

Algebraic Numbers in Isabelle/HOL*

René Thiemann, Akihisa Yamada, and Sebastiaan Joosten

March 29, 2023

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.	6
3.2	Algebraic Numbers – Definition, Inverse, and Roots	7
4	Resultants	17
4.1	Bivariate Polynomials	17
4.1.1	Evaluation of Bivariate Polynomials	18
4.1.2	Swapping the Order of Variables	20
4.2	Resultant	22
4.2.1	Sylvester matrices and vector representation of polynomials	23
4.2.2	Homomorphism and Resultant	24

*Supported by FWF (Austrian Science Fund) project Y757.

4.2.3	Resultant as Polynomial Expression	25
4.2.4	Resultant as Nonzero Polynomial Expression	27
5	Algebraic Numbers: Addition and Multiplication	28
5.1	Addition of Algebraic Numbers	29
5.1.1	<i>poly-add</i> has desired root	29
5.1.2	<i>poly-add</i> is nonzero	30
5.1.3	Summary for addition	32
5.2	Division of Algebraic Numbers	33
5.2.1	Summary for division	34
5.3	Multiplication of Algebraic Numbers	35
5.4	Summary: Closure Properties of Algebraic Numbers	35
5.5	More on algebraic integers	36
6	Separation of Roots: Sturm	37
6.1	Interface for Separating Roots	38
6.2	Implementing Sturm on Rational Polynomials	40
7	Getting Small Representative Polynomials via Factorization	42
8	The minimal polynomial of an algebraic number	44
9	Algebraic Numbers – Preliminary Implementation	46
10	Cauchy’s Root Bound	49
11	Real Algebraic Numbers	50
11.1	Real Algebraic Numbers – Innermost Layer	52
11.1.1	Basic Definitions	52
11.2	Real Algebraic Numbers = Rational + Irrational Real Algebraic Numbers	55
11.2.1	Definitions and Algorithms on Raw Type	55
11.2.2	Definitions and Algorithms on Quotient Type	56
11.2.3	Sign	56
11.2.4	Normalization: Bounds Close Together	56
11.2.5	Comparisons	60
11.2.6	Negation	60
11.2.7	Inverse	61
11.2.8	Floor	62
11.2.9	Generic Factorization and Bisection Framework	62
11.2.10	Addition	65
11.2.11	Multiplication	66
11.2.12	Root	68
11.2.13	Embedding of Rational Numbers	70
11.2.14	Definitions and Algorithms on Type with Invariant	73

11.3 Real Algebraic Numbers as Implementation for Real Numbers	80
12 Real Roots	81
13 Complex Roots of Real Valued Polynomials	83
13.1 Compare Instance for Complex Numbers	86
14 Interval Arithmetic	87
14.1 Syntactic Class Instantiations	87
14.2 Class Instantiations	88
14.3 Membership	89
14.4 Convergence	89
14.5 Complex Intervals	90
15 Complex Algebraic Numbers	93
15.1 Complex Roots	94
16 Show for Real Algebraic Numbers – Interface	100
17 Show for Real (Algebraic) Numbers – Approximate Representation	101
18 Show for Real (Algebraic) Numbers – Unique Representation	101
19 Algebraic Number Tests	102
19.1 Stand-Alone Examples	103
19.2 Example Application: Compute Norms of Eigenvalues	103
20 Explicit Constants for External Code	104
20.1 Operations on Real Algebraic Numbers	104
20.2 Operations on Complex Algebraic Numbers	105
20.3 Export Constants in Haskell	105

1 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 $p :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$ *poly*

assumes *pr*: *primitive p*

shows $p \text{ dvd } 1 \iff \text{degree } p = 0$

<proof>

lemma *dvd-all-coeffs-imp-dvd*:

assumes $\forall a \in \text{set } (\text{coeffs } p). c \text{ dvd } a$ **shows** $[:c:] \text{ dvd } p$

<proof>

lemma *irreducible-content*:

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

assumes *irreducible p* **shows** $\text{degree } p = 0 \vee \text{primitive } p$

<proof>

lemma *linear-irreducible-field*:

fixes $p :: 'a :: \text{field}$ *poly*

assumes *deg*: $\text{degree } p = 1$ **shows** *irreducible p*

<proof>

lemma *linear-irreducible-int*:

fixes $p :: \text{int}$ *poly*

assumes *deg*: $\text{degree } p = 1$ **and** *cp*: *content p dvd 1*

shows *irreducible p*

<proof>

lemma *irreducible-connect-rev*:

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

assumes *irr*: *irreducible p* **and** *deg*: $\text{degree } p > 0$

shows $\text{irreducible}_d p$

<proof>

3.1 Polynomial Evaluation of Integer and Rational Polynomials in Fields.

abbreviation *ipoly* **where** $\text{ipoly } f \ x \equiv \text{poly } (\text{of-int-poly } f) \ x$

lemma *poly-map-poly-code[code-unfold]*: $\text{poly } (\text{map-poly } h \ p) \ x = \text{fold-coeffs } (\lambda a \ b. h \ a + x * b) \ p \ 0$

<proof>

abbreviation *real-of-int-poly* $:: \text{int } \text{poly} \Rightarrow \text{real } \text{poly}$ **where**

$\text{real-of-int-poly} \equiv \text{of-int-poly}$

abbreviation *real-of-rat-poly* $:: \text{rat } \text{poly} \Rightarrow \text{real } \text{poly}$ **where**

$real-of-rat-poly \equiv map-poly\ of-rat$

lemma *of-rat-of-int[simp]*: $of-rat \circ of-int = of-int$ $\langle proof \rangle$

lemma *ipoly-of-rat[simp]*: $ipoly\ p\ (of-rat\ y) = of-rat\ (ipoly\ p\ y)$
 $\langle proof \rangle$

lemma *ipoly-of-real[simp]*:
 $ipoly\ p\ (of-real\ x :: 'a :: \{field, real-algebra-1\}) = of-real\ (ipoly\ p\ x)$
 $\langle proof \rangle$

lemma *finite-ipoly-roots*: **assumes** $p \neq 0$
shows $finite\ \{x :: real.\ ipoly\ p\ x = 0\}$
 $\langle proof \rangle$

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 $algebraic\ x \longleftrightarrow (\exists p.\ ipoly\ p\ x = 0 \wedge p \neq 0)$
 $\langle proof \rangle$

Definition of being algebraic with explicit witness polynomial.

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

lemma *representsI[intro]*: $ipoly\ p\ x = 0 \Longrightarrow p \neq 0 \Longrightarrow p\ represents\ x$
 $\langle proof \rangle$

lemma *representsD*:
assumes $p\ represents\ x$ **shows** $p \neq 0$ **and** $ipoly\ p\ x = 0$ $\langle proof \rangle$

lemma *representsE*:
assumes $p\ represents\ x$ **and** $p \neq 0 \Longrightarrow ipoly\ p\ x = 0 \Longrightarrow thesis$
shows *thesis* $\langle proof \rangle$

lemma *represents-imp-degree*:
fixes $x :: 'a :: field-char-0$
assumes $p\ represents\ x$ **shows** $degree\ p \neq 0$
 $\langle proof \rangle$

lemma *representsE-full[elim]*:
assumes $rep: p\ represents\ x$
and $main: p \neq 0 \Longrightarrow ipoly\ p\ x = 0 \Longrightarrow degree\ p \neq 0 \Longrightarrow thesis$
shows *thesis*
 $\langle proof \rangle$

lemma *represents-of-rat[simp]*: p represents (of-rat x) = p represents x \langle proof \rangle
lemma *represents-of-real[simp]*: p represents (of-real x) = p represents x \langle proof \rangle

lemma *algebraic-iff-represents*: algebraic $x \iff (\exists p. p$ represents $x)$
 \langle proof \rangle

lemma *represents-irr-non-0*:
assumes *irr*: irreducible p **and** *ap*: p represents x **and** *x0*: $x \neq 0$
shows $\text{poly } p \ 0 \neq 0$
 \langle proof \rangle

The polynomial encoding a rational number.

definition *poly-rat* :: $\text{rat} \Rightarrow \text{int poly}$ **where**
 $\text{poly-rat } x = (\text{case quotient-of } x \text{ of } (n,d) \Rightarrow [:-n,d:])$

definition *abs-int-poly*:: $\text{int poly} \Rightarrow \text{int poly}$ **where**
 $\text{abs-int-poly } p \equiv \text{if lead-coeff } p < 0 \text{ then } -p \text{ else } p$

lemma *pos-poly-abs-poly[simp]*:
shows $\text{lead-coeff } (\text{abs-int-poly } p) > 0 \iff p \neq 0$
 \langle proof \rangle

lemma *abs-int-poly-0[simp]*: $\text{abs-int-poly } 0 = 0$
 \langle proof \rangle

lemma *abs-int-poly-eq-0-iff[simp]*: $\text{abs-int-poly } p = 0 \iff p = 0$
 \langle proof \rangle

lemma *degree-abs-int-poly[simp]*: $\text{degree } (\text{abs-int-poly } p) = \text{degree } p$
 \langle proof \rangle

lemma *abs-int-poly-dvd[simp]*: $\text{abs-int-poly } p \text{ dvd } q \iff p \text{ dvd } q$
 \langle proof \rangle

lemma (**in idom**) *irreducible-uminus[simp]*: irreducible $(-x) \iff$ irreducible x
 \langle proof \rangle

lemma *irreducible-abs-int-poly[simp]*:
irreducible $(\text{abs-int-poly } p) \iff$ irreducible p
 \langle proof \rangle

lemma *coeff-abs-int-poly[simp]*:
 $\text{coeff } (\text{abs-int-poly } p) \ n = (\text{if lead-coeff } p < 0 \text{ then } - \text{coeff } p \ n \text{ else } \text{coeff } p \ n)$
 \langle proof \rangle

lemma *lead-coeff-abs-int-poly[simp]*:
 $\text{lead-coeff } (\text{abs-int-poly } p) = \text{abs } (\text{lead-coeff } p)$

$\langle \text{proof} \rangle$

lemma *ipoly-abs-int-poly-eq-zero-iff*[simp]:

$\text{ipoly } (\text{abs-int-poly } p) (x :: 'a :: \text{comm-ring-1}) = 0 \iff \text{ipoly } p \ x = 0$
 $\langle \text{proof} \rangle$

lemma *abs-int-poly-represents*[simp]:

$\text{abs-int-poly } p \text{ represents } x \iff p \text{ represents } x \langle \text{proof} \rangle$

lemma *content-pCons*[simp]: $\text{content } (p\text{Cons } a \ p) = \text{gcd } a \ (\text{content } p)$

$\langle \text{proof} \rangle$

lemma *content-uminus*[simp]:

fixes $p :: 'a :: \text{ring-gcd poly}$ **shows** $\text{content } (-p) = \text{content } p$
 $\langle \text{proof} \rangle$

lemma *primitive-abs-int-poly*[simp]:

$\text{primitive } (\text{abs-int-poly } p) \iff \text{primitive } p$
 $\langle \text{proof} \rangle$

lemma *abs-int-poly-inv*[simp]: $\text{smult } (\text{sgn } (\text{lead-coeff } p)) \ (\text{abs-int-poly } p) = p$

$\langle \text{proof} \rangle$

definition *cf-pos* :: $\text{int poly} \Rightarrow \text{bool}$ **where**

$\text{cf-pos } p = (\text{content } p = 1 \wedge \text{lead-coeff } p > 0)$

definition *cf-pos-poly* :: $\text{int poly} \Rightarrow \text{int poly}$ **where**

$\text{cf-pos-poly } f = (\text{let}$
 $c = \text{content } f;$
 $d = (\text{sgn } (\text{lead-coeff } f) * c)$
 $\text{in } \text{sdiv-poly } f \ d)$

lemma *sgn-is-unit*[intro!]:

fixes $x :: 'a :: \text{linordered-idom}$

assumes $x \neq 0$

shows $\text{sgn } x \ \text{dvd } 1 \langle \text{proof} \rangle$

lemma *cf-pos-poly-0*[simp]: $\text{cf-pos-poly } 0 = 0 \langle \text{proof} \rangle$

lemma *cf-pos-poly-eq-0*[simp]: $\text{cf-pos-poly } f = 0 \iff f = 0$

$\langle \text{proof} \rangle$

lemma

shows *cf-pos-poly-main*: $\text{smult } (\text{sgn } (\text{lead-coeff } f) * \text{content } f) \ (\text{cf-pos-poly } f) = f$ (is ?g1)

and *content-cf-pos-poly[simp]*: $\text{content } (cf\text{-pos-poly } f) = (\text{if } f = 0 \text{ then } 0 \text{ else } 1)$ (**is** ?g2)
and *lead-coeff-cf-pos-poly[simp]*: $\text{lead-coeff } (cf\text{-pos-poly } f) > 0 \longleftrightarrow f \neq 0$ (**is** ?g3)
and *cf-pos-poly-dvd[simp]*: $cf\text{-pos-poly } f \text{ dvd } f$ (**is** ?g4)
 <proof>

lemma *irreducible-connect-int*:

fixes $p :: \text{int poly}$
assumes $ir: \text{irreducible}_a p$ **and** $c: \text{content } p = 1$
shows $\text{irreducible } p$
 <proof>

lemma

fixes $x :: 'a :: \{\text{idom}, \text{ring-char-0}\}$
shows *ipoly-cf-pos-poly-eq-0[simp]*: $\text{ipoly } (cf\text{-pos-poly } p) x = 0 \longleftrightarrow \text{ipoly } p x = 0$
and *degree-cf-pos-poly[simp]*: $\text{degree } (cf\text{-pos-poly } p) = \text{degree } p$
and *cf-pos-cf-pos-poly[intro]*: $p \neq 0 \implies cf\text{-pos } (cf\text{-pos-poly } p)$
 <proof>

lemma *cf-pos-poly-eq-1*: $cf\text{-pos-poly } f = 1 \longleftrightarrow \text{degree } f = 0 \wedge f \neq 0$ (**is** ?l \longleftrightarrow ?r)
 <proof>

lemma *irr-cf-poly-rat[simp]*: $\text{irreducible } (\text{poly-rat } x)$
 $\text{lead-coeff } (\text{poly-rat } x) > 0$ $\text{primitive } (\text{poly-rat } x)$
 <proof>

lemma *poly-rat[simp]*: $\text{ipoly } (\text{poly-rat } x) (\text{of-rat } x :: 'a :: \text{field-char-0}) = 0$ $\text{ipoly } (\text{poly-rat } x) x = 0$
 $\text{poly-rat } x \neq 0$ $\text{ipoly } (\text{poly-rat } x) y = 0 \longleftrightarrow y = (\text{of-rat } x :: 'a)$
 <proof>

lemma *poly-rat-represents-of-rat*: $(\text{poly-rat } x) \text{ represents } (\text{of-rat } x)$ <proof>

lemma *ipoly-smult-0-iff*: **assumes** $c: c \neq 0$

shows $(\text{ipoly } (\text{smult } c p) x = (0 :: \text{real})) = (\text{ipoly } p x = 0)$
 <proof>

lemma *not-irreducibleD*:

assumes $\neg \text{irreducible } x$ **and** $x \neq 0$ **and** $\neg x \text{ dvd } 1$
shows $\exists y z. x = y * z \wedge \neg y \text{ dvd } 1 \wedge \neg z \text{ dvd } 1$ <proof>

lemma *cf-pos-poly-represents[simp]*: *(cf-pos-poly p) represents x* \longleftrightarrow *p represents x*

<proof>

lemma *coprime-prod*:

a $\neq 0 \implies$ *c* $\neq 0 \implies$ *coprime (a * b) (c * d) \implies coprime b (d::'a::\{semiring-gcd\})*

<proof>

lemma *smult-prod*:

*smult a b = monom a 0 * b*

<proof>

lemma *degree-map-poly-2*:

assumes *f (lead-coeff p) $\neq 0$*

shows *degree (map-poly f p) = degree p*

<proof>

lemma *irreducible-cf-pos-poly*:

assumes *irr: irreducible p* **and** *deg: degree p $\neq 0$*

shows *irreducible (cf-pos-poly p) (is irreducible ?p)*

<proof>

locale *dvd-preserving-hom = comm-semiring-1-hom +*

assumes *hom-eq-mult-hom-imp: hom x = hom y * hz \implies \exists z. hz = hom z \wedge x = y * z*

begin

lemma *hom-dvd-hom-iff[simp]*: *hom x dvd hom y* \longleftrightarrow *x dvd y*

<proof>

sublocale *unit-preserving-hom*

<proof>

sublocale *zero-hom-0*

<proof>

end

lemma *smult-inverse-monom: p $\neq 0 \implies$ smult (inverse c) (p::rat poly) = 1* \longleftrightarrow

p = [: c :]

<proof>

lemma *of-int-monom: of-int-poly p = [:rat-of-int c:]* \longleftrightarrow *p = [: c :]* *<proof>*

lemma *degree-0-content*:

fixes *p :: int poly*

assumes *deg: degree p = 0* **shows** *content p = abs (coeff p 0)*

<proof>

lemma *prime-elem-imp-gcd-eq*:
fixes $x::'a::\text{ring-gcd}$
shows $\text{prime-elem } x \implies \text{gcd } x \ y = \text{normalize } x \vee \text{gcd } x \ y = 1$
 $\langle \text{proof} \rangle$

lemma *irreducible-pos-gcd*:
fixes $p :: \text{int poly}$
assumes $\text{ir: irreducible } p$ **and** $\text{pos: lead-coeff } p > 0$ **shows** $\text{gcd } p \ q \in \{1, p\}$
 $\langle \text{proof} \rangle$

lemma *irreducible-pos-gcd-twice*:
fixes $p \ q :: \text{int poly}$
assumes $p: \text{irreducible } p$ $\text{lead-coeff } p > 0$
and $q: \text{irreducible } q$ $\text{lead-coeff } q > 0$
shows $\text{gcd } p \ q = 1 \vee p = q$
 $\langle \text{proof} \rangle$

interpretation *of-rat-hom*: $\text{field-hom-0}' \text{ of-rat} \langle \text{proof} \rangle$

lemma *poly-zero-imp-not-unit*:
assumes $\text{poly } p \ x = 0$ **shows** $\neg p \ \text{dvd } 1$
 $\langle \text{proof} \rangle$

lemma *poly-prod-mset-zero-iff*:
fixes $x :: 'a :: \text{idom}$
shows $\text{poly } (\text{prod-mset } F) \ x = 0 \iff (\exists f \in \# F. \text{poly } f \ x = 0)$
 $\langle \text{proof} \rangle$

lemma *algebraic-imp-represents-irreducible*:
fixes $x :: 'a :: \text{field-char-0}$
assumes $\text{algebraic } x$
shows $\exists p. p \ \text{represents } x \wedge \text{irreducible } p$
 $\langle \text{proof} \rangle$

lemma *algebraic-imp-represents-irreducible-cf-pos*:
assumes $\text{algebraic } (x::'a::\text{field-char-0})$
shows $\exists p. p \ \text{represents } x \wedge \text{irreducible } p \wedge \text{lead-coeff } p > 0 \wedge \text{primitive } p$
 $\langle \text{proof} \rangle$

lemma *gcd-of-int-poly*: $\text{gcd } (\text{of-int-poly } f) (\text{of-int-poly } g :: 'a :: \{\text{field-char-0}, \text{field-gcd}\})$
 $\text{poly} =$
 $\text{smult } (\text{inverse } (\text{of-int } (\text{lead-coeff } (\text{gcd } f \ g)))) (\text{of-int-poly } (\text{gcd } f \ g))$
 $\langle \text{proof} \rangle$

lemma *algebraic-imp-represents-unique*:
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes $\text{algebraic } x$
shows $\exists! p. p \ \text{represents } x \wedge \text{irreducible } p \wedge \text{lead-coeff } p > 0$ (**is** $\text{Ex1 } ?p$)

<proof>

lemma *ipoly-poly-compose*:
 fixes $x :: 'a :: idom$
 shows $ipoly (p \circ_p q) x = ipoly p (ipoly q x)$
<proof>

lemma *algebraic-0[simp]*: *algebraic 0*
<proof>

lemma *algebraic-1[simp]*: *algebraic 1*
<proof>

Polynomial for unary minus.

definition *poly-uminus* :: $'a :: ring-1 poly \Rightarrow 'a poly$ **where** [code del]:
 $poly-uminus p \equiv \sum_{i \leq degree p} monom ((-1)^i * coeff p i) i$

lemma *poly-uminus-pCons-pCons[simp]*:
 $poly-uminus (pCons a (pCons b p)) = pCons a (pCons (-b) (poly-uminus p))$ (**is**
 $?l = ?r$)
<proof>

fun *poly-uminus-inner* :: $'a :: ring-1 list \Rightarrow 'a poly$
where $poly-uminus-inner [] = 0$
 | $poly-uminus-inner [a] = [a:]$
 | $poly-uminus-inner (a\#b\#cs) = pCons a (pCons (-b) (poly-uminus-inner cs))$

lemma *poly-uminus-code[code,simp]*: $poly-uminus p = poly-uminus-inner (coeffs p)$
<proof>

lemma *poly-uminus-inner-0[simp]*: $poly-uminus-inner as = 0 \iff Poly as = 0$
<proof>

lemma *degree-poly-uminus-inner[simp]*: $degree (poly-uminus-inner as) = degree (Poly as)$
<proof>

lemma *ipoly-uminus-inner[simp]*:
 $ipoly (poly-uminus-inner as) (x::'a::comm-ring-1) = ipoly (Poly as) (-x)$
<proof>

lemma *represents-uminus*: **assumes** $alg: p$ *represents* x
 shows $(poly-uminus p)$ *represents* $(-x)$
<proof>

lemma *content-poly-uminus-inner[simp]*:
 fixes $as :: 'a :: ring-gcd list$

shows $\text{content } (\text{poly-uminus-inner } as) = \text{content } (\text{Poly } as)$
 ⟨proof⟩

