of-2 } z = \text{real-of-1 } x \ast \text{real-of-1 } y) \text{ proof --} \)
\end{verbatim}
\begin{verbatim}
  obtain \( p1 \ l1 \ r1 \text{ where } xt[\text{simp}]: x = (p1,l1,r1) \text{ by (cases } x) \)
  obtain \( p2 \ l2 \ r2 \text{ where } yt[\text{simp}]: y = (p2,l2,r2) \text{ by (cases } y) \)
  let \( ?xt = (p1,l1,r1) \)
  let \( ?yt = (p2,l2,r2) \)
  let \( ?x = \text{real-of-1 } ?xt \)
  let \( ?y = \text{real-of-1 } ?yt \)
  let \( ?mx = \uminus-1 ?xt \)
  let \( ?my = \uminus-1 ?yt \)
  let \( ?r = \text{real-of-rat} \)
\end{verbatim}
\begin{verbatim}
  from \text{invariant-1-2-of-rat[OF } x, \text{of 0]} \text{ have } x0: \( ?x < 0 \lor ?x > 0 \) \text{ by auto} \)
  from \text{invariant-1-2-of-rat[OF } y, \text{of 0]} \text{ have } y0: \( ?y < 0 \lor ?y > 0 \) \text{ by auto} \)
\end{verbatim}
\begin{verbatim}
  from \text{uminus-1-2[OF } x] \text{ have } mx: \text{invariant-1-2 } ?mx \text{ and } [\text{simp}]: \( ?mx = - ?x \)
  by auto \)
  from \text{uminus-1-2[OF } y] \text{ have } my: \text{invariant-1-2 } ?my \text{ and } [\text{simp}]: \( ?my = - ?y \)
  by auto \)
\end{verbatim}
\begin{verbatim}
  have \( \text{id: } r1 > 0 \iff ?x > 0 \land 0 < r1 \land ?y > 0 \iff ?x < 0 \land r2 > 0 \iff ?y > 0 \land r2 < 0 \)
\end{verbatim}
\begin{verbatim}
    \( \iff ?y < 0 \)
\end{verbatim}
\begin{verbatim}
  using \( x \ y \text{ by auto} \)
\end{verbatim}
\begin{verbatim}
  show \( ?\text{thesis} \)
\end{verbatim}
\begin{verbatim}
  proof (cases \( ?x > 0 \))
  case x0: \text{True} \)
  show \( ?\text{thesis} \)
\end{verbatim}
\begin{verbatim}
  proof (cases \( ?y > 0 \))
  case y0: \text{True} \)
  with \text{ x y x0 } \text{mult-1-pos[OF } x \ y] \text{ show } ?\text{thesis by auto} \)
\end{verbatim}
\begin{verbatim}
next \)
\end{verbatim}
\begin{verbatim}
  case False \)
  with \text{ y0 } \text{have } y0: \( ?y < 0 \) \text{ by auto} \)
  with \text{ x0 } \text{have } z: \( z = \text{uminus-2 } (\text{mult-1-pos } ?xt \ ?myt) \)
    unfolding \( z \ xt \ yt \text{ mult-1smsps split id by } \text{simp} \)
\end{verbatim}
\begin{verbatim}
  from \text{x0 y0 } \text{mult-1-pos[OF } x \ y] \text{uminus-2[of mult-1-pos } ?xt \ ?myt]\)
  show \( ?\text{thesis unfolding } z \text{ by } \text{simp} \)
\end{verbatim}
\begin{verbatim}
qed \)
\end{verbatim}
\begin{verbatim}
next
\end{verbatim}

153
case False
with \(x0\) have \(?x0 < 0\) by simp
show \(?\text{thesis}\)
proof (cases \(?y > 0\))
  case y0: True
  with \(x0\ y \text{id}\) have \(z = \text{uminus-2} (\text{mult-1-pos} \ ?mxt \ ?yt)\) by simp
  from \(x0\ y0 \text{ mult-1-pos}[OF \text{mx} \ ?y]\) \(\text{uminus-2}[\text{of \ mult-1-pos} \ ?mxt \ ?yt]\)
  show \(?\text{thesis unfolding} \ z\) by auto
next
  case False
  with \(y0\) have \(?y < 0\) by simp
  with \(x0\ y \text{ have}\) \(z = \text{mult-1-pos} \ ?mxt \ ?myt\) by auto
  with \(x0\ y0 \text{ y mul-1-pos}[OF \text{mx} \ ?my]\)
  show \(?\text{thesis unfolding} \ z\) by auto
qed
qed

lemma mult-rat-1: fixes \(x\) assumes \(y: \text{invariant-1}\ ?y\)
defines \(z: z \equiv \text{mult-rat-1} \ x \ ?y\)
shows \(\text{invariant-2} \ z \land (\text{real-of-2} \ z = \text{of-rat} \ x \ \times \ \text{real-of-1} \ ?y)\)
proof (cases \(?y\))
  case \(\text{yt}\): (fields \(p2\ l2\ r2\))
  let \(?yt = (p2, l2, r2)\)
  let \(?x = \text{real-of-rat} \ x\)
  let \(?y = \text{real-of-1} \ ?yt\)
  let \(?myt = \text{mult-rat-1-pos} (-x) \ ?yt\)
  note \(y = y[unfolded \ yt]\)
  note \(z = z[unfolded \ yt]\)
  show \(?\text{thesis}\)
  proof (cases \(x\ 0::\text{rat rule:linorder-cases}\))
    case greater
    with \(z\) have \(z = \text{mult-rat-1-pos} \ x \ ?yt\) by simp
    from \(\text{mult-rat-1-pos}[OF \ x \ ?y]\)
    show \(?\text{thesis unfolding} \ yt \ z\) by auto
  next
    case less
    then have \(x: -x > 0\) by auto
    hence \(z = \text{uminus-2} \ ?myt\) unfolding \(z\) by simp
    from \(\text{mult-rat-1-pos}[OF \ x \ ?y]\) have \(\text{rc: invaraint-2} \ ?myt\)
    and \(\text{rr: real-of-2} \ ?myt = - ?x * ?y\) by (auto simp: \text{hom-distribs})
    from \(\text{uminus-2}[OF \ \text{rc}]\) \(\text{rr}\) show \(?\text{thesis unfolding} \ z[\text{symmetric}]\) unfolding \(\text{yt[\text{symmetric}]}\)
    by simp
  qed (auto simp: \text{z})
qed
end

154
declare mult-1.simps[simp del]
declare mult-rat-1.simps[simp del]

8.2.12 Root

definition ipoly-root-delta :: int poly ⇒ real where
  ipoly-root-delta p = Min {insert 1 { abs (x - y) | x y. ipoly p x = 0 ∧ ipoly p y = 0 ∧ x ≠ y}} / 4

lemma ipoly-root-delta: assumes p ≠ 0
  shows ipoly-root-delta p > 0
    2 ≤ card (Collect (root-cond (p, l, r))) ⟹ ipoly-root-delta p ≤ real-of-rat (r - l) / 4
proof
  let ?z = 0 :: real
  let ?R = {x. ipoly p x = ?z}
  define S where S = insert 1 ?set
  from finite-ipoly-roots[of assms] have finR: finite ?R and fin: finite (?R × ?R)
    by auto
  have finite ?set
    by (rule finite-subset[OF - finite-image[OF fin, of λ (x, y). abs (x - y)]], force)
  hence fin: finite S and ne: S ≠ {} and pos: ∀ x. x ∈ S ⟹ x > 0
    unfolding S-def by auto
  have delta: ipoly-root-delta p = Min S / 4 unfolding ipoly-root-delta-def S-def
    using pos: Min S > 0 using fin ne pos by auto
  show ipoly-root-delta p > 0 unfolding delta using pos by auto
  let ?S = Collect (root-cond (p, l, r))
  assume 2 ≤ card ?S
  hence 2 ≤ Suc (Suc 0) ≤ card ?S by simp
  from 2[unfolded card-le-Suc-iff[of - ?S]] obtain x T where
    ST: ?S = insert x T and xT: x ∉ T and I: Suc 0 ≤ card T by auto
  from I[unfolded card-le-Suc-iff[of - T]] obtain y where yT: y ∈ T by auto
  from ST xT yT have: x ∈ ?S and y ∈ ?S and xy: x ≠ y by auto
  hence abs (x - y) ∈ S unfolding S-def root-cond-def[abs-def] by auto
  with fin have Min S ≤ abs (x - y) by auto
  with pos have le: Min S / 2 ≤ abs (x - y) / 2 by auto
  from x y have abs (x - y) ≤ of-rat r - of-rat l unfolding root-cond-def[abs-def]
    by auto
  also have ... = of-rat (r - l) by (auto simp: of-rat-diff)
  finally have abs (x - y) / 2 ≤ of-rat (r - l) / 2 by auto
  with le show ipoly-root-delta p ≤ real-of-rat (r - l) / 4 unfolding delta by auto
qed

lemma sgn-less-eq-1-rat: fixes a b :: rat
  shows sgn a = 1 ⟹ a ≤ b ⟹ sgn b = 1
  by (metis (no_types, hide_lams) not_less one-neq-neg-one one-neq-zero order_trans sgn-rat-def)
lemma sgn-less-eq-1-real: fixes a b :: real
  shows sgn a = 1 =⇒ a ≤ b =⇒ sgn b = 1
  by (metis (no-types, hide-lams) not-less one-neq-neg-one one-neq-zero order-trans sgn-real-def)

definition compare-1-rat :: real-alg-1 ⇒ rat ⇒ order where
  compare-1-rat rai = (let p = poly-real-alg-1 rai in
  if degree p = 1 then let x = Rat.Fract (− coeff p 0) (coeff p 1)
  in (λ y. compare y x)
  else (λ y. compare-rat-1 y rai))

lemma compare-real-of-rat: compare (real-of-rat x) (of-rat y) = compare x y
  unfolding compare-rat-def compare-real-def comparator-of-def of-rat-less by auto

lemma compare-1-rat: assumes rc: invariant-1 y
  shows compare-1-rat y x = compare (of-rat x) (real-of-1 y)
  proof (cases degree (poly-real-alg-1 y) Suc 0 rule: linorder-cases)
  case less with invariant-1-degree-0[of rc] show ?thesis by auto
  next
  case deg: greater
  with rc have rc: invariant-1-2 y by auto
  from deg compare-rat-1[of rc, of x]
  show ?thesis unfolding compare-1-rat-def by auto
  next
  case deg: equal
  obtain p l r where: y = (p,l,r) by (cases y)
  note rc = invariant-1D[of rc[unfolded y]]
  from deg have p: degree p = Suc 0
  and id: compare-1-rat y x = compare x (Rat.Fract (− coeff p 0) (coeff p 1))
  unfolding compare-1-rat-def by (auto simp: Let-def y)
  from rc(1)[unfolded split] have ipoly p (real-of-1 y) = 0
  unfolding y by auto
  with degree-1-ipoly[of p, of real-of-1 y]
  have id': real-of-1 y = real-of-rat (Rat.Fract (− coeff p 0) (coeff p 1)) by simp
  show ?thesis unfolding id id' compare-real-of-rat ..
qed

context
  fixes n :: nat
begin
private definition initial-lower-bound :: rat ⇒ rat where
  initial-lower-bound l = (if l ≤ 1 then l else of-int (root-rat-floor n l))

private definition initial-upper-bound :: rat ⇒ rat where
  initial-upper-bound r = (of-int (root-rat-ceiling n r))

context
  fixes cmpx :: rat ⇒ order

...
fun tighten-bound-root ::
rat × rat ⇒ rat × rat where

tighten-bound-root (l',r') = (let
m' = (l' + r') / 2;
m = m' ^ n
in case cmpx m of
Eq ⇒ (m',m')
| Lt ⇒ (m',r')
| Gt ⇒ (l',m')

lemma tighten-bound-root: assumes sgn: sgn il = 1 real-of-1 x ≥ 0 and
il: real-of-rat il ≤ root n (real-of-1 x) and
ir: root n (real-of-1 x) ≤ real-of-rat ir and
rai: invariant-1 x and

cmpx: cmpx = compare-1-rat x and

n: n ≠ 0

shows converges-to (λ i. (tighten-bound-root ^^ i) (il, ir))

(root n (real-of-1 x)) (is converges-to ?f ?x)

unfolding converges-to-def

proof (intro conjI impI allI)

{ fix x :: real
  have x ≥ 0 ⇒ (root n x) ^ n = x using n by simp
} note root-exp-cancel = this

{ fix x :: real
  have x ≥ 0 ⇒ root n (x ^ n) = x using n
  using real-root-pos-unique by blast
} note root-exp-cancel' = this

from il ir have real-of-rat il ≤ of-rat ir by auto
hence ir-il: il ≤ ir by (auto simp: af-rat-less-eq)

from n have n': n > 0 by auto

{ fix i
  have im-interval (?f i) ?x ∧ sub-interval (?f i) (il,ir) ∧ (i ≠ 0 → sub-interval
  (?f i) (?f (i - 1)))
  ∧ snd (?f i) - fst (?f i) ≤ (ir - il) / 2 ^ i
  proof (induct i)
  case 0
  show ?case using il ir by auto
  next
  case (Suc i)
  obtain l' r' where id: (tighten-bound-root ^^ i) (il, ir) = (l',r')
  by (cases (tighten-bound-root ^^ i) (il, ir), auto)
  let ?m' = (l' + r') / 2
  let ?m = ?m' ^ n
  define m where m = ?m
  note IH = Suc[unfolded id split snd-conv fst-conv]
from IH have sub-interval \((l', r')\) \((il, ir)\) by auto  
hence \(il\); \(il \leq l' \leq ir \) by auto  
with \(sgn\) have \(l'0; l' > 0\) using \(sgn-1\)-pos \(sgn-less-eq\)-1-rat by blast  
from IH have \(br\)'x; in-interval \((l', r')\) \(?x\) by auto  
hence \(br\)'r; real-of-rat \(l' \leq of-rat r'\) by auto  
hence \(br\)'L; \(l' \leq r'\) unfolding \(af-rat-less-eq\)  
with \(l'0\) have \(r'0; r' > 0\) by auto  

note compare = compare-1-rat[\(OF\) \(rai\), of \(?m\), folded cmpx]  
from IH have \(r'; = r' - l' \leq (ir - il) / 2 \leq i \) by auto  
have \(r' = (l' + r') / 2 = (r' - l') / 2\) by (simp add: field-simps)  
also have \(\ldots \leq (ir - il) / 2 \leq \) by using  
by (rule divide-right-mono, auto)  
finally have size: \(r' - (l' + r') / 2 \leq (ir - il) / 2 \leq i\) by simp  
also have \(r' - (l' + r') / 2 = (l' + r') / 2 = l'\) by auto  
finally have size': \(l' + r') / 2 - l' \leq (ir - il) / 2 \leq i\) by simp  

have root \(n\) \((real-of-rat \(?m\)) = root \(n\) \((real-of-rat \(?m\)) \) by (simp add: hom-distribs)  
also have \(\ldots = real-of-rat \(?m\)  
by (rule root-exp-cancel', insert \(l'0\) \(br\)', auto)  
finally have root: \(root\) \(n\) \((of-rat \(?m\)) = of-rat \(?m'\).  
show \(?case\)  
proof \((cases\) cmpx \(?m\))  

case \(Eq\)  
from compare[unfolded \(Eq\)] have \(real-of-1\) \(x = of-rat \(?m\)  
unfolding compare-real-def comparator-of-def by (auto split: if-splits)  
from \(arg\)-cong[\(OF\) \(this\), \(of\) \(root\) \(n\)] have \(\?x = root\) \(n\) \((of-rat \(?m\)) .  
also have \(\ldots = root\) \(n\) \((real-of-rat \(?m\)) \)  
using \(n\) \((real\)-root-power\) by (auto simp: hom-distribs)  
also have \(\ldots = of-rat \(?m'\)  
by (rule root-exp-cancel, insert IH \(sgn\) \((2)\) \(l'0\) \(r'0\), auto)  
finally have \(x: \?x = of-rat \(?m'\).  
show \(?thesis\) using \(x\) \(id\) \(Eq\) \(br\)'i\ll' \(ir\)-il by (auto simp: Let-def)  

next  
case \(Lt\)  
from compare[unfolded \(Lt\)] have \(lt: of-rat \(?m\) \(\leq real-of-1\) \(x\)  
unfolding compare-real-def comparator-of-def by (auto split: if-splits)  
have \(id': if\) \((Suc\) \(i\)) \(= \(?m',r')\) \(if\) \((Suc\) \(i - 1\)) \(= \((l',r')\)  
using \(Lt\) \(id\) by (auto simp add: Let-def)  
from \(real\)-root-le-mono[\(OF\) \(n\) \(\ll\)]  
have \(of-rat\) \(\?m'\) \(\leq \?x\) unfolding root by simp  
with \(br\)'x \(br''\) have \(ineq': real-of-rat\) \(l' + real-of-rat\) \(r' \leq \?x * 2\) by (auto simp: hom-distribs)  
show \(?thesis\) unfolding \(id''\)  
by (auto simp: Let-def hom-distribs, insert size \(ineq' \) \(br\)'i\ll' \(lr\)'x ir-il, auto)  

next  
case \(Gt\)  
from compare[unfolded \(Gt\)] have \(lt: of-rat \(?m\) \(\geq real-of-1\) \(x\)  
unfolding compare-real-def comparator-of-def by (auto split: if-splits)  
have \(id': if\) \((Suc\) \(i\)) \(= \((l',\?m')\) \(if\) \((Suc\) \(i - 1\)) \(= \((l',r')\)  

158
using Gt id by (auto simp add: Let-def)
from real-root-le_mono[OF n lt]
have \$x \leq \text{of-rat } \$m'\quad \text{unfolding root by simp}
with \$l' x \$r'\quad \text{have ineq': } \$x \times 2 \leq \text{real-of-rat } \$l' + \text{real-of-rat } \$r'\quad \text{by (auto simp: hom-distribs)}
\quad \text{show } \$thesis \quad \text{unfolding id''}
\quad \text{by (auto simp: Let-def hom-distribs, insert size' ineq' lr' ill' lr'' x ir-il, auto)}
\quad \text{qed}
\quad \text{qed}
\} \quad \text{note main = this}
\text{fix } i\quad \text{from main[of i] show in-interval (\$f i) \$x by auto}
\text{from main[of Suc i] show sub-interval (\$f (Suc i)) (\$f i) by auto}
\text{fix eps :: real}
\quad \text{assume eps: } 0 < \text{eps}
\quad \text{define c where c = eps / (max (real-of-rat (ir - il)) 1)}
\quad \text{have c0: } c > 0 \quad \text{using eps unfolding c-def by auto}
\quad \text{from exp-tends-to-zero[OF - - this, of 1/2] obtain i where c: (1/2)^i \leq c by auto}
\text{obtain l' r' where fi: } \text{if i = (l',r')} \quad \text{by force}
\text{from main[of i, unfolded fi] have lec: } r' - l' \leq (\text{ir - il}) / 2 ^ i \quad \text{by auto}
\quad \text{have iril: real-of-rat (ir - il) \geq 0 using ir-il by (auto simp: of-rat-less-eq)}
\text{show } \exists n \text{ la ra. } \text{if n = (la, ra) \& real-of-rat ra - real-of-rat la \leq eps}
\text{proof (intro conjI exI, rule fi)}
\quad \text{have real-of-rat r' - of-rat l' = real-of-rat (r' - l') by (auto simp: hom-distribs)}
\quad \text{also have } \ldots \leq real-of-rat ((\text{ir - il}) / 2 ^ i) \quad \text{using le unfolding of-rat-less-eq .}
\text{also have } \ldots = (\text{real-of-rat (ir - il)} \times ((1/2)^i) \quad \text{by (simp add: field-simps hom-distribs)}
\text{also have } \ldots \leq (\text{real-of-rat (ir - il)}) \times c
\quad \text{by (rule mult-left_mono[OF c iril])}
\text{also have } \ldots \leq \text{eps}
\text{proof (cases real-of-rat (ir - il) \leq 1)}
\quad \text{case True}
\quad \text{hence c = eps unfolding c-def by (auto simp: hom-distribs)}
\quad \text{thus } \$thesis \quad \text{using eps True by auto}
\quad \text{next}
\quad \text{case False}
\quad \text{hence max (real-of-rat (ir - il)) I = real-of-rat (ir - il) \quad real-of-rat (ir - il) \neq 0}
\quad \quad \text{by (auto simp: hom-distribs)}
\quad \quad \text{hence (real-of-rat (ir - il)) \times c = eps unfolding c-def by auto}
\quad \quad \text{thus } \$thesis \quad \text{by simp}
\text{qed}
\text{finally show real-of-rat r' - of-rat l' \leq eps .}
\text{qed}
\text{qed}
\text{end}
private fun root-pos-1 :: real-alg-1 ⇒ real-alg-2 where
root-pos-1 (p,l,r) = (select-correct-factor-int-poly
(tighten-bound-root (compare-1-rat (p,l,r)))
(λ x. x)
(initial-lower-bound l, initial-upper-bound r)
(poly-nth-root n p))

fun root-1 :: real-alg-1 ⇒ real-alg-2 where
root-1 (p,l,r) = (if n = 0 ∨ r = 0 then Rational 0
else if r > 0 then root-pos-1 (uminus-1 (p,l,r))
else uminus-2 (root-pos-1 (uminus-1 (p,l,r))))

context
assumes n: n ≠ 0
begin

lemma initial-upper-bound: assumes x: x > 0 and xr: x ≤ of-rat r
shows sgn (initial-upper-bound r) = 1 root n x ≤ of-rat (initial-upper-bound r)
proof –
  have n: n > 0 using n by auto
  note d = initial-upper-bound-def
  let ?r = initial-upper-bound r
  from x xr have r0: r > 0 by (meson not-less of-rat-le-0-iff order-trans)
  hence of-rat r > (0 :: real) by auto
  hence root n (of-rat r) > 0 using n by simp
  hence l ≤ ceiling (root n (of-rat r)) by auto
  hence (1 :: rat) ≤ of-int (ceiling (root n (of-rat r))) by linarith
  also have . . . = ?r unfolding d by simp
  finally show sgn ?r = 1 unfolding sgn-rat-def by auto
  have root n x ≤ root n (of-rat r)
    unfolding real-root-le-iff[OF n] by (rule xr)
  also have . . . ≤ of-rat ?r unfolding d by simp
  finally show root n x ≤ of-rat ?r.
qed

lemma initial-lower-bound: assumes l: l > 0 and lx: of-rat l ≤ x
shows sgn (initial-lower-bound l) = 1 of-rat (initial-lower-bound l) ≤ root n x
proof –
  have n: n > 0 using n by auto
  note d = initial-lower-bound-def
  let ?l = initial-lower-bound l
  from l lx have x0: x > 0 by (meson not-less of-rat-le-0-iff order-trans)
  have sgn ?l = 1 ∧ of-rat ?l ≤ root n x
  proof (cases l ≤ l)
    case True
    hence l: ?l = l and l0: of-rat l ≥ (0 :: real) and l1: of-rat l ≤ (1 :: real)
      using l unfolding True d by auto
  qed
have $sgn:\ ?l = 1$ using $l$ unfolding $ll$ by auto
have of-rat $?l = 0$ of-rat $l$ unfolding $ll$ by simp
also have of-rat $l \leq \text{root} n$ (of-rat $l$) using real-root-increasing[OF $\vdash ll$, of $\vdash n$]

by (cases $n = 1$, auto)
also have $\ldots \leq \text{root} n x$ using $lx$ unfolding real-root-le-iff[OF $n$].
finally show $\vdash$thesis using $sgn$ by auto

next
case $\text{False}$
hence $l$: $(1 :: \text{real}) \leq \text{of-rat} l$ and $ll$: $?l = \text{of-int} (\text{floor} (\text{root} n (\text{of-rat} l)))$

unfolding $d$ by auto

hence $\text{root} n l \leq \text{root} n (\text{of-rat} l)$

unfolding real-root-le-iff[OF $n$] by auto

hence $1 \leq \text{root} n (\text{of-rat} l)$ using $n$ by auto

from floor-mon[OF $\vdash this$] have $1 \leq ?l$

using one-le-floor unfolding $ll$ by fastforce

hence $sgn$: $sgn ?l = 1$ by simp
have of-rat $?l \leq \text{root} n x$ using $lx$ unfolding real-root-le-iff[OF $n$].
finally have of-rat $?l \leq \text{root} n x$.

with $sgn$ show $\vdash$thesis by auto

qed

thus $sgn ?l = 1$ of-rat $?l \leq \text{root} n x$ by auto

qed

lemma root-pos-1:
assumes $x$: invariant-1 $x$ and $p$: rai-ub $x > 0$
defines $y$: $y \equiv \text{root-pos-1} x$
shows invariant-2 $y \land \text{real-of-2} y = \text{root} n (\text{real-of-1} x)$

proof (cases $x$)
case $(\text{fields} p \ l \ r)$
let $?r = \text{initial-upper-bound} r$
from $x$ fields have rai: invariant-1 $(p,l,r)$ by auto

note $\ast = \text{invariant-1D}[OF \vdash this]$

let $?x = \text{the-unique-root} (p,l,r)$

from pos[unfolded fields] *

have sgnl: $sgn l = 1$ by auto

from sgnl have $\vdash ll$: $l > 0$ by (unfold sgn-1-pos)

hence $\vdash ll$: real-of-rat $l > 0$ by auto

from $\vdash ll$ have $x0$: $?x > 0$ by linarith

note $\vdash il = \text{initial-lower-bound}[OF \vdash ll lx]$

from $\vdash il$ have $?x \leq \text{of-rat} r$ by auto

note $\vdash iu = \text{initial-upper-bound}[OF \vdash x0 \vdash this]$

let $?p = \text{poly-nth-root} n p$

from $x0$ have id: $\text{root} n ?x \sim n = ?x$ using $n$ real-root-pow-pos by blast

have rc: root-cond $(?p, \ ?l, ?r)$ (root $n$ ?x)

using $\vdash il iu \ast$ by (intro root-condI, auto simp: ipoly-nth-root id)
hence root: ipoly ?p (root n (real-of-1 x)) = 0

unfolding root-cond-def fields by auto

from * have p ≠ 0 by auto

hence p': ?p ≠ 0 using poly-nth-root-0[of n p] n by auto

have tbr: 0 ≤ real-of-1 x
real-of-rat (initial-lower-bound l) ≤ root n (real-of-1 x)

root n (real-of-1 x) ≤ real-of-rat (initial-upper-bound r)

using x0 iil(2) iu(2) fields by auto

from select-correct-factor-int-poly[OF tighten-bound-root[OF il(1)[folded fields]]
tbr x refl n
refl root p]

show ?thesis by (simp add: y fields)

qed

end

lemma root-1: assumes x: invariant-1 x
defines y: y ≡ root-1 x
shows invariant-2 y ∧ (real-of-2 y = root n (real-of-1 x))
proof (cases n = 0 ∨ rai-ab x = 0)
case True
with x have n = 0 ∨ real-of-1 x = 0 by (cases x, auto)
then have root n (real-of-1 x) = 0 by auto
then show ?thesis unfolding y root-1.simps
using x by (cases x, auto)

next
case False with x have n: n ≠ 0 and x0: real-of-1 x ≠ 0 by (simp, cases x, auto)

note rt = root-pos-1

show ?thesis

proof (cases rai-ab x 0::rat rule:linorder-cases)
case greater
with rt[OF n x this] n show ?thesis by (unfold y, cases x, simp)

next
case less
let ?um = uminusch-1
let ?rt = root-pos-1

from n less y x0 have y: y = uminusch-2 (?rt (?um x)) by (cases x, auto)
from uminusch-1[OF x] have umx: invariant-1 (?um x) and umx2: real-of-1 (?um x) = - real-of-1 x by auto

with x less have 0 < rai-ab (uminusch-1 x)

by (cases x, auto simp: uminusch-1.simps Let-def)

from rt[OF n umx this] umx2 have rumx: invariant-2 (?rt (?um x))
and rumx2: real-of-2 (?rt (?um x)) = root n (- real-of-1 x)

by auto

from uminusch-2[OF rumx] rumx2 y real-root-minus show ?thesis by auto

next
case equal with x0 x show ?thesis by (cases x, auto)

qed
\textbf{8.2.13 Embedding of Rational Numbers}

**definition** of-rat-1 :: rat \Rightarrow real-alg-1 where
of-rat-1 x \equiv (\text{poly-rat } x, x, x)

**lemma** of-rat-1:
\text{shows invariant-1 (of-rat-1 x) and real-of-1 (of-rat-1 x) = of-rat x}
\text{unfolding of-rat-1-def}
\text{by (atomize(full), intro invariant-1-realI unique-rootI poly-condI, auto )}

**fun** info-2 :: real-alg-2 \Rightarrow rat + int poly \times nat where
info-2 (Rational x) = Inl x
| info-2 (Irrational n (p,l,r)) = Inr (p,n)

**lemma** info-2-card: \text{assumes rc: invariant-2 x}
\text{shows info-2 x = Inr (p,n) \Rightarrow poly-cond p \land ipoly p (real-of-2 x) = 0 \land degree p \geq 2}
\text{\land card (roots-below p (real-of-2 x)) = n}
info-2 x = Inl y \Rightarrow real-of-2 x = of-rat y
\text{proof (atomize(full), goal-cases)}
\text{case 1}
\text{show ?case}
\text{proof (cases x)}
\text{case (Irrational m rai)}
\text{then obtain q l r where x: x = Irrational m (q,l,r) by (cases rai, auto)}
\text{show ?thesis}
\text{proof (cases q = p \land m = n)}
\text{case False}
\text{thus ?thesis using x by auto}
\text{next}
\text{case True}
\text{with x have x: x = Irrational n (p,l,r) by auto}
\text{from rc[unfolded x, simplified] have inv: invariant-1-2 (p,l,r) and}
\text{n: card (roots-below p (real-of-2 x)) = n and 1: degree p \neq 1}
\text{by (auto simp: x)}
\text{from inv have degree p \neq 0 unfolding irreducible-def by auto}
\text{with 1 have degree p \geq 2 by linarith}
\text{thus ?thesis unfolding n using inv x by (auto elim!: invariant-1E)}
\text{qed}
\text{qed auto}
\text{qed}

**lemma** real-of-2-Irrational: \text{invariant-2 (Irrational n rai) \Rightarrow real-of-2 (Irrational n rai) \neq of-rat x}
\text{proof}
assume invariant-2 (Irrational n rai) and rat: real-of-2 (Irrational n rai) = 
real-of-rat x

hence real-of-1 rai ∈ ℚ invariant-1-2 rai by auto

from invariant-1-2-of-rat[of this(2)] rat show False by auto

qed

lemma info-2: assumes
ix: invariant-2 x and iy: invariant-2 y

shows info-2 x = info-2 y ←→ real-of-2 x = real-of-2 y

proof (cases x)
case x: (Irrational n1 rai1)

note ix = ix[unfolded x]

show ?thesis
proof (cases y)
case (Rational y)

with real-of-2-Irrational[of ix, of y]

show ?thesis unfolding x by (cases rai1, auto)

next
case y: (Irrational n2 rai2)

obtain p1 l1 r1 where rai1: rai1 = (p1,l1,r1) by (cases rai1)

obtain p2 l2 r2 where rai2: rai2 = (p2,l2,r2) by (cases rai2)

let ?rx = the-unique-root (p1,l1,r1)

let ?ry = the-unique-root (p2,l2,r2)

have id: (info-2 x = info-2 y) = (p1 = p2 ∧ n1 = n2)

(real-of-2 x = real-of-2 y) = (?rx = ?ry)

unfolding x y rai1 rai2 by auto

from ix[unfolded x rai1]

have ix: invariant-1 (p1, l1, r1) and deg1: degree p1 > 1 and n1: n1 = card

(roots-below p1 ?rx) by auto

noteIx = invariant-1D[of ix]

from deg1 have p1-0: p1 ≠ 0 by auto

from iy[unfolded y rai2]

have iy: invariant-1 (p2, l2, r2) and degree p2 > 1 and n2: n2 = card

(roots-below p2 ?ry) by auto

noteIy = invariant-1D[of iy]

show ?thesis unfolding id

proof
assume eq: ?rx = ?ry

fromIx

have algx: p1 represents ?rx ∧ irreducible p1 ∧ lead-coeff p1 > 0 unfolding

represents-def by auto

from iy

have algx: p2 represents ?rx ∧ irreducible p2 ∧ lead-coeff p2 > 0 unfolding

represents-def eq by (auto elim!: invariant-1E)

from algx have algebraic ?rx unfolding algebraic-altdef-ipoly by auto

note unique = algebraic-imp-represents-unique[of this]

with algx algx have id: p2 = p1 by auto

from eq id n1 n2 show p1 = p2 ∧ n1 = n2 by auto

next

164
assume \( p1 = p2 \land n1 = n2 \)

hence \( id: p1 = p2 n1 = n2 \) by auto

hence \( card: card (roots-below p1 ?rx) = card (roots-below p1 ?ry) \) unfolding

\( n1 n2 \) by auto

show \( ?rx = ?ry \)

proof (cases \( ?rx \ ?ry \) rule: linorder-cases)
  case less
  have \( roots-below p1 ?rx = roots-below p1 ?ry \)
  proof (intro card-subset-eq finite-subset[OF - ipoly-roots-finite] card)
    from less show \( roots-below p1 ?rx \subseteq roots-below p1 ?ry \) by auto
  qed (insert p1-0, auto)
  then show \( ?thesis \) using id less unique-rootD(3)[OF Iy(4)] by (auto simp: less-eq-real-def)
next
  case equal
  then show \( ?thesis \) using \( id \) by (auto)
next
  case greater
  have \( roots-below p2 ?ry = roots-below p1 ?rx \)
  proof (intro card-subset-eq card[symmetric] finite-subset[OF - ipoly-roots-finite[OF p1-0]])
    from greater show \( roots-below p1 ?ry \subseteq roots-below p1 ?rx \) by auto
  qed auto
  hence \( roots-below p2 ?ry = roots-below p2 ?rx \) unfolding \( id \) by auto
  thus \( ?thesis \) using \( id greater unique-rootD(3)[OF Ix(4)] \) by (auto simp: less-eq-real-def)
  qed
  qed
  qed
next
  case \( x \): (Rational \( x \))
  show \( ?thesis \)
  proof (cases \( y \))
    case (Rational \( y \))
    thus \( ?thesis \) using \( x \) by auto
  next
    case \( y \): (Irrational \( n rai \))
    with real-of-2-Irrational[OF iy[unfolded y], of \( x \)] show \( ?thesis \) unfolding \( x \) by (cases rai, auto)
  qed
  qed

lemma info-2-unique: invariant-2 \( x \) \( \Rightarrow \) invariant-2 \( y \) \( \Rightarrow \)
real-of-2 \( x \) = real-of-2 \( y \) \( \Rightarrow \) info-2 \( x \) = info-2 \( y \)
using info-2 by blast

lemma info-2-inj: invariant-2 \( x \) \( \Rightarrow \) invariant-2 \( y \) \( \Rightarrow \) info-2 \( x \) = info-2 \( y \) \( \Rightarrow \)
real-of-2 \( x \) = real-of-2 \( y \)
using info-2 by blast
context
  fixes cr1 cr2 :: rat ⇒ rat ⇒ nat

begin
partial-function (tailrec) compare-1 :: int poly ⇒ int poly ⇒ rat ⇒ rat ⇒ rat ⇒
rat ⇒ rat ⇒ rat ⇒ order where
  [code]: compare-1 p1 p2 l1 r1 l2 r2 sr1 l2 r2 sr2 = (if r1 < l2 then Lt else if r2 < l1
then Gt
else let
  (l1',r1',sr1') = tighten-poly-bounds p1 l1 r1 sr1;
(l2',r2',sr2') = tighten-poly-bounds p2 l2 r2 sr2
in compare-1 p1 p2 l1' r1' sr1' l2' r2' sr2')

lemma compare-1:
  assumes ur1: unique-root (p1,l1,r1)
  and ur2: unique-root (p2,l2,r2)
  and pc: poly-cond2 p1 poly-cond2 p2
  and diff: the-unique-root (p1,l1,r1) ≠ the-unique-root (p2,l2,r2)
  and sr: sr1 = sgn (ipoly p1 r1) sr2 = sgn (ipoly p2 r2)
  shows compare-1 p1 p2 l1 r1 l2 r2 sr2
= compare (the-unique-root (p1,l1,r1))
  (the-unique-root (p2,l2,r2))

proof −
  let ?r = real-of-rat

  { fix d x y
    assume d: d = (r1 - l1) + (r2 - l2) and xy: x = the-unique-root (p1,l1,r1) y
    = the-unique-root (p2,l2,r2)
    define delta where delta = abs (x - y) / 4
    have delta: delta > 0 and diff: x ≠ y unfolding delta-def using diff xy by auto
    let ?rel' = {(x, y). 0 ≤ y ∧ delta-gt delta x y}
    let ?rel = inv-image ?rel' ?r
    have SN: SN ?rel by (rule SN-inv-image[OF delta-gt-SN[OF delta]])
    from d ur1 ur2
    have thesis unfolding xy[symmetric] using xy sr
    proof (induct d arbitrary: l1 r1 l2 r2 sr1 l2 r2 sr2 rule: SN-induct[OF SN])
      case (1 d l1 r1 l2 r2)
      note IH = I(1)
      note d = I(2)
      note wr = I(3-4)
      note xy = I(5-6)
      note sr = I(7-8)
      note simps = compare-1.simps[of p1 p2 l1 r1 l2 r2 sr1 l2 r2 sr2]
      note urx = unique-rootD[OF ur1(1), folded xy]
      note ury = unique-rootD[OF ur2(2), folded xy]
      show ?case (is ?l = -)
      proof (cases r1 < l2)
        case True

166
hence \( l :: \forall l_1 \text{ and } l_2 \text{ unfolding } \text{simps-of-rat-less} \text{ by auto} \)

\( \text{show } \neg \text{thesis unfolding } l \text{ using } l_1 \text{ True } \text{wtx}(2) \text{ wry}(1) \)

\( \text{by (auto simp: compare-real-def comparator-of-def)} \)

next

case False

\( \text{note } le = \text{this} \)

\( \text{show } \neg \text{thesis unfolding } l \text{ using } l_1 \text{ True } \text{wry}(2) \text{ wry}(1) \)

\( \text{by (auto simp: compare-real-def comparator-of-def)} \)

next

case False

obtain \( l_1' r_1' s_1' \text{ where } \text{th1: tighten-poly-bounds} p_1 l_1 r_1 s_1 = (l_1', r_1', s_1') \)

\( \text{by (cases rule: prod-cases3, auto)} \)

obtain \( l_2' r_2' s_2' \text{ where } \text{th2: tighten-poly-bounds} p_2 l_2 r_2 s_2 = (l_2', r_2', s_2') \)

\( \text{by (cases rule: prod-cases3, auto)} \)

\( \text{from False le th1 th2 have } l :: \forall l_1 = \text{compare-1 } p_1 p_2 l_1' r_1' s_1' l_2' r_2' s_2' \)

unfolding simps by auto

from tighten-poly-bounds\[\text{OF th1 ur(1) pc(1) sr(1)}\]

\( \text{have } \text{rc1: root-cond } (p_1, l_1', r_1') \text{ (the-unique-root } (p_1, l_1, r_1)) \)

\( \text{and } \text{bnd1: } l_1 \leq l_1' \text{ l}_1' \leq r_1' r_1' \leq r_1 \text{ and } \text{d1: } r_1' - l_1' = (r_1 - l_1) / 2 \)

\( \text{and } s_1' s_1' = \text{sgn } (\text{ipoly p1 r1'}) \text{ by auto} \)

from pc have \( p_1 \neq 0 p_2 \neq 0 \text{ by auto} \)

from unique-root-sub-interval\[\text{OF ur(1) rc1 bnd1(1,3)} \text{ xy ur this}\]

\( \text{have } \text{ur1: unique-root } (p_1, l_1', r_1') \text{ and } x : x = \text{the-unique-root } (p_1, l_1', r_1') \)

\( \text{by (auto intro!: the-unique-root-eqI)} \)

from tighten-poly-bounds\[\text{OF th2 ur(2) pc(2) sr(2)}\]

\( \text{have } \text{rc2: root-cond } (p_2, l_2', r_2') \text{ (the-unique-root } (p_2, l_2, r_2)) \)

\( \text{and } \text{bnd2: } l_2 \leq l_2' l_2' \leq r_2' r_2' \leq r_2 \text{ and } \text{d2: } r_2' - l_2' = (r_2 - l_2) / 2 \)

\( \text{and } s_2' s_2' = \text{sgn } (\text{ipoly p2 r2'}) \text{ by auto} \)

from unique-root-sub-interval\[\text{OF ur(2) rc2 bnd2(1,3)} \text{ xy ur pc}\]

\( \text{have } \text{ur2: unique-root } (p_2, l_2', r_2') \text{ and } y : y = \text{the-unique-root } (p_2, l_2', r_2') \)

\( \text{by auto} \)

define \( d' \text{ where } d' = d/2 \)

\( \text{have } d' : d' = r_1' - l_1' + (r_2' - l_2') \text{ unfolding } d'\text{-def } d_1 d_2 \text{ by (simp add: field-simps)} \)

have \( d' : d' \geq 0 \text{ using bnd1 bnd2 unfolding } d' \text{ by auto} \)

have \( dd : d - d' = d/2 \text{ unfolding } d'\text{-def by simp} \)

have \( \text{abs } (x - y) \leq 2 * ?r d \)

proof (rule contr)

assume \( \neg \text{thesis} \)

hence \( l : 2 * ?r d < \text{abs } (x - y) \text{ by auto} \)

have \( r_1' - l_1' \leq d r_2' - l_2' \leq d \text{ unfolding } d \text{ using bnd1 bnd2 by auto} \)

from this[folded of-rat-less-eq\[\text{where } 'a = \text{real}]] \text{ lt}
have |r (r1 - l1) < abs (x - y) / 2 |r (r2 - l2) < abs (x - y) / 2
and dd: |r r1 - |r l1 ≤ |r d |r r2 - |r l2 ≤ |r d by (auto simp: of-rat-diff)
from le have r1 ≥ l2 by auto hence r1l2: |r r1 ≥ |r l2 unfolding of-rat-less-eq by auto
from False have r2 ≥ l1 by auto hence r2l1: |r r2 ≥ |r l1 unfolding of-rat-less-eq by auto
show False
proof (cases x ≤ y)
case True
from urx(1-2) dd(1) have |r r1 ≤ x + |r d by auto
with r1l2 have |r l2 ≤ x + |r d by auto
with Truc lt ury(2) dd(2) show False by auto
next
case False
from ury(1-2) dd(2) have |r r2 ≤ y + |r d by auto
with r2l1 have |r l1 ≤ y + |r d by auto
with False lt urx(2) dd(1) show False by auto
qed
qed
hence dd': delta-gt delta (|r d) (|r d')
unfolding delta-gt-def delta-def using dd by (auto simp: hom-distribs)
show ?thesis unfolding l
by (rule IH[OF - d' ur1 ur2 x y sr1 sr2], insert d'0 dd', auto)
qed
qed
}
thus ?thesis by auto
qed
end

fun real-alg-1 :: real-alg-2 ⇒ real-alg-1 where
real-alg-1 (Rational r) = of-rat-1 r
| real-alg-1 (Irrational n rai) = rai

lemma real-alg-1: real-of-1 (real-alg-1 x) = real-of-2 x
by (cases x, auto simp: of-rat-1)

definition root-2 :: nat ⇒ real-alg-2 ⇒ real-alg-2 where
root-2 n x = root-1 n (real-alg-1 x)

lemma root-2: assumes invariant-2 x
shows real-of-2 (root-2 n x) = root n (real-of-2 x)
invariant-2 (root-2 n x)
proof (atomize(full), cases x, goal-cases)
case (1 y)
show ?case by (simp add: root-2-def)
next
  case (2 i rai)
  from root-1[of rai n] assms 2 real-alg-2
  show ?case by (auto simp: root-2-def)
qed

fun add-2 :: real-alg-2 ⇒ real-alg-2 ⇒ real-alg-2 where
  add-2 (Rational r) (Rational q) = Rational (r + q)
| add-2 (Rational r) (Irrational n x) = Irrational n (add-rat-1 r x)
| add-2 (Irrational n x) (Rational q) = Irrational n (add-rat-1 q x)
| add-2 (Irrational n x) (Irrational m y) = add-1 x y

lemma add-2: assumes x: invariant-2 x and y: invariant-2 y
  shows real-of-2 (add-2 x y) (is ?g1)
  and real-of-2 (add-2 x y) = real-of-2 x + real-of-2 y (is ?g2)
  using assms add-rat-1 add-1
  by (atomize (full), (cases x; cases y), auto simp: hom-distribs)

fun mult-2 :: real-alg-2 ⇒ real-alg-2 ⇒ real-alg-2 where
  mult-2 (Rational r) (Rational q) = Rational (r ∗ q)
| mult-2 (Rational r) (Irrational n y) = mult-rat-1 r y
| mult-2 (Irrational n x) (Rational q) = mult-rat-1 q x
| mult-2 (Irrational n x) (Irrational m y) = mult-1 x y

lemma mult-2: assumes invariant-2 x invariant-2 y
  shows real-of-2 (mult-2 x y) = real-of-2 x ∗ real-of-2 y
  invariant-2 (mult-2 x y)
  using assms
  by (atomize(full), (cases x; cases y; auto simp: mult-rat-1 mult-1 hom-distribs))

fun to-rat-2 :: real-alg-2 ⇒ rat option where
  to-rat-2 (Rational r) = Some r
| to-rat-2 (Irrational n rai) = None

lemma to-rat-2: assumes rc: invariant-2 x
  shows to-rat-2 x = (if real-of-2 x ∈ ℚ then Some (THE q. real-of-2 x = of-rat q) else None)
proof (cases x)
  case (Irrational n rai)
  from real-of-2-Irrational[of rc[unfolded this]] show ?thesis
    unfolding Irrational Rats-def by auto
qed simp

fun equal-2 :: real-alg-2 ⇒ real-alg-2 ⇒ bool where
  equal-2 (Rational r) (Rational q) = (r = q)
| equal-2 (Irrational n (p,-)) (Irrational m (q,-)) = (p = q ∧ n = m)
| equal-2 (Rational r) (Irrational - yy) = False
lemma equal-2[simp]: assumes rc: invariant-2 x invariant-2 y
shows equal-2 x y = (real-of-2 x = real-of-2 y)
using info-2[OF rc]
by (cases x; cases y, auto)

fun compare-2 :: real-alg-2 ⇒ real-alg-2 ⇒ order where
compare-2 (Rational r) (Rational q) = (compare r q)
| compare-2 (Irrational n (p,l,r)) (Irrational m (q,l',r')) = (if p = q ∧ n = m then Eq
  else compare-1 p q l r (sgn (ipoly p r)) l' r' (sgn (ipoly q r')))
| compare-2 (Rational r) (Irrational - xx) = (compare-rat-1 r xx)
| compare-2 (Irrational - xx) (Rational r) = (invert-order (compare-rat-1 r xx))

lemma compare-2: assumes rc: invariant-2 x invariant-2 y
shows compare-2 x y = compare (real-of-2 x) (real-of-2 y)
proof (cases x)
case (Rational r) note xx = this
  show ?thesis
  proof (cases y)
  case (Rational q) note yy = this
  from compare-rat-1 rc
  show ?thesis unfolding xx yy by (simp add: compare-rat-def compare-real-def
  comparator-of-def of-rat-less)
next
case (Irrational n yy) note yy = this
  from compare-rat-1 rc
  show ?thesis unfolding xx yy by (simp add: of-rat-I)
qed
next
case (Irrational n xx) note xx = this
  show ?thesis
  proof (cases y)
  case (Rational q) note yy = this
  from compare-rat-1 rc
  show ?thesis unfolding xx yy by simp
next
case (Irrational m yy) note yy = this
  obtain p l r where xxx: xx = (p,l,r) by (cases xx)
  obtain q l' r' where yyy: yy = (q,l',r') by (cases yy)
  note rc = rc[unfolded xx xxx yy yyy]
  from rc have I: invariant-1-2 (p,l,r) invariant-1-2 (q,l',r') by auto
  then have unique-root (p,l,r) unique-root (q,l',r') poly-cond2 p poly-cond2 q
  by auto
  from compare-1[OF this - refl refl]
  show ?thesis using equal-2[OF rc] unfolding xx xxx yy yyy by simp
qed
qed
fun sgn-2 :: real-alg-2 ⇒ rat where
  sgn-2 (Rational r) = sgn r
  | sgn-2 (Irrational n rai) = sgn-1 rai

lemma sgn-2: invariant-2 x \implies real-of-rat (sgn-2 x) = sgn (real-of-2 x)
  using sgn-1 by (cases x, auto simp: real-of-rat-sgn)

fun floor-2 :: real-alg-2 ⇒ int where
  floor-2 (Rational r) = floor r
  | floor-2 (Irrational n rai) = floor-1 rai

lemma floor-2: invariant-2 x \implies floor-2 x = floor (real-of-2 x)
  by (cases x, auto simp: floor-1)

8.2.14 Definitions and Algorithms on Type with Invariant

lift-definition of-rat-3 :: rat ⇒ real-alg-3 is of-rat-2
  by (auto simp: of-rat-2)

lemma of-rat-3: real-of-3 (of-rat-3 x) = of-rat x
  by (transfer, auto simp: of-rat-2)

lift-definition root-3 :: nat ⇒ real-alg-3 ⇒ real-alg-3 is root-2
  by (auto simp: root-2)

lemma root-3: real-of-3 (root-3 n x) = root n (real-of-3 x)
  by (transfer, auto simp: root-2)

lift-definition equal-3 :: real-alg-3 ⇒ real-alg-3 ⇒ bool is equal-2 .

lemma equal-3: equal-3 x y = (real-of-3 x = real-of-3 y)
  by (transfer, auto)

lift-definition compare-3 :: real-alg-3 ⇒ real-alg-3 ⇒ order is compare-2 .

lemma compare-3: compare-3 x y = (compare (real-of-3 x) (real-of-3 y))
  by (transfer, auto simp: compare-2)

lift-definition add-3 :: real-alg-3 ⇒ real-alg-3 ⇒ real-alg-3 is add-2
  by (auto simp: add-2)

lemma add-3: real-of-3 (add-3 x y) = real-of-3 x + real-of-3 y
  by (transfer, auto simp: add-2)

lift-definition mult-3 :: real-alg-3 ⇒ real-alg-3 ⇒ real-alg-3 is mult-2
  by (auto simp: mult-2)
lemma mult-3: real-of-3 (mult-3 x y) = real-of-3 x * real-of-3 y
  by (transfer, auto simp: mult-2)

lift-definition sgn-3 :: real-alg-3 ⇒ rat is sgn-2.

lemma sgn-3: real-of-rat (sgn-3 x) = sgn (real-of-3 x)
  by (transfer, auto simp: sgn-2)

lift-definition to-rat-3 :: real-alg-3 ⇒ rat option is to-rat-2.

lemma to-rat-3: to-rat-3 x = (if real-of-3 x ∈ ℚ then Some (THE q. real-of-3 x = of-rat q) else None)
  by (transfer, simp add: to-rat-2)

lift-definition floor-3 :: real-alg-3 ⇒ int is floor-2.

lemma floor-3: floor-3 x = floor (real-of-3 x)
  by (transfer, auto simp: floor-2)

lift-definition info-3 :: real-alg-3 ⇒ rat + int poly × nat is info-2.

lemma info-3-fun: real-of-3 x = real-of-3 y ⇒ info-3 x = info-3 y
  by (transfer, intro info-2-unique, auto)

lift-definition info-real-alg :: real-alg ⇒ rat + int poly × nat is info-3
  by (metis info-3-fun)

lemma info-real-alg:
  info-real-alg x = Inr (p, n) ⇒ p represents (real-of x) ∧ card { y. y ≤ real-of x ∧ ipoly p y = 0} = n ∧ irreducible p
  info-real-alg x = Inl q ⇒ real-of x = of-rat q

proof (atomize (full), transfer, transfer, goal-cases)
  case (1 x p n q)
  from 1 have x: invariant-2 x by auto
  note info = info-2-card[OF this]
  show ?case
  proof (cases x)
    case irr: (Irrational m rai)
    from info(1)[of p n]
    show ?thesis unfolding irr by (cases rai, auto simp: poly-cond-def)
  qed (insert 1 info, auto)
  qed

instantiation real-alg :: plus
begin


lift-definition plus-real-alg :: real-alg ⇒ real-alg ⇒ real-alg is add-3
  by (simp add: add-3)
instance ..
end

lemma plus-real-alg: (real-of x) + (real-of y) = real-of (x + y)
  by (transfer, rule add-3[symmetric])

instantiation real-alg :: minus
begin
definition minus-real-alg :: real-alg ⇒ real-alg ⇒ real-alg where
  minus-real-alg x y = x + (−y)
instance ..
end

lemma minus-real-alg: (real-of x) − (real-of y) = real-of (x − y)
  unfolding minus-real-alg-def minus-real-def uminus-real-alg plus-real-alg ..

lift-definition of-rat-real-alg :: rat ⇒ real-alg is of-rat-3.

lemma of-rat-real-alg: real-of-rat x = real-of (of-rat-real-alg x)
  by (transfer, rule of-rat-3[symmetric])

instantiation real-alg :: zero
begin
definition zero-real-alg :: real-alg where zero-real-alg ≡ of-rat-real-alg 0
instance ..
end

lemma zero-real-alg: 0 = real-of 0
  unfolding zero-real-alg-def by (simp add: of-rat-real-alg[symmetric])

instantiation real-alg :: one
begin
definition one-real-alg :: real-alg where one-real-alg ≡ of-rat-real-alg 1
instance ..
end

lemma one-real-alg: 1 = real-of 1
  unfolding one-real-alg-def by (simp add: of-rat-real-alg[symmetric])

instantiation real-alg :: times
begin
lift-definition times-real-alg :: real-alg ⇒ real-alg ⇒ real-alg is mult-3
by (simp add: mult-3)
instance ..
end

lemma times-real-alg: \((\text{real-of } x) \ast (\text{real-of } y) = \text{real-of } (x \ast y)\)
by (transfer, rule mult-3 [symmetric])

instantiation \text{real-alg} :: \text{inverse}
begin
lift-definition inverse-real-alg :: \text{real-alg} \Rightarrow \text{real-alg} is inverse-3
by (simp add: inverse-3)
definition divide-real-alg :: \text{real-alg} \Rightarrow \text{real-alg} \Rightarrow \text{real-alg} where
divide-real-alg \( x \ y = x \ast \text{inverse } y \)
instance ..
end

lemma inverse-real-alg: \(\text{inverse } (\text{real-of } x) = \text{real-of } (\text{inverse } x)\)
by (transfer, rule inverse-3 [symmetric])

lemma divide-real-alg: \((\text{real-of } x) \div (\text{real-of } y) = \text{real-of } (x \div y)\)
unfolding divide-real-alg-def times-real-alg [symmetric] divide-real-def inverse-real-alg ..

instance real-alg :: ab-group-add
apply intro-classes
apply (transfer, unfold add-3, force)
apply (unfold zero-real-alg-def, transfer, unfold add-3 of-rat-3, force)
apply (transfer, unfold add-3 of-rat-3, force)
apply (transfer, unfold add-3 uminus-3 of-rat-3, force)
apply (unfold minus-real-alg-def, force)
done

instance real-alg :: field
apply intro-classes
apply (transfer, unfold mult-3, force)
apply (transfer, unfold mult-3, force)
apply (unfold one-real-alg-def, transfer, unfold mult-3 of-rat-3, force)
apply (transfer, unfold mult-3 add-3, force simp: field-simps)
apply (unfold zero-real-alg-def, transfer, unfold of-rat-3, force)
apply (transfer, unfold mult-3 inverse-3 of-rat-3, force simp: field-simps)
apply (unfold divide-real-alg-def, force)
apply (transfer, unfold inverse-3 of-rat-3, force)
done

instance real-alg :: numeral ..

174
lift-definition root-real-alg :: nat ⇒ real-alg ⇒ real-alg is root-3
by (simp add: root-3)

lemma root-real-alg: root n (real-of x) = real-of (root-real-alg n x)
by (transfer, rule root-3[symmetric])

lift-definition sgn-real-alg-rat :: real-alg ⇒ rat is sgn-3
by (insert sgn-3, metis to-rat-of-rat)

lemma sgn-real-alg-rat: real-of-rat (sgn-real-alg-rat x) = sgn (real-of x)
by (transfer, auto simp: sgn-3)

instantiation real-alg :: sgn
begin
definition sgn-real-alg :: real-alg ⇒ real-alg where
sgn-real-alg x = of-rat-real-alg (sgn-real-alg-rat x)
instance ..
end

lemma sgn-real-alg: sgn (real-of x) = real-of (sgn x)
unfolding sgn-real-alg-def of-rat-real-alg[symmetric]
by (transfer, simp add: sgn-3)

instantiation real-alg :: equal
begin
lift-definition equal-real-alg :: real-alg ⇒ real-alg ⇒ bool is equal-3
by (simp add: equal-3)
instance
proof
fix x y :: real-alg
show equal-class.equal x y = (x = y)
  by (transfer, simp add: equal-3)
qed
end

lemma equal-real-alg: HOL.equal (real-of x) (real-of y) = (x = y)
unfolding equal-real-def by (transfer, auto)

instantiation real-alg :: ord
begin

definition less-real-alg :: real-alg ⇒ real-alg ⇒ bool where
[code def]: less-real-alg x y = (real-of x < real-of y)
definition less-eq-real-alg :: real-alg ⇒ real-alg ⇒ bool where
  [code def]: less-eq-real-alg x y = (real-of x ≤ real-of y)

instance ..
end

lemma less-real-alg: less (real-of x) (real-of y) = (x < y) unfolding less-real-alg-def ..
lemma less-eq-real-alg: less-eq (real-of x) (real-of y) = (x ≤ y) unfolding less-eq-real-alg-def ..

instantiation real-alg :: compare-order
begin
  lift-definition compare-real-alg :: real-alg ⇒ real-alg ⇒ order is compare-3
  by (simp add: compare-3)

lemma compare-real-alg: compare (real-of x) (real-of y) = (compare x y)
  by (transfer, simp add: compare-3)
instance proof (intro-classes, unfold compare-real-alg[abs-def])
  show le-of-comp (λx y. compare (real-of x) (real-of y)) = (≤)
    by (intro ext, auto simp: compare-real-def comparator-of-def le-of-comp-def)
  show lt-of-comp (λx y. compare (real-of x) (real-of y)) = (〈)
    by (intro ext, auto simp: compare-real-def comparator-of-def lt-of-comp-def)
  show comparator (λx y. compare (real-of x) (real-of y))
    unfolding comparator-def
    proof (intro conjI impI allI)
      fix x y z :: real-alg
      let ?r = real-of
      note rc = comparator-compare[where 'a = real, unfolded comparator-def]
      from rc show invert-order (compare (?r x) (?r y)) = compare (?r y) (?r x)
        by blast
      from rc show compare (?r x) (?r y) = Lt ⇒ compare (?r y) (?r z) = Lt
        ⇒ compare (?r x) (?r z) = Lt by blast
      assume compare (?r x) (?r y) = Eq
      with rc have ?r x = ?r y by blast
      thus x = y unfolding real-of-inj.
    qed
    qed
end

lemma less-eq-real-alg-code[code]:
  (less-eq :: real-alg ⇒ real-alg ⇒ bool) = le-of-comp compare
  (less :: real-alg ⇒ real-alg ⇒ bool) = lt-of-comp compare
  by (rule ord-defs(1)[symmetric], rule ord-defs(2)[symmetric])

176
instantiation real-alg :: abs
begin

definition abs-real-alg :: real-alg \Rightarrow real-alg where
  abs-real-alg x = (if real-of x < 0 then uminus x else x)
instance ..
end

lemma abs-real-alg: abs (real-of x) = real-of (abs x)
  unfolding abs-real-alg-def abs-real-def if-distrib
  by (auto simp: uminus-real-alg)

lemma sgn-real-alg-sound: sgn x = (if x = 0 then 0 else if 0 < real-of x then 1 else -1)
  (is - = ?r)
proof
  have real-of (sgn x) = sgn (real-of x) by (simp add: sgn-real-alg)
  also have \ldots = real-of ?r unfolding sgn-real-def if-distrib
  by (auto simp: less-real-alg-def
       zero-real-alg-def one-real-alg-def of-rat-real-alg
       equal-real-alg[ symmetric] equal-real-alg[ symmetric]
       equal-real-def uminus-real-alg[ symmetric])
  finally show sgn x = ?r unfolding equal-real-alg[ symmetric] equal-real-def by simp
qed

lemma real-of-of-int: real-of-rat (rat-of-int z) = real-of (of-int z)
proof (cases z \geq 0)
  case True
  define n where n = nat z
  from True have z: z = int n unfolding n-def by simp
  show ?thesis unfolding z
    by (induct n, auto simp: zero-real-alg plus-real-alg[ symmetric] one-real-alg
           hom-distribs)
next
  case False
  define n where n = nat (-z)
  from False have z: z = - int n unfolding n-def by simp
  show ?thesis unfolding z
    by (induct n, auto simp: zero-real-alg plus-real-alg[ symmetric] one-real-alg
           uminus-real-alg[ symmetric]
           minus-real-alg[ symmetric] hom-distribs)
qed

instance real-alg :: linordered-field
apply standard
  apply (unfold less-eq-real-alg-def plus-real-alg[ symmetric], force)
apply (unfold abs-real-alg-def less-real-alg-def zero-real-alg[ symmetric], rule refl)
apply (unfold less-real-alg-def times-real-alg[ symmetric], force)
apply (rule sgn-real-alg-sound)
done

instantiation real-alg :: floor-ceiling
begin
lift-definition floor-real-alg :: real-alg ⇒ int is floor-3
  by (auto simp: floor-3)

lemma floor-real-alg: floor (real-of x) = floor x
  by (transfer, auto simp: floor-3)

instance
proof
  fix x :: real-alg
  show of-int ⌊x⌋ ≤ x ∧ x < of-int (⌈x⌉ + 1) unfolding floor-real-alg[symmetric]
    using floor-correct[of real-of x] unfolding less-eq-real-alg-def less-real-alg-def
    real-of-of-int[symmetric] by (auto simp: hom-distrib)
  hence x ≤ of-int (⌈x⌉ + 1) by auto
  thus ∃z. x ≤ of-int z by blast
qed
end

definition real-alg-of-real :: real ⇒ real-alg where
  real-alg-of-real x = (if (∃y. x = real-of y) then (THE y. x = real-of y) else 0)
lemma real-alg-of-real-code[code]: real-alg-of-real (real-of x) = x
  using real-of-inj unfolding real-alg-of-real-def by auto

lift-definition to-rat-real-alg-main :: real-alg ⇒ rat option is to-rat-3
  by (simp add: to-rat-3)
lemma to-rat-real-alg-main: to-rat-alg ⇒ rat option is to-rat-3
  by (transfer, simp add: to-rat-3)
definition to-rat-real-alg :: real-alg ⇒ rat where
to-rat-real-alg x = (case to-rat-real-alg-main x of Some q ⇒ q | None ⇒ 0)
definition is-rat-real-alg :: real-alg ⇒ bool where
  is-rat-real-alg x = (case to-rat-real-alg-main x of Some q ⇒ True | None ⇒ False)
lemma is-rat-real-alg: is-rat (real-of x) = (is-rat-real-alg x)
  unfolding is-rat-real-alg-def is-rat to-rat-real-alg-main by auto
lemma to-rat-real-alg: to-rat (real-of x) = (to-rat-real-alg x)
  unfolding to-rat to-rat-real-alg-def to-rat-real-alg-main by auto
8.3 Real Algebraic Numbers as Implementation for Real Numbers

lemmas real-alg-code-eqns =
  one-real-alg
  zero-real-alg
  uminus-real-alg
  root-real-alg
  minus-real-alg
  plus-real-alg
  times-real-alg
  inverse-real-alg
  divide-real-alg
  equal-real-alg
  less-real-alg
  less-eq-real-alg
  compare-real-alg
  sgn-real-alg
  abs-real-alg
  floor-real-alg
  is-rat-real-alg
  to-rat-real-alg

code_datatype real-of

declare [[code drop:
  plus :: real ⇒ real ⇒ real
  uminus :: real ⇒ real
  minus :: real ⇒ real ⇒ real
  times :: real ⇒ real ⇒ real
  inverse :: real ⇒ real
  divide :: real ⇒ real ⇒ real
  floor :: real ⇒ int
  HOL.equal :: real ⇒ real ⇒ bool
  compare :: real ⇒ real ⇒ order
  less-eq :: real ⇒ real ⇒ bool
  less :: real ⇒ real ⇒ bool
  0 :: real
  1 :: real
  sgn :: real ⇒ real
  abs :: real ⇒ real
  root]]

declare real-alg-code-eqns [code equation]

lemma Ratreal-code[code]:
  Ratreal = real-of o of-rat-real-alg
  by (transfer, transfer) (simp add: fun-eq-iff of-rat-2)

lemma real-of-post[code-post]: real-of (Real-Alg-Quotient (Real-Alg-Invariant (Rational

179
proof (transfer)
  fix x
  show real-of-3 (Real-Alg-Invariant (Rational x)) = real-of-rat x
    by (simp add: Real-Alg-Invariant-inverse real-of-3.rep-eq)
qed

eend

9 Real Roots

This theory contains an algorithm to determine the set of real roots of a rational polynomial. It further contains an algorithm which tries to determine the real roots of real-valued polynomial, which incorporates Yun-factorization and closed formulas for polynomials of degree 2.

theory Real-Roots
imports
  Real-Algebraic-Numbers
begin

hide-const (open) UnivPoly.coeff
hide-const (open) Module.smult

Division of integers, rounding to the upper value.