Multiplicative inverse is represented by *reflect-poly*.

lemma *inverse-pow-minus*: **assumes** $x \neq (0 :: 'a :: \text{field})$
and $i \leq n$
shows $\text{inverse } x \wedge n * x \wedge i = \text{inverse } x \wedge (n - i)$
 ⟨proof⟩

lemma (in *inj-idom-hom*) *reflect-poly-hom*:
 $\text{reflect-poly } (\text{map-poly } \text{hom } p) = \text{map-poly } \text{hom } (\text{reflect-poly } p)$
 ⟨proof⟩

lemma *ipoly-reflect-poly*: **assumes** $x: (x :: 'a :: \text{field-char-0}) \neq 0$
shows $\text{ipoly } (\text{reflect-poly } p) x = x \wedge (\text{degree } p) * \text{ipoly } p (\text{inverse } x)$ (is ?l = ?r)
 ⟨proof⟩

lemma *represents-inverse*: **assumes** $x: x \neq 0$
and $\text{alg}: p \text{ represents } x$
shows $(\text{reflect-poly } p) \text{ represents } (\text{inverse } x)$
 ⟨proof⟩

lemma *inverse-roots*: **assumes** $x: (x :: 'a :: \text{field-char-0}) \neq 0$
shows $\text{ipoly } (\text{reflect-poly } p) x = 0 \iff \text{ipoly } p (\text{inverse } x) = 0$
 ⟨proof⟩

context
fixes $n :: \text{nat}$
begin

Polynomial for n-th root.

definition *poly-nth-root* :: $'a :: \text{idom } \text{poly} \Rightarrow 'a \text{ poly}$ **where**
 $\text{poly-nth-root } p = p \circ_p \text{monom } 1 n$

lemma *ipoly-nth-root*:
fixes $x :: 'a :: \text{idom}$
shows $\text{ipoly } (\text{poly-nth-root } p) x = \text{ipoly } p (x \wedge n)$
 ⟨proof⟩

context
assumes $n: n \neq 0$
begin

lemma *poly-nth-root-0[simp]*: $\text{poly-nth-root } p = 0 \iff p = 0$
 ⟨proof⟩

lemma *represents-nth-root*:
assumes $y: y \wedge n = x$ **and** $\text{alg}: p \text{ represents } x$
shows $(\text{poly-nth-root } p) \text{ represents } y$
 ⟨proof⟩

lemma *represents-nth-root-odd-real:*

assumes *alg: p represents x and odd: odd n*
shows *(poly-nth-root p) represents (root n x)*
<proof>

lemma *represents-nth-root-pos-real:*

assumes *alg: p represents x and pos: x > 0*
shows *(poly-nth-root p) represents (root n x)*
<proof>

lemma *represents-nth-root-neg-real:*

assumes *alg: p represents x and neg: x < 0*
shows *(poly-uminus (poly-nth-root (poly-uminus p))) represents (root n x)*
<proof>
end
end

lemma *represents-csqr:*

assumes *alg: p represents x* **shows** *(poly-nth-root 2 p) represents (csqr x)*
<proof>

lemma *represents-sqr:*

assumes *alg: p represents x and pos: x ≥ 0*
shows *(poly-nth-root 2 p) represents (sqr x)*
<proof>

lemma *represents-degree:*

assumes *p represents x* **shows** *degree p ≠ 0*
<proof>

Polynomial for multiplying a rational number with an algebraic number.

definition *poly-mult-rat-main where*

poly-mult-rat-main n d (f :: 'a :: idom poly) = (let fs = coeffs f; k = length fs in
*poly-of-list (map (λ (fi, i). fi * d ^ i * n ^ (k - Suc i)) (zip fs [0 ..< k])))*

definition *poly-mult-rat :: rat ⇒ int poly ⇒ int poly where*

poly-mult-rat r p ≡ case quotient-of r of (n,d) ⇒ poly-mult-rat-main n d p

lemma *coeff-poly-mult-rat-main: coeff (poly-mult-rat-main n d f) i = coeff f i * n*
*^(degree f - i) * d ^ i*
<proof>

lemma *degree-poly-mult-rat-main: n ≠ 0 ⇒ degree (poly-mult-rat-main n d f) =*
(if d = 0 then 0 else degree f)
<proof>

lemma *ipoly-mult-rat-main:*

fixes *x :: 'a :: {field,ring-char-0}*

assumes $d \neq 0$ **and** $n \neq 0$
shows $\text{ipoly } (\text{poly-mult-rat-main } n \ d \ p) \ x = \text{of-int } n \ \wedge \ \text{degree } p * \text{ipoly } p \ (x * \text{of-int } d / \text{of-int } n)$
 $\langle \text{proof} \rangle$

lemma $\text{degree-poly-mult-rat}[\text{simp}]$: **assumes** $r \neq 0$ **shows** $\text{degree } (\text{poly-mult-rat } r \ p) = \text{degree } p$
 $\langle \text{proof} \rangle$

lemma ipoly-mult-rat :
assumes $r0: r \neq 0$
shows $\text{ipoly } (\text{poly-mult-rat } r \ p) \ x = \text{of-int } (\text{fst } (\text{quotient-of } r)) \ \wedge \ \text{degree } p * \text{ipoly } p \ (x * \text{inverse } (\text{of-rat } r))$
 $\langle \text{proof} \rangle$

lemma $\text{poly-mult-rat-main-0}[\text{simp}]$:
assumes $n \neq 0 \ d \neq 0$ **shows** $\text{poly-mult-rat-main } n \ d \ p = 0 \iff p = 0$
 $\langle \text{proof} \rangle$

lemma $\text{poly-mult-rat-0}[\text{simp}]$: **assumes** $r0: r \neq 0$ **shows** $\text{poly-mult-rat } r \ p = 0 \iff p = 0$
 $\langle \text{proof} \rangle$

lemma $\text{represents-mult-rat}$:
assumes $r: r \neq 0$ **and** p **represents** x **shows** $(\text{poly-mult-rat } r \ p)$ **represents** $(\text{of-rat } r * x)$
 $\langle \text{proof} \rangle$

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

definition $\text{poly-add-rat} :: \text{rat} \Rightarrow \text{int poly} \Rightarrow \text{int poly}$ **where**
 $\text{poly-add-rat } r \ p \equiv \text{case quotient-of } r \ \text{of } (n, d) \Rightarrow$
 $(\text{poly-mult-rat-main } d \ 1 \ p \circ_p \ [:-n, d:])$

lemma $\text{poly-add-rat-code}[\text{code}]$: $\text{poly-add-rat } r \ p \equiv \text{case quotient-of } r \ \text{of } (n, d) \Rightarrow$
 $\text{let } p' = (\text{let } fs = \text{coeffs } p; k = \text{length } fs \ \text{in } \text{poly-of-list } (\text{map } (\lambda(fi, i). fi * d \ \wedge \ (k - \text{Suc } i)) \ (\text{zip } fs \ [0..<k])))$;
 $p'' = p' \circ_p \ [:-n, d:]$
 $\text{in } p''$
 $\langle \text{proof} \rangle$

lemma $\text{degree-poly-add-rat}[\text{simp}]$: $\text{degree } (\text{poly-add-rat } r \ p) = \text{degree } p$
 $\langle \text{proof} \rangle$

lemma ipoly-add-rat : $\text{ipoly } (\text{poly-add-rat } r \ p) \ x = (\text{of-int } (\text{snd } (\text{quotient-of } r)) \ \wedge \ \text{degree } p) * \text{ipoly } p \ (x - \text{of-rat } r)$
 $\langle \text{proof} \rangle$

lemma *poly-add-rat-0*[simp]: $\text{poly-add-rat } r \ p = 0 \iff p = 0$
 ⟨proof⟩

lemma *add-rat-roots*: $\text{ipoly } (\text{poly-add-rat } r \ p) \ x = 0 \iff \text{ipoly } p \ (x - \text{of-rat } r) = 0$
 ⟨proof⟩

lemma *represents-add-rat*:
assumes p represents x **shows** $(\text{poly-add-rat } r \ p)$ represents $(\text{of-rat } r + x)$
 ⟨proof⟩

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*:
 $\text{ipoly } (\text{poly-add-rat } r \ p) \ (x::'a::\text{linordered-field}) < 0 \iff \text{ipoly } p \ (x - \text{of-rat } r) < 0$
 $\text{ipoly } (\text{poly-add-rat } r \ p) \ (x::'a::\text{linordered-field}) > 0 \iff \text{ipoly } p \ (x - \text{of-rat } r) > 0$
 ⟨proof⟩

lemma *sgn-ipoly-add-rat*[simp]:
 $\text{sgn } (\text{ipoly } (\text{poly-add-rat } r \ p) \ (x::'a::\text{linordered-field})) = \text{sgn } (\text{ipoly } p \ (x - \text{of-rat } r))$ (is $\text{sgn } ?l = \text{sgn } ?r$)
 ⟨proof⟩

lemma *deg-nonzero-represents*:
assumes deg : $\text{degree } p \neq 0$ **shows** $\exists x :: \text{complex. } p$ represents x
 ⟨proof⟩

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.Ring-Hom-Poly

Subresultants.More-Homomorphisms

Berlekamp-Zassenhaus.Unique-Factorization-Poly

begin

4.1.1 Evaluation of Bivariate Polynomials

definition $\text{poly2} :: 'a::\text{comm-semiring-1} \text{ poly poly} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
where $\text{poly2 } p \ x \ y = \text{poly } (\text{poly } p \ [: y :]) \ x$

lemma poly2-by-map : $\text{poly2 } p \ x = \text{poly } (\text{map-poly } (\lambda c. \text{poly } c \ x) \ p)$
(proof)

lemma poly2-const[simp] : $\text{poly2 } [: [a] :] \ x \ y = a$ (proof)

lemma $\text{poly2-smult[simp,hom-distrib]}$: $\text{poly2 } (\text{smult } a \ p) \ x \ y = \text{poly } a \ x * \text{poly2 } p \ x \ y$ (proof)

interpretation poly2-hom : $\text{comm-semiring-hom } \lambda p. \text{poly2 } p \ x \ y$ (proof)

interpretation poly2-hom : $\text{comm-ring-hom } \lambda p. \text{poly2 } p \ x \ y$ (proof)

interpretation poly2-hom : $\text{idom-hom } \lambda p. \text{poly2 } p \ x \ y$ (proof)

lemma $\text{poly2-pCons[simp,hom-distrib]}$: $\text{poly2 } (\text{pCons } a \ p) \ x \ y = \text{poly } a \ x + y * \text{poly2 } p \ x \ y$ (proof)

lemma poly2-monom : $\text{poly2 } (\text{monom } a \ n) \ x \ y = \text{poly } a \ x * y \wedge n$ (proof)

lemma $\text{poly-poly-as-poly2}$: $\text{poly2 } p \ x \ (\text{poly } q \ x) = \text{poly } (\text{poly } p \ q) \ x$ (proof)