definition div-ceiling :: int ⇒ int ⇒ int where
  div-ceiling x y = (let q = x div y in if q * y = x then q else q + 1)

definition root-bound :: int poly ⇒ rat where
  root-bound p ≡ let
    n = degree p;
    m = 1 + div-ceiling (max-list-non-empty (map (λi. abs (coeff p i)) [0..<n]))
      (abs (lead-coeff p))
  — round to the next higher number $2^m$, so that bisection will
  — stay on integers for as long as possible
  in of-int (2 ^ (log-ceiling 2 m))

partial-function (tailrec) roots-of-2-main ::
  int poly ⇒ root-info ⇒ (rat ⇒ rat ⇒ nat) ⇒ (rat × rat)list ⇒ real-alg-2 list ⇒
  real-alg-2 list where
  [code]:
  roots-of-2-main p ri cr lrs rais = (case lrs of Nil ⇒ rais
    | (l,r) # lrs ⇒ let c = cr l r in
      if c = 0 then roots-of-2-main p ri cr lrs rais
      else if c = 1 then roots-of-2-main p ri cr lrs (real-alg-2'' ri p l r # rais)
      else let m = (l + r) / 2 in roots-of-2-main p ri cr ((m,r) # (l,m) # lrs) rais)

definition roots-of-2-irr :: int poly ⇒ real-alg-2 list where
  roots-of-2-irr p = (if degree p = 1
    then [Rational (Rat.Fract (- coeff p 0) (coeff p 1)) ] else
let ri = root-info p;
cr = root-info.1-r ri;
B = root-bound p
in (roots-of-2-main p ri cr [(−B,B)] ()))

lemma root-imp-deg-nonzero: assumes p ≠ 0 poly p x = 0
shows degree p ≠ 0
proof
assume degree p = 0
from degree0-coeffs[OF this] assms show False by auto
qed

lemma cauchy-root-bound: fixes x :: 'a :: real_normed_field
assumes x: poly p x = 0 and p: p ≠ 0
shows norm x ≤ 1 + max-list-non-empty (map (λ i. norm (coeff p i)) [0 ..< degree p])
/ norm (lead-coeff p) (is - ≤ - + ?max / ?nlc)
proof
let ?n = degree p
let ?p = coeff p
let ?lc = lead-coeff p
define ml where ml = ?max / ?nlc
from p have lc: ?lc ≠ 0 by auto
hence nlc: norm ?lc > 0 by auto
from root-imp-deg-nonzero[OF p x] have*: 0 ∈ set [0 ..< degree p] by auto
have 0 ≤ norm (?p 0) by simp
also have . . . ≤ ?max
by (rule max-list-non-empty, insert *, auto)
finally have max0: ?max ≥ 0.
with nlc have ml0: ml ≥ 0 unfolding ml-def by auto
hence easy: norm x ≤ 1 ⇒ ?thesis unfolding ml-def[symmetric] by auto
show ?thesis
proof (cases norm x ≤ 1)
  case True
  thus ?thesis using easy by auto
next
  case False
  hence nx: norm x > 1 by simp
  hence x0: x ≠ 0 by auto
  hence xn0: 0 < norm x ~ ?n by auto
  from x[unfolded poly-altdef] have x ~ ?n * ?lc = x ~ ?n * ?lc − (∑ i≤?n. x ~ i * ?p i)
    unfolding poly-altdef by (simp add: ac-simps)
  also have (∑ i≤?n. x ~ i * ?p i) = x ~ ?n * ?lc + (∑ i < ?n. x ~ i * ?p i)
    by (subst sum.remove[of - ?n], auto intro: sum.cong)
  finally have x ~ ?n * ?lc = − (∑ i < ?n. x ~ i * ?p i) by simp
  with lc have x ~ ?n = − (∑ i < ?n. x ~ i * ?p i) / ?lc by (simp add: field-simps)
  from arg-cong[OF this, of norm]
have \( \text{norm } x \sim ?n = \text{norm } (\sum_{i < ?n} x \sim i \ast ?p i) / ?lc) \) unfolding \( \text{norm-power} \) by \( \text{simp} \)
also have \( (\sum_{i < ?n} x \sim i \ast ?p i) / ?lc = (\sum_{i < ?n} x \sim i \ast ?p i / ?lc) \)
by (rule sum-divide-distrib)
also have \( \text{norm } \ldots \leq (\sum_{i < ?n} \text{norm } (x \sim i \ast (?p i / ?lc))) \)
by (simp add: field-simps, rule \text{norm-sum})
also have \( \ldots = (\sum_{i < ?n} \text{norm } x \sim i \ast \text{norm } (?p i / ?lc)) \)
unfolding \( \text{norm-mult} \), \( \text{norm-power} \)
also have \( \ldots \leq (\sum_{i < ?n} \text{norm } x \sim i \ast ml) \)
proof (rule sum-mono)
fix \( i \)
assume \( i \in \{..<?n\} \)
hence \( i : i < ?n \) by simp
show \( \text{norm } x \sim i \ast \text{norm } (?p i / ?lc) \leq \text{norm } x \sim i \ast ml \)
proof (rule \text{mult-left-mono})
show \( 0 \leq \text{norm } x \sim i \) using \( ?x \) by auto
show \( \text{norm } (?p i / ?lc) \leq ml \) unfolding \( \text{norm-divide} \), \( ml-def \)
by (rule \text{divide-right-mono}[OF max-list-non-empty], insert \( ?lc i \), auto)
qed
qed
also have \( \ldots = ml \ast (\sum_{i < ?n} \text{norm } x \sim i) \)
unfolding \( \text{sum-distrib-right}[\text{symmetric}] \) by simp
also have \( \sum_{i < ?n} \text{norm } x \sim i = (\text{norm } x \sim ?n - 1) / (\text{norm } x - 1) \)
by (rule \text{geometric-sum}, insert \( nx \), auto)
finally have \( \text{norm } x \sim ?n \leq ml \ast (\text{norm } x \sim ?n - 1) / (\text{norm } x - 1) \) by simp
from \( \text{mult-left-mono}[OF this, of \text{norm } x - 1] \)
have \( (\text{norm } x - 1) \ast (\text{norm } x \sim ?n) \leq ml \ast (\text{norm } x \sim ?n - 1) \) using \( ?x \) by auto
also have \( \ldots = (ml \ast (1 - 1 / (\text{norm } x \sim ?n))) \ast \text{norm } x \sim ?n \)
using \( ?x \) False \( ?x0 \) by (simp add: field-simps)
finally have \( (\text{norm } x - 1) \ast (\text{norm } x \sim ?n) \leq (ml \ast (1 - 1 / (\text{norm } x \sim ?n))) \)
* \( \text{norm } x \sim ?n \).
from \( \text{mult-right-le-imp-le}[OF this \ xn0] \)
have \( \text{norm } x - 1 \leq ml \ast (1 - 1 / (\text{norm } x \sim ?n)) \) by simp
hence \( \text{norm } x \leq 1 + ml - ml / (\text{norm } x \sim ?n) \) by (simp add: field-simps)
also have \( \ldots \leq 1 + ml \) using \( ml0 \) \( xn0 \) by auto
finally show \( ?thesis \) unfolding \( ml-def \).
qed
qed
lemma \( \text{div-le-div-ceiling} : x \text{ div } y \leq \text{div-ceiling } x y \)
unfolding \( \text{div-ceiling-def} \), \( \text{Let-def} \) by auto

lemma \( \text{div-ceiling} : \text{assumes } q : q \neq 0 \)
shows \( (\text{of-int } x :: \text{ floor-ceiling}) \leq \text{of-int } q \leq \text{of-int } (\text{div-ceiling } x q) \)
proof (cases \( q \text{ dvd } x \))
case True
then obtain \( k \) where \( xqk : x = q \ast k \) unfolding \( \text{dvd-def} \) by auto
unique xqk using q by simp

next

case False
{
  assume x div q * q = x
  hence x = q * (x div q) by (simp add: ac-simps)
  hence q dvd x unfolding dvd-def by auto
  with False have False by simp
}

hence id: div-ceiling x q = x div q + 1 unfolding div-ceiling-def Let-def using q by auto

show ?thesis unfolding id unfolding xqk using q by simp

qed

lemma max-list-non-empty-map: assumes hom: \( \forall x y. \max(f x) (f y) = f(\max x y) \)

shows zs \( \neq [] \implies \max\text{-list-non-empty}(\map f zs) = f(\max\text{-list-non-empty}zs) \)

by (induct zs rule: max-list-non-empty.induct, auto simp: hom)

lemma root-bound: assumes root-bound p = B and deg: degree p > 0

shows \( \ipoly p (x :: \text{real}) = 0 \implies \norm x \leq \of-rat B B \geq 0 \)

proof -
  let ?r = real-of-rat
  let ?i = real-of-int
  let ?p = real-of-int-poly p
  define n where n = degree p
  let ?lc = coeff p n
  let ?list = map (\xi. abs (coeff p \xi)) [0..<n]
  let ?list' = (map (\xi. ?i (abs (coeff p \xi)))) [0..<n]
  define m where m = max-list-non-empty ?list
  define m-up where m-up = 1 + div-ceiling m (abs ?lc)
  define C where C = rat-of-int (2 \^ log-ceiling 2 m-up)
  from deg have p0: p \( \neq 0 \) by auto
  from p0 have alc0: abs ?lc \( \neq 0 \) unfolding n-def by auto
  from deg have mem: abs (coeff p 0) \( \in \) set ?list unfolding n-def by auto
  from max-list-non-empty[OF this, folded m-def]
  have m0: m \( \geq 0 \) by auto
  have div-ceiling m (abs ?lc) \( \geq 0 \)
  by (rule order-trans[OF div-le-div-ceiling[of m abs ?lc]], subst
  pos-imp-zdiv-nonneg-iff, insert p0 m0, auto simp: n-def)
  hence m-up \( \geq 1 \) unfolding m-up-def by auto
  have m-up \( \leq 2 \^ (\log-ceiling 2 m-up) \) using m-up log-ceiling-sound(1) by auto
  hence Cmap: C \( \geq \) of-int m-up unfolding C-def by linarith
  with map have C: C \( \geq 1 \) by auto
  from assms(1)[unfolded root-bound-def Let-def]
  have B: C = of-rat B unfolding C-def m-up-def n-def m-def by auto
  note dc = div-le-div-ceiling[of m abs ?lc]
with $C$ show $B \geq 0$ unfolding $B$ by auto
assume ipoly $p$ $x = 0$
hence $rt$: poly $?p$ $x = 0$ by simp
from root-imp-deg-nonzero[OF - this] $p0$ have $n0$: $n \neq 0$ unfolding $n$-def by auto
from cauchy-root-bound[OF $rt$ $p0$]
have $rt$:
  norm $x$ \leq $1 + max-list-non-empty $?list' / $?i (abs $?lc)
by (simp add: $n$-def)
also have $?list' = map $?i $?list$ by simp
also have max-list-non-empty $\ldots = $?i $m$ unfolding $m$-def
by (rule max-list-non-empty-map, insert mem, auto)
also have $1 + m / $?i (abs $?lc) \leq $?i $m$-up
unfolding $m$-up-def using div-ceiling[OF alc0, of $m$] by auto
finally have norm $x \leq $?r $C$,
thus norm $x \leq $?r $B$ unfolding $B$ by simp
qed

fun pairwise-disjoint :: 'a set list \Rightarrow bool
where
  pairwise-disjoint [] = True
| pairwise-disjoint ($x$ # $xs$) = (($x$ \cap ($\bigcup\{y | y \in set $xs$. y\} = \{\})) \land pairwise-disjoint $xs$)

lemma roots-of-2-irr: assumes $pc$: poly-cond $p$ and $deg$: degree $p > 0$
shows real-of-int $c$ \in Ball (set (roots-of-2-irr $p$)) (is $?one$)
  Ball (set (roots-of-2-irr $p$)) invariant-2 (is $?two$)
  distinct (map real-of-int (roots-of-2-irr $p$)) (is $?three$)
proof -
  note $d$ = roots-of-2-irr-def
from poly-condD[OF $pc$]
have $mon$: lead-coeff $p > 0$ and $irr$: irreducible $p$ by auto
let $?norm = real-alg-2'$
have $?one \land $?two \land $?three
proof (cases degree $p = 1$)
  case True
  define $c$ where $c = $coeff $p$ 0
  define $d$ where $d = $coeff $p$ 1
  from True have $rr$: roots-of-2-irr $p = [\text{Rational (Rat.Frat (- c) (d))}]$
  unfolding $d$-def $c$-def by auto
  from degree1-coefs[OF $True$] have $p$: $p = [x,d]$ and $d$: $d \neq 0$
  unfolding $c$-def $d$-def by auto
  have $*: real-of-int $c$ + $x$ * real-of-int $d = 0 \Rightarrow x = (real-of-int $c$ / real-of-int $d$)
f for $x$
  using $d$ by (simp add: field-simps)
  show $?thesis$ unfolding $rr$ using $d$ * unfolding $p$ using of-rat-1[of $\text{Rat.Frat (- c) (d)}$]
  by (auto simp: Fract-of-int-quotient hom-distribs)
next
  case False

184
let \(?r\) = real-of-rat
let \(?rp\) = map-poly \(?r\)
define ri where ri = root-info p
define cr where cr = root-info.l-r ri
define bnds where bnds = \([\{\text{root-bound } p, \text{root-bound } p\}]\)
define empty where empty = \((\text{Nil :: real-alg-2 list})\)

have empty: Ball (set empty) invariant-2 \& distinct (map real-of-2 empty)

unfolding empty-def by auto
from mon have \(p\): \(p \neq 0\) by auto
from root-info[OF irr_deg] have ri: root-info-cond ri \(p\) unfolding ri-def .
from False
have rr: roots-of-2-irr \(p\) = roots-of-2-main \(p\) ri cr bnds empty
  unfolding d ri-def cr-def Let-def bnds-def empty-def

have bnds: \(\bigwedge l r. \ (l, r) \in \text{set bnds} \implies \ l \leq r\)
unfolding bnds-def by auto

have ipoly \(p\) \(x = 0\) \implies \ (?r (- root-bound \(p\)) \(\leq x \land x \leq \ ?r\) (root-bound \(p\)) \text{ for } \(x\))
using root-bound(\(1\))|\(of \ x\) by (auto simp: hom-distribs)

hence rts: \(\{x. \ \text{ipoly } \(p\) \(x = 0\)\}\)
= real-of-2 \(\cdot\) set empty \(\cup\) \(\{x. \ \exists l r. \ \text{root-cond } (p, l, r) \ \text{x} \ \land (l, r) \in \text{set bnds}\}\)
unfolding empty-def bnds-def by (force simp: root-cond-def)
define rts where rts lr = Collect (root-cond \((p, lr)\)) \text{ for } lr
have disj: pairwise-disjoint (real-of-2 \(\cdot\) set empty \# map rts bnds)
unfolding empty-def bnds-def by auto
from deg False have deg1: degree \(p\) \(> 1\) by auto
define delta where delta = ipoly-root-delta \(p\)
note delta = ipoly-root-delta[OF \(p\), folded delta-def]
define rel' where rel' = \(\{(x, y). \ 0 \leq y \land \text{delta-gt } x y\}\)^{-1}
define mm where mm = \((\lambda \text{bnds. mset (map } (\lambda l, r). \ ?r - \ ?r l) \text{ bnds)\})
define rel where rel = inv-image (multI rel') mm
have wf: \(\text{wf rel unfolding rel'-def}\)
by (rule wf-inv-image[OF \(\text{wlf-multI [OF SN-imp-wf [OF delta-gt-SN [OF delta(I)]]]]}\])
let \(?main\) = roots-of-2-main \(p\) ri cr
have real-of-2 \(\cdot\) set (?main bnds empty) =
  real-of-2 \(\cdot\) set empty \(\cup\)
  \(\{x. \ \exists l r. \ \text{root-cond } (p, l, r) \ \text{X} \ \land (l, r) \in \text{set bnds}\}\) \land
  Ball (set (?main bnds empty)) invariant-2 \& distinct (map real-of-2 (?main bnds empty)) \(\text{(is \ ?one' \ \& \ ?two' \ \& \ ?three')}\)
using empty bnds disj

proof (induct bnds arbitrary: empty rule: wf-induct[OF wf])
case (1 lrs ss)

note rais = I(2)[rule-format]
note lrs = I(3)
note disj = I(4)
note IH = I(1)[rule-format]

note simp = roots-of-2-main.simps[of \(p\) \(p\) \(p\) \(cr\) \(cr\) \(rais\)]
show ?case

proof (cases lrs)
case Nil
  with rais show \(?thesis unfolding simp by auto
next
  case (Cons \(l \cdot lrs\))
  obtain \(l \cdot r\) where \(lr' \cdot lr = (l, r)\) by force
  \{ fix \(lr'\)
    assume \(lt:: l' \cdot r'. (l', r') \in set lr' \implies l' \leq r' \land \text{delta-gt} \delta (\?r r - ?r l) (\?r r' - ?r l')\)
    have \(l:: \text{mm} (lr' \cdot lrs) = \text{mm} lrs + \text{mm} lr'\) unfolding mm-def by (auto simp: ac-simps)
    have \(r:: \text{mm} lrss = \text{mm} lrs + \{\# ?r r - ?r l \#\}\) unfolding Cons lr'
    rel-def mm-def
    by auto
    have \(\text{mm} (lr' \cdot lrs), \text{mm lrss} \in \text{mult1 rel' unfolding l r mult1-def}\)
    proof (rule, unfold split, intro ex1 conj1, unfold add-mset-add-single[symmetric],
      rule refl, rule refl, intro allI impl)
      fix \(d\)
      assume \(d \in \# \text{mm lr'}\)
      then obtain \(l' \cdot r'\) where \(d:: d = \?r r' - \?r l'\) and \(lrs:: (l', r') \in set lr'\)
        unfolding mm-def in-multiset-in-set by auto
        from \(lt(OF lr']\)
        show \((d, \?r r - \?r l) \in rel'\) unfolding d rel'-def
        by (auto simp: of-rat-less-eq)
    qed
    hence \((lr' \cdot lrs, lrss) \in rel unfolding rel-def by auto\)
  \} note rel = this
  from rel[of Nil] have easy-rel: \((lrs, lrss) \in rel by auto\)
  define \(c:: c = cr l r\)
  from simp Cons lr' have simp: \(?main lrs rais =\)
    \((\text{if} c = 0 \text{ then } \text{?main lrs rais else if } c = 1 \text{ then }\)
    \(\text{?main lrs (real-\text{alg-2'} ri p l r \# rais)}\)
    \(\text{else let } m = (1 + r) / 2 \text{ in } \text{?main ((m, r) \# (l, m) \# lrs) rais}\)\)
    unfolding c-def simp Cons lr' using real-\text{alg-2'}(OF False] by auto
  note lrs = lrs[unfolded Cons lr']
  from lrs have \(lr:: l \leq r\) by auto
  from root-info-condD(1)[OF ri lr, folded cr-def]
  have \(c:: c = \text{card} \{x. \text{root-cond} (p, l, r) x\}\) unfolding c-def by auto
  let \(?rt = \lambda lrs. \{x. \exists l r. \text{root-cond} (p, l, r) x \land (l, r) \in set lrs\}\)
  have \(\text{rts:: rt lrs = rt lrs \cup \{x. \text{root-cond} (p, l, r) x\}}\) \((\text{is } \text{?rt1 = ?rt2 \cup ?rt3}\)\)
    unfolding Cons lr' by auto
  show \(?thesis\)
  proof (cases c = 0)
    case True
    with simp have simp: \(?main lrs rais = ?main lrs rais by simp\)
    from disj have disj: pairwise-disjoint (real-of-2' set rais \# map rts lrs)
      unfolding Cons by auto
    from finite-iopoly-roots[OF p] True[unfolded c] have empty: \(?rt3 = \{}\)
      unfolding root-cond-def[abs-def] split by simp
  186
with rts have rts: ?rt1 = ?rt2 by auto
show ?thesis unfolding simp rts
by (rule IH[OF easy-rel rais lrs disj], auto)

next
  case False
  show ?thesis
  proof (cases c = 1)
    case True
    let ?rai = real-alg-2' ri p l r
    from True simp have simp: ?main lrs rais = ?main lrs (?rai ≠ rais)
    by auto
    from card-1-Collect-ext[OF c[symmetric, unfolded True]]
    have ur: unique-root (p,l,r) .
    from real-alg-2'OF ur pc ri
    have rai: invariant-2 ?rai real-of-2 ?rai = the-unique-root (p,l,r)
    by auto
    have rais: ?rai ∈ set (real-alg-2' # rais)
    using rc1 ur rai by (auto intro: the-unique-root-eqI theI')
    have Ball (set (real-alg-2' ri p l r # rais)) invariant-2 ∧
    distinct (map real-of-2 (roots-of-2-main p ri cr lrs (?rai ≠ rais)))
    (is (one ∧ ?two ∧ ?three)
    proof (rule IH[OF easy-rel, OF ?rai ≠ rais, OF conjI lrs])
      have Ball (set (real-alg-2' ri p l r # rais)) invariant-2 using rais by auto
      have real-of-2 (real-alg-2' ri p l r) /∈ set (map real-of-2 rais)
      using disj rt3 unfolding Cons br' rts-def by auto
      thus distinct (map real-of-2 (real-alg-2' ri p l r # rais)) using dist by auto
      show pairwise-disjoint (real-of-2 ' set (real-alg-2' ri p l r # rais) #
      map rts lrs)
      using disj rt3 unfolding Cons br' rts-def by auto
      qed auto
      hence ?one ?two ?three by blast+
      show ?thesis unfolding simp rts rt3
      by (rule conjI[OF OF - conjI[of ⟨?two⟩ ⟨?three⟩]], unfold ⟨?one⟩, auto)
  next
    case False
    let ?m = (l+r)/2
    let ?lrs = [(?m,r),(l,?m)] @ lrs
    from False c ≠ 0 have simp: ?main lrs rais = ?main lrs rais
    unfolding simp by (auto simp: Let-def)
    from False c ≠ 0 have c ≥ 2 by auto
    from delta(2)[OF this[unfolded c]] have delta: delta ≤ ?r (r - l) / 4 by auto
    have lrs: ?l. (l,r) ∈ set ?lrs → l ≤ r
using lr lrs by (fastforce simp: field-simps)
have ?r ?m ∈ Q unfolding Rats-def by blast
with poly-cond-degree- gt-1[OF pc deg1, of ?r ?m]
have disj1: ?r ?m ∉ rts lr for lr unfolding rts-def root-cond-def by auto
have disj2: rts (?m, r) ∩ rts (l, ?m) = {} using disj1[of (l,?m)] disj1[of (?m,r)]

unfolding rts-def root-cond-def by auto
have disj3: (rts (l,?m) ∪ rts (?m,r)) = rts (l,r) unfolding rts-def root-cond-def by (auto simp: hom-distribs)

have disj4: real-of-2 : set rais ∩ rts (l,r) = {} using disj unfolding Cons lr' by auto

have disj: pairwise-disjoint (real-of-2 ; set rais ≠ map rts ([|?[m,,r] (l,,?m)|] @ lrs)) using disj disj2 disj3 disj4 by (auto simp: Cons lr')

have (?[lrs,lrs]) ∈ rel proof (rule rel, intro conjI)
fix l' r'
assume mem: (l', r') ∈ set [|?[m,r] (l,?m)]
from mem lr show l' ≤ r' by auto
from mem have diff: (?r r' - ?r l) / 2 by auto
(auto simp: eq-diff-eq minus-diff-eq mult2-right of-rat-add of-rat-mult of-rat-numeral-eq)

show delta-gt delta (?[r r' - ?r l] (?[r r' - ?r l]) unfolding diff
delta-gt-def by (rule order.trans[OF delta], insert lr,
auto simp: field-simps of-rat-diff of-rat-eq)

qed

note IH = IH[OF this, of rais, OF rais lrs disj]

have real-of-2 ; set (?main ?lrs rais) =
real-of-2 ; set rais ∩ ?rt lrs ∧
Ball (set (?main ?lrs rais)) invariant-2 ∧ distinct (map real-of-2 (?main ?lrs rais))
(is ?one ∧ ?two)
by (rule IH)

hence ?one ?two by blast

have cong: a b c. b = c ⟹ a ∪ b = a ∪ c by auto

have id: ?rt lrs = ?rt lrs ∪ ?rt [|?[m,r] (l,?m)] by auto

show ?thesis unfolding rts simp (?one ?two id

proof (rule conjI[OF cong[OF cong conjI]])

have ∨ x. root-cond (p,l) x = (root-cond (p,l,?m) x ∨ root-cond (p,?m,r) x)

unfolding root-cond-def by (auto simp: hom-distribs)

hence id: Collect (root-cond (p,l,r)) = [x. (root-cond (p,l,?m) x ∨ root-cond (p,?m,r) x)]
by auto

show ?rt [|?[m,r] (l,?m)] = Collect (root-cond (p,l,r)) unfolding id
list.simps by blast

show ∀ a ∈ set (?main ?lrs rais). invariant-2 a using (?two a) by auto

show distinct (map real-of-2 (?main ?lrs rais)) using (?two a) by auto

qed
qed
qed
qed

hence idd: ?one' and cond: ?two' ?three' by blast+
define res where res = roots-of-2-main p ri cr bnds empty
have e: set empty = {} unfolding empty-def by auto
from idd[folded res-def] e have idd: ?one ?two ?three
by auto
show ?thesis
unfolding rr unfolding rts id e norm-def using cond
unfolding res-def[symmetric] image-empty e idd[symmetric] by auto
qed
thus ?one ?two ?three by blast+
qed

definition roots-of-2 :: int poly ⇒ real-alg-2 list where
roots-of-2 p = concat (map roots-of-2-irr
(factors-of-int-poly p))

lemma roots-of-2:
shows p ≠ 0 ⇒ real-of-2 ' set (roots-of-2 p) = {x. ipoly p x = 0}
Ball (set (roots-of-2 p)) invariant-2
distinct (map real-of-2 (roots-of-2 p))
proof –
let ?rr = roots-of-2 p
note d = roots-of-2-def
note frp1 = factors-of-int-poly
{
  fix q r
  assume q ∈ set ?rr
  then obtain s where
  s: s ∈ set (factors-of-int-poly p) and
  q: q ∈ set (roots-of-2-irr s)
  unfolding d by auto
  from frp1(1)[OF refl s] have poly-cond s degree s > 0 by (auto simp: poly-cond-def)
  from roots-of-2-irr[OF this] q
  have invariant-2 q by auto
}
thus Ball (set ?rr) invariant-2 by auto
{
  assume p: p ≠ 0
  have real-of-2 ' set ?rr = (⋃ ((λ p. real-of-2 ' set (roots-of-2-irr p)) ' (set (factors-of-int-poly p))))
  (is - = ?rrr)
  unfolding d set-concat set-map by auto
  also have . . . = {x. ipoly p x = 0}

189
proof -
{ 
  fix x
  assume x ∈ ?rrr
  then obtain q s where 
  s: s ∈ set (factors-of-int-poly p) and 
  q: q ∈ set (roots-of-2-irr s) and 
  x: x = real-of-2 q by auto
  from frp1(1)[OF refl s] have s0: s ≠ 0 and pt: poly-cond s degree s > 0 
  by (auto simp: poly-cond-def)
  from roots-of-2-irr[OF pt] q have rt: ipoly s x = 0 unfolding x by auto
  from frp1(2)[OF refl p, of x] rt s have rt: ipoly p x = 0 by auto
} moreover
{ 
  fix x :: real
  assume rt: ipoly p x = 0
  from rt frp1(2)[OF refl p] obtain s where s: s ∈ set (factors-of-int-poly p) 
  and rt: ipoly s x = 0 by auto
  from frp1(1)[OF refl s] have s0: s ≠ 0 and ty: poly-cond s degree s > 0 
  by (auto simp: poly-cond-def)
  from roots-of-2-irr(1)[OF ty] rt obtain q where 
  q: q ∈ set (roots-of-2-irr s) and 
  x: x = real-of-2 q by blast
  have x ∈ ?rrr unfolding x using q s by auto
} ultimately show ?thesis by auto
qed
finally show real-of-2 · set ?rr = {x. ipoly p x = 0} by auto
} 
show distinct (map real-of-2 (roots-of-2 p))
proof (cases p = 0)
case True
by auto
next
next p: False
note frp1 = frp1[OF refl]
let ?fp = factors-of-int-poly p
let ?cc = concat (map roots-of-2-irr ?fp)
show ?thesis unfolding roots-of-2-def distinct-conv-nth length-map
proof (intro allI impl notI)
fix i j
  assume ij: i < length ?cc j < length ?cc i ≠ j and id: map real-of-2 ?cc ! i 
= map real-of-2 ?cc ! j
  from ij id have id: real-of-2 (?cc ! i) = real-of-2 (?cc ! j) by auto
  from nth-concat-diff[OF ij, unfolded length-map] obtain j1 k1 j2 k2 where 
  s: (j1,k1) ≠ (j2,k2)
  show
\[ j_1 < \text{length} \ ?fp \ j_2 < \text{length} \ ?fp \ \text{and} \]
\[ k_1 < \text{length} \ (\text{map roots-of-2-irr} \ ?fp \ ! j_1) \]
\[ k_2 < \text{length} \ (\text{map roots-of-2-irr} \ ?fp \ ! j_2) \]
\[ \forall cc \ ! i = \text{map roots-of-2-irr} \ ?fp \ ! j_1 \ ! k_1 \]
\[ \forall cc \ ! j = \text{map roots-of-2-irr} \ ?fp \ ! j_2 \ ! k_2 \ \text{by blast} \]

**hence**: \( k_1 < \text{length} \ (\text{roots-of-2-irr} \ ?fp \ ! j_1) \)
\( k_2 < \text{length} \ (\text{roots-of-2-irr} \ ?fp \ ! j_2) \)
\[ \forall cc \ ! i = \text{roots-of-2-irr} \ ?fp \ ! j_1 \ ! k_1 \]
\[ \forall cc \ ! j = \text{roots-of-2-irr} \ ?fp \ ! j_2 \ ! k_2 \]

by auto

from \(*\) have \( \text{mem:} \ ?fp \ ! j_1 \in \text{set} \ ?fp \ ?fp \ ! j_2 \in \text{set} \ ?fp \ \text{by auto} \)
from \(\text{frp1(1)[OF mem1(1)] \ frp1(1)[OF mem2(2)]}\)
have \(pc1: \text{poly-cond} \ (\ ?fp \ ! j_1) \ degree \ (\ ?fp \ ! j_1) > 0 \) and \(pc10: \ ?fp \ ! j_1 \neq 0 \)
and \(pc2: \text{poly-cond} \ (\ ?fp \ ! j_2) \ degree \ (\ ?fp \ ! j_2) > 0 \)
by (auto simp: poly-cond-def)

show False

proof (cases \( j_1 = j_2 \))
  case True
  with \(*\) have \( \text{neq:} \ k_1 \neq k_2 \ \text{by auto} \)
  from \(*\)\[\text{unfolded True}\] id *
  \[\text{have} \ \text{map real-of-2} \ (\text{roots-of-2-irr} \ ?fp \ ! j_2)) ! k_1 = \text{real-of-2} \ (?cc ! j) \]
  \[\text{map real-of-2} \ (\text{roots-of-2-irr} \ ?fp \ ! j_2)) ! k_1 = \text{real-of-2} \ (?cc ! j) \]
  by auto

  hence \(\text{distinct} \ (\text{map real-of-2} \ (\text{roots-of-2-irr} \ ?fp \ ! j_2))\)

  unfolding \(\text{distinct-conv-nth \ using} \ \(*)\ \text{True} \ \text{by auto} \)

  with \(\text{roots-of-2-irr(3)[OF \ pc2]}\) show False by auto

next
  case neq: False
  with \(\text{frp1(4)[of \ p]} \)\[\text{* have} \ \text{neq:} \ ?fp \ ! j_1 \neq ?fp \ ! j_2 \ \text{unfolding \ distinct-conv-nth} \]
  by auto
  let \(?x = \text{real-of-2} \ (?cc ! i) \)
  define \(x \ \text{where} \ x = ?x \)
  from \(*\)\[\text{have} \ x \in \text{real-of-2} \ \text{set} \ (\text{roots-of-2-irr} \ ?fp \ ! j_1) \ \text{unfolding \ x-def} \]
  by auto
  with \(\text{roots-of-2-irr(1)[OF \ pc1]}\) have \(x1: \text{ipoly} \ (\ ?fp \ ! j_1) \ x = 0 \ \text{by auto} \)
  from \(*\)\[\text{id have} \ x \in \text{real-of-2} \ \text{set} \ (\text{roots-of-2-irr} \ ?fp \ ! j_2) \ \text{unfolding \ x-def} \]
  by (metis \text{image-eqI} \text{nth-mem})
  with \(\text{roots-of-2-irr(1)[OF \ pc2]}\) have \(x2: \text{ipoly} \ (\ ?fp \ ! j_2) \ x = 0 \ \text{by auto} \)
  have \(\text{ipoly} \ p \ x = 0 \ \text{using} \ x1 \ \text{mem \ unfolding} \ \text{roots-of-2-def \ by} \ (\text{metis} \ \text{frp1(2) \ p}) \)
  from \(\text{frp1(3)[OF \ p \ this]}\) \(x1 \ x2 \ \text{neq} \ \text{mem} \ \text{show False \ by \ blast} \)
  qed
  qed
  qed

lift-definition \[\text{roots-of-3 ::} \ \text{int poly} \Rightarrow \text{real-alg-3 list is roots-of-2} \]
by (insert \text{roots-of-2, auto simp: list-all-iff})
lemma roots-of-3:
  shows \( p \neq 0 \implies \text{real-of-3 } \set \text{(roots-of-3 } p) = \{ x. \text{ipoly } p \ x = 0 \} \)
  distinct (map real-of-3 (roots-of-3 } p))
proof –
  show \( p \neq 0 \implies \text{real-of-3 } \set \text{(roots-of-3 } p) = \{ x. \text{ipoly } p \ x = 0 \} \)
    by (transfer; intro roots-of-2, auto)
  show distinct (map real-of-3 (roots-of-3 } p))
    by (transfer; insert roots-of-2, auto)
qed

lift-definition roots-of-real-alg :: \text{int poly } \Rightarrow \text{real-alg list } \text{is roots-of-3 } .

lemma roots-of-real-alg:
  \( p \neq 0 \implies \text{real-of } \set \text{(roots-of-real-alg } p) = \{ x. \text{ipoly } p \ x = 0 \} \)
  distinct (map real-of (roots-of-real-alg } p))
proof –
  show \( p \neq 0 \implies \text{real-of } \set \text{(roots-of-real-alg } p) = \{ x. \text{ipoly } p \ x = 0 \} \)
    by (transfer', insert roots-of-3, auto)
  show distinct (map real-of (roots-of-real-alg } p))
    by (transfer, insert roots-of-3(2), auto)
qed

It follows an implementation for \text{roots-of-3}, since the current definition
does not provide a code equation.

color{context}
begin

private typedef real-alg-2-list = \{ \text{xs.Ball}(\text{set xs}) \text{invariant-2} \} by (intro exI[of - Nil], auto)

setup-lifting type-definition-real-alg-2-list

private lift-definition roots-of-2-list :: \text{int poly } \Rightarrow \text{real-alg-2-list } \text{is roots-of-2}
  by (insert roots-of-2, auto)
private lift-definition real-alg-2-list-nil :: \text{real-alg-2-list } \Rightarrow \text{bool } \lambda \text{xs. case xs of Nil } \Rightarrow \text{True } | - \Rightarrow \text{False } .

private fun real-alg-2-list-hd-intern :: \text{real-alg-2 list } \Rightarrow \text{real-alg-2 where}
  \text{real-alg-2-list-hd-intern}(\text{Cons } x \text{xs}) = x
  | \text{real-alg-2-list-hd-intern Nil} = \text{of-rat-2 } 0

private lift-definition real-alg-2-list-hd :: \text{real-alg-2-list } \Rightarrow \text{real-alg-3 } \text{is real-alg-2-list-hd-intern}
proof (goal-cases)
  case (1 \text{xs})
  thus \( q \text{case using of-rat-2[of } 0 \) by (cases \text{xs}, auto)
qed

private lift-definition real-alg-2-list-tl :: \text{real-alg-2-list } \Rightarrow \text{real-alg-2-list } \text{is tl}
proof (goal-cases)

192
case (1 xs)
thus ?case by (cases xs, auto)
qed

private lift-definition real-alg-2-list-length :: real-alg-2-list ⇒ nat is length.

private lemma real-alg-2-list-length[simp]: ¬ real-alg-2-list-nil xs ⇒ real-alg-2-list-length (real-alg-2-list-tl xs) < real-alg-2-list-length xs
  by (transfer, auto split: list.splits)

private function real-alg-2-list-convert :: real-alg-2-list ⇒ real-alg-3 list where
  real-alg-2-list-convert xs = (if real-alg-2-list-nil xs then [] else real-alg-2-list-hd xs
  ≠ real-alg-2-list-convert (real-alg-2-list-tl xs)) by pat-completeness auto

termination by (relation measure real-alg-2-list-length, auto)

private definition roots-of-3-impl :: int poly ⇒ real-alg-3 list where
  roots-of-3-impl p = real-alg-2-list-convert (roots-of-2-list p)

private lift-definition real-alg-2-list-convert-id :: real-alg-2-list ⇒ real-alg-3 list is id
  by (auto simp: list-all-iff)

lemma real-alg-2-list-convert: real-alg-2-list-convert xs = real-alg-2-list-convert-id xs
proof (induct xs rule: wf-induct[OF wf-measure[of real-alg-2-list-length], rule-format])
  case (1 xs)
  show ?case
    proof (cases real-alg-2-list-nil xs)
    case True
    hence real-alg-2-list-convert xs = [] by auto
    also have [] = real-alg-2-list-convert-id xs using True
      by (transfer', auto split: list.splits)
    finally show ?thesis.
    next
    case False
    hence real-alg-2-list-convert xs = real-alg-2-list-hd xs ≠ real-alg-2-list-convert (real-alg-2-list-tl xs) by simp
    also have real-alg-2-list-convert (real-alg-2-list-tl xs) = real-alg-2-list-convert-id (real-alg-2-list-tl xs)
      by (rule 1, insert False, simp)
    also have real-alg-2-list-hd xs ≠ ... = real-alg-2-list-convert-id xs using False
      by (transfer', auto split: list.splits)
    finally show ?thesis.
    qed
  qed

lemma roots-of-3-code[code]: roots-of-3 p = roots-of-3-impl p
unfolding roots-of-3-impl-def real-alg-2-list-convert
by (transfer, simp)
end

definition real-roots-of-int-poly :: int poly ⇒ real list where
real-roots-of-int-poly p = map real-of (roots-of-real-alk p)

definition real-roots-of-rat-poly :: rat poly ⇒ real list where
real-roots-of-rat-poly p = map real-of (roots-of-real-alk (snd (rat-to-int-poly p)))

abbreviation rpoly :: rat poly ⇒ 'a :: field-char-0 ⇒ 'a
where rpoly f ≡ poly (map-poly to-rat f)

lemma real-roots-of-int-poly: p ≠ 0 ⇒ set (real-roots-of-int-poly p) = {x. ipoly p x = 0}
distinct (real-roots-of-int-poly)
proof -
obtain c q where cq: rat-to-int-poly p = (c,q) by force
from rat-to-int-poly[OF this]
have pq: p = smult (inverse (of-int c)) (of-int-poly q)
  and c ≠ 0 by auto
have id: {x. rpoly p x = (0 :: real)} = {x. ipoly q x = 0}
  unfolding pq by (simp add: c of-rat-of-int-poly hom-distribs)
show distinct (real-roots-of-rat-poly p) unfolding real-roots-of-rat-poly-def cq
  snd-conv using roots-of-real-alk(2)[of q] .
assume p ≠ 0
with pq c have q: q ≠ 0 by auto
show set (real-roots-of-rat-poly p) = {x. rpoly p x = 0} unfolding id
  unfolding real-roots-of-rat-poly-def cq snd-conv using roots-of-real-alk(1)[OF q]
  by auto
qed

The upcoming functions no longer demand an integer or rational polynomial as input.

definition roots-of-real-main :: real poly ⇒ real list where
roots-of-real-main p ≡ let n = degree p in
  if n = 0 then [] else if n = 1 then [roots1 p] else if n = 2 then rroots2 p
  else (real-roots-of-rat-poly (map-poly to-rat p))

definition roots-of-real-poly :: real poly ⇒ real list option where
roots-of-real-poly p ≡ let (c,pis) = gun-factorization gcd p in
  if (c ≠ 0 ∧ (∀ (p,i) ∈ set pis. degree p ≤ 2 ∨ (∀ x ∈ set (coeffs p). x ∈ Q)))
  then

194
lemma roots-of-real-main: assumes p: p ≠ 0 and deg: degree p ≤ 2 ∨ set (coeffs p) ⊆ ℚ
   shows set (roots-of-real-main p) = {x. poly p x = 0} (is ?l = ?r)

proof –
  note d = roots-of-real-main-def Let-def
  show ?thesis
  proof (cases degree p = 0)
    case True
    hence ?l = {} unfolding d by auto
    with roots0[OF p True] show ?thesis by auto
  next
  case False
  proof (cases degree p = 1)
    case True
    hence ?l = {roots1 p} unfolding d by auto
    with roots1[OF True] show ?thesis by auto
  next
  case False
  proof (cases degree p = 2)
    case True
    hence ?l = set (rroots2 p) unfolding d by auto
    with rroots2[OF True] show ?thesis by auto
  next
    case False
  note 1 = this
  let ?q = map-poly to-rat p
  from 0 1 2 have l: ?l = set (real-roots-of-rat-poly ?q) unfolding d by auto
  from deg 0 1 2 have rat: set (coeffs p) ⊆ ℚ by auto
  have p = map-poly (of-rat o to-rat) p
    by (rule sym, rule map-poly-idI, insert rat, auto)
  also have . . . = real-of-rat-poly ?q
    by (subst map-poly-map-poly, auto simp: to-rat)
  finally have id: {x. poly p x = 0} = {x. poly (real-of-rat-poly ?q) x = 0}
  and q: ?q ≠ 0
    using p by auto
    from real-roots-of-rat-poly(1)[OF q, folded id] show ?thesis by simp
  qed
  qed
  qed
  qed

   shows set xs = {x. poly p x = 0}

proof –
  obtain c pis where yun: yun-factorization gcd p = (c, pis) by force
  from rt[unfolded roots-of-real-poly-def yun split Let-def]
have \( c \neq 0 \) and \( \text{pis} : \bigwedge p \ i. (p, i) \in \text{set pis} \implies \deg p \leq 2 \lor (\forall x \in \text{set (coeffs p)}. x \in \mathbb{Q}) \) 

and \( \text{xs} : \text{xs} = \text{concat (map (roots-of-real-main \circ \text{fst} \text{pis})} \) 

by (auto split: if-splits)

note \( \text{yun} = \text{square-free-factorizationD(1,2,4)}[\text{OF yun-factorization(1)}][\text{OF yun}] \) 

from \( \text{yun(1)} \) have \( p : p = \text{smult} c \left( \prod (a, i) \in \text{set pis}. a ^{\sim \text{Suc} i} \right) \) 

unfolding \( p \) using \( c \) by auto

also have \( \ldots = \bigcup ((\lambda p \cdot \{x. \text{poly p x = 0}\}) ' \text{fst} ' \text{set pis}) \) \( \text{(is - = ?r)} \) 

by (subst poly-prod-0, force+)

finally have \( r : \{x. \text{poly p x = 0}\} = \ ?r . \)

\{ 
  fix \( p \ i \)
  assume \( p : (p,i) \in \text{set pis} \)
  have \( \text{set (roots-of-real-main p)} = \{x. \text{poly p x = 0}\} \)
  by (rule roots-of-real-main, insert yun(2)[\text{OF p}][\text{pis}[\text{OF p}], auto)
\}

note \( \text{main} = \text{this} \)

have \( \text{xs} = \bigcup ((\lambda (p, i). \text{set (roots-of-real-main p)}) ' \text{set pis}) \) unfolding \( \text{xs} \) 

\( \text{o-def} \)

by auto

also have \( \ldots = \ ?r \) using \( \text{main} \) by auto

finally show \( \?\text{thesis unfolding r by simp} \)

qed

end

10 Complex Roots of Real Valued Polynomials

We provide conversion functions between polynomials over the real and the complex numbers, and prove that the complex roots of real-valued polynomial always come in conjugate pairs. We further show that also the order of the complex conjugate roots is identical.

As a consequence, we derive that every real-valued polynomial can be factored into real factors of degree at most 2, and we prove that every polynomial over the reals with odd degree has a real root.

theory Complex-Roots-Real-Poly

imports
  HOL-Computational-Algebra.Fundamental-Theorem-Algebra
  Polynomial-Factorization.Order-Polynomial
  Polynomial-Factorization.Explicit-Roots
  Polynomial-Interpolation.Ring-Hom-Poly

begin

interpretation of-real-poly-hom: map-poly-dom-hom complex-of-real..

lemma real-poly-real-coeff: assumes \( \text{set (coeffs p) } \subseteq \mathbb{R} \)

shows \( \text{coeff p x } \in \mathbb{R} \)
proof

  have coeff p x ∈ range (coeff p) by auto

  from this[unfolded range-coeff] assms show thesis by auto

qed

lemma complex-conjugate-root:

assumes real: set (coeffs p) ⊆ ℝ and rt: poly p c = 0

shows poly p (cnj c) = 0

proof

  let ?c = cnj c

  { fix x
    have coeff p x ∈ ℝ
      by (rule real-poly-real-coeff[OF real])
    hence cnj (coeff p x) = coeff p x by (cases coeff p x, auto)
  } note cnj-coeff = this

  have poly p ?c = poly (∑ x≤degree p. monom (coeff p x) x) ?c
    unfolding poly-as-sum-of-monomms ..
    also have .. . = (∑ x≤degree p . coeff p x * cnj (c ^ x))
      unfolding poly-sum poly-monom complex-cnj-power ..
    also have .. . = cnj (∑ x≤degree p . coeff p x * c ^ x)
      unfolding complex-cnj-mult cnj-coeff ..
    also have .. . = cnj (∑ x≤degree p . coeff p x * c ^ x)
      unfolding cnj-sum ..
    also have (∑ x≤degree p . coeff p x * c ^ x) =
      poly (∑ x≤degree p . monom (coeff p x) x) c
      unfolding poly-sum poly-monom ..
    also have .. . = 0 unfolding poly-as-sum-of-monomms rt ..
    also have cnj 0 = 0 by simp
    finally show thesis .

qed

context

fixes p :: complex poly

assumes coeffs: set (coeffs p) ⊆ ℝ

begin

lemma map-poly-Re-poly: fixes x :: real

  shows poly (map-poly Re p) x = poly p (of-real x)

proof

  have id: map-poly (of-real o Re) p = p
    by (rule map-poly-idI, insert coeffs, auto)
  show thesis unfolding arg-cong[OF id, of poly, symmetric]
    by (subst map-poly-map-poly[symmetric], auto)

qed

lemma map-poly-Re-coeffs:

  coeffs (map-poly Re p) = map Re (coeffs p)

proof (rule coeffs-map-poly)
have lead-coeff p ∈ range (coeff p) by auto
hence x: lead-coeff p ∈ ℝ using coeffs by (auto simp: range-coeff)
show (Re (lead-coeff p) = 0) = (p = 0)
  using of-real-Re[OF x] by auto
qed

lemma map-poly-Re-0: map-poly Re p = 0 =⇒ p = 0
  using map-poly-Re-coeffs by auto
end

lemma real-poly-add:
  assumes set (coeffs p) ⊆ ℝ set (coeffs q) ⊆ ℝ
  shows set (coeffs (p + q)) ⊆ ℝ
proof –
  define pp where pp = coeffs p
  define qq where qq = coeffs q
  show ?thesis using assms
    unfolding coeffs-plus-eq-plus-coeffs pp-def [symmetric] qq-def [symmetric]
    by (induct pp qq rule: plus-coeffs induct, auto simp: cCons-def)
qed

lemma real-poly-sum:
  assumes ⋀ x. x ∈ S =⇒ set (coeffs (f x)) ⊆ ℝ
  shows set (coeffs (sum f S)) ⊆ ℝ
using assms proof (induct S rule: infinite-finite-induct)
case (insert x S)
hence id: sum f (insert x S) = f x + sum f S by auto
show ?case unfolding id
  by (rule real-poly-add[OF - insert 3], insert insert, auto)
qed auto

lemma real-poly-smult: fixes p :: 'a :: {idom,real-algebra-1} poly
  assumes c ∈ ℝ set (coeffs p) ⊆ ℝ
  shows set (coeffs (smult c p)) ⊆ ℝ
using assms by (auto simp: coeffs-smult)

lemma real-poly-pCons:
  assumes c ∈ ℝ set (coeffs p) ⊆ ℝ
  shows set (coeffs (pCons c p)) ⊆ ℝ
using assms by (auto simp: cCons-def)

lemma real-poly-mult: fixes p :: 'a :: {idom,real-algebra-1} poly
  assumes p: set (coeffs p) ⊆ ℝ and q: set (coeffs q) ⊆ ℝ
  shows set (coeffs (p * q)) ⊆ ℝ using p
proof (induct p)
case (pCons a p)
show ?case unfolding mult-pCons-left
  by (intro real-poly-add real-poly-smult real-poly-pCons pCons(2) q,
      insert pCons(1,3), auto simp: cCons-def if-splits)
qed simp

lemma real-poly-power: fixes p :: 'a :: {idom, real-algebra-1} poly
  assumes p: set (coeffs p) ⊆ R
  shows set (coeffs (p ^ n)) ⊆ R
proof (induct n)
  case (Suc n)
  from real-poly-mult[OF p this]
  show ?case by simp
qed simp

lemma real-poly-prod: fixes f :: 'a ⇒ 'b :: {idom, real-algebra-1} poly
  assumes ∃ x. x ∈ S =⇒ set (coeffs (f x)) ⊆ R
  shows set (coeffs (prod f S)) ⊆ R
using assms
proof (induct S rule: infinite-finite-induct)
  case (insert x S)
  hence id: prod f (insert x S) = f x * prod f S by auto
  show ?case unfolding id
    by (rule real-poly-mult[OF - insert(3), insert insert, auto])
qed auto

lemma real-poly-uminus:
  assumes set (coeffs p) ⊆ R
  shows set (coeffs (¬p)) ⊆ R
using assms unfolding coeffs-uminus by auto

lemma real-poly-minus:
  assumes set (coeffs p) ⊆ R set (coeffs q) ⊆ R
  shows set (coeffs (p - q)) ⊆ R
using assms unfolding diff-conv-add-uminus
by (intro real-poly-uminus real-poly-add, auto)

lemma fixes p :: 'a :: real-field poly
  assumes p: set (coeffs p) ⊆ R and *: set (coeffs q) ⊆ R
  shows real-poly-div: set (coeffs (q div p)) ⊆ R
    and real-poly-mod: set (coeffs (q mod p)) ⊆ R
proof (atomize(full), insert *, induct q)
  case 0
  thus ?case by auto
next
  case (pCons a q)
  from pCons(1,3) have a: a ∈ R and q: set (coeffs q) ⊆ R by auto
  note res = pCons
show \(?\)case
proof (cases \(p = 0\))
  case True
  with res pCons(3) show \(?\)thesis by auto
next
  case False
  from pCons have IH: set \((\text{coeffs} (q \div p)) \subseteq \mathbb{R}\) set \((\text{coeffs} (q \mod p)) \subseteq \mathbb{R}\)
by auto
  define \(c\) where \(c = \text{coeff} (\text{pCons} a (q \mod p)) (\text{degree} p) / \text{coeff} p (\text{degree} p)\)
  { 
    have \(\text{coeff} (\text{pCons} a (q \mod p)) (\text{degree} p) \in \mathbb{R}\) 
    by (rule \text{real-poly-real-coeff}, insert \text{IH} a, intro \text{real-poly-pCons})
    moreover have \(\text{coeff} p (\text{degree} p) \in \mathbb{R}\) 
    by (rule \text{real-poly-real-coeff}[OF \(p\)])
    ultimately have \(c \in \mathbb{R}\) unfolding \(\text{c-def}\) by simp
  } note \(c = \text{this}\)
  from False have \(r\): pCons a q div p = pCons c (q div p) and \(s\): pCons a q mod p = pCons a (q mod p) − smult c p
  unfolding \(\text{c-def div-pCons-eq mod-pCons-eq}\) by simp-all 
  show \(?\)thesis unfolding \(r\ s\) using a p c IH by (intro conjI \text{real-poly-pCons} real-poly-minus real-poly-smult)
qed
qed

lemma \text{real-poly-factor}: fixes \(p :: 'a ::\text{real-field poly}\)
  assumes \(\text{set} (\text{coeffs} (p * q)) \subseteq \mathbb{R}\)
  \(\text{set} (\text{coeffs} p) \subseteq \mathbb{R}\)
  \(p \neq 0\)
  shows \(\text{set} (\text{coeffs} q) \subseteq \mathbb{R}\)
proof −
  have \(q = p * q \div p\) using \(<p \neq 0\>\) by simp
  hence \(\text{id}: \text{coeffs} q = \text{coeffs} (p * q \div p)\) by simp
  show \(?\)thesis unfolding \(\text{id}\)
  by (rule \text{real-poly-div}, insert \text{assms}, auto)
qed

lemma \text{complex-conjugate-order}: assumes \(\text{real}: \text{set} (\text{coeffs} p) \subseteq \mathbb{R}\)
  \(p \neq 0\)
  shows \(\text{order} (\text{cnj} c) p = \text{order} c p\)
proof −
  define \(n\) where \(n = \text{degree} p\)
  have \(\text{degree} p \leq n\) unfolding \(\text{n-def}\) by auto
  thus \(?\)thesis using \text{assms}
proof (induct \(n\) arbitrary: \(p\))
  case (0 \(p\))
  { 
    fix \(x\)
    have \(\text{order} x p \leq \text{degree} p\)
  }
by (rule order-degree[OF 0(3)])
hence order x p = 0 using 0 by auto

thus ?case by simp

next
  case (Suc m p)
  note order = order[OF p ≠ 0]
  let ?c = cnj c
  show ?case
  proof (cases poly p c = 0)
    case True
    note rt1 = this
    from complex-conjugate-root[OF Suc True] have rt2: poly p ?c = 0.
    show ?thesis
    proof (cases c ∈ I R)
      case True
      hence ?c = c by (cases c, auto)
      thus ?thesis by auto
    next
      case False
      hence neq: ?c ≠ c by (simp add: Reals-cnj-iff)
      let ?fac1 = [:- c, 1 ]
      let ?fac2 = [:- ?c, 1 ]
      let ?fac = ?fac1 * ?fac2
      from rt1 have ?fac1 dvd p unfolding poly-eq-0-iff-dvd .
      from this[unfolded dvd-def] obtain q where p = ?fac1 * q by auto
      from rt2[unfolded p poly-mult] neq have poly q ?c = 0 by auto
      hence ?fac2 dvd q unfolding poly-eq-0-iff-dvd .
      from this[unfolded dvd-def] obtain r where q = ?fac2 * r by auto
      have p: p = ?fac * r unfolding p q by algebra
      from (p ≠ 0) have nz: ?fac1 ≠ 0 ?fac2 ≠ 0 ?fac ≠ 0 r ≠ 0 unfolding p
      by auto
      have id: ?fac = [?c * c, -(?c + c), I ] by simp
      have cfac: coeffs ?fac = [ ?c * c, -(?c + c), I ] unfolding id by simp
      have cfac: set (coeffs ?fac) ⊆ R unfolding cfac by (cases c, auto simp: Reals-cnj-iff)
      have degree p = degree ?fac + degree r unfolding p
      by (rule degree-mult-eq, insert nz, auto)
      also have degree ?fac = degree ?fac1 + degree ?fac2
      by (rule degree-mult-eq, insert nz, auto)
      finally have degree p = 2 + degree r by simp
      with Suc have deg: degree r ≤ m by auto
      from real-poly-factor[OF Suc(3)[unfolded p] cfac] nz have set (coeffs r) ⊆ R
      by auto
      from Suc(1)[OF deg this |r ≠ 0] have IH: order ?c r = order c r .
      { fix cc
        have order cc p = order cc ?fac + order cc r unfolding p ≠ 0
        by (rule order-mult)
also have \( \text{order } cc \ ?\text{fac} = \text{order } cc \ ?\text{fac1} + \text{order } cc \ ?\text{fac2} \)
by (rule order-mult, rule nz)
also have \( \text{order } cc \ ?\text{fac1} = (\text{if } cc = c \text{ then } 1 \text{ else } 0) \)
unfolding order-linear \( \text{by simp} \)
also have \( \text{order } cc \ ?\text{fac2} = (\text{if } cc = ?c \text{ then } 1 \text{ else } 0) \)
unfolding order-linear \( \text{by simp} \)
finally have \( \text{order } cc \ p = \)
\((\text{if } cc = c \text{ then } 1 \text{ else } 0) + (\text{if } cc = \text{cnj } c \text{ then } 1 \text{ else } 0) + \text{order } cc \ r \).
\( \)
\( \)}\ note \( \text{order } = \text{this} \)
show \( ?\text{thesis} \)
unfolding order-IH \( \text{by auto} \)
qed
next
case False \note \rr1 = \text{this} 
\{ 
\assume poly p \ ?c = 0 
from complex-conjugate-root[\( \text{OF Suc(3)} \) \this] \rr1 
have False \text{ by auto} 
\}
\hence \rr2: poly p \ ?c \neq 0 \text{ by auto} 
from \rr1 \rr2 \show \( ?\text{thesis} \)
unfolding order-root \( \text{by simp} \)
qed
qed

lemma \( \text{map-poly-of-real-Re}: \) \assumes \( \text{set } (\text{coeffs } p) \subseteq \mathbb{R} \)
\shows \( \text{map-poly of-real } (\text{map-poly Re } p) = p \)
\by (subst \text{map-poly-map-poly}, \text{force+}, \text{rule map-poly-idI}, \text{insert \assms}, \text{auto})

lemma \( \text{map-poly-Re-of-real}: \) \map-poly Re \( \) (\text{map-poly of-real } p) = p
\by (subst \text{map-poly-map-poly}, \text{force+}, \text{rule map-poly-idI}, \text{auto})

lemma \( \text{map-poly-Re-mult}: \) \assumes \( p: \text{set } (\text{coeffs } p) \subseteq \mathbb{R} \)
and \( q: \text{set } (\text{coeffs } q) \subseteq \mathbb{R} \) shows \( \text{map-poly Re } (p \ast q) = \text{map-poly Re } p \ast \text{map-poly Re } q \)
pro \( \)
\let \( \text{r} = \text{map-poly Re} \)
\let \( \text{c} = \text{map-poly complex-of-real} \)
\have \( \text{r} (p \ast q) = \text{r} (\text{c } (\text{r} p) \ast \text{c } (\text{r} q)) \)
\unfolding \( \text{map-poly-of-real-Re[OF } p] \text{ map-poly-of-real-Re[OF } q] \text{ by simp} \)
also have \( \text{c } (\text{r} p) \ast \text{c } (\text{r} q) = \text{c } (\text{r} p \ast \text{r} q) \text{ by } (\text{simp add: hom-distribs}) \)
also have \( \text{r} \ldots = \text{r} p \ast \text{r} q \text{ unfolding map-poly-Re-of-real } \ldots \)
finally show \( ?\text{thesis} \).
qed

lemma \( \text{map-poly-Re-power}: \) \assumes \( p: \text{set } (\text{coeffs } p) \subseteq \mathbb{R} \)
\shows \( \text{map-poly Re } (p^n) = (\text{map-poly Re } p)^n \)
pro \( \)
\case (Suc \( n \))
let \(?r = \text{map-poly } Re\)

have \(?r \ (p \cdot \text{Suc } n) = ?r \ (p \ast p^n)\) by \text{simp}
also have \(\ldots = ?r \ p \ast ?r \ (p \cdot n)\)
  by (rule map-poly-Re-mult[OF \p \ real-poly-power[OF \p]])
also have \(?r \ (p \cdot n) = (?r \ p) \cdot n\) by (rule Suc)
finally show \(?\text{case by simp}\)
qed \text{simp}

lemma \text{real-degree-2-factorization-exists-complex: fixes } \p :: \text{complex poly}
\text{assumes } \pR : \text{set (coeffs } \p) \subseteq \mathbb{R}
\text{shows } \exists \ q s. \ \p = \text{prod-list } qs \land \ (\forall q \in \text{set } qs. \ \text{set (coeffs } q) \subseteq \mathbb{R} \land \ \text{degree } q \leq 2)
\text{proof –}
obtain n where \text{degree } \p = n \text{ by auto}
thus \text{thesis using } \pR
\text{proof (induct } n \text{ arbitrary; } \p \text{ rule: less-induct)}
case (\text{less } n \ p)
hence \pR : \text{set (coeffs } \p) \subseteq \mathbb{R} \text{ by auto}
show \text{?case}
proof (cases } n \leq 2)
case True
  thus \text{thesis using } \pR
    by (intro exI[of - \p], auto)
next
case False
  hence \text{degp: degree } \p \geq 2 \text{ using less(2) by auto}
  hence \lnot \text{constant (poly } \p) \text{ by (simp add: constant-degree)}
from \text{fundamental-theorem-of-algebra}[OF this] obtain \ x where \ x : \text{poly } \p \ x = 0 \text{ by auto}
from \x have \text{dvd: } [-x, 1:] \text{ dvd } \p \text{ using poly-eq-0-iff-dvd by blast}
have \exists \ \f. \ \f \text{ dvd } \p \land \text{ set (coeffs } \f) \subseteq \mathbb{R} \land \ 1 \leq \text{degree } \f \land \text{degree } \f \leq 2
\text{proof (cases } x \in \mathbb{R})
case True
  with \text{dvd show } \text{?thesis}
    by (intro exI[of - [-x, 1:]], auto)
next
case False
let \(?x = \text{cnj } x\)
let \(?a = ?x \ast x\)
let \(?b = - ?x - x\)
from \text{complex-conjugate-root}[OF \pR \ x]
\text{have } \xx : \text{poly } \p \ ?x = 0 \text{ by auto}
from \text{False have } \text{diff: } x \neq ?x \text{ by (simp add: Reals-cnj-iff)}
from \text{dvd obtain } \r \text{ where } \p = [-x, 1:] \ast \r \text{ unfolding dvd-def by auto}
from \text{xx[unfolded this] diff have } \text{poly } \r \ ?x = 0 \text{ by simp}
\text{hence } [-?x, 1:] \text{ dvd } \r \text{ using poly-eq-0-iff-dvd by blast}
then obtain \ s \text{ where } \r = [-?x, 1:] \ast \ s \text{ unfolding dvd-def by auto}
\text{have } \p = (-x, 1:] \ast [-?x, 1:] \ast \ s \text{ unfolding } \text{poly } \r \text{ by algebra}
also have \ [-x, 1:] \ast [-?x, 1:] = [-?a, ?b, 1:] \text{ by simp}
finally have \ [-?a, ?b, 1:] \text{ dvd } \p \text{ unfolding dvd-def by auto}

203
moreover have \( ?a \in \mathbb{R} \) by (simp add: Reals-cnj-iff)
moreover have \( ?b \in \mathbb{R} \) by (simp add: Reals-cnj-iff)
ultimately show \( ?\text{thesis} \) by (intro exI[of - ?:a,?:b,?c], auto)

qed
then obtain \( f \) where \( \text{dvd}: f \text{ dvd } p \) and \( \text{fr}: \text{ set } (\text{coeffs } f) \subseteq \mathbb{R} \) and \( \text{degf}: 1 \leq \text{degree } f \) \( \text{degree } f \leq 2 \) by auto
from \( \text{dvd} \) obtain \( r \) where \( p: p = f \ast r \) unfolding \( \text{dvd-def} \) by auto
from \( \text{degp} \) have \( p0: p \neq 0 \) by auto
with \( p \) have \( f0: f \neq 0 \) and \( r0: r \neq 0 \) by auto
from \( \text{real-poly-factor}(\text{OF } pR[\text{unfolded } p] \text{ fr } f0] \) have \( rR: \text{ set } (\text{coeffs } r) \subseteq \mathbb{R} \).
have \( \text{deg}: \text{degree } p = \text{degree } f + \text{degree } r \) unfolding \( p \)
by (rule degree-mult-eq[\text{OF } f0 r0])
with \( \text{degf less}(2) \) have \( \text{degr}: \text{degree } r < n \) by auto
from \( \text{less}(1)[\text{OF this } \text{fr } rR] \) obtain \( qs \)
where \( \text{IH}: r = \text{prod-list } qs \) (\( \forall q \in \text{set } qs. \text{ set } (\text{coeffs } q) \subseteq \mathbb{R} \) \( \land \) degree \( q \leq 2 \)) by auto
from \( \text{IH}(1) \) have \( p: p = \text{prod-list } (f \# qs) \) unfolding \( p \) by auto
with \( \text{IH}(2) \) \( \text{fr } \text{degf} \) show \( ?\text{thesis} \)
by (intro exI[of - f \# qs], auto)

qed

lemma real-degree-2-factorization-exists: fixes \( p :: \text{real poly} \)
shows \( \exists qs. p = \text{prod-list } qs \land (\forall q \in \text{set } qs. \text{ degree } q \leq 2) \)

proof --
let \( \text{?cp} = \text{map-poly } \text{complex-of-real} \)
let \( \text{?rp} = \text{map-poly } \text{Re} \)
let \( \text{?p} = \text{?cp } p \)
have \( \text{set } (\text{coeffs } ?p) \subseteq \mathbb{R} \) by auto
from \( \text{real-degree-2-factorization-exists-complex}[\text{OF this}] \)
obtain \( qs \) where \( p: ?p = \text{prod-list } qs \) and
\( qs: \forall q. q \in \text{set } qs \Longrightarrow \text{ set } (\text{coeffs } q) \subseteq \mathbb{R} \land \text{ degree } q \leq 2 \) by auto
have \( p: p = ?rp \) (\( \text{prod-list } qs \) unfolding \( \text{arg-cong}[\text{OF } p, \text{of } ?rp, \text{symmetric}] \)
by (subst \( \text{map-poly-map-poly } \text{force}, \text{rule sym}, \text{rule map-poly-idI}, \text{auto} \)
from \( qs \) have \( \exists rs. \text{prod-list } qs = ?cp \) (\( \text{prod-list } rs \) \( \land \) (\( \forall r \in \text{set } rs. \text{ degree } r \leq 2 \))

proof (induct \( qs \))
case \( \text{Nil} \)
show \( ?\text{case} \) by (auto intro!: exI[of - \text{Nil}])
next
case (\( \text{Cons } q \) \( qs \))
then obtain \( rs \) where \( qs: \text{prod-list } qs = ?cp \) (\( \text{prod-list } rs \) 
and \( rs: \forall q. q \in \text{set } rs \Longrightarrow \text{ degree } q \leq 2 \)) by force+
from \( \text{Cons}(2)[\text{of } q] \) have \( q: \text{ set } (\text{coeffs } q) \subseteq \mathbb{R} \) and \( dq: \text{ degree } q \leq 2 \) by auto
define \( r \) where \( r = ?rp q \)
have \( q: q = ?cp r \) unfolding \( r\text{-def} \)
by (subst \( \text{map-poly-map-poly } \text{force}, \text{rule sym}, \text{rule map-poly-idI}, \text{insert } q, \text{auto} \)