The following lemma is an extension rule for bivariate polynomials.

lemma poly2-ext :

fixes $p \ q :: 'a :: \{\text{ring-char-0, idom}\} \text{ poly poly}$

assumes $\bigwedge x \ y. \text{poly2 } p \ x \ y = \text{poly2 } q \ x \ y$ shows $p = q$

(proof)

abbreviation (input) $\text{coeff-lift2} == \lambda a. [: [a :] :]$

lemma coeff-lift2-lift : $\text{coeff-lift2} = \text{coeff-lift} \circ \text{coeff-lift}$ (proof)

definition $\text{poly-lift} = \text{map-poly } \text{coeff-lift}$

definition $\text{poly-lift2} = \text{map-poly } \text{coeff-lift2}$

lemma $\text{degree-poly-lift[simp]}$: $\text{degree } (\text{poly-lift } p) = \text{degree } p$
(proof)

lemma poly-lift-0[simp] : $\text{poly-lift } 0 = 0$ (proof)

lemma $\text{poly-lift-0-iff[simp]}$: $\text{poly-lift } p = 0 \longleftrightarrow p = 0$
(proof)

lemma $\text{poly-lift-pCons[simp]}$:

$\text{poly-lift } (\text{pCons } a \ p) = \text{pCons } [: a :] \ (\text{poly-lift } p)$

(proof)

lemma *coeff-poly-lift*[simp]:

fixes $p :: 'a :: \text{comm-monoid-add poly}$

shows $\text{coeff } (\text{poly-lift } p) \ i = \text{coeff-lift } (\text{coeff } p \ i)$

$\langle \text{proof} \rangle$

lemma *pcompose-conv-poly*: $p\text{compose } p \ q = \text{poly } (\text{poly-lift } p) \ q$

$\langle \text{proof} \rangle$

interpretation *poly-lift-hom*: *inj-comm-monoid-add-hom poly-lift*

$\langle \text{proof} \rangle$

interpretation *poly-lift-hom*: *inj-comm-semiring-hom poly-lift*

$\langle \text{proof} \rangle$

interpretation *poly-lift-hom*: *inj-comm-ring-hom poly-lift* $\langle \text{proof} \rangle$

interpretation *poly-lift-hom*: *inj-idom-hom poly-lift* $\langle \text{proof} \rangle$

lemma (in *comm-monoid-add-hom*) *map-poly-hom-coeff-lift*[simp, *hom-distrib*]:

$\text{map-poly } \text{hom } (\text{coeff-lift } a) = \text{coeff-lift } (\text{hom } a) \ \langle \text{proof} \rangle$

lemma (in *comm-ring-hom*) *map-poly-coeff-lift-hom*:

$\text{map-poly } (\text{coeff-lift } \circ \text{hom}) \ p = \text{map-poly } (\text{map-poly } \text{hom}) \ (\text{map-poly } \text{coeff-lift } p)$

$\langle \text{proof} \rangle$

lemma *poly-poly-lift*[simp]:

fixes $p :: 'a :: \text{comm-semiring-0 poly}$

shows $\text{poly } (\text{poly-lift } p) \ [x] = [\text{poly } p \ x]$

$\langle \text{proof} \rangle$

lemma *degree-poly-lift2*[simp]:

$\text{degree } (\text{poly-lift2 } p) = \text{degree } p \ \langle \text{proof} \rangle$

lemma *poly-lift2-0*[simp]: $\text{poly-lift2 } 0 = 0 \ \langle \text{proof} \rangle$

lemma *poly-lift2-0-iff*[simp]: $\text{poly-lift2 } p = 0 \iff p = 0$

$\langle \text{proof} \rangle$

lemma *poly-lift2-pCons*[simp]:

$\text{poly-lift2 } (p\text{Cons } a \ p) = p\text{Cons } [[:a:]] \ (\text{poly-lift2 } p)$

$\langle \text{proof} \rangle$

lemma *poly-lift2-lift*: $\text{poly-lift2} = \text{poly-lift} \circ \text{poly-lift}$ (is ?l = ?r)

$\langle \text{proof} \rangle$

lemma *poly2-poly-lift*[simp]: $\text{poly2 } (\text{poly-lift } p) \ x \ y = \text{poly } p \ y \ \langle \text{proof} \rangle$

lemma *poly-lift2-nonzero*:

assumes $p \neq 0$ **shows** $\text{poly-lift2 } p \neq 0$

$\langle \text{proof} \rangle$

4.1.2 Swapping the Order of Variables

definition

$poly\text{-}y\text{-}x\ p \equiv \sum_{i \leq \text{degree } p} \sum_{j \leq \text{degree } (coeff\ p\ i)} monom\ (monom\ (coeff\ (coeff\ p\ i)\ j)\ i)\ j$

lemma *poly-y-x-fix-y-deg*:

assumes *ydeg*: $\forall i \leq \text{degree } p. \text{degree } (coeff\ p\ i) \leq d$

shows $poly\text{-}y\text{-}x\ p = (\sum_{i \leq \text{degree } p} \sum_{j \leq d} monom\ (monom\ (coeff\ (coeff\ p\ i)\ j)\ i)\ j)$

(**is** $- = \text{sum } (\lambda i. \text{sum } (?f\ i)\ -) -$)

<proof>

lemma *poly-y-x-fixed-deg*:

fixes $p :: 'a :: comm\text{-}monoid\text{-}add\ poly\ poly$

defines $d \equiv Max\ \{ \text{degree } (coeff\ p\ i) \mid i. i \leq \text{degree } p \}$

shows $poly\text{-}y\text{-}x\ p = (\sum_{i \leq \text{degree } p} \sum_{j \leq d} monom\ (monom\ (coeff\ (coeff\ p\ i)\ j)\ i)\ j)$

<proof>

lemma *poly-y-x-swapped*:

fixes $p :: 'a :: comm\text{-}monoid\text{-}add\ poly\ poly$

defines $d \equiv Max\ \{ \text{degree } (coeff\ p\ i) \mid i. i \leq \text{degree } p \}$

shows $poly\text{-}y\text{-}x\ p = (\sum_{j \leq d} \sum_{i \leq \text{degree } p} monom\ (monom\ (coeff\ (coeff\ p\ i)\ j)\ i)\ j)$

<proof>

lemma *poly2-poly-y-x[simp]*: $poly2\ (poly\text{-}y\text{-}x\ p)\ x\ y = poly2\ p\ y\ x$

<proof>

context begin

private lemma *poly-monom-mult*:

fixes $p :: 'a :: comm\text{-}semiring\text{-}1$

shows $poly\ (monom\ p\ i * q \wedge j)\ y = poly\ (monom\ p\ j * [:y:] \wedge i)\ (poly\ q\ y)$

<proof>

lemma *poly-poly-y-x*:

fixes $p :: 'a :: comm\text{-}semiring\text{-}1\ poly\ poly$

shows $poly\ (poly\ (poly\text{-}y\text{-}x\ p)\ q)\ y = poly\ (poly\ p\ [:y:])\ (poly\ q\ y)$

<proof>

end

interpretation *poly-y-x-hom*: *zero-hom poly-y-x* *<proof>*

interpretation *poly-y-x-hom*: *one-hom poly-y-x* *<proof>*

lemma *map-poly-sum-commute*:

assumes $h\ 0 = 0\ \forall p\ q. h\ (p + q) = h\ p + h\ q$

shows $\text{sum } (\lambda i. \text{map}\text{-}poly\ h\ (f\ i))\ S = \text{map}\text{-}poly\ h\ (\text{sum } f\ S)$

$\langle proof \rangle$

lemma *poly-y-x-const*: $poly-y-x \ [:p:] = poly-lift \ p \ (\mathbf{is} \ ?l = ?r)$
 $\langle proof \rangle$

lemma *poly-y-x-pCons*:
shows $poly-y-x \ (pCons \ a \ p) = poly-lift \ a + map-poly \ (pCons \ 0) \ (poly-y-x \ p)$
 $\langle proof \rangle$

lemma *poly-y-x-pCons-0*: $poly-y-x \ (pCons \ 0 \ p) = map-poly \ (pCons \ 0) \ (poly-y-x \ p)$
 $\langle proof \rangle$

lemma *poly-y-x-map-poly-pCons-0*: $poly-y-x \ (map-poly \ (pCons \ 0) \ p) = pCons \ 0 \ (poly-y-x \ p)$
 $\langle proof \rangle$

interpretation *poly-y-x-hom*: $comm-monoid-add-hom \ poly-y-x \ :: \ 'a \ :: \ comm-monoid-add \ poly \ poly \Rightarrow -$
 $\langle proof \rangle$

poly-y-x is bijective.