204
have \( dr: \text{degree } r \leq 2 \) using \( dq \) unfolding \( q \) by \( \text{simp add: degree-map-poly} \)

show \( ?\text{case} \)

by \( \text{(rule exI[of - r \# rs], unfold prod-list.Cons qs q, insert dr rs, auto simp: hom-distribs)} \)

qed

then obtain \( rs \) where \( id: \text{prod-list qs} = ?\text{cp (prod-list rs)} \) and \( \text{deg: } \forall r \in \text{set rs. degree } r \leq 2 \) by auto

show \( ?\text{thesis unfolding } p \ id \)

by \( \text{(intro exI, rule conjI[OF - deg], subst map-poly-map-poly, force, rule map-poly-idI, auto)} \)

qed

lemma odd-degree-imp-real-root: assumes \( \text{odd (degree } p) \)

shows \( \exists x. \text{poly } p \ x = (0 :: \text{real}) \)

proof

from \( \text{real-degree-2-factorization-exists[of } p \) obtain \( qs \) where

\( id: \text{p = prod-list qs} \) and \( \text{qs: } \forall q. q \in \text{set qs } \Rightarrow \text{degree } q \leq 2 \) by auto

show \( ?\text{thesis using assms qs unfolding id} \)

proof \( \text{(induct } qs) \)

case \( \text{Cons q qs} \)

from \( \text{Cons(3)[of } q \) have \( dq: \text{degree } q \leq 2 \) by auto

show \( ?\text{case} \)

proof \( \text{(cases degree } q = 1) \)

case True

from \( \text{roots1[OF this]} \) show \( ?\text{thesis by auto} \)

next

case False

with \( dq \) have \( \text{deg: degree } q = 0 \lor \text{degree } q = 2 \) by arith

from \( \text{Cons(2)} \) have \( q * \text{prod-list qs} \neq 0 \) by fastforce

hence \( q \neq 0 \) prod-list qs \( \neq 0 \) by auto

from \( \text{degree-mult-eq[OF this]} \)

have \( \text{degree (prod-list (q \# qs)) = degree } q + \text{degree (prod-list qs)} \) by simp

from \( \text{Cons(2)[unfolded this]} \) deg have \( \text{odd (degree (prod-list qs))} \) by auto

from \( \text{Cons(1)[OF this Cons(3)]} \) obtain \( x \) where \( \text{poly (prod-list qs) x = 0} \) by auto

thus \( ?\text{thesis by auto} \)

qed simp

qed

end

10.1 Compare Instance for Complex Numbers

We define some code equations for complex numbers, provide a comparator for complex numbers, and register complex numbers for the container framework.

theory Compare-Complex
imports
  HOL.Complex
  Polynomial-Interpolation.Missing-Unsorted
  Deriving.Compare-Real
  Containers.Set-Impl
begin

declare [[code drop: Gcd-fin]]
declare [[code drop: Lcm-fin]]
definition gcds :: 'a::semiring-gcd list ⇒ 'a where [simp, code-abbrev]: gcds xs = gcd-list xs

lemma [code]:
gcds xs = fold gcd xs 0
by (simp add: Gcd-fin.set-eq-fold)
definition lcms :: 'a::semiring-gcd list ⇒ 'a where [simp, code-abbrev]: lcms xs = lcm-list xs

lemma [code]:
lcms xs = fold lcm xs 1
by (simp add: Lcm-fin.set-eq-fold)

lemma in-reals-code [code-unfold]:
x ∈ ℝ ←→ Im x = 0
by (fact complex-is-Real-iff)
definition is-norm-1 :: complex ⇒ bool where is-norm-1 z = ((Re z)² + (Im z)² = 1)

lemma is-norm-1[simp]: is-norm-1 x = (norm x = 1)
unfolding is-norm-1-def norm-complex-def by simp
definition is-norm-le-1 :: complex ⇒ bool where is-norm-le-1 z = ((Re z)² + (Im z)² ≤ 1)

lemma is-norm-le-1[simp]: is-norm-le-1 x = (norm x ≤ 1)
unfolding is-norm-le-1-def norm-complex-def by simp

instantiation complex :: finite-UNIV
begin
definition finite-UNIV = Phantom(complex) False
instance
  by (intro-classes, unfold finite-UNIV-complex-def, simp add: infinite-UNIV-char-0)
end

instantiation complex :: compare
begin
definition compare-complex :: complex ⇒ complex ⇒ order where
  compare-complex x y = compare (Re x, Im x) (Re y, Im y)

instance
proof (intro-classes, unfold-locales; unfold compare-complex-def)
  fix x y z :: complex
  let ?c = compare :: (real × real) comparator
  interpret comparator ?c by (rule comparator-compare)
  show invert-order (?c (Re x, Im x) (Re y, Im y)) = ?c (Re y, Im y) (Re x, Im x)
    by (rule sym)
  \{
    assume ?c (Re x, Im x) (Re y, Im y) = Lt
    ?c (Re y, Im y) (Re z, Im z) = Lt
    thus ?c (Re x, Im x) (Re z, Im z) = Lt
      by (rule trans)
  \}
  \{
    assume ?c (Re x, Im x) (Re y, Im y) = Eq
    from weak-eq[OF this] show x = y unfolding complex-eq-iff by auto
  \}
qed
end

derive (eq) cceq complex real
derive (compare) ccompare complex
derive (compare) ccompare real
derive (dlist) set-impl complex real
end

11 Interval Arithmetic

We provide basic interval arithmetic operations for real and complex intervals. As application we prove that complex polynomial evaluation is continuous w.r.t. interval arithmetic. To be more precise, if an interval sequence converges to some element $x$, then the interval polynomial evaluation of $f$ tends to $f(x)$.

theory Interval-Arithmetic
imports
  Algebraic-Numbers-Prelim
begin

  Intervals
datatype ('a) interval = Interval (lower: 'a) (upper: 'a)

  hide-const(open) lower upper

  207
definition to-interval where to-interval a ≡ Interval a a

abbreviation of-int-interval :: int ⇒ 'a :: ring-1 interval where
    of-int-interval x ≡ to-interval (of-int x)

11.1 Syntactic Class Instantiations

instantiation interval :: (zero) zero begin
    definition zero-interval where 0 ≡ Interval 0 0
    instance..
end

instantiation interval :: (one) one begin
    definition 1 = Interval 1 1
    instance..
end

instantiation interval :: (plus) plus begin
    fun plus-interval where Interval lx ux + Interval ly uy = Interval (lx + ly) (ux + uy)
    instance..
end

instantiation interval :: (uminus) uminus begin
    fun uminus-interval where − Interval l u = Interval (−u) (−l)
    instance..
end

instantiation interval :: (minus) minus begin
    fun minus-interval where Interval lx ux − Interval ly uy = Interval (lx − uy) (ux − ly)
    instance..
end

instantiation interval :: ({ord,times}) times begin
    fun times-interval where
        Interval lx ux * Interval ly uy =
            (let x1 = lx * ly; x2 = lx * uy; x3 = ux * ly; x4 = ux * uy
                in Interval (min x1 (min x2 (min x3 x4))) (max x1 (max x2 (max x3 x4))))
    instance..
end

instantiation interval :: ({ord,times,inverse}) inverse begin
    fun inverse-interval where
        inverse (Interval l u) = Interval (inverse u) (inverse l)
    definition divide-interval :: 'a interval ⇒ - where
        divide-interval X Y = X * (inverse Y)
    instance..
end
11.2 Class Instantiations

instance interval :: (semigroup-add) semigroup-add
proof
  fix a b c :: 'a interval
  show a + b + c = a + (b + c) by (cases a, cases b, cases c, auto simp: ac-simps)
qed

instance interval :: (monoid-add) monoid-add
proof
  fix a :: 'a interval
  show 0 + a = a by (cases a, auto simp: zero-interval-def)
  show a + 0 = a by (cases a, auto simp: zero-interval-def)
qed

instance interval :: (ab-semigroup-add) ab-semigroup-add
proof
  fix a b :: 'a interval
  show a + b = b + a by (cases a, cases b, auto simp: ac-simps)
qed

instance interval :: (comm-monoid-add) comm-monoid-add by (intro-classes, auto)

Intervals do not form an additive group, but satisfy some properties.

lemma interval-uminus-zero[simp]:
  shows -(0 :: 'a :: group-add interval) = 0
  by (simp add: zero-interval-def)

lemma interval-diff-zero[simp]:
  fixes a :: 'a :: cancel-comm-monoid-add interval
  shows a - 0 = a by (cases a, simp add: zero-interval-def)

Without type invariant, intervals do not form a multiplicative monoid, but satisfy some properties.

instance interval :: ({linorder, mult-zero}) mult-zero
proof
  fix a :: 'a interval
  show a * 0 = 0 0 * a = 0 by (atomize(full), cases a, auto simp: zero-interval-def)
qed

11.3 Membership

fun in-interval :: 'a :: order ⇒ 'a interval ⇒ bool ((-/ (\_ \_)) [51, 51] 50) where
  y ∈ i, Interval lx ux = (lx ≤ y ∧ y ≤ ux)

lemma in-interval-to-interval[intro!]: a ∈_i to-interval a
  by (auto simp: to-interval-def)

lemma plus-in-interval:
  fixes x y :: 'a :: ordered-comm-monoid-add
shows $x \in_i X \implies y \in_i Y \implies x + y \in_i X + Y$
by (cases $X$, cases $Y$, auto dest:add-mono)

lemma `uminus-in-interval`:
fixes $x :: 'a :: ordered-ab-group-add$
shows $x \in_i X \implies -x \in_i -X$
by (cases $X$, auto)

lemma `minus-in-interval`:
fixes $x y :: 'a :: ordered-ab-group-add$
shows $x \in_i X \implies y \in_i Y \implies x - y \in_i X - Y$
by (cases $X$, cases $Y$, auto dest:diff-mono)

lemma `times-in-interval`:
fixes $x y :: 'a :: linordered-ring$
assumes $x \in_i X \quad y \in_i Y$
shows $x \ast y \in_i X \ast Y$
proof
obtain $X1 \ X2$ where $X:Interval \ X1 \ X2 = X$
by (cases $X$,auto)
obtain $Y1 \ Y2$ where $Y:Interval \ Y1 \ Y2 = Y$
by (cases $Y$,auto)
from $assms \ X \ Y$ have $assms: X1 \leq x \ast x \leq X2 \quad Y1 \leq y \ast y \leq Y2$
by auto
have $(X1 \ast Y1 \leq x \ast y \vee X1 \ast Y2 \leq x \ast y \vee X2 \ast Y1 \leq x \ast y \vee X2 \ast Y2 \leq x \ast y) \wedge$
$(X1 \ast Y1 \geq x \ast y \vee X1 \ast Y2 \geq x \ast y \vee X2 \ast Y1 \geq x \ast y \vee X2 \ast Y2 \geq x \ast y)$
proof (cases $x \leq 0$::'a rule: linorder-cases)
case $x0: \ less$
show $\ ?thesis$
proof (cases $y \ < \ 0$)
case $y0: True$
from $y0 \ x0 \ assms$ have $x \ast y \leq X1 \ast y$ by (intro mult-right-mono-neg, auto)
also from $x0 \ y0 \ assms$ have $X1 \ast y \leq X1 \ast Y1$ by (intro mult-left-mono-neg, auto)
finally have $1: x \ast y \leq X1 \ast Y1$.
show $\ ?thesis$ proof (cases $X2 \leq 0$
  case $True$
  with $assms \ X2 \ast Y2 \leq X2 \ast y$ by (auto intro: mult-left-mono-neg)
  also from $assms \ y0 \ have \ ... \ leq x \ast y$ by (auto intro: mult-right-mono-neg)
  finally have $X2 \ast Y2 \leq x \ast y$.
  with $1$ show $\ ?thesis$ by auto
next
  case $False$
  with $assms \ X2 \ast Y1 \leq X2 \ast y$ by (auto intro: mult-left-mono)
  also from $assms \ y0 \ have \ ... \ leq x \ast y$ by (auto intro: mult-right-mono-neg)
  finally have $X2 \ast Y1 \leq x \ast y$.
  with $1$ show $\ ?thesis$ by auto
qed
next
  case $False$
qed
then have \( y_0 \): \( y \geq 0 \) by \textit{auto}.

From \( x_0 \ y_0 \)\ assumptions have \( X_1 \ast Y_2 \leq x \ast Y_2 \) by \textit{(intro mult-right-mono, auto)}.

Also from \( y_0 \ x_0 \)\ assumptions have \( \ldots \leq x \ast y \) by \textit{(intro mult-left-mono-neg, auto)}.

Finally have 1: \( X_1 \ast Y_2 \leq x \ast y \).

Show \( \textit{thesis} \).

\textit{proof}\((\text{cases } X_2 \leq 0)\)

\textit{case } X_2: \textit{True}

From \( \text{assms } y_0 \)\ have \( x \ast y \leq X_2 \ast y \) by \textit{(intro mult-right-mono)}.

Also from \( \text{assms } X_2 \)\ have \( \ldots \leq X_2 \ast Y_1 \) by \textit{(auto intro: mult-left-mono-neg)}.

Finally have \( x \ast y \leq X_2 \ast Y_1 \).

With 1 show \( \textit{thesis by auto} \).

Next

\textit{case } X_2: \textit{False}

From \( \text{assms } y_0 \)\ have \( x \ast y \leq X_2 \ast y \) by \textit{(intro mult-right-mono)}.

Also from \( \text{assms } X_2 \)\ have \( \ldots \leq X_2 \ast Y_2 \) by \textit{(auto intro: mult-left-mono)}.

Finally have \( x \ast y \leq X_2 \ast Y_2 \).

With 1 show \( \textit{thesis by auto} \).

Qed

Qed

Next

\textit{case } \textit{simp}: \textit{equal}

With \( \text{assms} \) show \( \textit{thesis} \) by \textit{(cases } Y_2 \leq 0, \text{ auto intro: mult-sign-intros)}.

Next

\textit{case } x_0: \textit{greater}

Show \( \textit{thesis} \).

\textit{proof}\((\text{cases } y < 0)\)

\textit{case } y_0: \textit{True}

From \( \text{assms } x_0 \)\ have \( X_2 \ast Y_1 \leq X_2 \ast y \) by \textit{(intro mult-left-mono, auto)}.

Also from \( \text{assms } X_2 \)\ have \( \ldots \leq X_1 \ast Y_2 \) by \textit{(auto intro: mult-right-mono-neg, auto)}.

Finally have 1: \( X_2 \ast Y_1 \leq x \ast y \).

Show \( \textit{thesis} \).

\textit{proof}\((\text{cases } Y_2 \leq 0)\)

\textit{case } Y_2: \textit{True}

From \( \text{assms } x_0 \)\ have \( x \ast y \leq x \ast Y_2 \) by \textit{(auto intro: mult-left-mono)}.

Also from \( \text{assms } Y_2 \)\ have \( \ldots \leq X_1 \ast Y_2 \) by \textit{(auto intro: mult-right-mono-neg)}.

Finally have \( x \ast y \leq X_1 \ast Y_2 \).

With 1 show \( \textit{thesis by auto} \).

Next

\textit{case } Y_2: \textit{False}

From \( \text{assms } x_0 \)\ have \( x \ast y \leq x \ast Y_2 \) by \textit{(auto intro: mult-left-mono)}.

Also from \( \text{assms } Y_2 \)\ have \( \ldots \leq X_2 \ast Y_2 \) by \textit{(auto intro: mult-right-mono)}.

Finally have \( x \ast y \leq X_2 \ast Y_2 \).

With 1 show \( \textit{thesis by auto} \).

Qed

Next

\textit{case } y_0: \textit{False}

From \( \text{assms } x_0 \)\ have \( x \ast y \leq X_2 \ast y \) by \textit{(intro mult-right-mono, auto)}.

Also from \( y_0 \ x_0 \)\ \textit{assms have }\( \ldots \leq X_2 \ast Y_2 \) by \textit{(intro mult-left-mono, auto)}.

211
finally have 1: $x \cdot y \leq X2 \cdot Y2$.

show ?thesis

proof (cases $X1 \leq 0$)
  case True
  with assms have $X1 \cdot Y2 \leq X1 \cdot y$ by (auto intro: mult-left-mono-neg)
  also from assms $y0$ have ... $\leq x \cdot y$ by (auto intro: mult-right-mono)
  finally have $X1 \cdot Y2 \leq x \cdot y$.

with 1 show ?thesis by auto

next
  case False
  with assms have $X1 \cdot Y1 \leq X1 \cdot y$ by (auto intro: mult-left-mono)
  also from assms $y0$ have ... $\leq x \cdot y$ by (auto intro: mult-right-mono)
  finally have $X1 \cdot Y1 \leq x \cdot y$.

with 1 show ?thesis by auto

qed

hence $\min: \min (X1 \cdot Y1) (\min (X1 \cdot Y2)) (\min (X2 \cdot Y1)) (\min (X2 \cdot Y2)) \leq x \cdot y$

and $\max: x \cdot y \leq \max (X1 \cdot Y1) (\max (X1 \cdot Y2)) (\max (X2 \cdot Y1)) (\max (X2 \cdot Y2))$

by (auto simp:min-le-iff-disj le-max-iff-disj)

show ?thesis using min max $X$ $Y$ by (auto simp: Let-def)

qed

11.4 Convergence

definition interval-tendsto :: ('a :: topological-space interval) ⇒ 'a ⇒ bool
  (infixr “−→i”) where
  $(X \rightarrow i x) \equiv ((\text{interval.upper} \circ X) \rightarrow i x) \land ((\text{interval.lower} \circ X) \rightarrow i x)$

lemma interval-tendstoI[intro]:
  assumes $(\text{interval.upper} \circ X) \rightarrow i x$ and $(\text{interval.lower} \circ X) \rightarrow i x$
  shows $X \rightarrow i x$

using assms by (auto simp:interval-tendsto-def)

lemma const-interval-tendsto: $(\lambda i. \text{to-interval} a) \rightarrow i a$

by (auto simp: o-def to-interval-def)

lemma interval-tendsto-0: $(\lambda i. 0) \rightarrow i 0$

by (auto simp: o-def zero-interval-def)

lemma plus-interval-tendsto:
  fixes $x$ $y$ :: 'a :: topological-monoid-add
  assumes $X \rightarrow i x$ $Y \rightarrow i y$
  shows $(\lambda i. X i + Y i) \rightarrow i x + y$

proof
  have *: $X i + Y i = \text{Interval} \ (\text{interval.lower} (X i) + \text{interval.lower} (Y i))$
  $(\text{interval.upper} (X i) + \text{interval.upper} (Y i))$ for i

  by (cases $X i$; cases $Y i$, auto)
from assms show \( \text{thesis unfolding} \) * interval-tendsto-def o-def by (auto intro: tendsto-intros)

\( \text{qed} \)

**lemma** uminus-interval-tendsto:

\( \text{fixes } x :: \lambda a :: \text{topological-group-add} \)

\( \text{assumes } X \longrightarrow \gamma_i x \)

\( \text{shows (} \lambda i. - X i \longrightarrow \gamma_i -x \) \)

**proof**

\( \text{have } \ast: - X i = \text{Interval} (\text{interval.upper} (X i)) (\text{interval.lower} (X i)) \text{ for } i \)

by (cases \( X i \), auto)

from assms show \( \text{thesis unfolding} \) o-def * interval-tendsto-def by (auto intro: tendsto-intros)

\( \text{qed} \)

**lemma** minus-interval-tendsto:

\( \text{fixes } x y :: \lambda a :: \text{topological-group-add} \)

\( \text{assumes } X \longrightarrow \gamma_i x Y \longrightarrow \gamma_i y \)

\( \text{shows (} \lambda i. X i - Y i \longrightarrow \gamma_i x - y \) \)

**proof**

\( \text{have } \ast: X i - Y i = \text{Interval} (\text{interval.lower} (X i) - \text{interval.upper} (Y i)) (\text{interval.upper} (X i) - \text{interval.lower} (Y i)) \text{ for } i \)

by (cases \( X i \); cases \( Y i \), auto)

from assms show \( \text{thesis unfolding} \) o-def * interval-tendsto-def by (auto intro: tendsto-intros)

\( \text{qed} \)

**lemma** times-interval-tendsto:

\( \text{fixes } x y :: \{ \text{linorder-topology, real-normed-algebra} \} \)

\( \text{assumes } X \longrightarrow \gamma_i x Y \longrightarrow \gamma_i y \)

\( \text{shows (} \lambda i. X i * Y i \longrightarrow \gamma_i x * y \) \)

**proof**

\( \text{have } \ast: (\text{interval.lower} (X i * Y i)) = ( \)

\( \text{let } lx = (\text{interval.lower} (X i)); uax = (\text{interval.upper} (X i)); lby = (\text{interval.lower} (Y i)); uy = (\text{interval.upper} (Y i)); \)

\( x1 = lx * ly; x2 = lx * uy; x3 = ux * ly; x4 = ux * uy \text{ in} \)

\( (\text{min x1 (min x2 (min x3 x4)))) (\text{interval.upper} (X i * Y i)) = ( \)

\( \text{let } lx = (\text{interval.lower} (X i)); uax = (\text{interval.upper} (X i)); lby = (\text{interval.lower} (Y i)); uy = (\text{interval.upper} (Y i)); \)

\( x1 = lx * ly; x2 = lx * uy; x3 = ux * ly; x4 = ux * uy \text{ in} \)

\( (\text{min x1 (max x2 (max x3 x4)))) \text{ for } i \)

by (cases \( X i \); cases \( Y i \), auto simp; Let-def)+

\( \text{have } (\lambda i. (\text{interval.lower} (X i * Y i))) \longrightarrow \text{min } (x * y) (\text{min } (x * y) (\text{min } (x * y) (x * y))) \)

\( \text{using assms unfolding interval-tendsto-def o-def} \)

by (intro tendsto-min tendsto-intros, auto)

moreover

\( \text{have } (\lambda i. (\text{interval.upper} (X i * Y i))) \longrightarrow \text{max } (x * y) (\text{max } (x * y) (\text{max } (x * y) (x * y))) \)

213
using assms unfolding interval-tendsto-def * Let-def o-def
by (intro tendsto-max tendsto-intros, auto)
ultimately show ?thesis unfolding interval-tendsto-def o-def by auto
qed

lemma interval-tendsto-neq:
fixes a b :: real
assumes (λ i. f i) −−−→ i a and a ≠ b
shows ∃ n. ¬ b ∈ i f n
proof
  let ?d = norm (b − a) / 2
  from assms have d: ?d > 0 by auto
  from assms[1][unfolded interval-tendsto-def]
  have cvg: (interval.lower o f) −−−→ a (interval.upper o f) −−−→ a by auto
  from LIMSEQ-D[OF cvg(1) d] obtain n1 where
    n1: ∃ n. n ≥ n1 ⇒ norm (interval.lower o f) n − a < ?d by auto
  from LIMSEQ-D[OF cvg(2) d] obtain n2 where
    n2: ∃ n. n ≥ n2 ⇒ norm (interval.upper o f) n − a < ?d by auto
  define n where n = max n1 n2
  from n1[of n] n2[of n] have bnd:
    norm (interval.lower o f) n − a < ?d
    norm (interval.upper o f) n − a < ?d
  unfolding n-def by auto
  show ?thesis by (rule exI[of - n], insert bnd, cases f n, auto,argo)
qed

11.5 Complex Intervals
datatype complex-interval = Complex-Interval (Re-interval: real interval) (Im-interval: real interval)
definition in-complex-interval :: complex ⇒ complex-interval ⇒ bool ((/ / ∈c -)
[51, 51] 50) where
  y ∈c x ≡ (case x of Complex-Interval r i ⇒ Re y ∈c r ∧ Im y ∈c i)
instantiation complex-interval :: comm-monoid-add begin
definition 0 ≡ Complex-Interval 0 0
fun plus-complex-interval :: complex-interval ⇒ complex-interval ⇒ complex-interval
where
  Complex-Interval rx ix + Complex-Interval ry iy = Complex-Interval (rx + ry) (ix + iy)
instance
  proof
    fix a b c :: complex-interval
    show a + b + c = a + (b + c) by (cases a, cases b, cases c, simp add: ac-simps)
    show a + b = b + a by (cases a, cases b, simp add: ac-simps)
show 0 + a = a by (cases a, simp add: ac-simps zero-complex-interval-def)
qed
end

lemma plus-complex-interval: x ∈_e X ⇒ y ∈_e Y ⇒ x + y ∈_e X + Y
unfolding in-complex-interval-def using plus-in-interval by (cases X, cases Y, auto)

definition of-int-complex-interval :: int ⇒ complex-interval where
of-int-complex-interval x = Complex-Interval (of-int-interval x) 0

lemma of-int-complex-interval-0 [simp]: of-int-complex-interval 0 = 0
by (simp add: of-int-complex-interval-def zero-complex-interval-def to-interval-def
zero-interval-def)

lemma of-int-complex-interval: of-int i ∈_e of-int-complex-interval i
unfolding in-complex-interval-def of-int-complex-interval-def
by (auto simp: zero-complex-interval-def zero-interval-def)

instantiation complex-interval :: mult-zero begin

fun times-complex-interval where
Complex-Interval rx ix * Complex-Interval ry iy =
Complex-Interval (rx * ry - ix * iy) (rx * iy + ix * ry)

instance proof
fix a :: complex-interval
show 0 * a = 0 a * 0 = 0 by (atomize(full), cases a, auto simp: zero-complex-interval-def)
qed
end

instantiation complex-interval :: minus begin

fun minus-complex-interval where
Complex-Interval R I - Complex-Interval R' I' = Complex-Interval (R-R')
(I-I')

instance..
end

lemma times-complex-interval: x ∈_e X ⇒ y ∈_e Y ⇒ x * y ∈_e X * Y
unfolding in-complex-interval-def
by (cases X, cases Y, auto intro: times-in-interval minus-in-interval plus-in-interval)

definition ipoly-complex-interval :: int poly ⇒ complex-interval ⇒ complex-interval
where
ipoly-complex-interval p x = fold-coefs (λa b. of-int-complex-interval a + x * b)
lemma ipoly-complex-interval-0[simp]:
  ipoly-complex-interval 0 x = 0
  by (auto simp: ipoly-complex-interval-def)

lemma ipoly-complex-interval-pCons[simp]:
  ipoly-complex-interval (pCons a p) x = of-int-complex-interval a + x * (ipoly-complex-interval p x)
  by (cases p = 0; cases a = 0, auto simp: ipoly-complex-interval-def)

lemma ipoly-complex-interval: assumes x: x ∈ c X
  shows ipoly p x ∈ c ipoly-complex-interval p X
proof
  define xs where xs = coeffs p
  have 0: in-complex-interval 0 0 (is in-complex-interval ?Z ?z)
    unfolding in-complex-interval-def zero-complex-interval-def zero-interval-def
    by auto
  define Z where Z = ?Z
  define z where z = ?z
  from 0 have 0: in-complex-interval Z z unfolding Z-def z-def by auto
  note x = times-complex-interval [OF x]
    by (induct xs arbitrary: Z z, auto intro!: plus-complex-interval of-int-complex-interval x)
qed

definition complex-interval-tendsto (infix −−→c 55) where
  C −−→c c ≡ ((Re-interval ◦ C) −−→i Re c) ∧ ((Im-interval ◦ C) −−→i Im c)

lemma complex-interval-tendstoI[intro!]:
  (Re-interval ◦ C) −−→i Re c ⇒ (Im-interval ◦ C) −−→i Im c ⇒ C −−→c c
  by (simp add: complex-interval-tendsto-def)

lemma of-int-complex-interval-tendsto: (λi. of-int-complex-interval n) −−→c of-int n
  by (auto simp: o-def of-int-complex-interval-def intro!:const-interval-tendsto_interval-tendsto-0)

lemma Im-interval-plus: Im-interval (A + B) = Im-interval A + Im-interval B
  by (cases A; cases B, auto)

lemma Re-interval-plus: Re-interval (A + B) = Re-interval A + Re-interval B
  by (cases A; cases B, auto)
lemma $Im\text{-interval-minus}$: $Im\text{-interval} (A - B) = Im\text{-interval} A - Im\text{-interval} B$

by (cases $A$; cases $B$, auto)

lemma $Re\text{-interval-minus}$: $Re\text{-interval} (A - B) = Re\text{-interval} A - Re\text{-interval} B$

by (cases $A$; cases $B$, auto)

lemma $Re\text{-interval-times}$: $Re\text{-interval} (A \ast B) = Re\text{-interval} A \ast Re\text{-interval} B - Im\text{-interval} A \ast Im\text{-interval} B$

by (cases $A$; cases $B$, auto)

lemma $Im\text{-interval-times}$: $Im\text{-interval} (A \ast B) = Re\text{-interval} A \ast Im\text{-interval} B + Im\text{-interval} A \ast Re\text{-interval} B$

by (cases $A$; cases $B$, auto)

lemma $plus\text{-complex\text{-}interval\text{-}tendsto}$:

$A \longrightarrow a \Rightarrow B \longrightarrow b \Rightarrow (\lambda i. A i + B i) \longrightarrow (a + b)$

unfolding complex\text{-}interval\text{-}tendsto-def

by (auto intro: plus\text{-}interval\text{-}tendsto simp: o\text{-}def Re\text{-}interval\text{-}plus Im\text{-}interval\text{-}plus)

lemma $minus\text{-complex\text{-}interval\text{-}tendsto}$:

$A \longrightarrow a \Rightarrow B \longrightarrow b \Rightarrow (\lambda i. A i - B i) \longrightarrow (a - b)$

unfolding complex\text{-}interval\text{-}tendsto-def

by (auto intro: minus\text{-}interval\text{-}tendsto simp: o\text{-}def Re\text{-}interval\text{-}minus Im\text{-}interval\text{-}minus)

lemma $times\text{-complex\text{-}interval\text{-}tendsto}$:

$A \longrightarrow a \Rightarrow B \longrightarrow b \Rightarrow (\lambda i. A i \ast B i) \longrightarrow (a \ast b)$

unfolding complex\text{-}interval\text{-}tendsto-def

by (auto intro: minus\text{-}interval\text{-}tendsto times\text{-}interval\text{-}tendsto plus\text{-}interval\text{-}tendsto simp: o\text{-}def Re\text{-}interval\text{-}times Im\text{-}interval\text{-}times)

lemma $ipoly\text{-complex\text{-}interval\text{-}tendsto}$:

assumes $C \longrightarrow c$

shows $(\lambda i. ipoly\text{-}complex\text{-}interval\ p (C i)) \longrightarrow ipoly\ p\ c$

proof (induct $p$)

case $\theta$

show $?case$ by (auto simp: o\text{-}def zero\text{-}complex\text{-}interval\text{-}def zero\text{-}interval\text{-}def complex\text{-}interval\text{-}tendsto-def)

next

case $(p\text{Cons}\ a\ p)$

show $?case$

apply (unfold ipoly\text{-}complex\text{-}interval\text{-}pCons of\text{-}int\text{-}hom.map\text{-}poly\text{-}pCons\text{-}hom poly\text{-}pCons)

apply (intro plus\text{-}complex\text{-}interval\text{-}tendsto times\text{-}complex\text{-}interval\text{-}tendsto assms p\text{Cons}\ of\text{-}int\text{-}complex\text{-}interval\text{-}tendsto)

done

qed
lemma complex-interval-tendsto-neq: assumes (λ i. f i) −→ₘ a
and a ≠ b
shows ∃ n. ¬ b ∈ₘ f n
proof –
from assms(1)[unfolded complex-interval-tendsto-def o-def]
have cvg: (λx. Re-interval (f x)) −→ₘ, Re a (λx. Im-interval (f x)) −→ₘ,
Im a by auto
from assms(2) have Re a ≠ Re b ∨ Im a ≠ Im b
  using complex.expand by blast
thus ?thesis
proof
  assume Re a ≠ Re b
  from interval-tendsto-neq[OF cvg(1) this] show ?thesis
unfolding in-complex-interval-def by (metis (no-types, lifting) complex-interval.case-eq-if)
next
  assume Im a ≠ Im b
  from interval-tendsto-neq[OF cvg(2) this] show ?thesis
unfolding in-complex-interval-def by (metis (no-types, lifting) complex-interval.case-eq-if)
qed
qed
end

12 Complex Algebraic Numbers

Since currently there is no immediate analog of Sturm’s theorem for the
complex numbers, we implement complex algebraic numbers via their real
and imaginary part.

The major algorithm in this theory is a factorization algorithm which
factors a rational polynomial over the complex numbers.

This algorithm is then combined with explicit root algorithms to try to
factor arbitrary complex polynomials.

theory Complex-Algebraic-Numbers
imports
  Real-Roots
  Complex-Roots-Real-Poly
  Compare-Complex
  Jordan-Normal-Form. Char-Poly
  Berlekamp-Zassenhaus. Code-Abort-Gcd
  Interval-Arithmetic
begin

12.1 Complex Roots

hide-const (open) UnivPoly.coeff
hide-const (open) Module.smult
hide-const (open) Coset.order
abbreviation complex-of-int-poly :: int poly ⇒ complex poly where complex-of-int-poly ≡ map-poly of-int

abbreviation complex-of-rat-poly :: rat poly ⇒ complex poly where complex-of-rat-poly ≡ map-poly of-rat

lemma poly-complex-to-real: (poly (complex-of-int-poly p) (complex-of-real x) = 0) = (poly (real-of-int-poly p) x = 0)
proof
  have id: of-int = complex-of-real o real-of-int by auto
interpret cr: semiring-hom complex-of-real by (unfold-locales, auto)
show ?thesis unfolding id by (subst map-poly-map-poly [symmetric], force+)
qed

lemma represents-cnj: assumes p represents x shows p represents (cnj x)
proof
  from assms have p: p ≠ 0 and ipoly p x = 0 by auto
  hence rt: poly (complex-of-int-poly p) x = 0 by auto
  have poly (complex-of-int-poly p) (cnj x) = 0
    by (rule complex-conjugate-root [OF rt], subst coeffs-map-poly, auto)
  with p show ?thesis by auto
qed

definition poly-2i :: int poly where poly-2i ≡ [: 4, 0, 1:]

lemma represents-2i: poly-2i represents (2 * i)
unfolding represents-def poly-2i-def by simp

definition root-poly-Re :: int poly ⇒ int poly where root-poly-Re p = cf-pos-poly (poly-mult-rat (inverse 2) (poly-add p p))

lemma root-poly-Re-code[code]: root-poly-Re p = (let fs = coeffs (poly-add p p); k = length fs
  in cf-pos-poly (poly-of-list (map (λ(f, i). f * 2 ^ i) (zip fs [0..<k])))))
proof
  have [simp]: quotient-of (1 / 2) = (1,2) by eval
qed

definition root-poly-Im :: int poly ⇒ int list where root-poly-Im p = (let fs = factors-of-int-poly (poly-add p (poly-uminus p))
  in remdups ((if (∃ f ∈ set fs. coeff f 0 = 0) then [[0,1]] else [])) @
  [ cf-pos-poly (poly-div f poly-2i) . f ← fs, coeff f 0 ≠ 0])
lemma represents-root-poly:
assumes ipoly p x = 0 and p: p ≠ 0
shows (root-poly-Re p) represents (Re x)
and ∃ q ∈ set (root-poly-Im p). q represents (Im x)

proof –
let ?Rep = root-poly-Re p
let ?Imp = root-poly-Im p
from assms have ap: p represents x by auto
from represents-cnq[OF this] have apc: p represents (cnj x)
from represents-mul-rat[OF - represents-add[OF ap apc], of inverse 2]
have ?Rep represents (1 / 2 * (x + cnj x)) unfolding root-poly-Re-def Let-def
  by (auto simp: hom-distrib)
also have 1 / 2 * (x + cnj x) = of-real (Re x)
  by (simp add: complex-add-cnj)
finally have Rep: ?Rep ≠ 0 and rt: ipoly ?Rep (complex-of-real (Re x)) = 0

unfolding represents-def by auto
from rt[unfolded poly-complex-to-real]
have ipoly ?Rep (Re x) = 0.
with Rep show ?Rep represents (Re x) by auto
let ?q = poly-add p (poly-uminus p)
from represents-add[OF ap, of poly-uminus p - cnj x] represents-uminus[OF apc]

have apq: ?q represents (x - cnj x) by auto
from factors-int-poly-represents[OF this] obtain pi where pi: pi ∈ set (factors-of-int-poly ?q)
  and appi: pi represents (x - cnj x) and irr-pi: irreducible pi by auto
have id: inverse (2 * i) * (x - cnj x) = of-real (Im x)
apply (cases x) by (simp add: complex-split imaginary-unit.ctr legacy-Complex-simps)
from represents-2i have 12: poly-2i represents (2 * i) by simp
have ∃ qi ∈ set ?Imp. qi represents (inverse (2 * i) * (x - cnj x))
proof (cases x - cnj x = 0)
case False
have poly poly-2i 0 ≠ 0 unfolding poly-2i-def by auto
from represents-div[OF appi 12 this]
  represents-irr-non-0[OF irr-pi appi False, unfolded poly-0-coeff-0] pi
  show ?thesis unfolding root-poly-Im-def Let-def by (auto intro: bexI[of -
cf-pos-poly (poly-div pi poly-2i)])
next
case True
hence id2: Im x = 0 by (simp add: complex-cq-iff)
from appi[unfolded True represents-def] have coeff pi 0 = 0 by (cases pi, auto)
with pi have mem: [:0,1:] ∈ set ?Imp unfolding root-poly-Im-def Let-def by auto
  have [:0,1:] represents (complex-of-real (Im x)) unfolding id2 represents-def
  by simp
  with mem show ?thesis unfolding id by auto
qed
then obtain qi where qi: qi ∈ set ?Imp qi ≠ 0 and rt: ipoly qi (complex-of-real (Im x)) = 0

220
unfolding \( \text{id represents-def by auto} \)
from qi rt[unfolded poly-complex-to-real]
show \( \exists \ qi \in \text{set ?Imp. qi represents (Im x) by auto} \)
qed

Determine complex roots of a polynomial, intended for polynomials of degree 3 or higher, for lower degree polynomials use roots1 or roots2

hide-const (open) eq

primrec remdups-gen :: ('a ⇒ 'a ⇒ bool) ⇒ 'a list ⇒ 'a list where
remdups-gen eq [] = []
| remdups-gen eq (x # xs) = (if \( \exists \ y \in \text{set xs. eq x y} \) then
  remdups-gen eq xs else x # remdups-gen eq xs)

lemma real-of-3-remdups-equal-3[simp]: real-of-3 ' set (remdups-gen equal-3 xs) = real-of-3 ' set xs
  by (induct xs, auto simp: equal-3)

lemma distinct-remdups-equal-3: distinct (map real-of-3 (remdups-gen equal-3 xs))
  by (induct xs, auto, auto simp: equal-3)

lemma real-of-3-code [code]: real-of-3 x = real-of (Real-Alg-Quotient x)
  by (transfer, auto)

definition real-parts-3 p = roots-of-3 (root-poly-Re p)

definition pos-imaginary-parts-3 p =
  remdups-gen equal-3 (filter \( \lambda \ x. \ \text{sgn-3 x} = 1 \) (concat (map roots-of-3 (root-poly-Im p))))

lemma real-parts-3: assumes p: p ≠ 0 and ipoly p x = 0
  shows Re x ∈ real-of-3 ' set (real-parts-3 p)
  unfolding real-parts-3-def using represents-root-poly(1)[OF assms(2,1)]
  roots-of-3(1) unfolding represents-def by auto

lemma distinct-real-parts-3: distinct (map real-of-3 (real-parts-3 p))
  unfolding real-parts-3-def using roots-of-3(2).

lemma pos-imaginary-parts-3: assumes p: p ≠ 0 and ipoly p x = 0 and Im x > 0
  shows Im x ∈ real-of-3 ' set (pos-imaginary-parts-3 p)
  proof –
  from represents-root-poly(2)[OF assms(2,1)] obtain q where
    q: q ∈ set (root-poly-Im p) q represents Im x by auto
  from roots-of-3(1)[of q] have Im x ∈ real-of-3 ' set (roots-of-3 q) using q
    unfolding represents-def by auto
  then obtain i3 where i3: i3 ∈ set (roots-of-3 q) and id: Im x = real-of-3 i3
    by auto
from \( \text{Im } x > 0 \) have \( \text{sgn} (\text{Im } x) = 1 \) by simp 

hence \( \text{sgn} : \text{sgn-3 } i3 = 1 \) unfolding id by (metis of-rat-eq-1-iff sgn-3) 

show ?thesis unfolding pos-imaginary-parts-3-def real-of-3-remdups-equal-3 id 
using sgn i3 q(1) by auto 

qed 

lemma distinct-pos-imaginary-parts-3: distinct (map real-of-3 (pos-imaginary-parts-3 p)) 
unfolding pos-imaginary-parts-3-def by (rule distinct-remdups-equal-3) 

lemma remdups-gen-subset: set (remdups-gen eq xs) \( \subseteq \) set xs 
by (induct xs, auto) 

lemma positive-pos-imaginary-parts-3: assumes \( x \in \text{set } (\text{pos-imaginary-parts-3 } p) \) shows \( 0 < \text{real-of-3 } x \) 
proof – 
from subsetD[OF remdups-gen-subset assms unfolded pos-imaginary-parts-3-def] 
have \( \text{sgn-3 } x = 1 \) by auto 
thus ?thesis using sgn-3[of x] by (simp add: sgn-1-pos) 
qed 

definition pair-to-complex ri \( \equiv \) case ri of \((r, i)\) \( \Rightarrow \) Complex (real-of-3 r) (real-of-3 i) 

fun get-itvl-2 :: real-alg-2 \( \Rightarrow \) real interval where 
get-itvl-2 (Irrational n (p,l,r)) = Interval (of-rat l) (of-rat r) 
| get-itvl-2 (Rational r) = (let rr = of-rat r in Interval rr rr) 

lemma get-bounds-2: assumes invariant-2 x 
shows real-of-2 x \( \in \), get-itvl-2 x 
proof (cases x) 
  case (Irrational n plr) 
  with assms obtain p l r where plr: plr = (p,l,r) by (cases plr, auto) 
  from assms Irrational plr have inv1: invariant-1 (p,l,r) 
  and id: real-of-2 x = real-of-1 (p,l,r) by auto 
  thus ?thesis unfolding id using invariant-1D(1)[OF inv1] by (auto simp: plr Irrational) 
  qed (insert assms, auto simp: Let-def) 

lift-definition get-itvl-3 :: real-alg-3 \( \Rightarrow \) real interval is get-itvl-2 . 

lemma get-itvl-3: real-of-3 x \( \in \), get-itvl-3 x 
by (transfer, insert get-bounds-2, auto) 

fun tighten-bounds-2 :: real-alg-2 \( \Rightarrow \) real-alg-2 where 
tighten-bounds-2 (Irrational n (p,l,r)) = (case tighten-poly-bounds p l r (sgn (ipoly p r)) 
of (l',r',-) \Rightarrow Irrational n (p,l',r'))
lemma tighten-bounds-2: assumes inv: invariant-2 x
  shows real-of-2 (tighten-bounds-2 x) = real-of-2 x invariant-2 (tighten-bounds-2 x)
  proof (atomize(full), cases x)
    case (Irrational n plr)
    show real-of-2 (tighten-bounds-2 x) = real-of-2 x ∧
      invariant-2 (tighten-bounds-2 x) ∧
      (get-itvl-2 x = Interval l r) →
      get-itvl-2 (tighten-bounds-2 x) = Interval l' r' → r' - l' = (r - l) / 2
    proof -
    obtain p l r where plr: plr = (p,l,r) by (cases plr, auto)
    let ?tb = tighten-poly-bounds p l r (sgn (ipoly p r))
    obtain l' r' sr' where tb: ?tb = (l',r',sr') by (cases ?tb, auto)
    have id: tighten-bounds-2 x = irrational n (p,l',r') unfolding Irrational plr
      using tb by auto
    from inv[unfolded Irrational plr] have inv: invariant-1-2 (p, l, r)
      n = card {y. y ≤ real-of-1 (p, l, r) ∧ ipoly p y = 0} by auto
    have rof: real-of-2 x = real-of-1 (p, l, r)
      real-of-2 (tighten-bounds-2 x) = real-of-1 (p, l', r') using irrational plr id by auto
    from inv have inv1: invariant-1 (p, l, r) and poly-cond2 p by auto
    hence rc: ∃z. root-cond (p, l, r) x poly-cond2 p by auto
    note tb' = tighten-poly-bounds[OF tb rc refl]
    have eq: real-of-1 (p, l, r) = real-of-1 (p, l', r') using tb' inv1
      using invariant-1-sub-interval(2) by presburger
    from inv1 tb' have invariant-1 (p, l', r') by (metis invariant-1-sub-interval(1))
    hence inv2: invariant-2 (tighten-bounds-2 x) unfolding id using inv eq by auto
    thus ?thesis unfolding rof eq unfolding id unfolding Irrational plr
      using tb'(1-4) arg-cong[OF tb'(5), of real-of-rat] by (auto simp: hom-distribs)
    qed
  qed (auto simp: Let-def)

lift-definition tighten-bounds-3 :: real-alg-3 ⇒ real-alg-3 is tighten-bounds-2
using tighten-bounds-2 by auto

lemma tighten-bounds-3:
  real-of-3 (tighten-bounds-3 x) = real-of-3 x
  get-itvl-3 x = Interval l r ⇒
  get-itvl-3 (tighten-bounds-3 x) = Interval l' r' ⇒ r' - l' = (r - l) / 2
by (transfer, insert tighten-bounds-2, auto)

partial-function (tailrec) filter-list-length
  :: ('a ⇒ 'a) ⇒ ('a ⇒ bool) ⇒ nat ⇒ 'a list ⇒ 'a list where
  [code]: filter-list-length f p n xs = (let ys = filter p xs

223
in if length ys = n then ys else
filter-list-length f p n (map f ys)

lemma filter-list-length: assumes length (filter P xs) = n
and \( \forall i. x : xs \Rightarrow P x \Rightarrow \neg p ((f \sim i) x) \)
and \( \forall x : xs \Rightarrow \neg P x \Rightarrow \exists i. \neg p ((f \sim i) x) \)
and \( \forall x. y : g (f x) = y x \)
and \( P : \forall x. P (f x) = P x \)
shows map g (filter-list-length f p n xs) = map g (filter P xs)
proof –
from assms(3) have \( \forall x. \exists i. x : xs \Rightarrow \neg P x \Rightarrow \neg p ((f \sim i) x) \)
by auto
from choice[OF this] obtain i where i: \( \forall x. x : xs \Rightarrow \neg P x \Rightarrow \neg p ((f \sim i) x) \)
by auto
define m where m = max-list (map i xs)
have m: \( \forall x. x : xs \Rightarrow \neg P x \Rightarrow \exists i. \neg p ((f \sim i) x) \)
using max-list[of - map i xs, folded m-def] i by auto
show ?thesis using assms(1-2) m
proof (induct m arbitrary: xs rule: less-induct)
case (less m xs)
define ys where ys = filter p xs
have xs-ys: filter P xs = filter P ys unfolding ys-def filter-filter
by (rule filter-cong[OF refl], insert less(3)[of - 0], auto)
have filter (P o f) ys = filter P ys using P unfolding o-def by auto
hence id3: filter P (map f ys) = map f (filter P ys) unfolding filter-map by simp
hence id2: map g (filter P (map f ys)) = map g (filter P ys) by (simp add: g)
show ?case
proof (cases length ys = n)
case True
hence id: filter-list-length f p n xs = ys unfolding ys-def
filter-list-length.simps[of - - - xs] Let-def by auto
show ?thesis using True unfolding id xs-ys using less(2)
by (metis filter-id-conv length-filter-less less-le xs-ys)
next
case False
{
assume m = 0
from less(4)[unfolded this] have Pp: \( \forall x : xs \Rightarrow \neg P x \Rightarrow \neg p x \) for x
by auto
with xs-ys False[folded less(2)] have False
by (metis (mono-tags, lifting) filter-True mem-Collct-eq set-filter ys-def)
} note m0 = this
then obtain M where mM: m = Suc M by (cases m, auto)
hence m: M < m by simp
from False have id: filter-list-length f p n xs = filter-list-length f p n (map f ys)
unfolding ys-def filter-list-length.simps[of - - - xs] Let-def by auto

224
show \textit{thesis unfolding} \(id\) \(xs-ys\) \(id2[\text{symmetric}]\)
proof (rule less(1)[OF \(m\)])
fix \(y\)
assume \(y \in \text{set} (\text{map} \ f \ ys)\)
then obtain \(x\) where \(x: x \in \text{set} \ xs \ p \ x \and y: y = f \ x\) unfolding \(ys\)-def
by auto
\{
assume \(\neg \ P \ y\)
hence \(\neg \ P \ x\) unfolding \(y \ P\).
from less(4)[OF \(x(1)\) this] obtain \(i \ where \ i: i \leq m \and p: \neg \ p \ ((f \ ^{\sim} \ i) \ y)\)
\}
\{ fix \(i\)
assume \(P \ y\)
hence \(P \ x\) unfolding \(y \ P\).
\} fix \(i\)
assume \(P \ y\)
hence \(P \ x\) unfolding \(y \ P\).
next
show length (filter \(P\) (\text{map} \(f \ ys\))) = \(n\) unfolding \(id3\) \(length\)-map using \(xs-ys\)
less(2) by auto
qed
qed
qed

definition \textit{complex-roots-of-int-poly3} :: int poly \Rightarrow complex list where
complex-roots-of-int-poly3 \(p\) \equiv \text{let} \(n = \text{degree} \ p\);
\(rrts = \text{real-roots-of-int-poly} \ p\);
\(nr = \text{length} \ rrts\);
\(crts = \text{map} (\lambda \ r. \ Complex \ r \ 0) \ rrts\)
in
if \(n = nr\) then \(crts\)
else let \(nr-crts = n - nr\) in if \(nr-crts = 2\) then
let \(pp = \text{real-of-int-poly} \ p \ \text{div} \ (\text{prod-list} \ (\text{map} (\lambda \ x. [:-x,1:]) \ rrts ));\)
\(cpp = \text{map-poly} (\lambda \ r. \ Complex \ r \ 0) \ pp\)
in \(crts @ \text{roots2} \ cpp\) else
let \(nr-pos-crts = nr-crts \text{ div} \ 2;\)
\(rxs = \text{real-parts-3} \ p;\)
\(ixs = \text{pos-imaginary-parts-3} \ p;\)
\(rts = [(rx, ix). \ rx < - rxs, \ ix < - ixs];\)
\(crts' = \text{map} \ \text{pair-to-complex}\)
\((\text{filter-list-length} \ (\text{map-prod} \ \text{tighten-bounds-3} \ \text{tighten-bounds-3}))\)
\((\lambda \ (r, i). \ 0 \in_c \ \text{ipoly-complex-interval} \ p \ (\text{Complex-Interval} \ (\text{get-itvl-3} \ r) \ (\text{get-itvl-3} \ i)))\) \text{nr-pos-crts} \ \text{rts}
\text{in} \ \text{crts} @ \ (\text{concat} \ (\text{map} \ (\lambda \ x. \ [x, \ \text{cnj} \ x]) \ \text{crts}'))

\text{definition} \ \text{complex-roots-of-int-poly-all} :: \ \text{int} \ \text{poly} \Rightarrow \ \text{complex} \ \text{list} \ \text{where}
\text{complex-roots-of-int-poly-all} \ \text{p} = (\text{let} \ \text{n} = \text{degree} \ \text{p} \ \text{in}
  \text{if} \ \text{n} \geq 3 \ \text{then} \ \text{complex-roots-of-int-poly3} \ \text{p}
  \text{else if} \ \text{n} = 1 \ \text{then} \ [\text{roots1} \ (\text{map-poly} \ \text{of-int} \ \text{p})]
  \text{else if} \ \text{n} = 2 \ \text{then} \ \text{croots2} \ (\text{map-poly} \ \text{of-int} \ \text{p})
  \text{else}
)\)

\text{lemma} \ \text{in-real-itvl-get-bounds-tighten}: \ \text{real-of-3} \ \text{x} \ \in \ \text{i} \ \text{get-itvl-3} \ (\text{(tighten-bounds-3} \ \sim \ \text{n)} \ \text{x})
\text{proof} \ (\text{induct} \ \text{n} \ \text{arbitrary}: \ \text{x})
\text{case} \ 0
  \text{thus} \ ?\text{case using} \ \text{get-itvl-3}[\text{of} \ \text{x}] \ \text{by} \ \text{simp}
\text{next}
\text{case} \ (\text{Suc} \ \text{n} \ \text{x})
  \text{have} \ \text{id}: \ (\text{tighten-bounds-3} \ \sim \ (\text{Suc} \ \text{n})) \ \text{x} = (\text{tighten-bounds-3} \ \sim \ \text{n}) \ (\text{tighten-bounds-3} \ \text{x})
  \text{by} \ (\text{metis} \ \text{comp-apply} \ \text{funpow-Suc-right})
  \text{show} \ ?\text{case unfolding} \ \text{id} \ \text{tighten-bounds-3}(\text{Suc} \ \text{of} \ \text{x}, \ \text{symmetric}) \ \text{by} \ (\text{rule} \ \text{Suc})
\text{qed}

\text{lemma} \ \text{sandwich-real}:
\text{fixes} \ \text{l \ r} :: \ \text{nat} \Rightarrow \ \text{real}
\text{assumes} \ \text{la}: \ \text{l} \longrightarrow \ \text{a} \ \text{and} \ \text{ra}: \ \text{r} \longrightarrow \ \text{a}
\text{and} \ \text{lm}: \ \bigwedge \ i. \ \text{l} \ \text{i} \ \leq \ \text{m} \ \text{i} \ \text{and} \ \text{mr}: \ \bigwedge \ i. \ \text{m} \ \text{i} \ \leq \ \text{r} \ \text{i}
\text{shows} \ \text{m} \longrightarrow \ \text{a}
\text{proof} \ (\text{rule} \ \text{LIMSEQ-I})
\text{fix} \ \text{e} :: \ \text{real}
\text{assume} \ \text{0} < \ \text{e}
\text{hence} \ \text{e} = \text{0} < \ \text{e} / 2 \ \text{by} \ \text{simp}
\text{from} \ \text{LIMSEQ-D}[\text{OF} \ \text{la} \ \text{e}] \ \text{obtain} \ \text{n1} \ \text{where} \ \text{n1}: \ \bigwedge \ \text{n}. \ \text{n} \ \geq \ \text{n1} \ \Longrightarrow \ \text{norm} \ (\text{l} \ \text{n} - \ \text{a}) < \text{e}/2 \ \text{by} \ \text{auto}
\text{from} \ \text{LIMSEQ-D}[\text{OF} \ \text{m} \ \text{e}] \ \text{obtain} \ \text{n2} \ \text{where} \ \text{n2}: \ \bigwedge \ \text{n}. \ \text{n} \ \geq \ \text{n2} \ \Longrightarrow \ \text{norm} \ (\text{r} \ \text{n} - \ \text{a}) < \text{e}/2 \ \text{by} \ \text{auto}
\text{show} \ \exists \ \text{no}. \ \forall \ \text{n}\geq\text{no}. \ \text{norm} \ (\text{m} \ \text{n} - \ \text{a}) < \ \text{e}
\text{proof} \ (\text{rule} \ \text{exI}[\text{of} - \ \text{max} \ \text{n1} \ \text{n2}], \ \text{intro} \ \text{allI} \ \text{impl})
\text{fix} \ \text{n}
\text{assume} \ \text{max} \ \text{n1} \ \text{n2} \ \leq \ \text{n}
\text{with} \ \text{n1} \ \text{n2} \ \text{have} \ \ast: \ \text{norm} \ (\text{l} \ \text{n} - \ \text{a}) < \text{e}/2 \ \text{norm} \ (\text{r} \ \text{n} - \ \text{a}) < \text{e}/2 \ \text{by} \ \text{auto}
\text{from} \ \text{lm}[\text{of} \ \text{n}] \ \text{mr}[\text{of} \ \text{n}] \ \text{have} \ \text{norm} \ (\text{m} \ \text{n} - \ \text{a}) \ \leq \ \text{norm} \ (\text{l} \ \text{n} - \ \text{a}) + \ \text{norm} \ (\text{r} \ \text{n} - \ \text{a}) \ \text{by} \ \text{simp}
\text{with} \ \ast \ \text{show} \ \text{norm} \ (\text{m} \ \text{n} - \ \text{a}) < \ \text{e} \ \text{by} \ \text{auto}
\text{qed}
\text{qed}
lemma real-of-tighten-bounds-many[simp]: real-of-3 ((tighten-bounds-3 ^^ i) x) = real-of-3 x
  apply (induct i) using tighten-bounds-3 by auto

definition lower-3 where lower-3 x i ≡ interval.lower (get-itvl-3 ((tighten-bounds-3 ^^ i) x))
definition upper-3 where upper-3 x i ≡ interval.upper (get-itvl-3 ((tighten-bounds-3 ^^ i) x))

lemma interval-size-3: upper-3 x i − lower-3 x i = (upper-3 x 0 − lower-3 x 0) / 2^i
proof (induct i)
case (Suc i)
  have upper-3 x (Suc i) − lower-3 x (Suc i) = (upper-3 x i − lower-3 x i) / 2
    unfolding upper-3-def lower-3-def using tighten-bounds-3 get-itvl-3 by auto
  with Suc show ?case by auto
qed auto

lemma interval-size-3-tendsto-0: (λi. (upper-3 x i − lower-3 x i)) −→ 0
by (subst interval-size-3, auto intro: LIMSEQ-divide-realpow-zero)

lemma dist-tendsto-0-imp-tendsto: (λi. |f i − a| :: real) −→ 0 =⇒ f −→ a
using LIM-zero-cancel tendsto-rabs-zero-iff by blast

lemma upper-3-tendsto: upper-3 x −→ real-of-3 x
proof (rule dist-tendsto-0-imp-tendsto, rule sandwich-real)
  fix i
  obtain l r where lr: get-itvl-3 ((tighten-bounds-3 ^^ i) x) = Interval l r
    by (metis interval.collapse)
  with get-itvl-3[of (tighten-bounds-3 ^^ i) x]
  show |(upper-3 x i) − real-of-3 x| ≤ (upper-3 x i − lower-3 x i)
    unfolding upper-3-def lower-3-def by auto
  qed (insert interval-size-3-tendsto-0, auto)

lemma lower-3-tendsto: lower-3 x −→ real-of-3 x
proof (rule dist-tendsto-0-imp-tendsto, rule sandwich-real)
  fix i
  obtain l r where lr: get-itvl-3 ((tighten-bounds-3 ^^ i) x) = Interval l r
    by (metis interval.collapse)
  with get-itvl-3[of (tighten-bounds-3 ^^ i) x]
  show |lower-3 x i − real-of-3 x| ≤ (upper-3 x i − lower-3 x i)
    unfolding upper-3-def lower-3-def by auto
  qed (insert interval-size-3-tendsto-0, auto)

lemma tends-to-tight-bounds-3: (λx. get-itvl-3 ((tighten-bounds-3 ^^ x) y)) −→, real-of-3 y
  using lower-3-tendsto[of y] upper-3-tendsto[of y] unfolding lower-3-def upper-3-def
  interval-tendsto-def o-def by auto

227
lemma complex-roots-of-int-poly3: assumes p: p ≠ 0 and sf: square-free p
shows set (complex-roots-of-int-poly3 p) = {x. ipoly p x = 0} (is ?l = ?r)
distinct (complex-roots-of-int-poly3 p)

proof –
interpret map-poly-inj-dom-hom of-real..
define q where q = real-of-int-poly p
let ?q = map-poly complex-of-real q
from p have qb: q ≠ 0 unfolding q-def by auto
hence q: ?q ≠ 0 by auto
define rr where rr = real-roots-of-int-poly p
define rrts where rrts = map (λr. Complex r 0) rr

note d = complex-roots-of-int-poly3-def[of p, unfolded Let-def, folded rr-def, folded rrts-def]

have rr: set rr = {x. ipoly p x = 0} unfolding rr-def
have rrts: set rrts = {x. poly ?q x = 0 ∧ x ∈ ℝ} unfolding rrts-def set-map rr

complex-of-real-def[symmetric] by (auto elim: Reals-cases)
have dist: distinct rr unfolding rr-def using real-roots-of-int-poly(2) .
from dist have dist1: distinct rrts unfolding rrts-def distinct-map inj-on-def by auto

have lrv: length rr = card {x. poly (real-of-int-poly p) x = 0}
unfolding rr-def using real-roots-of-int-poly[of p] p distinct-card by fastforce
have lv: length rr = card {x. poly ?q x = 0 ∧ x ∈ ℝ} unfolding lrv q-def[symmetric]

proof –
have card {x. poly q x = 0} ≤ card {x. poly (map-poly complex-of-real q) x = 0 ∧ x ∈ ℝ} (is ?l ≤ ?r)
by (rule card- injustice-on-le[of of-real], insert poly-roots-finite[OF q], auto simp: inj-on-def)
moreover have ?l ≥ ?r
by (rule card-injustice-on-le[of Re, OF - - poly-roots-finite[OF q0]], auto simp: inj-on-def elim!: Reals-cases)
ultimately show ?l = ?r by simp
qed
have conv: ∃ x. ipoly p x = 0 ←→ poly ?q x = 0
unfolding q-def by (subst map-poly-map-map, auto simp: o-def)
have r: ?r = {x. poly ?q x = 0} unfolding conv ..
have ?t = {x. ipoly p x = 0} ∧ distinct (complex-roots-of-int-poly3 p)
proof (cases degree p = length rr)
case False note oFalse = this
show ?thesis
proof (cases degree p = length rr = 2)
case False
let ?rr = (degree p = length rr) div 2
define cpxI where cpxI = pos-imaginary-parts-3 p
define cpxR where cpxR = real-parts-3 p
let ?rrs = [(rx, ix)]. rr <- cpxR, ix <- cpxI
define cpx where cpx = map pair-to-complex (filter (λ c. ipoly p (pair-to-complex

228
\(c) = 0\)  
\[\text{let } ?LL = \text{cpx } @ \text{map cnj cpx}\]
\[\text{let } ?LL' = \text{concat } (\text{map } (\lambda x. [x, \text{cnj} x]) \text{ cpx})\]
\[\text{let } ?ll = \text{rrts } @ \text{ ?LL}\]
\[\text{let } ?ll' = \text{rrts } @ \text{ ?LL'}\]
\[\text{have cpx: set cpx } \subseteq \text{ ?r unfolding cpx-def by auto}\]
\[\text{have cpx: cnj } \text{ set cpx } \subseteq \text{ ?r using cpx unfolding r}\]
\[\text{by (auto intro: complex-conjugate-root[of ?q] simp: Reals-def)}\]
\[\text{have set ?ll } \subseteq \text{ ?r using rrts cpx cxpx unfolding r by auto}\]

moreover

{  
  fix x :: complex  
  assume rt: ipoly p x = 0
  
  {  
    fix x  
    assume rt: ipoly p x = 0  
    and gt: Im x > 0  
    define rx where rx = Re x  
    let ?x = Complex rx (Im x)  
    have x. x = ?x by (cases x, auto simp: rx-def)  
    from rt x have rt': ipoly p ?x = 0 by auto
    from real-parts-3[OF p rt, folded rx-def] pos-imaginary-parts-3[OF p rt gt]  
    rt'
    have ?x ∈ set cpx unfolding cpx-def cpxI-def cpxR-def
    by (force simp: pair-to-complex-def[abs-def])  
    hence x ∈ set cpx using x by simp
  }
  note gt = this
  have cases: Im x = 0 ∨ Im x > 0 ∨ Im x < 0 by auto
  from rt have rt': ipoly p (cnj x) = 0 unfolding conv
  by (intro complex-conjugate-root[of ?q x], auto simp: Reals-def)
  {  
    assume Im x > 0
    from gt[OF rt this] have x ∈ set ?ll by auto
  }
  moreover
  
  {  
    assume Im x < 0
    hence Im (cnj x) > 0 by simp
    from gt[OF rt' this] have cnj (cnj x) ∈ set ?ll unfolding set-append set-map by blast
    hence x ∈ set ?ll by simp
  }
  moreover
  
  {  
    assume Im x = 0
    hence x ∈ R using complex-is-Real-iff by blast
    with rt rtts have x ∈ set ?ll unfolding conv by auto
  }
ultimately have $x \in \text{set ?ll}$ using cases by blast

ultimately have lr: set ?ll = \{ x. ipoly p x = 0 \} by blast
let ?rr = map real-of-3 cpxR
let ?pi = map real-of-3 cpxI
have dist2: distinct ?rr unfolding cpxR-def by (rule distinct-real-parts-3)
have dist3: distinct ?pi unfolding cpxI-def by (rule distinct-pos-imaginary-parts-3)
have idd: concat (map (map pair-to-complex) (map (\lambda rx. \text{map} (Pair rx) cpxI)) cpxR)

= concat (map (\lambda. \text{map} (\lambda i. \text{Complex} (\text{real-of-3} r) (\text{real-of-3} i)) cpxI)) cpxR

unfolding pair-to-complex-def by (auto simp: o-def)
have dist4: distinct cpx unfolding cpx-def

proof (rule distinct-map-filter, unfold map-concat idd, unfold distinct-conv-nth, intro allI impI, goal-cases)
case (1 i j)
from nth-concat-diff[OF 1, unfolded length-map] dist2[unfolded distinct-conv-nth]
dist3[unfolded distinct-conv-nth] show ?case by auto
qed

have dist5: distinct (map cnj cpx) using dist4 unfolding distinct-map by
(auto simp: inj-on-def)

{ fix x :: complex
  have rrts: $x \in \text{set rrts} \implies \text{Im} x = 0$ unfolding rrts-def by auto
  have cpx: $\forall x. x \in \text{set cpx} \implies \text{Im} x > 0$ unfolding cpx-def cpxI-def
    by (auto simp: pair-to-complex-def[abs-def] positive-pos-imaginary-parts-3)
  have cpx': $x \in \text{cnj } ' \text{set cpx} \implies \text{sgn} (\text{Im} x) = -1$ using cpx by auto
  have $x \notin \text{set rrts} \cap \text{set cpx} \cup \text{set rrts} \cap \text{cnj } ' \text{set cpx} \cup \text{set cpx} \cap \text{cnj } ' \text{set cpx}$
    using rrts cpx[of x] cpx' by auto
  } note dist6 = this

have dist: distinct ?ll unfolding distinct-append using dist6 by (auto simp: distI dist4 dist5)
let ?p = complex-of-int-poly p
have pp: ?p \neq 0 using p by auto
from p square-free-of-int-poly[OF sf] square-free-rsquarefree
have rsf: rsquarefree ?p by auto
from dist lr have length ?ll = card \{ x. poly ?p x = 0 \}
  by (metis distinct-card)
also have \ldots = degree p
  using rsf unfolding rsquarefree-card-degree[OF pp] by simp
finally have deg-len: degree p = length ?ll by simp
let \?P = \lambda c. \text{ipoly} p (\text{pair-to-complex} c) = 0
let \?itvl = \lambda r i. \text{ipoly-complex-interval} p (\text{Complex-Interval} (\text{get-itvl-3} r) (\text{get-itvl-3} i))
let \?itvl = \lambda (r,i). \text{get-itvl} r i
let \?p = (\lambda (r,i). 0 \in_c (\?itvl r i))
let \?tb = tighten-bounds-3
let \?f = map-prod \?tb \?tb