lemma *poly-y-x-poly-lift*:
fixes $p \ :: \ 'a \ :: \ comm-monoid-add \ poly$
shows $poly-y-x \ (poly-lift \ p) = [:p:]$
 $\langle proof \rangle$

lemma *poly-y-x-id[simp]*:
fixes $p \ :: \ 'a \ :: \ comm-monoid-add \ poly \ poly$
shows $poly-y-x \ (poly-y-x \ p) = p$
 $\langle proof \rangle$

interpretation *poly-y-x-hom*:
 $bijjective \ poly-y-x \ :: \ 'a \ :: \ comm-monoid-add \ poly \ poly \Rightarrow -$
 $\langle proof \rangle$

lemma *inv-poly-y-x[simp]*: $Hilbert-Choice.inv \ poly-y-x = poly-y-x \ \langle proof \rangle$

interpretation *poly-y-x-hom*: $comm-monoid-add-isom \ poly-y-x$
 $\langle proof \rangle$

lemma *pCons-as-add*:
fixes $p \ :: \ 'a \ :: \ comm-semiring-1 \ poly$
shows $pCons \ a \ p = [:a:] + monom \ 1 \ 1 * p \ \langle proof \rangle$

lemma *mult-pCons-0*: $(*) \ (pCons \ 0 \ 1) = pCons \ 0 \ \langle proof \rangle$