230
proof (rule filter-list-length)
    have length (filter ?P ?rts) = length cpx
        unfolding cpx-def by simp
    also have \ldots = ?nr unfolding deg-len by (simp add: rrts-def)
finally show length (filter ?P ?rts) = ?nr by auto
next
fix n x
assume x : ?P x
obtain r i where xri: x = (r,i) by force
have id: (if \ldots n) x = ((?tb \ldots n) r, (?tb \ldots n) i) for n unfolding xri
    by (induct n, auto)
have px: pair-to-complex x = Complex (real-of-3 r) (real-of-3 i)
    unfolding xri pair-to-complex-def by auto
show ?p ((if \ldots n) x)
    unfolding id split
    by (rule ipoly-complex-interval[of pair-to-complex x - p, unfolded x], unfold px,
        auto simp: in-complex-interval-def in-real-itvl-get-bounds-tighten)
next
fix x
assume x : x \in set ?rts \land ?P x
let ?x = pair-to-complex x
obtain r i where xri: x = (r,i) by force
have id: (if \ldots n) x = ((?tb \ldots n) r, (?tb \ldots n) i) for n unfolding xri
    by (induct n, auto)
have px: ?x = Complex (real-of-3 r) (real-of-3 i)
    unfolding xri pair-to-complex-def by auto
have cvg: (\lambda ia. Complex-Interval (get-itvl-3 ((?tb \ldots ia) r)) (get-itvl-3 ((?tb
\ldots ia) i))) \longrightarrow_c ipoly p ?x
    unfolding id split px
proof (rule ipoly-complex-interval-tendsto)
    show (\lambda ia. Complex-Interval (get-itvl-3 ((?tb \ldots ia) r)) (get-itvl-3 ((?tb
\ldots ia) i))) \longrightarrow_c Complex (real-of-3 r) (real-of-3 i)
    unfolding complex-interval-tendsto-def by (simp add: tends-to-tight-bounds-3 o-def)
qed
from complex-interval-tendsto-neq[OF this x(2)]
show \exists i. \sim p ((if \ldots i) x) unfolding id by auto
next
show pair-to-complex (if x) = pair-to-complex x for x
    by (cases x, auto simp: pair-to-complex-def tighten-bounds-3(1))
next
show ?P (if x) = ?P x for x
    by (cases x, auto simp: pair-to-complex-def tighten-bounds-3(1))
qed
have l: complex-roots-of-int-poly3 p = ?ll'
unfolding \ldots filter cpx-def[symmetric] cpxI-def[symmetric] cpxR-def[symmetric]

231
using False oFalse

by auto

have distinct ??l' = (distinct rts ∧ distinct ??ll' ∧ set rts ∩ set ??ll = {}) unfolding distinct-append ..

also have set ??ll' = set ??ll by auto

also have distinct ??ll' = distinct ??ll by (induct cpoly, auto)

finally have distinct ??l' = distinct ??ll unfolding distinct-append by auto

with dist have distinct ??ll' by auto

with br l show ?thesis by auto

next

case True

let ?c2 = map-poly of-real :: real poly ⇒ complex poly

define pp where pp = complex-of-int-poly p

have id: pp = map-poly af-of-real q unfolding q-def pp-def

by (subst map-poly-map-poly, auto simp: o-def)

let ?rts = map (λ x. [:−x,1;]) rr

define rts where rts = prod-list ?rts

let ?c2 = ?c2 (q div rts)

have pq: (∀ x. ipoly p x = 0) ←→ poly q x = 0 unfolding q-def by simp

from True have 2: degree q = card {x. poly q x = 0} = 2 unfolding pq

symmetric

unfolding q-def by simp

from True have id: degree p = length rr ←→ False
degree p − length rr = 2 ←→ True by auto

have l: ??l = of-real {x. poly q x = 0} ∪ set (croots2 ?c2)

unfolding d rts-def id if-False if-True set-append rts Reals-def

by (fold complex-of-real-def q-def, auto)

from dist

have len-rr: length rr = card {x. poly q x = 0} unfolding rr

symmetric

by (simp add: distinct-card)

have rr': (∀ r. r ∈ set rr ⇒ poly q r = 0) using rr unfolding q-def by simp

with dist have q = q div prod-list ?rts * prod-list ?rts

proof (induct rr arbitrary: q)

case (Cons r rr)

note dist = Cons(2)

let ?p = q div [:−r,1;]

from Cons.prems(2) have poly q r = 0 by simp

hence [:−r,1;] dvd q using poly-eq-0-iff-dvd by blast

from dvd-mult-div-cancel[OF this]

have q = ?p * [:−r,1;] by simp

moreover have ?p = ?p div (Π x∈rr. [:−x,1;]) * (Π x∈rr. [:−x,1;])

proof (rule Cons.hyps)

show distinct rr using dist by auto

fix s

assume s ∈ set rr

with dist Cons(3) have s ≠ r poly q s = 0 by auto

hence poly (?p * [:−1 * r,1;]) s = 0 using calculation by force

thus poly ?p s = 0 by (simp add: s ≠ r)
qed
ultimately have \( q : q = q \div (\prod_{x\in r} [-x, 1]) \)
* \([-r, 1]\)
  by auto
also have \( \ldots = (\prod_{x\in r} [-x, 1])) * (\prod_{x\in r \neq r} [-x, 1]) \)
  unfolding mult.assoc by simp
also have \( q \div (\prod_{x\in r} [-x, 1]) = q \div (\prod_{x\in r \neq r} [-x, 1]) \)
  unfolding poly-div-mult-right[symmetric] by simp
finally show \(?case\).
qed simp

**hence q-div: q = q \div rts \div rts unfolding rts-def.**
from q-div q0 have q \div rts \neq 0 rts \neq 0 by auto
from degree-mult-eq[OF this] have degree q = degree (q \div rts) + degree rts
using q-div by simp
also have degree rts = length rr unfolding rts-def by (rule degree-linear-factors)
also have \( \ldots = \text{card} \{x. \text{poly} q x = 0\} \) unfolding len-rr by simp
finally have deg2: degree \(?c2 = 2\) using 2 by simp
note croots2 = croots2[OF deg2, symmetric]
have \( q \cap \text{croots} \) using q-div by simp
also have \( \ldots = q \cap \text{croots} \cap \text{croots} \) unfolding hom-distrib by simp
finally have q-prod: \( q = q \cap \text{croots} \cap \text{croots} \).
from croots2 l
have l: \( l = \text{of-real} \cap \{x. \text{poly} q x = 0\} \cup \{x. \text{poly} \text{croots} x = 0\} \) by simp
from r[unfolded q-prod]
have r: \( r = \{x. \text{poly} (\text{croots} rts) x = 0\} \cup \{x. \text{poly} \text{croots} x = 0\} \) by auto
also have \( \text{croots} \cup \text{croots} \cap \text{croots} \) by (simp add: rts-def o-def
of-real-poly-hom-hom-prod-list)
also have \( \{x. \text{poly} \ldots x = 0\} = \text{of-real} \cap \text{set} rr \)
  unfolding poly-prod-list-zero-iff by auto
also have set rr = \( \{x. \text{poly} q x = 0\} \) unfolding rr q-def by simp
finally have br: \( \text{br} = \text{unfolding} l \) by simp
show \(?thesis\)

proof (intro conj[OF br])
from sf have sf: square-free q unfolding q-def by (rule square-free-of-int-poly)
  \{
  interpret field-hom-0' complex-of-real ..
  from sf have square-free ?q unfolding square-free-map-poly .
  } note sf = this
have l: complex-roots-of-int-poly3 p = rts \& croots2 \& \text{croots}
  unfolding d \text{rts-def id} if-False if-True set-append rts q-def complex-of-real-def
by auto
have dist2: distinct (croots2 \& \text{croots}) unfolding croots2-def Let-def by auto

\{
  fix x
  assume x: \( x \in \text{set} \) (croots2 \& \text{croots}) \( x \in \text{set} \) rts
from x(1)[unfolded croots2] have x1: poly \& \text{croots} \& \text{croots} \& \text{croots} x = 0 by auto
from x(2) have x2: poly \& \text{croots} \& \text{croots} \& \text{croots} x = 0
  unfolding rts-def rts-def complex-of-real-def[symmetric]

233
by (auto simp: poly-prod-list-zero-iff o-def)
from square-free-multD[OF sf[unfolded q-prod], of \[-x, L\]]
x1 x2 have False unfolding poly-eq-0-iff-dvd by auto
}
note dist3 = this
show distinct (complex-roots-of-int-poly3 p) unfolding l distinct-append
by (intro conjI dist1 dist2, insert dist3, auto)
qed
qed
next
case True
have card \{x. poly ?q x = 0\} \leq degree ?q
by (rule poly-roots-degree[OF q])
also have \ldots = degree p unfolding q-def by simp
finally have le: card \{x. poly ?q x = 0 \& x \in \mathbb{R}\} \leq card \{x. poly ?q x = 0 \& x \in \mathbb{R}\}
by auto
have \{x. poly ?q x = 0 \& x \in \mathbb{R}\} = \{x. poly ?q x = 0\}
by (rule card-seteq[OF - - le], insert poly-roots-finite[OF q], auto)
thus distinct (complex-roots-of-int-poly3 p) \?l = \?r by auto
qed

lemma complex-roots-of-int-poly-all: assumes sf: degree p \geq 3 \implies square-free p
shows p \neq 0 \implies set (complex-roots-of-int-poly-all p) = \{x. ipoly p x = 0\}
(is - \implies set \?l = \?r)
and distinct (complex-roots-of-int-poly-all p)
proof –
  note d = complex-roots-of-int-poly-all-def Let-def
  have (p \neq 0 \implies set \?l = \?r) \& (distinct (complex-roots-of-int-poly-all p))
  proof (cases degree p \geq 3)
    case True
     hence p: p \neq 0 by auto
    from True complex-roots-of-int-poly3[OF p] sf show \?thesis unfolding d by auto
  next
    case False
    let ?p = map-poly (of-int :: int \Rightarrow complex) p
    have deg: degree ?p = degree p
    by (simp add: degree-map-poly)
    show \?thesis
    proof (cases degree p = 1)
      case True
       hence l: \?l = [roots1 ?p] unfolding d by auto
      from True have degree ?p = 1 unfolding deg by auto
      from roots1[OF this] show \?thesis unfolding l roots1-def by auto
    next
      case False
      show \?thesis
      proof (cases degree p = 2)
case True
    hence l: ?l = roots2 ?p unfolding d by auto
    from True have degree ?p = 2 unfolding deg by auto
    from roots2[OF this] show ?thesis unfolding l by (simp add: roots2-def)

next
    case False
    with ⟨degree p ≠ 1; degree p ≠ 2; (degree p ≥ 3) ⟩ have True: degree p = 0 by auto
    hence l: ?l = [] unfolding d by auto
    from True have degree ?p = 0 unfolding deg by auto
    from roots0[OF - this] show ?thesis unfolding l by simp
    qed
    qed
    qed

thus p ≠ 0 =⇒ set ?l = ?r distinct (complex-roots-of-int-poly-all p) by auto
qed

It now comes the preferred function to compute complex roots of an integer polynomial.

definition complex-roots-of-int-poly :: int poly ⇒ complex list where
    complex-roots-of-int-poly p = (let ps = (if degree p ≥ 3 then factors-of-int-poly p else [p])
    in concat (map complex-roots-of-int-poly-all ps))

definition complex-roots-of-rat-poly :: rat poly ⇒ complex list where

lemma complex-roots-of-int-poly:
    shows p ≠ 0 =⇒ set (complex-roots-of-int-poly p) = {x. ipoly p x = 0} (is - =⇒ ?l = ?r)
    and distinct (complex-roots-of-int-poly p)
proof −
    have (p ≠ 0 =⇒ ?l = ?r) ∧ (distinct (complex-roots-of-int-poly p))
    proof (cases degree p ≥ 3)
        case False
        hence complex-roots-of-int-poly p = complex-roots-of-int-poly-all p
        unfolding complex-roots-of-int-poly-def Let-def by auto
    next
    case True
    |
    fix q
    assume q ∈ set (factors-of-int-poly p)
    from factors-of-int-poly(1)[OF refl this] irreducible-imp-square-free[of q]
    have 0: q ≠ 0 and sf: square-free q by auto
    from complex-roots-of-int-poly-all(1)[OF sf 0] complex-roots-of-int-poly-all(2)[OF sf]
have set (complex-roots-of-int-poly-all q) = {x. ipoly q x = 0}
distinct (complex-roots-of-int-poly-all q)

} note all = this
from True have
\[ I = (\bigcup (\lambda p. \text{set (complex-roots-of-int-poly-all p))} \times \text{set (factors-of-int-poly p))} \]
unfolding complex-roots-of-int-poly-def Let-def by auto
also have \[ \ldots = (\bigcup (\lambda p. \{x. \text{ipoly p x = 0}\}) \times \text{set (factors-of-int-poly p))} \]
using all by blast
finally have \[ I = (\bigcup (\lambda p. \{x. \text{ipoly p x = 0}\}) \times \text{set (factors-of-int-poly p))} \].

have br: p ≠ 0 \rightarrow I = \{ r \text{ using l factors-of-int-poly(2)[OF refl, of p]} \text{ by auto} \}
show ?thesis
proof (rule conjI[OF br])
from True have id: complex-roots-of-int-poly p =
concat (map complex-roots-of-int-poly-all (factors-of-int-poly p))
unfolding complex-roots-of-int-poly-def Let-def by auto
show distinct (complex-roots-of-int-poly p) unfolding id distinct-conv-nth
proof (intro all_impl, goal-cases)
case (1 i j)
let ?fp = factors-of-int-poly p
let ?rr = complex-roots-of-int-poly-all
let ?cc = concat (map ?rr (factors-of-int-poly p))
from nth-concat-diff[OF I, unfolded length-map]
obtain j1 k1 j2 k2 where
*: (j1,k1) ≠ (j2,k2)
j1 < length ?fp j2 < length ?fp and
k1 < length (map ?rr ?fp ! j1)
k2 < length (map ?rr ?fp ! j2)
?cc ! i = map ?rr ?fp ! j1 ! k1
?cc ! j = map ?rr ?fp ! j2 ! k2 by blast
hence **: k1 < length (?rr (?fp ! j1))
k2 < length (?rr (?fp ! j2))
?cc ! i = ?rr (?fp ! j1) ! k1
?cc ! j = ?rr (?fp ! j2) ! k2
by auto
from * have mem: ?fp ! j1 \in set ?fp ?fp ! j2 \in set ?fp by auto
show ?cc ! i ≠ ?cc ! j
proof (cases j1 = j2)
case True
with * have k1 ≠ k2 by auto
with all(2)[OF mem(2)] **(1-2) show ?thesis unfolding **(3-4)
unfolding True
distinct-conv-nth by auto
next
case False
from degree p ≥ 3 have p: p ≠ 0 by auto
note fp = factors-of-int-poly(2-3)[OF refl this]
show ?thesis unfolding **(3-4)
proof

  define x where x = ?rr (?fp ! j2) ! k2
  assume id: ?rr (?fp ! j1) ! k1 = ?rr (?fp ! j2) ! k2
  from ** have x1: x ∈ set (?rr (?fp ! j1)) unfolding x-def id[symmetric]
    by auto
  from ** have x2: x ∈ set (?rr (?fp ! j2)) unfolding x-def by auto
  from all(1)[OF mem(1)] x1 have x1: ipoly (?fp ! j1) x = 0 by auto
  from all(1)[OF mem(2)] x2 have x2: ipoly (?fp ! j2) x = 0 by auto
  from False factors-of-int-poly(4)[OF refl, of p] have neq: ?fp ! j1 ≠ ?fp
    using * unfolding distinct-conv-nth by auto
  have poly (complex-of-int-poly p) x = 0 by (meson fip(1) mem(2) x2)
  show False by blast
  qed
  qed
  qed
  qed
  thus p ≠ 0 ⇒ ?l = ?r distinct (complex-roots-of-int-poly p) by auto
  qed

lemma complex-roots-of-rat-poly:
  p ≠ 0 ⇒ set (complex-roots-of-rat-poly p) = {x. rpoly p x = 0} (is - ⇒ ?l = ?r)
  distinct (complex-roots-of-rat-poly p)
proof
  obtain c q where cq: rat-to-int-poly p = (c,q) by force
  from rat-to-int-poly[OF this]
  have pq: p = smult (inverse (of-int c)) (of-int-poly q)
    and c: c ≠ 0 by auto
  show distinct (complex-roots-of-rat-poly p) unfolding complex-roots-of-rat-poly-def
    using complex-roots-of-int-poly(2) .
  assume: p: p ≠ 0
  with pq c have q: q ≠ 0 by auto
  have id: {x. rpoly p x = (0 :: complex)} = {x. ipoly q x = 0}
    unfolding pq by (simp add: c of-rat-of-int-poly hom-distribs)
  show ?l = ?r unfolding complex-roots-of-rat-poly-def cq snd-conv id
    complex-roots-of-int-poly(1)[OF q] ..
  qed

definition roots-of-complex-main :: complex poly ⇒ complex list where
  roots-of-complex-main p ≡ let n = degree p in
    if n = 0 then [] else if n = 1 then [roots1 p] else if n = 2 then croots2 p
    else (complex-roots-of-rat-poly (map-poly to-rat p))

definition roots-of-complex-poly :: complex poly ⇒ complex list option where
  roots-of-complex-poly p ≡ let (c,pis) = yun-factorization qed p in
if \( c \neq 0 \land (\forall (p,i) \in \text{set pis}. \text{degree} p \leq 2 \lor (\forall x \in \text{set (coeffs p)}. x \in \mathbb{Q})) \)
then
Some (concat (map (roots-of-complex-main o fst) pis)) else None

lemma roots-of-complex-main: assumes \( p \neq 0 \) and \( \text{deg}: \text{degree} p \leq 2 \lor \text{set (coeffs p)} \subseteq \mathbb{Q} \)
shows \( \text{set (roots-of-complex-main p)} = \{x. \text{poly} p x = 0\} \) (is set \( ?l = ?r \)
and distinct (roots-of-complex-main p)
proof –

note \( d = \text{roots-of-complex-main-def Let-def} \)

have \( \text{set ?l} = \text{?r} \land \text{distinct (roots-of-complex-main p)} \)

proof (cases degree \( p = 0 \))

case True
hence \( ?l = \] unfolding \( d \) by auto
with roots0[OF \( p \) True] show \( ?\text{thesis} \) by auto

next

case False note \( 0 = \text{this} \)
show \( ?\text{thesis} \)

proof (cases degree \( p = 1 \))

case True
hence \( ?l = \text{roots1 p} \) unfolding \( d \) by auto
with roots1[OF True] show \( ?\text{thesis} \) by auto

next

case False note \( 1 = \text{this} \)
show \( ?\text{thesis} \)

proof (cases degree \( p = 2 \))

case True
hence \( ?l = \text{croots2 p} \) unfolding \( d \) by auto
with croots2[OF True] show \( ?\text{thesis} \) by (auto simp: croots2-def Let-def)

next

case False note \( 2 = \text{this} \)

let \( ?q = \text{map-poly to-rat} \)

from \( 0 1 2 \) have \( l: ?l = \text{complex-roots-of-rat-poly} \?q \) unfolding \( d \) by auto

from \( \text{deg 0 1 2} \) have \( \text{rat: set (coeffs p)} \subseteq \mathbb{Q} \) by auto

have \( p = \text{map-poly (of-rat o to-rat)} \)

by (rule sym, rule map-poly-idI, insert rat, auto)

also have \( \ldots = \text{complex-of-rat-poly} \?q \)

by (subst map-poly-map-poly, auto simp: to-rat)

finally have \( \text{id: \{x. \text{poly} p x = 0\} = \{x. \text{poly (complex-of-rat-poly} \?q) x = 0\} \)

and \( \?q \neq 0 \)

using \( p \) by auto

from \( \text{complex-roots-of-rat-poly[of} \?q, \text{folded} \text{id} \} \) \( q \)

show \( ?\text{thesis} \) by auto

qed

qed

thus \( \text{set ?l} = \text{?r} \) distinct \( ?l \) by auto

qed

238
12.2 Factorization of Complex Polynomials

```plaintext
definition factorize-complex-main :: complex poly ⇒ (complex × (complex × nat) list) option where
    factorize-complex-main p ≡ let (c,pis) = yun-factorization gcd p in
    if ((∀ (p,i) ∈ set pis. degree p ≤ 2 ∨ (∀ x ∈ set (coeffs p). x ∈ ℚ))) then
        Some (c, concat (map (λ (p,i). map (λ r. (r,i)) (roots-of-complex-main p)) pis))
    else None
```

```plaintext
lemma factorize-complex-poly: assumes rt: factorize-complex-main p = Some (c,xis)
shows p = smult c (Π (x,i)←xis. x^-Suc i)
proof
  obtain d pis where yun: yun-factorization gcd p = (d,pis) by force
```

239
from rt[unfolded factorize-complex-main-def yun split Let-def]
have pis: \( p \vdash (p, i) : \text{set} \ \text{pis} \implies \deg p \leq 2 \lor (\forall x : \text{coeffs} \ p. \ x \in \mathbb{Q}) \)
and xis: \( \text{xis} = \text{concat} (\text{map} (\lambda (p, i). \text{map} (\lambda r. (r, i)) \ (\text{roots-of-complex-main} \ \ p)) \ \text{pis}) \)
and \( d : d = e \)
by (auto split: if-splits)

note yun = \text{yun-factorization} \ \text{[OF yun[unfolded d]}}
note yun = \text{square-free-factorizationD} \ \text{[OF yun(1)] yun(2)[unfolded snd-conv]}

let \( ?\exp = \lambda \text{pis} : \prod (x, i) : \text{concat} (\text{map} (\lambda (p, i). \text{map} (\lambda r. (r, i)) \ (\text{roots-of-complex-main} \ \ p)) \ \text{pis}) \).
from yun(1) have \( p : \text{smult} c \ \prod (a, i) : \text{set} \ \text{pis} \ a \sim \text{Suc} \ i \).
also have \( \prod (a, i) : \text{set} \ a \sim \text{Suc} \ i = \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i \)
by (rule prod.distinct-set-conv-list[OF yun(5)])
also have \( \ldots = ?\exp \ \text{pis} \ \text{using} \ \text{pis} \ yun(2,6) \)

\text{proof (induct pis)}

\text{case (Cons \ \text{pi} \ pis)}

obtain \( p \ i \ \text{where} \ \text{pi} : (p, i) \ \text{by force} \)

let \( \text{rts} = \text{roots-of-complex-main} \ p \)

\text{note Cons = Cons[unfolded pi]}\)

have IH: \( \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i = (?\exp \ \text{pis}) \)
by (rule Cons(1)[OF Cons(2-4)], auto)
from Cons(2-4)[of \( p \ i \)] have deg: \( \deg p \leq 2 \lor (\forall x : \text{coeffs} \ p. \ x \in \mathbb{Q}) \)
and \( p : \text{square-free} \ p \ \deg p \neq 0 \ p \neq 0 \ \text{monic} \ p \ \text{by auto} \)
have \( \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i = p \sim \text{Suc} \ i \ast \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i \)

unfolding \( \text{pi} \) \ \text{by simp} \)
also have \( \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i = (?\exp \ \text{pis} \ \text{by} \ \text{rule IH}) \)
finally have id: \( \prod (a, i) : \text{concat} \ a \sim \text{Suc} \ i = p \sim \text{Suc} \ i \ast ?\exp \ \text{pis} \ \text{by simp} \)

have \( ?\exp (p, i) = (?\exp \ [p, i]) \ast ?\exp \ \text{pis} \ \text{unfolding} \ \text{pi} \ \text{by simp} \)
also have \( ?\exp (p, i) = \prod (x, i) : \text{concat} \ (\text{map} (\lambda (r, i) \ \text{rts}) : \text{Suc} \ i) \)
by simp \)
also have \( \ldots = \prod (x) : \text{Suc} \ i \)

unfolding \( \text{prod-list-power} \ \text{by} \ \text{rule arg-cong[of - prod-list], auto} \)
also have \( \prod (x) : \text{Suc} \ i \)

\text{proof –}

from fundamental-theorem-algebra-factorized[of p, unfolded \ text{monic} \ p]

obtain as where as: \( p = \prod (a : \text{as} : \text{Suc} \ i) \ \text{by auto} \)
also have \( \ldots = \prod (a : \text{as} : \text{Suc} \ i) \)

\text{proof (rule sym, rule prod.distinct-set-conv-list, rule \text{contr})}

\ \text{assume \sim \ distinct as}

from not-distinct-decomp[OF this] obtain as1 as2 as3 a where
a: as1 = as1 @ [a] @ as2 @ [a] @ as3 by blast
\text{define q where q = \( \prod (a : \text{as} : \text{Suc} \ i) : \text{Suc} \ i) \)
have \( p = \prod (a : \text{as} : \text{Suc} \ i) \ \text{by fact} \)
also have \( \ldots = \prod (a : \text{as} = (a @ [a]) : \text{Suc} \ i) \ast q \)

unfolding \( q\text{-def} \ a \text{-map-append \ prod-list.append} \ \text{by (simp only: ac-simps)\)}
also have \( \ldots = [:\sim :-a, I :] \ast [-a, I :] \ast q \ \text{by simp} \)
finally have \( p = ([:\sim :-a, I :] \ast [-a, I :] \ast q \ \text{by simp} \)
hence \([-a,1] \ast [-a,1]:\) dvd \(p\) unfolding dvd-def ..
with (square-free \(p\):
 unfolded square-free-def, THEN conjunct2, rule-format, of \([-a,1]:\])
  show \(\text{False}\) by auto
qed
also have set as = \(\{x.\ \text{poly}\ p\ x = 0\}\) unfolding as poly-prod-list
  by (simp add: a-def, induct as, auto)
also have .. = set ?rts
  by (rule roots-of-complex-main(1)[symmetric], insert p deg, auto)
also have \(\prod a\in\text{set}\ ?rts.\ [:- a, 1:]\) = \(\prod a\leftarrow\text{rts}.\ [:- a, 1:]\)
  by (rule prod.distinct-set-conv-list[OF roots-of-complex-main(2)], insert deg p, auto)
finally show ?thesis by simp
qed
also have ?exp \(\pi\#\pis\) = \(p^\text{Suc}\ i \ast ?exp\ \pis\) by simp
finally show ?thesis unfolding \(\pi\#\pis\) by simp
qed

lemma distinct-factorize-complex-main:
  assumes factorize-complex-main \(p\) = Some \(\text{fctrs}\)
  shows distinct (map fst (snd \(\text{fctrs}\)))
proof -
  from assms have solvable: 
    \(\forall x\in\text{set}\ \text{snd}\ (\text{yun-factorization gcd } p).\ \text{degree}\ (\text{fst} x) \leq 2\ \vee\)
    \((\forall x\in\text{set}\ \text{coeffs}\ (\text{fst} x).\ x \in \mathbb{Q})\)
    by (auto simp add: factorize-complex-main-def case-prod-unfold
         Let-def map-concat o-def split: if-splits)
  have sqf: square-free-factorization \(p\)
    \((\text{fst} (\text{yun-factorization gcd } p),\ \text{snd} (\text{yun-factorization gcd } p))\)
    by (rule yun-factorization simp)
  have map \(\text{fst}\ (\text{snd} \text{fctrs}) =\)
    \(\text{concat}\ (\lambda x. (\text{roots-of-complex-main}\ \text{fst} x))\ (\text{snd} \text{(yun-factorization gcd } p))\)
    using assms by (auto simp add: factorize-complex-main-def case-prod-unfold
         Let-def map-concat o-def split: if-splits)
  also have distinct . . .
proof (rule distinct-concat, goal-cases)
  case 1
  show ?case
proof (subst distinct-map, safe)
  from square-free-factorizationD(5)[OF sqf]
  show distinct (snd (yun-factorization gcd p)).
  show inj-on (\(\lambda x. (\text{roots-of-complex-main}\ \text{fst} x))\) (set (snd (yun-factorization gcd p)))
proof (rule inj-onI, clarify, goal-cases)
case \((1 \ a1 \ b1 \ a2 \ b2)\)  
\{
  \text{assume neq: } (a1, b1) \not= (a2, b2) 
  \text{from } 1(1,2)[\text{THEN square-free-factorizationD}(2)[OF sqf]] 
  \text{have degree } a1 \not= 0 \text{ degree } a2 \not= 0 \text{ by blast}+ 
  \text{hence [simp]: } a1 \not= 0 \text{ a2 }\not= 0 \text{ by auto} 
  \text{from square-free-factorizationD}(3)[OF sqf 1(1,2) neq] 
  \text{have coprime } a1 \text{ a2 by simp} 
  \text{from solvable 1(1) have } \{z. \text{ poly } a1 \ z = 0\} = \text{ set (roots-of-complex-main a1)} 
\}

by (intro roots-of-complex-main(1) [symmetric]) auto
also have set (roots-of-complex-main a1) = set (roots-of-complex-main a2) 
using 1(3) by (subt (1 2) set-remdups [symmetric]) (simp only: fst-conv)
also from solvable 1(2) have \(\ldots = \{z. \text{ poly } a2 \ z = 0\}\) .
with coprime-imp-no-common-roots (coprime a1 a2): 
  have \(\{z. \text{ poly } a1 \ z = 0\} = \{\} \text{ by auto} \) 
with fundamental-theorem-of-algebra constant-degree 
  have degree a1 = 0 by auto
with (degree a1 \not= 0) have False by contradiction

thus \(?case\text{ by blast} 
qed

qed

next

\text{case } (2 \ ys) 
then obtain \(f \ b \) where \(fb: (f, b) \in \text{ set (snd (yun-factorization gcd p))} \) 
and \(ys: \text{ys }= \text{roots-of-complex-main f by auto} \) 
from square-free-factorizationD(2)[OF sqf fb] have \(0: f \not= 0 \text{ by auto} \)
from solvable[rule-format, OF fb] have \(f: \text{degree } f \leq 2 \lor (\text{set (coeffs } f) \subseteq \mathbb{Q}) \) 
by auto 

show \(?case unfolding \ ys 
  by (rule roots-of-complex-main[OF 0 f])

next

\text{case } (3 \ ys \ zs) 
then obtain \(a1 \ b1 \ a2 \ b2 \) where \(ab: \) 
  \((a1, b1) \in \text{ set (snd (yun-factorization gcd p))} \) 
  \((a2, b2) \in \text{ set (snd (yun-factorization gcd p))} \) 
  \(ys = \text{roots-of-complex-main a1 zs }= \text{roots-of-complex-main a2} \) 
  by auto

with \(\exists\) have \(\text{neg: } (a1,b1) \not= (a2,b2) \text{ by auto} \)
from \(ab(1,2)[\text{THEN square-free-factorizationD}(2)[OF sqf]] \)
  have [simp]: \(a1 \not= 0 \text{ a2 }\not= 0 \text{ by auto} \)
from square-free-factorizationD(3)[OF sqf \(ab(1,2)\) neq] have \(\text{coprime } a1 \text{ a2 by simp} \)

  have set \(ys = \{z. \text{ poly } a1 \ z = 0\} \) set \(zs = \{z. \text{ poly } a2 \ z = 0\} \) 
  by (insert solvable ab(1,2), subst ab,
rule roots-of-complex-main; (auto) []+ 
with coprime-imp-no-common-roots ⟨coprime a1 a2⟩ show ?case by auto 
qed

finally show ?thesis .
qed

lemma factorize-complex-poly: assumes fp: factorize-complex-poly p = Some (c,qis) 
shows 
p = smult c (Π (q, i)←qis. q ^ i) 
(q,i) ∈ set qis => irreducible q ∧ i ≠ 0 ∧ monic q ∧ degree q = 1 
proof –
from fp[unfolded factorize-complex-poly-def] 
obtain pis where fp: factorize-complex-main p = Some (c,pis) 
and qis: qis = map (λ (r, i). ([− r, 1], Suc i)) pis 
by auto 
from factorize-complex-main[OF fp] have p: p = smult c (Π (x, i)←pis. [− x, 1]) ^ Suc i .
show p = smult c (Π (q, i)←qis. q ^ i) unfolding p qis 
by (rule arg-cong[of - - λ p. smult c (prod-list p)], auto) 
show (q,i) ∈ set qis => irreducible q ∧ i ≠ 0 ∧ monic q ∧ degree q = 1 
using linear-irreducible-field[of q] unfolding qis by auto 
qed
end

13 Real Factorization

This theory contains an algorithm to completely factorize real polynomials 
with rational coefficients. It internally does a complex polynomial factor-
ization, and then combines all non-real roots with their conjugates.