lemma *pCons-0-as-mult*:
shows $pCons \ (0 \ :: \ 'a \ :: \ comm-semiring-1) = (\lambda p. \ pCons \ 0 \ 1 * p) \ \langle proof \rangle$

```

lemma map-poly-pCons-0-as-mult:
  fixes p :: 'a :: comm-semiring-1 poly poly
  shows map-poly (pCons 0) p = [:pCons 0 1:] * p
  ⟨proof⟩

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⟩

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⟩

interpretation poly-y-x-hom:
  comm-semiring-isom poly-y-x :: 'a :: comm-semiring-1 poly poly ⇒ -
  ⟨proof⟩

interpretation poly-y-x-hom: comm-ring-isom poly-y-x ⟨proof⟩
interpretation poly-y-x-hom: idom-isom poly-y-x ⟨proof⟩

lemma Max-degree-coeff-pCons:
  Max { degree (coeff (pCons a p) i) | i. i ≤ degree (pCons a p) } =
  max (degree a) (Max { degree (coeff p x) | x. x ≤ degree p })
  ⟨proof⟩

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 - = ?d p)
  ⟨proof⟩

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 $p\ n\ j \equiv$
 $vec\ n\ (\lambda i. \text{if } i \leq j \wedge j \leq \text{degree } p + i \text{ then } \text{coeff } p\ (\text{degree } p + i - j) \text{ else } 0)$

lemma *vec-of-poly-rev-shifted-dim[simp]*: $dim\text{-}vec\ (vec\text{-of-poly-rev-shifted } p\ n\ j) = n$

<proof>

lemma *col-sylvester*:

fixes $p\ q$

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

assumes $j: j < m+n$

shows $col\ (sylvester\text{-}mat\ p\ q)\ j =$

$vec\text{-of-poly-rev-shifted } p\ n\ j\ @_v\ vec\text{-of-poly-rev-shifted } q\ m\ j\ (\text{is } ?l = ?r)$

<proof>

lemma *inj-on-diff-nat2*: $inj\text{-}on\ (\lambda i. (n::nat) - i)\ \{..n\}\ \langle proof \rangle$

lemma *image-diff-atMost*: $(\lambda i. (n::nat) - i)\ \{..n\} = \{..n\}\ (\text{is } ?l = ?r)$

<proof>

lemma *sylvester-sum-mat-upper*:

fixes $p\ q :: 'a :: \text{comm-semiring-1 poly}$

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

assumes $i: i < n$

shows $(\sum j < m+n. \text{monom } (sylvester\text{-}mat\ p\ q\ \$\$ (i,j))\ (m + n - Suc\ j)) =$
 $\text{monom } 1\ (n - Suc\ i) * p\ (\text{is } \text{sum } ?f - = ?r)$

<proof>

lemma *sylvester-sum-mat-lower*:

fixes $p\ q :: 'a :: \text{comm-semiring-1 poly}$

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

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

shows $(\sum j < m+n. \text{monom } (sylvester\text{-}mat\ p\ q\ \$\$ (i,j))\ (m + n - Suc\ j)) =$
 $\text{monom } 1\ (m + n - Suc\ i) * q\ (\text{is } \text{sum } ?f - = ?r)$

<proof>

definition *vec-of-poly* $p \equiv \text{let } m = \text{degree } p \text{ in } vec\ (Suc\ m)\ (\lambda i. \text{coeff } p\ (m-i))$

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

lemma *poly-of-vec-of-poly[simp]*:

fixes $p :: 'a :: \text{comm-monoid-add poly}$

shows $poly\text{-of-}vec\ (vec\text{-of-poly } p) = p$

<proof>

lemma *poly-of-vec-0[simp]*: $\text{poly-of-vec } (0_v \ n) = 0$ *<proof>*

lemma *poly-of-vec-0-iff[simp]*:

fixes $v :: 'a :: \text{comm-monoid-add vec}$

shows $\text{poly-of-vec } v = 0 \iff v = 0_v \ (\text{dim-vec } v)$ (**is** $?v = - \iff - = ?z$)
<proof>

lemma *degree-sum-smaller*:

assumes $n > 0$ *finite* A

shows $(\bigwedge x. x \in A \implies \text{degree } (f \ x) < n) \implies \text{degree } (\sum_{x \in A}. f \ x) < n$

<proof>

lemma *degree-poly-of-vec-less*:

fixes $v :: 'a :: \text{comm-monoid-add vec}$

assumes $\text{dim}: \text{dim-vec } v > 0$

shows $\text{degree } (\text{poly-of-vec } v) < \text{dim-vec } v$

<proof>

lemma *coeff-poly-of-vec*:

$\text{coeff } (\text{poly-of-vec } v) \ i = (\text{if } i < \text{dim-vec } v \text{ then } v \ \$ \ (\text{dim-vec } v - \text{Suc } i) \ \text{else } 0)$

(**is** $?l = ?r$)

<proof>

lemma *vec-of-poly-rev-shifted-scalar-prod*:

fixes $p \ v$

defines $q \equiv \text{poly-of-vec } v$

assumes $m[\text{simp}]: \text{degree } p = m$ **and** $n: \text{dim-vec } v = n$

assumes $j: j < m+n$

shows $\text{vec-of-poly-rev-shifted } p \ n \ (n+m-\text{Suc } j) \cdot v = \text{coeff } (p * q) \ j$ (**is** $?l = ?r$)

<proof>

lemma *syvester-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 $\text{poly-of-vec } (\text{transpose-mat } (\text{syvester-mat } p \ q) * v \ v) =$

$\text{poly-of-vec } (\text{vec-first } v \ n) * p + \text{poly-of-vec } (\text{vec-last } v \ m) * q$ (**is** $?l = ?r$)

<proof>

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 $p \ q :: 'a \ \text{poly}$

shows $\text{hom } (\text{resultant-sub } m \ n \ p \ q) = \text{resultant-sub } m \ n \ (\text{map-poly } \text{hom } p)$

(*map-poly* $\text{hom } q$)

(**is** $?l = ?r'$)

<proof>

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 ($l - \text{Suc } j$) (*mk-poly-sub* A l j)

definition *mk-poly* $A = \text{mk-poly-sub}$ (*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

<proof> **lemma** *mk-poly-sub-carrier*:

assumes $A \in \text{carrier-mat } nr \ nc$ **shows** *mk-poly-sub* A l $j \in \text{carrier-mat } nr \ nc$

<proof> **lemma** *mk-poly-dim[simp]*:

dim-col (*mk-poly* A) = *dim-col* A
dim-row (*mk-poly* A) = *dim-row* A

<proof> **lemma** *mk-poly-sub-others[simp]*:

assumes $l \neq j'$ **and** $i < \text{dim-row } A$ **and** $j' < \text{dim-col } A$

shows *mk-poly-sub* A l j $\$ \$ (i, j') = A$ $\$ \$ (i, j')$

<proof> **lemma** *mk-poly-others[simp]*:

assumes $i: i < \text{dim-row } A$ **and** $j: j < \text{dim-col } A - 1$

shows *mk-poly* A $\$ \$ (i, j) = [: A$ $\$ \$ (i, j) :]$

<proof> **lemma** *mk-poly-delete[simp]*:

assumes $i: i < \text{dim-row } A$

shows *mat-delete* (*mk-poly* A) i (*dim-col* $A - 1$) = *map-mat coeff-lift* (*mat-delete* A i (*dim-col* $A - 1$))

<proof> **lemma** *col-mk-poly-sub[simp]*:

assumes $l \neq j'$ **and** $j' < \text{dim-col } A$

shows *col* (*mk-poly-sub* A l j) $j' = \text{col } A$ j'

<proof> **lemma** *det-mk-poly-sub*:

assumes $A: (A :: 'a :: \text{comm-ring-1 poly mat}) \in \text{carrier-mat } n \ n$ **and** $i: i < n$

shows *det* (*mk-poly-sub* A ($n - 1$) i) = *det* A

<proof> **lemma** *det-mk-poly*:

fixes $A :: 'a :: \text{comm-ring-1 mat}$

shows *det* (*mk-poly* A) = $[: \text{det } A :]$

<proof> **fun** *mk-poly2-row* **where**

mk-poly2-row A d j pv $0 = pv$

| *mk-poly2-row* A d j pv (*Suc* n) =

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

private fun *mk-poly2-col* **where**

mk-poly2-col A pv $0 = pv$

| *mk-poly2-col* A pv (*Suc* m) =

$mk\text{-poly2-row } A \ m \ (dim\text{-col } A - Suc \ m) \ (mk\text{-poly2-col } A \ pv \ m) \ (dim\text{-row } A)$

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

private lemma $mk\text{-poly2-row-dim}[simp]$: $dim\text{-vec } (mk\text{-poly2-row } A \ d \ j \ pv \ i) = dim\text{-vec } pv$

<proof> **lemma** $mk\text{-poly2-col-dim}[simp]$: $dim\text{-vec } (mk\text{-poly2-col } A \ pv \ j) = dim\text{-vec } pv$

<proof> **lemma** $mk\text{-poly2-row}$:

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

shows $mk\text{-poly2-row } A \ d \ j \ pv \ n \ \$ \ i =$

$(if \ i < n \ then \ pv \ \$ \ i + monom \ (A \ \$ \ (i,j)) \ d \ else \ pv \ \$ \ i)$

<proof> **lemma** $mk\text{-poly2-row-col}$:

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

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

<proof> **lemma** $mk\text{-poly2-col}$:

fixes $pv :: 'a :: comm\text{-semiring-1} \ poly \ vec$ **and** $A :: 'a \ mat$

assumes i : $i < dim\text{-row } A$ **and** dim : $dim\text{-row } A = dim\text{-vec } pv$

shows $mk\text{-poly2-col } A \ pv \ j \ \$ \ i = pv \ \$ \ i + (\sum \ j' < j. monom \ (A \ \$ \ (i, dim\text{-col } A - Suc \ j')) \ j')$

<proof> **lemma** $mk\text{-poly2-pre}$:

fixes $A :: 'a :: comm\text{-semiring-1} \ mat$

assumes i : $i < dim\text{-row } A$

shows $mk\text{-poly2 } A \ \$ \ i = (\sum \ j' < dim\text{-col } A. monom \ (A \ \$ \ (i, dim\text{-col } A - Suc \ j')) \ j')$

<proof> **lemma** $mk\text{-poly2}$:

fixes $A :: 'a :: comm\text{-semiring-1} \ mat$

assumes i : $i < dim\text{-row } A$

and c : $dim\text{-col } A > 0$

shows $mk\text{-poly2 } A \ \$ \ i = (\sum \ j' < dim\text{-col } A. monom \ (A \ \$ \ (i,j')) \ (dim\text{-col } A - Suc \ j'))$

$(is \ ?l = sum \ ?f \ ?S)$

<proof> **lemma** $mk\text{-poly2-sylvester-upper}$:

fixes $p \ q :: 'a :: comm\text{-semiring-1} \ poly$

assumes i : $i < degree \ q$

shows $mk\text{-poly2} \ (sylvester\text{-mat } p \ q) \ \$ \ i = monom \ 1 \ (degree \ q - Suc \ i) * p$

<proof> **lemma** $mk\text{-poly2-sylvester-lower}$:

fixes $p \ q :: 'a :: comm\text{-semiring-1} \ poly$

assumes mi : $i \geq degree \ q$ **and** imn : $i < degree \ p + degree \ q$

shows $mk\text{-poly2} \ (sylvester\text{-mat } p \ q) \ \$ \ i = monom \ 1 \ (degree \ p + degree \ q - Suc \ i) * q$

<proof> **lemma** foo :

fixes $v :: 'a :: comm\text{-semiring-1} \ vec$

shows $monom \ 1 \ d \ \cdot_v \ map\text{-vec} \ coeff\text{-lift} \ v = map\text{-vec} \ (\lambda a. monom \ a \ d) \ v$

<proof> **lemma** $mk\text{-poly-sub-corresp}$:

assumes $dimA[simp]$: $dim\text{-col } A = Suc \ l$ **and** $dimpv[simp]$: $dim\text{-vec } pv = dim\text{-row } A$

and j : $j < dim\text{-col } A$

shows $pv + col \ (mk\text{-poly-sub} \ (map\text{-mat} \ coeff\text{-lift} \ A) \ l \ j) \ l =$

$mk\text{-poly2-col } A \text{ pv } (Suc \ j)$
 <proof> **lemma** *col-mk-poly-mk-poly2*:
 fixes $A :: 'a :: comm\text{-semiring-1 } mat$
 assumes $dim: dim\text{-col } A > 0$
 shows $col (mk\text{-poly } A) (dim\text{-col } A - 1) = mk\text{-poly2 } A$
 <proof> **lemma** *mk-poly-mk-poly2*:
 fixes $A :: 'a :: comm\text{-semiring-1 } mat$
 assumes $dim: dim\text{-col } A > 0$ and $i: i < dim\text{-row } A$
 shows $mk\text{-poly } A \ \$\$ (i, dim\text{-col } A - 1) = mk\text{-poly2 } A \ \$ i$
 <proof>

lemma *mk-poly-sylvester-upper*:
 fixes $p \ q :: 'a :: comm\text{-ring-1 } poly$
 defines $m \equiv degree \ p$ and $n \equiv degree \ q$
 assumes $i: i < n$
 shows $mk\text{-poly } (sylvester\text{-mat } p \ q) \ \$\$ (i, m + n - 1) = monom \ 1 \ (n - Suc \ i)$
 $* \ p \ (is \ ?l = ?r)$
 <proof>

lemma *mk-poly-sylvester-lower*:
 fixes $p \ q :: 'a :: comm\text{-ring-1 } poly$
 defines $m \equiv degree \ p$ and $n \equiv degree \ q$
 assumes $ni: n \leq i$ and $imn: i < m+n$
 shows $mk\text{-poly } (sylvester\text{-mat } p \ q) \ \$\$ (i, m + n - 1) = monom \ 1 \ (m + n -$
 $Suc \ i) * q \ (is \ ?l = ?r)$
 <proof>

The next lemma corresponds to Lemma 7.2.1.

lemma *resultant-as-poly*:
 fixes $p \ q :: 'a :: comm\text{-ring-1 } poly$
 assumes $degp: degree \ p > 0$ and $degq: degree \ q > 0$
 shows $\exists p' \ q'. degree \ p' < degree \ q \wedge degree \ q' < degree \ p \wedge$
 $[: resultant \ p \ q :] = p' * p + q' * q$
 <proof>

end

4.2.4 Resultant as Nonzero Polynomial Expression

lemma *resultant-zero*:
 fixes $p \ q :: 'a :: comm\text{-ring-1 } poly$
 assumes $deg: degree \ p > 0 \vee degree \ q > 0$
 and $xp: poly \ p \ x = 0$ and $xq: poly \ q \ x = 0$
 shows $resultant \ p \ q = 0$
 <proof>

lemma *poly-resultant-zero*:
 fixes $p \ q :: 'a :: comm\text{-ring-1 } poly \ poly$
 assumes $deg: degree \ p > 0 \vee degree \ q > 0$
 assumes $p0: poly2 \ p \ x \ y = 0$ and $q0: poly2 \ q \ x \ y = 0$

shows $\text{poly } (\text{resultant } p \ q) \ x = 0$
 $\langle \text{proof} \rangle$

lemma *resultant-as-nonzero-poly-weak*:

fixes $p \ q :: 'a :: \text{idom poly}$
assumes $\text{degp}: \text{degree } p > 0$ **and** $\text{degq}: \text{degree } q > 0$
and $r0: \text{resultant } p \ q \neq 0$
shows $\exists p' \ q'. \text{degree } p' < \text{degree } q \wedge \text{degree } q' < \text{degree } p \wedge$
 $[\text{resultant } p \ q :] = p' * p + q' * q \wedge p' \neq 0 \wedge q' \neq 0$
 $\langle \text{proof} \rangle$

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 $\exists p' \ q'. \text{degree } p' < n \wedge \text{degree } q' < m \wedge$
 $[\text{resultant } p \ q :] = p' * p + q' * q \wedge p' \neq 0 \wedge q' \neq 0$
 $\langle \text{proof} \rangle$

Corresponds to Lemma 7.2.3 of the textbook

lemma *resultant-zero-imp-common-factor*:

fixes $p \ q :: 'a :: \text{ufd poly}$
assumes $\text{deg}: \text{degree } p > 0 \vee \text{degree } q > 0$ **and** $r0: \text{resultant } p \ q = 0$
shows $\neg \text{coprime } p \ q$
 $\langle \text{proof} \rangle$

lemma *resultant-non-zero-imp-coprime*:

assumes $\text{nz}: \text{resultant } (f :: 'a :: \text{field poly}) \ g \neq 0$
and $\text{nz}': f \neq 0 \vee g \neq 0$
shows $\text{coprime } f \ g$
 $\langle \text{proof} \rangle$

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* $\lambda p. \text{coeff } p \ i \ \langle \text{proof} \rangle$

interpretation *coeff-hom: comm-monoid-add-hom* $\lambda p. \text{coeff } p \ i \langle \text{proof} \rangle$
interpretation *coeff-hom: group-add-hom* $\lambda p. \text{coeff } p \ i \langle \text{proof} \rangle$
interpretation *coeff-hom: ab-group-add-hom* $\lambda p. \text{coeff } p \ i \langle \text{proof} \rangle$
interpretation *coeff-0-hom: monoid-mult-hom* $\lambda p. \text{coeff } p \ 0 \langle \text{proof} \rangle$
interpretation *coeff-0-hom: semiring-hom* $\lambda p. \text{coeff } p \ 0 \langle \text{proof} \rangle$
interpretation *coeff-0-hom: comm-monoid-mult-hom* $\lambda p. \text{coeff } p \ 0 \langle \text{proof} \rangle$
interpretation *coeff-0-hom: comm-semiring-hom* $\lambda p. \text{coeff } p \ 0 \langle \text{proof} \rangle$

5.1 Addition of Algebraic Numbers

definition $x-y \equiv [[: 0, 1 :], -1 :]$

definition $\text{poly-}x\text{-minus-}y \ p = \text{poly-lift } p \circ_p \ x-y$

lemma *coeff-xy-power*:

assumes $k \leq n$

shows $\text{coeff } (x-y \wedge n :: 'a :: \text{comm-ring-1 poly poly}) \ k =$
 $\text{monom } (\text{of-nat } (n \ \text{choose } (n - k)) * (-1) \wedge k) \ (n - k)$

$\langle \text{proof} \rangle$

The following polynomial represents the sum of two algebraic numbers.

definition $\text{poly-add} :: 'a :: \text{comm-ring-1 poly} \Rightarrow 'a \ \text{poly} \Rightarrow 'a \ \text{poly}$ **where**
 $\text{poly-add } p \ q = \text{resultant } (\text{poly-}x\text{-minus-}y \ p) \ (\text{poly-lift } q)$

5.1.1 *poly-add* has desired root

interpretation *poly-}x\text{-minus-}y\text{-hom}*:

comm-ring-hom poly-}x\text{-minus-}y \langle \text{proof} \rangle

lemma *poly2-}x\text{-}y[simp]*:

fixes $x :: 'a :: \text{comm-ring-1}$

shows $\text{poly2 } x-y \ x \ y = x - y \langle \text{proof} \rangle$

lemma *degree-poly-}x\text{-minus-}y[simp]*:

fixes $p :: 'a :: \text{idom poly}$

shows $\text{degree } (\text{poly-}x\text{-minus-}y \ p) = \text{degree } p \langle \text{proof} \rangle$

lemma *poly-}x\text{-minus-}y\text{-pCons[simp]*:

$\text{poly-}x\text{-minus-}y \ (p\text{Cons } a \ p) = [[: a :]] + \text{poly-}x\text{-minus-}y \ p * x-y$
 $\langle \text{proof} \rangle$

lemma *poly-poly-poly-}x\text{-minus-}y[simp]*:

fixes $p :: 'a :: \text{comm-ring-1 poly}$

shows $\text{poly } (\text{poly } (\text{poly-}x\text{-minus-}y \ p) \ q) \ x = \text{poly } p \ (x - \text{poly } q \ x)$
 $\langle \text{proof} \rangle$

lemma *poly2-poly-}x\text{-minus-}y[simp]*:

fixes $p :: 'a :: \text{comm-ring-1 poly}$

shows $\text{poly2 } (\text{poly-}x\text{-minus-}y \ p) \ x \ y = \text{poly } p \ (x-y) \langle \text{proof} \rangle$

interpretation *x-y-mult-hom*: *zero-hom-0* $\lambda p :: 'a :: \text{comm-ring-1 poly poly. } x-y * p$
 $\langle \text{proof} \rangle$

lemma *x-y-nonzero[simp]*: $x-y \neq 0 \langle \text{proof} \rangle$

lemma *degree-x-y[simp]*: $\text{degree } x-y = 1 \langle \text{proof} \rangle$

interpretation *x-y-mult-hom*: *inj-comm-monoid-add-hom* $\lambda p :: 'a :: \text{idom poly poly. } x-y * p$
 $\langle \text{proof} \rangle$

interpretation *poly-x-minus-y-hom*: *inj-idom-hom* *poly-x-minus-y*
 $\langle \text{proof} \rangle$

lemma *poly-add*:

fixes $p q :: 'a :: \text{comm-ring-1 poly}$

assumes $q0: q \neq 0$ **and** $x: \text{poly } p x = 0$ **and** $y: \text{poly } q y = 0$

shows $\text{poly } (\text{poly-add } p q) (x+y) = 0$

$\langle \text{proof} \rangle$

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* $:: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$
 $\text{poly} \Rightarrow -$
 $\langle \text{proof} \rangle$

interpretation *poly-lift-hom*:

factor-preserving-hom *poly-lift* $:: 'a :: \text{idom poly} \Rightarrow 'a \text{ poly poly}$
 $\langle \text{proof} \rangle$

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

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

context *idom-isom* **begin**

sublocale *comm-semiring-isom* $\langle \text{proof} \rangle$

end

interpretation *poly-x-minus-y-hom*:

factor-preserving-hom *poly-x-minus-y* $:: 'a :: \text{idom poly} \Rightarrow 'a \text{ poly poly}$
 $\langle \text{proof} \rangle$

Now we show that results of *poly-x-minus-y* and *poly-lift* are coprime.

lemma *poly-y-x-const[simp]*: $\text{poly-y-x } [[:a:]] = [[:a:]] \langle \text{proof} \rangle$

context begin

private abbreviation $y-x == [[: 0, -1 :], 1 :]$

lemma $poly-y-x-x-y[simp]$: $poly-y-x x-y = y-x$ $\langle proof \rangle$ **lemma** $y-x[simp]$: **fixes** $x :: 'a :: comm-ring-1$ **shows** $poly2 y-x x y = y - x$
 $\langle proof \rangle$ **definition** $poly-y-minus-x p \equiv poly-lift p \circ_p y-x$

private lemma $poly-y-minus-x-0[simp]$: $poly-y-minus-x 0 = 0$ $\langle proof \rangle$ **lemma** $poly-y-minus-x-pCons[simp]$:
 $poly-y-minus-x (pCons a p) = [[: a :]] + poly-y-minus-x p * y-x$ $\langle proof \rangle$ **lemma** $poly-y-x-poly-x-minus-y$:
fixes $p :: 'a :: idom poly$
shows $poly-y-x (poly-x-minus-y p) = poly-y-minus-x p$
 $\langle proof \rangle$

lemma $degree-poly-y-minus-x[simp]$:
fixes $p :: 'a :: idom poly$
shows $degree (poly-y-x (poly-x-minus-y p)) = degree p$
 $\langle proof \rangle$

end

lemma $dvd-all-coeffs-iff$:
fixes $x :: 'a :: comm-semiring-1$
shows $(\forall pi \in set (coeffs p). x dvd pi) \longleftrightarrow (\forall i. x dvd coeff p i)$ **(is ?l = ?r)**
 $\langle proof \rangle$

lemma $primitive-imp-no-constant-factor$:
fixes $p :: 'a :: \{comm-semiring-1, semiring-no-zero-divisors\} poly$
assumes pr : $primitive p$ **and** F : $mset-factors F p$ **and** fF : $f \in \# F$
shows $degree f \neq 0$
 $\langle proof \rangle$

lemma $coprime-poly-x-minus-y-poly-lift$:
fixes $p q :: 'a :: ufd poly$
assumes $degp$: $degree p > 0$ **and** $degq$: $degree q > 0$
and pr : $primitive p$
shows $coprime (poly-x-minus-y p) (poly-lift q)$
 $\langle proof \rangle$

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 $poly-add p q \neq 0$
 $\langle proof \rangle$

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*:
 $map\text{-}poly\ (map\text{-}poly\ hom)\ (poly\text{-}x\text{-}minus\text{-}y\ p) = poly\text{-}x\text{-}minus\text{-}y\ (map\text{-}poly\ hom\ p)$
 ⟨*proof*⟩

lemma (in *comm-ring-hom*) *hom-poly-lift[simp]*:
 $map\text{-}poly\ (map\text{-}poly\ hom)\ (poly\text{-}lift\ q) = poly\text{-}lift\ (map\text{-}poly\ hom\ q)$
 ⟨*proof*⟩

lemma *lead-coeff-poly-x-minus-y*:
fixes $p :: 'a :: idom\ poly$
shows $lead\text{-}coeff\ (poly\text{-}x\text{-}minus\text{-}y\ p) = [lead\text{-}coeff\ p * ((- 1) ^ degree\ p):]$ (is ?l = ?r)
 ⟨*proof*⟩

lemma *degree-coeff-poly-x-minus-y*:
fixes $p\ q :: 'a :: \{idom, semiring\text{-}char\text{-}0\}\ poly$
shows $degree\ (coeff\ (poly\text{-}x\text{-}minus\text{-}y\ p)\ i) = degree\ p - i$
 ⟨*proof*⟩

lemma *coeff-0-poly-x-minus-y [simp]*: $coeff\ (poly\text{-}x\text{-}minus\text{-}y\ p)\ 0 = p$
 ⟨*proof*⟩

lemma (in *idom-hom*) *poly-add-hom*:
assumes $p0: hom\ (lead\text{-}coeff\ p) \neq 0$ **and** $q0: hom\ (lead\text{-}coeff\ q) \neq 0$
shows $map\text{-}poly\ hom\ (poly\text{-}add\ p\ q) = poly\text{-}add\ (map\text{-}poly\ hom\ p)\ (map\text{-}poly\ hom\ q)$
 ⟨*proof*⟩

lemma(in *zero-hom*) *hom-lead-coeff-nonzero-imp-map-poly-hom*:
assumes $hom\ (lead\text{-}coeff\ p) \neq 0$
shows $map\text{-}poly\ hom\ p \neq 0$
 ⟨*proof*⟩

lemma *ipoly-poly-add*:
fixes $x\ y :: 'a :: idom$
assumes $p0: (of\text{-}int\ (lead\text{-}coeff\ p) :: 'a) \neq 0$ **and** $q0: (of\text{-}int\ (lead\text{-}coeff\ q) :: 'a) \neq 0$
and $x: ipoly\ p\ x = 0$ **and** $y: ipoly\ q\ y = 0$
shows $ipoly\ (poly\text{-}add\ p\ q)\ (x+y) = 0$
 ⟨*proof*⟩

lemma (in *comm-monoid-gcd*) *gcd-list-eq-0-iff[simp]*: $listgcd\ xs = 0 \longleftrightarrow (\forall x \in set\ xs.\ x = 0)$
 ⟨*proof*⟩

lemma *primitive-field-poly[simp]*: *primitive* $(p :: 'a :: \text{field poly}) \longleftrightarrow p \neq 0$
 <proof>

lemma *ipoly-poly-add-nonzero*:
fixes $x y :: 'a :: \text{field}$
assumes $p \neq 0$ **and** $q \neq 0$ **and** $\text{ipoly } p \ x = 0$ **and** $\text{ipoly } q \ y = 0$
and $(\text{of-int } (\text{lead-coeff } p) :: 'a) \neq 0$ **and** $(\text{of-int } (\text{lead-coeff } q) :: 'a) \neq 0$
shows $\text{poly-add } p \ q \neq 0$
 <proof>

lemma *represents-add*:
assumes x : p *represents* x **and** y : q *represents* y
shows $(\text{poly-add } p \ q)$ *represents* $(x + y)$
 <proof>

5.2 Division of Algebraic Numbers

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

lemma *coeff-poly-x-mult-y*:
shows $\text{coeff } (\text{poly-x-mult-y } p) \ i = \text{monom } (\text{coeff } p \ i) \ i$ (**is** $?l = ?r$)
 <proof>

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

definition *poly-div :: 'a :: comm-ring-1 poly \Rightarrow 'a poly \Rightarrow 'a poly where*
 $\text{poly-div } p \ q = \text{resultant } (\text{poly-x-mult-y } p) \ (\text{poly-lift } q)$

poly-div has desired roots.

lemma *poly2-poly-x-mult-y*:
fixes $p :: 'a :: \text{comm-ring-1 poly}$
shows $\text{poly2 } (\text{poly-x-mult-y } p) \ x \ y = \text{poly } p \ (x * y)$
 <proof>

lemma *poly-div*:
fixes $p \ q :: 'a :: \text{field poly}$
assumes $q0$: $q \neq 0$ **and** x : $\text{poly } p \ x = 0$ **and** y : $\text{poly } q \ y = 0$ **and** $y0$: $y \neq 0$
shows $\text{poly } (\text{poly-div } p \ q) \ (x/y) = 0$
 <proof>

poly-div is nonzero.

interpretation *poly-x-mult-y-hom*: $\text{ring-hom } \text{poly-x-mult-y} :: 'a :: \{\text{idom}, \text{ring-char-0}\}$
 $\text{poly} \Rightarrow -$
 <proof>

interpretation *poly-x-mult-y-hom*: *inj-ring-hom poly-x-mult-y :: 'a :: {idom,ring-char-0}*
poly \Rightarrow -
 \langle *proof* \rangle

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** $?l = ?r$)
 \langle *proof* \rangle

interpretation *poly-x-mult-y-hom*: *unit-preserving-hom poly-x-mult-y :: 'a :: field-char-0*
poly \Rightarrow -
 \langle *proof* \rangle

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 $degree f = degree g$
 \langle *proof* \rangle

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 \neq 0 \vee poly q 0 \neq 0*
shows *coprime (poly-x-mult-y p) (poly-lift q)*
 \langle *proof* \rangle

lemma *poly-div-nonzero*:
fixes $p q :: 'a :: field-char-0$ *poly*
assumes *p0: p \neq 0* **and** *q0: q \neq 0* **and** *x: poly p x = 0* **and** *y: poly q y = 0*
and *p-0: poly p 0 \neq 0 \vee poly q 0 \neq 0*
shows *poly-div p q \neq 0*
 \langle *proof* \rangle

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*:
 $poly-x-mult-y (map-poly hom p) = map-poly (map-poly hom) (poly-x-mult-y p)$
 \langle *proof* \rangle

lemma (**in** *inj-comm-ring-hom*) *poly-div-hom*:
 $map-poly hom (poly-div p q) = poly-div (map-poly hom p) (map-poly hom q)$
 \langle *proof* \rangle

lemma *ipoly-poly-div*:
fixes $x y :: 'a :: field-char-0$

assumes $q \neq 0$ **and** $\text{ipoly } p \ x = 0$ **and** $\text{ipoly } q \ y = 0$ **and** $y \neq 0$
shows $\text{ipoly } (\text{poly-div } p \ q) \ (x/y) = 0$
 $\langle \text{proof} \rangle$

lemma *ipoly-poly-div-nonzero*:
fixes $x \ y :: 'a :: \text{field-char-0}$
assumes $p \neq 0$ **and** $q \neq 0$ **and** $\text{ipoly } p \ x = 0$ **and** $\text{ipoly } q \ y = 0$ **and** $\text{poly } p \ 0 \neq 0 \vee \text{poly } q \ 0 \neq 0$
shows $\text{poly-div } p \ q \neq 0$
 $\langle \text{proof} \rangle$

lemma *represents-div*:
fixes $x \ y :: 'a :: \text{field-char-0}$
assumes p *represents* x **and** q *represents* y **and** $\text{poly } q \ 0 \neq 0$
shows $(\text{poly-div } p \ q)$ *represents* (x / y)
 $\langle \text{proof} \rangle$

5.3 Multiplication of Algebraic Numbers

definition *poly-mult* **where** $\text{poly-mult } p \ q \equiv \text{poly-div } p \ (\text{reflect-poly } q)$

lemma *represents-mult*:
assumes px : p *represents* x **and** qy : q *represents* y **and** $q-0$: $\text{poly } q \ 0 \neq 0$
shows $(\text{poly-mult } p \ q)$ *represents* $(x * y)$
 $\langle \text{proof} \rangle$

5.4 Summary: Closure Properties of Algebraic Numbers

lemma *algebraic-representsI*: p *represents* $x \implies$ *algebraic* x
 $\langle \text{proof} \rangle$

lemma *algebraic-of-rat*: *algebraic* $(\text{of-rat } x)$
 $\langle \text{proof} \rangle$

lemma *algebraic-uminus*: *algebraic* $x \implies$ *algebraic* $(-x)$
 $\langle \text{proof} \rangle$

lemma *algebraic-inverse*: *algebraic* $x \implies$ *algebraic* $(\text{inverse } x)$
 $\langle \text{proof} \rangle$

lemma *algebraic-plus*: *algebraic* $x \implies$ *algebraic* $y \implies$ *algebraic* $(x + y)$
 $\langle \text{proof} \rangle$

lemma *algebraic-div*:
assumes x : *algebraic* x **and** y : *algebraic* y **shows** *algebraic* (x/y)
 $\langle \text{proof} \rangle$

lemma *algebraic-times*: *algebraic* $x \implies$ *algebraic* $y \implies$ *algebraic* $(x * y)$
 $\langle \text{proof} \rangle$

lemma algebraic-root: $\text{algebraic } x \implies \text{algebraic } (\text{root } n \ x)$
 ⟨proof⟩

lemma algebraic-nth-root: $n \neq 0 \implies \text{algebraic } x \implies y^n = x \implies \text{algebraic } y$
 ⟨proof⟩

5.5 More on algebraic integers

definition poly-add-sign :: $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a :: \text{comm-ring-1}$ **where**
 $\text{poly-add-sign } m \ n = \text{signof } (\lambda i. \text{if } i < n \text{ then } m + i \text{ else if } i < m + n \text{ then } i - n \text{ else } i)$

lemma lead-coeff-poly-add:
fixes $p \ q :: 'a :: \{\text{idom}, \text{semiring-char-0}\}$ **poly**
defines $m \equiv \text{degree } p$ **and** $n \equiv \text{degree } q$
assumes $\text{lead-coeff } p = 1$ $\text{lead-coeff } q = 1$ $m > 0$ $n > 0$
shows $\text{lead-coeff } (\text{poly-add } p \ q :: 'a \ \text{poly}) = \text{poly-add-sign } m \ n$
 ⟨proof⟩

lemma lead-coeff-poly-mult:
fixes $p \ q :: 'a :: \{\text{idom}, \text{ring-char-0}\}$ **poly**
defines $m \equiv \text{degree } p$ **and** $n \equiv \text{degree } q$
assumes $\text{lead-coeff } p = 1$ $\text{lead-coeff } q = 1$ $m > 0$ $n > 0$
assumes $\text{coeff } q \ 0 \neq 0$
shows $\text{lead-coeff } (\text{poly-mult } p \ q :: 'a \ \text{poly}) = 1$
 ⟨proof⟩

lemma algebraic-int-plus [intro]:
fixes $x \ y :: 'a :: \text{field-char-0}$
assumes $\text{algebraic-int } x$ $\text{algebraic-int } y$
shows $\text{algebraic-int } (x + y)$
 ⟨proof⟩

lemma algebraic-int-times [intro]:
fixes $x \ y :: 'a :: \text{field-char-0}$
assumes $\text{algebraic-int } x$ $\text{algebraic-int } y$
shows $\text{algebraic-int } (x * y)$
 ⟨proof⟩

lemma algebraic-int-power [intro]:
 $\text{algebraic-int } (x :: 'a :: \text{field-char-0}) \implies \text{algebraic-int } (x^n)$
 ⟨proof⟩

lemma algebraic-int-diff [intro]:
fixes $x \ y :: 'a :: \text{field-char-0}$
assumes $\text{algebraic-int } x$ $\text{algebraic-int } y$
shows $\text{algebraic-int } (x - y)$
 ⟨proof⟩

lemma *algebraic-int-sum* [*intro*]:
 $(\bigwedge x. x \in A \implies \text{algebraic-int } (f\ x :: 'a :: \text{field-char-0}))$
 $\implies \text{algebraic-int } (\text{sum } f\ A)$
 ⟨*proof*⟩

lemma *algebraic-int-prod* [*intro*]:
 $(\bigwedge x. x \in A \implies \text{algebraic-int } (f\ x :: 'a :: \text{field-char-0}))$
 $\implies \text{algebraic-int } (\text{prod } f\ A)$
 ⟨*proof*⟩

lemma *algebraic-int-nth-root-real-iff*:
 $\text{algebraic-int } (\text{root } n\ x) \longleftrightarrow n = 0 \vee \text{algebraic-int } x$
 ⟨*proof*⟩

lemma *algebraic-int-power-iff*:
 $\text{algebraic-int } (x \wedge^n :: 'a :: \text{field-char-0}) \longleftrightarrow n = 0 \vee \text{algebraic-int } x$
 ⟨*proof*⟩

lemma *algebraic-int-power-iff'* [*simp*]:
 $n > 0 \implies \text{algebraic-int } (x \wedge^n :: 'a :: \text{field-char-0}) \longleftrightarrow \text{algebraic-int } x$
 ⟨*proof*⟩

lemma *algebraic-int-sqrt-iff* [*simp*]: $\text{algebraic-int } (\text{sqrt } x) \longleftrightarrow \text{algebraic-int } x$
 ⟨*proof*⟩

lemma *algebraic-int-csqrt-iff* [*simp*]: $\text{algebraic-int } (\text{csqrt } x) \longleftrightarrow \text{algebraic-int } x$
 ⟨*proof*⟩

lemma *algebraic-int-norm-complex* [*intro*]:
assumes $\text{algebraic-int } (z :: \text{complex})$
shows $\text{algebraic-int } (\text{norm } z)$
 ⟨*proof*⟩

hide-const (open) x - y

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 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  $p :: 'a :: \{\text{semiring-gcd, semiring-no-zero-divisors}\}$  poly
  shows poly (primitive-part  $p$ )  $x = 0 \iff$  poly  $p$   $x = 0$ 
  <proof>