theory Real-Factorization
imports 
  Complex-Algebraic-Numbers
begin

fun delete-cnj :: complex ⇒ nat ⇒ (complex × nat) list ⇒ (complex × nat) list 
where 
delete-cnj x i ((y,j) ≠ yjs) = (if x = y then if Suc j = i then yjs else if Suc j > i 
then
  ((y,j − i) ≠ yjs) else delete-cnj x (i − Suc j) yjs else (y,j) ≠ delete-cnj x i yjs)
| delete-cnj - - [] = []

lemma delete-cnj-length[termination-simp]: length (delete-cnj x i yjs) ≤ length yjs 
by (induct x i yjs rule: delete-cnj.induct, auto)

fun complex-roots-to-real-factorization :: (complex × nat) list ⇒ (real poly × 
  nat)list where
complex-roots-to-real-factorization [] = []
| complex-roots-to-real-factorization ((x,i) ≠ xs) = (if x ∈ R then
  ([:−(Re x),I],Suc i) ≠ complex-roots-to-real-factorization xs else
  let xx = cnj x; ys = delete-cnj xx (Suc i) xs; p = map-poly Re ([:−x,I] *
  [:−xx,I])
  in (p,Suc i) ≠ complex-roots-to-real-factorization ys)

definition factorize-real-poly :: real poly ⇒ (real × (real poly × nat) list) option
where
 factorize-real-poly p ≡ map-option
  (λ (c,ris). (Re c, complex-roots-to-real-factorization ris))
  (factorize-complex-main (map-poly of-real p))

lemma monic-imp-nonzero: monic x −→ x ≠ 0 for x :: 'a :: semiring-1 poly by auto

lemma delete-cnj: assumes
  order x ([Π (x,i)←xis. [:− x, i] ∼ Suc i) ≥ si (si ≠ 0
  shows ([Π (x,i)←xis. [:− x, i] ∼ Suc i) =
  [:− x, I] ∼ si * ([Π (x,i)←delete-cnj x si xis. [:− x, i] ∼ Suc i)
using assms
proof (induct x si xis rule: delete-cnj.induct)
  case (2 x si)
  hence order x 1 ≥ 1 by auto
  hence [:−x,1] dvd 1 unfolding order-divides by simp
  from power-le-dvd[OF this, of 1] (si ≠ 0) have [:− x, I] dvd 1 by simp
  from divides-degree[OF this]
  show ?case by auto
next
  case (1 x i y j yjs)
  note IH = I(1−2)
  let ?yj = [:−y,i] ∼ Suc j
  let ?yjs = ([Π (x,i)←yjs. [:− x, i] ∼ Suc i)
  let ?x = [:− x, i] :]
  let ?xi = ?x ∼ i
  have monic ([Π (x,i)←(y, j) # yjs. [:− x, i] ∼ Suc i)
    by (rule monic-prod-list-pow) then have monic (??yj * ?yjs) by simp
  from monic-imp-nonzero[OF this] have yy0: ?yj * ?yjs ≠ 0 by auto
  have id: (Π (x,i)←(y, j) # yjs. [:− x, i] ∼ Suc i) = ?yj * ?yjs by simp
  from I(3−) have ord: i ≤ order x (?yj * ?yjs) and i: i ≠ 0 unfolding id by auto
  from ord[unfolded order-mult[OF yy0]] have ord: i ≤ order x ?yj + order x ?yjs
    .
  from this[unfolded order-linear-power']
  have ord: i ≤ (if y = x then Suc j else 0) + order x ?yjs by simp
  show ?case
proof (cases x = y)
  case False
  case False
from ord False have \(i \leq \text{order } x\) ?yjs by simp

note IH = IH(2)[OF False this \(\tilde{i}\)]
from False have del: delete-cnj \(x\) \(i\) ((\(y, j\)) \# yjs) = (y, j) \# delete-cnj \(x\) \(i\) yjs by simp

show ?thesis unfolding del id IH
by (simp add: ac-simps)

next

case True note xy = this

note IH = IH(1)[OF True]

show ?thesis
proof (cases Suc \(j\) \(\geq i\))

case False
from ord have ord: \(i - Suc j \leq \text{order } x\) ?yjs unfolding xy by simp

have \(?xi = ?x \sim (Suc j + (i - Suc j))\) using False by simp
also have \(\ldots = ?x \sim Suc j \ast ?x \sim (i - Suc j)\)
unfolding power-add by simp
finally have \(x_1: ?xi = ?x \sim Suc j \ast ?x \sim (i - Suc j)\).
from False have Suc \(j\) \(\neq i - i < Suc j\) \(i - Suc j \neq 0\) by auto

note IH = IH[OF this(1,2) ord this(3)]
from xy False have del: delete-cnj \(x\) \(i\) ((\(y, j\)) \# yjs) = delete-cnj \(x\) \((i - Suc j)\) yjs by auto

show ?thesis unfolding del id unfolding IH xi unfolding xy by simp

next

case True

hence Suc \(j = i \lor i < Suc j\) by auto

thus ?thesis

proof

assume \(i: Suc j = i\)

from xy \(i\) have del: delete-cnj \(x\) \(i\) ((\(y, j\)) \# yjs) = yjs by simp

show ?thesis unfolding id del unfolding xy \(i\) by simp

next

assume \(ij: i < Suc j\)

with xy \(i\) have del: delete-cnj \(x\) \(i\) ((\(y, j\)) \# yjs) = (y, j - i) \# yjs by simp

from ij have idd: Suc \(j = i + Suc (j - i)\) by simp

show ?thesis unfolding id del unfolding xy \(idd\) power-add by simp

qed

c qed

lemma factorize-real-poly: assumes \(fp\): factorize-real-poly \(p\) = Some \((c, \text{qis})\)
shows \(p = \text{smult } c \prod (q, i) \sim \text{qis}. q \sim i\)
\((q,j) \in \text{set } \text{qis} \implies \text{irreducible } q \land j \neq 0 \land \text{monic } q \land \text{degree } q \in \{1,2\}\)

proof –

interpret map-poly-inj-\text{idom-hom of-\text{real}}..

have \((p = \text{smult } c \prod (q, i) \sim \text{qis}. q \sim i) \land ((q,j) \in \text{set } \text{qis} \implies \text{irreducible } q \land j \neq 0 \land \text{monic } q \land \text{degree } q \in \{1,2\})

\)
proof (cases p = 0)
  case True
  have factorize-real-poly p = Some (0,[]) unfolding True
  by (simp add: factorize-real-poly-def factorize-complex-main-def yun-factorization-def)
  with fp have id: c = 0 qis = [] by auto
thus ?thesis unfolding True by simp
next
  case False note p0 = this
  let ?c = complex-of-real
  let ?r?p = map-poly Re
  let ?c?p = map-poly ?c
  let ?p = ?cp p
  from fp[unfolded factorize-real-poly-def]
  obtain d xis where fp: factorize-complex-main ?p = Some (d,xis)
  and c: c = Re d and qis: qis = complex-roots-to-real-factorization xis by auto
  from factorize-complex-main[OF fp] have p: ?p = smult d (π (x, i)←xis. :− x, 1): Suc i
  (is - = smult d ?q) .
  from arg-cong[OF this, of λ p. coeff p (degree p)]
  have coeff ?p (degree ?p) = coeff (smult d ?q) (degree (smult d ?q)) .
  also have coeff ?p (degree ?p) = ?c (coeff p (degree p)) by simp
  also have coeff (smult d ?q) (degree (smult d ?q)) = d * coeff ?q (degree ?q)
  by simp
  also have monic ?q by (rule monic-prod-list-pow)
  finally have d: d = ?c (coeff p (degree p)) by auto
  from arg-cong[OF this, of Re, folded c] have c: c = coeff p (degree p) by auto
  have set (coeffs ?p) ⊆ ℜ by auto
  with p have q: set (coeffs (smult d ?q)) ⊆ ℜ by auto
  from d p0 have d0: d ≠ 0 by auto
  have smult d ?q = [d:] * ?q by auto
  from real-poly-factor[OFfq[unfolded this]] d0 d
  have q: set (coeffs ?q) ⊆ ℜ by auto
  have p = ?r?p ?p
  by (rule sym, subst map-poly-map-poly, force, rule map-poly-idI, auto)
  also have ... = ?r?p (smult d ?q) unfolding p ..
  also have ?q = ?cp (?r?p ?q)
  by (rule sym, rule map-poly-map-of-real-Re, insert q, auto)
  also have d = ?c e unfolding d e ..
  also have smult (?c c) (?cp (?r?p ?q)) = ?cp (smult c (?r?p ?q)) by (simp add: hom-distribs)
  also have ?r?p ... = smult c (?r?p ?q)
  by (subst map-poly-map-poly, force, rule map-poly-idI, auto)
  finally have p: p = smult c (?r?p ?q) .
  let ?fact = complex-roots-to-real-factorization
  have ?r?q = (π (q, i)←qis. q i) ∧ ((q, i) ∈ set qis → irreducible q ∧ i ≠ 0 ∧ monic q ∧ degree q ∈ {1, 2})
  using q unfolding qis
proof (induct xis rule: complex-roots-to-real-factorization.induct)
  case 1
show \(\text{case by simp}\)

next

case \((2 \times i \times i)\)

note \(IH = 2(1-2)\)

note \(prems = 2(3)\)

let \(?xi = [-x, 1] \sim Suc i\)

let \(?xis = \prod (x, i) \times i. [-x, 1] \sim Suc i\)

have \(id: (\prod (x, i) \times i. [-x, 1] \sim Suc i) = ?xi \times ?xis\)

by simp

show \(\text{case}\)

proof

\((\text{cases } x \in \mathbb{R})\)

case True

have \(xi: \text{set (coeffs } ?xi) \subseteq \mathbb{R}\)

by (rule real-poly-power, insert True, auto)

have \(xis: \text{set (coeffs } ?xis) \subseteq \mathbb{R}\)

by (rule real-poly-factor[\(OF\) \(prems\)[unfolded \(id\)], rule linear-power-nonzero)

note \(IH = IH(1)[OF \ True \ xis]\)

have \(?rp (\ ?xi \times ?xis) = ?rp (\ ?xi \times ?xis)\)

by (rule map-poly-Re-mul[\(OF \ xis\)])

also have \(?rp (\ ?xi) = (\ ?rp (-x, 1)) \sim Suc i\)

by (rule map-poly-Re-power, insert True, auto)

also have \(?rp (-x, 1) = [- (Re x), 1] \ by \ auto\)

also have \(?rp (\ ?xis) = (\prod (a, b) \times \ ?fact \ xis. a \times b)\)

using \(IH\) by auto

also have \([-x \times I, 1] \sim Suc i \times (\prod (a, b) \times \ ?fact \ xis. a \times b) = (\prod (a, b) \times \ ?fact \ ((x, i) \times iis). a \times b)\) using \(True\) by simp

finally have \(idd: \ ?rp (\ ?xi \times ?xis) = (\prod (a, b) \times \ ?fact \ ((x, i) \times iis). a \times b)\).

show \(\text{thesis unfolding \ id \ idd}\)

proof

\((\text{intro conjI}, \ force, \ intro \ impI)\)

assume \((q, j) \in \text{set (} \ ?fact \ ((x, i) \times iis)\))

hence \((q,j) \in \text{set (} \ ?fact \ xis) \lor (q = \ [-Re x, 1] \land j = Suc i)\)

using \(True\) by auto

thus irreducible \(q \land j \neq 0 \land \text{monic} \ q \land \text{degree} \ q \in \{1, 2\}\)

proof

assume \((q,j) \in \text{set (} \ ?fact \ xis)\)

with \(IH\) show \(\text{thesis by auto}\)

next

assume \(q = \ [-Re x, 1] \land j = Suc i\)

with \(linear-irreducible-field[\ of \ [-Re x, 1]]\) show \(\text{thesis by auto}\)

qed

qed

next

case False

define \(xi\ where\ \ xi = [-Re x, Re x + Im x] \times Im x, - (2 \times Re x), 1]\)

obtain \(xx\ where\ xx = cnj x\ by\ auto\)

have \(xi: xi = \ ?rp (\ [-x, I] \times [-xx, I])\) unfolding \(xx\ \ xi-def\) by \(auto\)

have \(cpxi: cpxi = [-x, I] \times [-xx, I]\) unfolding \(xi-def\)

by \(\text{(cases } x, \ \text{auto simp: } xx\ \text{legacy-Complex-simps})\)

obtain \(yis\ where\ yis: yis = delete-cnj xx (Suc i)\ \times i\) by \(auto\)
from False have fact: (?fact ((x, i) # xis) = (xi, Suc i) # ?fact yis)
  unfolding xi-def xx yis by simp
note IH = IH(2)[OF False xx yis xi]
have irreducible xi
  apply (fold irreducible-connect-field)
proof (rule irreducible, I)
  show degree xixi > 0 unfolding xi by auto
    fix q p :: real poly
    assume degree q > 0 degree q < degree xi and qp: xi = q * p
    hence dq: degree q = 1 unfolding xi by auto
    have dxixi: degree xi = 2 xi ≠ 0 unfolding xi by auto
    with qp have q ≠ 0 p ≠ 0 by auto
    hence degree xi = degree q + degree p unfolding qp
      by (rule degree-mult-eq)
    with dq have dp: degree p = 1 unfolding dxixi by simp
  
    fix c :: complex
    assume rt: poly (?cp xi) c = 0
    hence poly (?cp q * ?cp p) c = 0 by (simp add: qp hom-distribs)
    hence (poly (?cp q) c = 0 ∨ poly (?cp p) c = 0) by auto
    hence c = roots1 (?cp q) ∨ c = roots1 (?cp p)
      unfolding roots1-def by auto
    hence c ∈ R unfolding roots1-def by auto
    hence c ≠ x using False by auto
  
  hence poly (?cp xi) x ≠ 0 by auto
  thus False unfolding cpixi by simp
  qed

  hence xixi': irreducible xi monic xi degree xi = 2
  unfolding xi by auto
  let ?xixi = [-- xx, I:] ^ Suc i
  let ?yis = ([] (x, i)←yis. [--, x, 1:] ^ Suc i)
  let ?yi = (?cp xi) ^ Suc i
  have yi: set (coeffs ?yi) ⊆ R
    by (rule real-poly-power, auto simp: xixi)
  have mon: monic ([] (x, i)←(x, i) # xis. [--, x, 1:] ^ Suc i)
    by (rule monic-prod-list-pow)
  from monic-imp-nonzero[OF this] have xixis: ?xi * ?xis ≠ 0 unfolding id
    by auto
    from False have xex: xx ≠ x unfolding xx by (cases x, auto simp: xixis)
  
from prems[unfolded id] have prems: set (coeffs (?xi * ?xis)) ⊆ R .
from id have [-- xx, 1:] ^ Suc i ded ?xi * ?xis by auto
from xxis this[unfolded order-divides]
  have order x (?xi * ?xis) ≥ Suc i by auto
  with complex-conjugate-order[OF prems xxis, of x, folded xx]
  have order xx (?xi * ?xis) ≥ Suc i by auto
  hence order xx ?xi + order xx ?xis ≥ Suc i unfolding order-mult[OF xxis]

. 248
also have order \( xx \) \(?xx = 0 \) unfolding order-linear-power' using \( xxx \) by simp

finally have order \( xx \) \(?xis \geq \) Suc \( i \) by simp

hence \( yis : \) \(?xis = \) ?xxi \(* \) ?yis unfolding \( yis \)

by (rule delete-cnj, simp)

hence \( \( ?xi \) \(* \) \(?xis = (\( ?xi \) \(* \) \(?xxi \) \) \(* \) \(?yis by (simp only: ac-simps) \)

also have \( \( \) \(?xi \) \(* \) \(?xis = (\( \) \(?xi \) \(* \) \(?xxi \) \) \(* \) \(?yis by (metis power-mult-distrib) \)

also have \( \( \) \(?xi \) \(* \) \(?xxi = \) \( \) \([- x, 1]; \) \(* \) \([- xx, 1]; \) \)^Suc \( i \) \)

by (rule delete-cnj, simp)

hence \( \) \(?xi \) \(* \) \(?xis = (\( \) \(?xi \) \(* \) \(?xxi \) \) \(* \) \(?yis \)

unfolding \( \) \(?xis by (fold hom-distrib, rule map-poly-Re-of-real) \)

also have \( \) \(?xi \) \(^\# \) \(?xis \)

unfolding \( \) \(?xis by simp \)

finally have \( \) \(?xi \) \(* \) \(?xis = (\( \) \(?xi \) \(* \) \(?yis \) \)

have \( \) \(?xis \)

by (rule real-poly-factor[OF \( R \) \( yis \) ], auto, auto simp: \( xi-def \))

note \( IH = IH'[OF \) \(?xis \) ]

have \( \) \(?ri \) \(* \) \(?xis = \) \(?ri \) \(* \) \(?yi \) \(* \) \(?ri \) \(* \) \(?yis unfolding \( \) \(?xis \)

by (rule map-poly-Re-mult[OF \( yi \) \( yis \) ])

also have \( \) \(?ri \) \(* \) \(?yi = \) \( \) \( xi \) \(^\# \) \(?xis \) i by (fold hom-distrib, rule map-poly-Re-of-real)

also have \( \) \(?ri \) \(* \) \(?yis = (\( \prod \) a, b \) \(* \) \(?fact \) \( yis \) a \(* b \) \)

using \( IH \) by auto

also have \( \) \(?xi = \) \( \) \( \prod \) a, b \(* \) \(?fact \) \( yis \) a \(* b \) \)

\( \) \( \prod \) a, b \(* \) \(?fact \) \( \) \( (x,i) \# \) \(?xis \) a \(* b \) unfolding \( \) \(?fact by simp \)

finally have \( \) \(?ri \) \(* \) \(?xis = (\( \) \( \prod \) a, b \(* \) \(?fact \) \( \) \( (x,i) \# \) \(?xis \) a \(* b \) \) \)

show \( \) \(?thesis unfolding \( \) \(?thesis by auto \)

qed

thus \( \) \(?thesis unfolding p by simp \)

qed

thus \( p = smult \) \( c (\( \prod \) q, i \) \(* \) \(?qis \) q \(* i \) \)

\( (q, i) \) \( \in \) set \( qis \) \) \( \Rightarrow \) irreducible \( q \) \(* i \) \( \neq \) \( \emptyset \) \(* \) \(?monic \) \( q \) \(* \) degree \( q \) \( \in \{ 1, 2 \} \) by blast+

qed

end

14 Show for Real Algebraic Numbers – Interface

We just demand that there is some function from real algebraic numbers to string and register this as show-function and use it to implement show-real.

Implementations for real algebraic numbers are available in one of the theories Show-Real-Precise and Show-Real-Approx.

theory Show-Real-Alg
imports
  Real-Algebraic-Numbers
  Show.Show-Real
begin

consts show-real-alg :: real-alg \( \Rightarrow \) string

definition shows-p-real-alg :: real-alg shows-p where
  shows-p-real-alg \( p \) \( x \) \( y = (\) \( show-real-alg x \) \( \& \) \( y \) \)
lemma show-law-real-alg [show-law-intros]:
  show-law showsp-real-alg r
by (rule show-lawI) (simp add: showsp-real-alg-def show-law-simps)

lemma showsp-real-alg-append [show-law-simps]:
  showsp-real-alg p r (x @ y) = showsp-real-alg p r x @ y
by (intro show-lawD show-law-intros)

local-setup ⟨
  Show-Generator.register-foreign-showsp @\{typ real-alg\} @\{term showsp-real-alg\}
@\{thm show-law-real-alg\}
⟩

derive show real-alg

  We now define show-real.

overloading show-real ≡ show-real
begin
  definition show-real ≡ show-real-alg o real-alg-of-real
end
end

15  Show for Real (Algebraic) Numbers – Approximate Representation

We implement the show-function for real (algebraic) numbers by calculating
the number precisely for three digits after the comma.

theory Show-Real-Approx
imports
  Show-Real-Alg
  Show.Show-Instances
begin

overloading show-real-alg ≡ show-real-alg
begin

definition show-real-alg[code]: show-real-alg x ≡ let
  x1000' = floor (1000 * x);
  (x1000,s) = (if x1000' < 0 then (−x1000', "−") else (x1000', "'"));
  (bef,aft) = divmod-int x1000 1000;
  a' = show aft;
  a = replicate (3−length a') (CHR "0") @ a'
in
  "−" @ s @ show bef @ "'" @ a

end

250
16 Show for Real (Algebraic) Numbers – Unique Representation

We implement the show-function for real (algebraic) numbers by printing them uniquely via their monic irreducible polynomial with a special cases for polynomials of degree at most 2.

theory Show-Real-Precise
imports Show-Real-Alg Show.Show-Instances
begin

datatype real-alg-show-info = Rat-Info rat | Sqrt-Info rat rat | Real-Alg-Info int poly nat

fun convert-info :: rat + int poly × nat ⇒ real-alg-show-info where
convert-info (Inl q) = Rat-Info q
| convert-info (Inr (f, n)) = (if degree f = 2 then (let a = coeff f 2; b = coeff f 1; c = coeff f 0;
  b2a = Rat Fract (−b) (2 * a);
  below = Rat Fract (bˆ2 - 4 * a * c) (4 * a * a)
  in Sqrt-Info b2a (if n = 1 then −below else below))
else Real-Alg-Info f n)

definition real-alg-show-info :: real-alg ⇒ real-alg-show-info where
real-alg-show-info x = convert-info (info-real-alg x)

We prove that the extracted information for showing an algebraic real number is correct.

lemma real-alg-show-info: real-alg-show-info x = Rat-Info r ⇒ real-of x = of-rat r
real-alg-show-info x = Sqrt-Info r sq ⇒ real-of x = of-rat r + sqrt (of-rat sq)
real-alg-show-info x = Real-Alg-Info p n ⇒ p represents (real-of x) ∧ n = card { y. y ≤ real-of x ∧ ipoly p y = 0}
(is ?l ⇒ ?r)
proof (atomize (full), goal-cases)
case 1
note d = real-alg-show-info-def
show ?case
proof (cases info-real-alg x)
case (Inl q)
  from info-real-alg(2) OF this this show ?thesis unfolding d by auto
next
case (Inr qm)

251
then obtain \( p n \) where id: info-real-alg \( x = \text{Inr} (p,n) \) by (cases qm, auto) from info-real-alg(\( l \))[OF id] have ap: \( p \) represents (real-of \( x \)) and n: \( n = \text{card} \{ y. y \leq \text{real-of} \ x \land \text{ipoly} \ p \ y = 0 \} \) and irr: irreducible \( p \) by auto

note id' = real-alg-show-info-def id convert-info.simps Fract-of-int-quotient

Let-def

have last: \( \exists l \Rightarrow \exists r \) unfolding id' using ap n by (auto split: if-splits)

{ assume *: real-alg-show-info \( x = \text{Sqrt-Info} \ r \ sq \)
  from this[unfolded id'] have deg: degree \( p = 2 \) by (auto split: if-splits)
  from degree2-coeffs[OF this] obtain a b c where \( p = [a,b,c] \) and a: a \neq 0 by auto
  hence coeffs: coeff \( p \ 0 = c \) coeff \( p \ 1 = b \) coeff \( p \ (\text{Suc} \ (\text{Suc} \ 0)) = a \) 2 = Suc (Suc 0) by auto

let \( ?a = \text{real-of-int} \ a \)
let \( ?b = \text{real-of-int} \ b \)
let \( ?c = \text{real-of-int} \ c \)
define A where \( A = \_ \ ?a \)
define B where \( B = \_ \ ?b \)
define C where \( C = \_ \ ?c \)
define R where \( R = \_ \ ?r \)
define \( ?sq = (B * B - 4 * A * C) / (4 * A * A) \)
define \( ?p = \text{real-of-int-poly} \ p \)
define \( ?\text{disc} = (B / (2 * A)) ^ \text{Suc} (\text{Suc} \ 0) - C / A \)
define \( D \) where \( D = ?\text{disc} \)
from arg-cong[OF \( p \), of map-poly real-of-int] have rp: \( ?p = [\_ \ C, B, A : ] \)
  using a by (auto simp: A-def B-def C-def)
from a have A: \( A \neq 0 \) unfolding A-def by auto
from * [unfolded id', deg, unfolded coeffs of-int-minus of-int-minus of-int-mult of-int-diff, simplified]
  have r: real-of-rat \( r = R \) and sq: sqrt (of-rat sq) = (if \( n = 1 \) then - sqrt \( ?sq \) else sqrt \( ?sq \))
    by (auto simp: A-def B-def C-def R-def real-sqrt-minus hom-distribs)

note sq
also have \( ?sq = D \) using A by (auto simp: field-simps D-def)
finally have sq: sqrt (of-rat sq) = (if \( n = 1 \) then - sqrt D else sqrt D) by simp

with rp have coeffs': coeff \( ?p \ 0 = C \) coeff \( ?p \ 1 = B \) coeff \( ?p \ (\text{Suc} \ (\text{Suc} \ 0)) = A \) 2 = Suc (Suc 0) by auto

from rp A have degree (real-of-int-poly \( p \)) = 2 by auto
note roots = rroots2[OF this, unfolded rroots2-def Let-def coeffs', folded D-def R-def]

from ap[unfolded represents-def] have root: ipoly \( p \) (real-of \( x \)) = 0 by auto
from root roots have D: \( (D < 0) = \text{False} \) by auto
note roots = roots[unfolded this if-False, folded R-def]

have real-of \( x \) = of-rat \( r \) + sqrt (of-rat sq)
proof (cases \( D = 0 \))

  case True
  show ?thesis using roots root unfolding sq r True by auto

next

  case False
  with \( D \) have \( D > 0 \) by auto
  from roots False have roots: \( \{ x. \ ipoly p x = 0 \} = \{ R + \sqrt{D}, R - \sqrt{D} \} \)

by auto

let ?Roots = \( \{ y. y \leq \realof x \land \ipoly p y = 0 \} \)

have \( x: \realof x \in ?Roots \) using root by auto

from root roots have choice: \( \realof x = R + \sqrt{D} \lor \realof x = R - \sqrt{D} \)

by auto

show ?thesis proof (cases n = 1)

  case True
  from card-1-singletonE[OF n[symmetric, unfolded this]] obtain y where id: ?Roots = \( \{ y \} \)

  from x small show ?thesis unfolding sq r id using True by auto

next

  case False
  from x obtain Y where Y: ?Roots = insert (real-of x) \( \{ \} \)

by auto

  with False[unfolded n] obtain z Z where Z: \( Y = \{ \} \) = insert z

by auto

  with roots choice \( D \) have \( \realof x = R + \sqrt{D} \) by force

thus ?thesis unfolding sq r id using False by auto

qed

qed

with last show ?thesis unfolding d by (auto simp: id Let-def)

qed

fun show-rai-info :: int ⇒ real-alg-show-info ⇒ string where

show-rai-info fl (Rat-Info r) = show r

| show-rai-info fl (Sqrt-Info r sq) = (let sqrt = "sqrt(\" @ show (abs sq) @ ")"

in if r = 0 then (if sq < 0 then "" else \[ \]) @ sqrt

  else ("\" @ show r @ (if sq < 0 then "" else "") @ sqrt) @ ")

| show-rai-info fl (Real-Alg-Info p n) = ""(root \" @ show n @ ") of "" @ show p @ ", in ("" @ show fl @ "," @ show (fl + 1) @ ")"

overloading show-real-alg ≡ show-real-alg

begin

definition show-real-alg[code]:

show-real-alg x ≡ show-rai-info (floor x) (real-alg-show-info x)
17 Algebraic Number Tests

We provide a sequence of examples which demonstrate what can be done with the implementation of algebraic numbers.

theory Algebraic-Number-Tests
imports
  Jordan-Normal-Form.Char-Poly
  Jordan-Normal-Form.Determinant-Impl
  Show.Show-Complex
  HOL-Library.Code-Target-Nat
  HOL-Library.Code-Target-Int
  Berlekamp-Zassenhaus.Factorize-Rat-Poly
  Real-Factorization
  Show-Real-Precise
begin

17.1 Stand-Alone Examples

abbreviation (input) show-lines x ≡ shows-lines x Nil

fun show-factorization :: 'a :: {semiring-1,show} × (('a poly × nat)list) ⇒ string
  where
    show-factorization (c,[]) = show c
  | show-factorization (c,[(p,i) # ps]) = show-factorization (c,ps) @ "" @ ("" @ show p @ "")" @ 
    (if i = 1 then [] else """" @ show i)

definition show-sf-factorization :: 'a :: {semiring-1,show} × (('a poly × nat)list) ⇒ string
  where
    show-sf-factorization x = show-factorization (map-prod id (map (map-prod id Suc)) x)

Determine the roots over the rational, real, and complex numbers.

definition testpoly = [:5/2, -7/2, 1/2, -5, 7, -1, 5/2, -7/2, 1/2:]
definition test = show-lines ( real-roots-of-rat-poly testpoly)

value [code] show-lines ( roots-of-rat-poly testpoly)
value [code] show-lines ( real-roots-of-rat-poly testpoly)
value [code] show-lines (complex-roots-of-rat-poly testpoly)

Factorize polynomials over the rational, real, and complex numbers.

value [code] show-sf-factorization (factorize-rat-poly testpoly)
value [code] show-factorization (the (factorize-real-poly testpoly))
value [code] show-factorization (the (factorize-complex-poly testpoly))

254
If the input is not a rational polynomial, factorization can fail.

\[
\text{value [code] factorize-real-poly [:sqrt 2,1,3,:]}
\]

fails

\[
\text{value [code] factorize-real-poly [:sqrt 2,1,3,:]}
\]

does not fail, reveals internal representation

\[
\text{value [code] show (factorize-real-poly [:sqrt 2,1,3,:])}
\]

does not fail, pretty printed

A sequence of calculations.

\[
\text{value [code] show (- sqrt 2 - sqrt 3)}
\]

\[
\text{lemma root 3 4 > sqrt (root 4 3) + [1/10 * root 3 7] by eval}
\]

\[
\text{lemma csqrt (4 + 3 * i) \notin \mathbb{R} by eval}
\]

\[
\text{value [code] show (csqrt (4 + 3 * i))}
\]

\[
\text{value [code] show (csqrt (1 + i))}
\]

17.2 Example Application: Compute Norms of Eigenvalues

For complexity analysis of some matrix \(A\) it is important to compute the spectral radius of a matrix, i.e., the maximal norm of all complex eigenvalues, since the spectral radius determines the growth rates of matrix-powers \(A^n\), cf. [4] for a formalized statement of this fact.

\[
\text{definition eigenvalues :: rat mat \Rightarrow complex list where}
\]

\[
\text{eigenvalues } A = \text{complex-roots-of-rat-poly (char-poly } A)
\]

definition testmat = mat-of-rows-list 3 [ 
    [1,-4,2],
    [1/5,7,9],
    [7,1,5 :: rat]
  ]

definition spectral-radius-test = show (Max (set [ norm ev. ev \leftarrow eigenvalues testmat]))

\[
\text{value [code] char-poly testmat}
\]

\[
\text{value [code] spectral-radius-test}
\]

end

18 Explicit Constants for External Code

theory Algebraic-Numbers-External-Code
  imports Algebraic-Number-Tests
begin
We define constants for most operations on real- and complex-algebraic numbers, so that they are easily accessible in target languages. In particular, we use target languages integers, pairs of integers, strings, and integer lists, resp., in order to represent the Isabelle types \( \text{int}/\text{nat} \), \( \text{rat} \), \( \text{string} \), and \( \text{int poly} \), resp.

**definition** decompose-rat = map-prod integer-of-int integer-of-int o quotient-of

### 18.1 Operations on Real Algebraic Numbers

**definition** zero-ra = (0 :: real-alg)

**definition** one-ra = (1 :: real-alg)

**definition** of-integer-ra = (of-int o int-of-integer :: integer ⇒ real-alg)

**definition** plus-ra = ((+) :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** minus-ra = ((−) :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** uminus-ra = (uminus :: real-alg ⇒ real-alg)

**definition** times-ra = ((∗) :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** divide-ra = ((/) :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** inverse-ra = (inverse :: real-alg ⇒ real-alg)

**definition** abs-ra = (abs :: real-alg ⇒ real-alg)

**definition** floor-ra = (integer-of-int o floor :: real-alg ⇒ integer)

**definition** ceiling-ra = (integer-of-int o ceiling :: real-alg ⇒ integer)

**definition** minimum-ra = (min :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** maximum-ra = (max :: real-alg ⇒ real-alg ⇒ real-alg)

**definition** equals-ra = ((=) :: real-alg ⇒ real-alg ⇒ bool)

**definition** less-ra = ((<) :: real-alg ⇒ real-alg ⇒ bool)

**definition** less-equal-ra = ((≤) :: real-alg ⇒ real-alg ⇒ bool)

**definition** compare-ra = (compare :: real-alg ⇒ real-alg ⇒ order)

**definition** roots-of-poly-ra = (roots-of-real-alg o poly-of-list o map int-of-integer :: integer list ⇒ real-alg list)

**definition** root-ra = (root-real-alg o nat-of-integer :: integer ⇒ real-alg ⇒ real-alg)

**definition** show-ra = ((String.implode o show) :: real-alg ⇒ String.literal)

**definition** is-rational-ra = (is-rat-real-alg :: real-alg ⇒ bool)

**definition** to-rational-ra = (decompose-rat o to-rat-real-alg :: real-alg ⇒ integer × integer)

**definition** sign-ra = (fst o to-rational-ra o sgnt :: real-alg ⇒ integer)

**definition** decompose-ra = (map-sum decompose-rat (map-prod (map integer-of-int o coeff) integer-of-nat) o info-real-alg ∘ real-alg ⇒ integer × integer + integer list × integer)

### 18.2 Operations on Complex Algebraic Numbers

**definition** zero-ca = (0 :: complex)

**definition** one-ca = (1 :: complex)

**definition** imag-unit-ca = (i :: complex)

**definition** of-integer-ca = (of-int o int-of-integer :: integer ⇒ complex)
definition of-rational-ca = ((\num (num, denom). of-rat (Rat.Fract (int-of-integer num)) (int-of-integer denom))) :: integer \times integer \Rightarrow complex

definition of-real-imag-ca = ((\real (real, imag). Complex (real-of real) (real-of imag))) :: real-alg \times real-alg \Rightarrow complex

definition plus-ca = ((+)) :: complex \Rightarrow complex \Rightarrow complex

definition minus-ca = ((−)) :: complex \Rightarrow complex \Rightarrow complex

definition uminus-ca = (uminus :: complex \Rightarrow complex)

definition times-ca = ((*)) :: complex \Rightarrow complex \Rightarrow complex

definition divide-ca = ((/)) :: complex \Rightarrow complex \Rightarrow complex

definition inverse-ca = (inverse :: complex \Rightarrow complex)

definition equals-ca = ((=)) :: complex \Rightarrow complex \Rightarrow bool

definition roots-of-poly-ca = (complex-roots-of-int-poly o poly-of-list o map int-of-integer) :: integer list \Rightarrow complex list

definition csqrt-ca = (csqrt :: complex \Rightarrow complex)

definition show-ca = ((String.implode o show)) :: complex \Rightarrow String.literal

definition real-of-ca = (real-alg-of-real o Re :: complex \Rightarrow real-alg)

definition imag-of-ca = (real-alg-of-real o Im :: complex \Rightarrow real-alg)

18.3 Export Constants in Haskell

export-code

order.Eq order.Lt order.Gt — for comparison
Inl Inr — make disjoint sums available for decomposition information

code

zero-ra
one-ra
of-integer-ra
of-rational-ra
plus-ra
minus-ra
uminus-ra
times-ra
divide-ra
inverse-ra
abs-ra
floor-ra
ceiling-ra
minimum-ra
maximum-ra
equals-ra
less-ra
less-equal-ra
compare-ra
roots-of-poly-ra
root-ra
show-ra
in Haskell module-name Algebraic-Numbers

References