```

```

lemma irreducible-primitive-part:
  assumes irreducible  $p$  and degree  $p > 0$ 
  shows primitive-part  $p = p$ 
  <proof>

```

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  $\Rightarrow$  rat  $\Rightarrow$  nat) (number-root: rat  $\Rightarrow$  nat)
hide-const (open) l-r
hide-const (open) number-root

```

```

definition count-roots-interval-sf :: real poly  $\Rightarrow$  (real  $\Rightarrow$  real  $\Rightarrow$  nat)  $\times$  (real  $\Rightarrow$  nat)
where
  count-roots-interval-sf  $p =$  (let  $ps =$  sturm-squarefree  $p$ 
    in (( $\lambda$  a b. sign-changes  $ps$  a - sign-changes  $ps$  b + (if poly  $p$  a = 0 then 1 else 0)),
    ( $\lambda$  a. sign-changes-neg-inf  $ps$  - sign-changes  $ps$  a)))

```

```

definition count-roots-interval :: real poly  $\Rightarrow$  (real  $\Rightarrow$  real  $\Rightarrow$  nat)  $\times$  (real  $\Rightarrow$  nat)
where
  count-roots-interval  $p =$  (let  $ps =$  sturm  $p$ 
    in (( $\lambda$  a b. sign-changes  $ps$  a - sign-changes  $ps$  b + (if poly  $p$  a = 0 then 1 else 0)),
    ( $\lambda$  a. sign-changes-neg-inf  $ps$  - sign-changes  $ps$  a)))

```

```

lemma count-roots-interval-iff: square-free  $p \implies$  count-roots-interval  $p =$  count-roots-interval-sf
 $p$ 
  <proof>

```

```

lemma count-roots-interval-sf: assumes  $p: p \neq 0$ 
  and  $cr: \text{count-roots-interval-sf } p = (cr, nr)$ 
  shows  $a \leq b \implies cr\ a\ b = (\text{card } \{x. a \leq x \wedge x \leq b \wedge \text{poly } p\ x = 0\})$ 

```

$nr\ a = \text{card } \{x. x \leq a \wedge \text{poly } p\ x = 0\}$
 ⟨proof⟩

lemma *count-roots-interval*: **assumes** *cr*: *count-roots-interval* $p = (cr, nr)$
and *sf*: *square-free* p
shows $a \leq b \implies cr\ a\ b = (\text{card } \{x. a \leq x \wedge x \leq b \wedge \text{poly } p\ x = 0\})$
 $nr\ a = \text{card } \{x. x \leq a \wedge \text{poly } p\ x = 0\}$
 ⟨proof⟩

definition *root-cond* :: $\text{int poly} \times \text{rat} \times \text{rat} \Rightarrow \text{real} \Rightarrow \text{bool}$ **where**
 $root\text{-}cond\ plr\ x = (\text{case } plr\ \text{of } (p, l, r) \Rightarrow \text{of-rat } l \leq x \wedge x \leq \text{of-rat } r \wedge \text{ipoly } p\ x = 0)$

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

lemma *root-info-condD*: $root\text{-}info\text{-}cond\ ri\ p \implies a \leq b \implies \text{root-info.l-r } ri\ a\ b = \text{card } \{x. \text{root-cond } (p, a, b)\ x\}$
 $root\text{-}info\text{-}cond\ ri\ p \implies \text{root-info.number-root } ri\ a = \text{card } \{x. x \leq \text{real-of-rat } a \wedge \text{ipoly } p\ x = 0\}$
 ⟨proof⟩

definition *count-roots-interval-sf-rat* :: $\text{int poly} \Rightarrow \text{root-info}$ **where**
 $count\text{-}roots\text{-}interval\text{-}sf\text{-}rat\ p = (\text{let } pp = \text{real-of-int-poly } p;$
 $(cr, nr) = \text{count-roots-interval-sf } pp$
 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* :: $\text{int poly} \Rightarrow \text{root-info}$ **where**
 [code del]: $count\text{-}roots\text{-}interval\text{-}rat\ p = (\text{let } pp = \text{real-of-int-poly } p;$
 $(cr, nr) = \text{count-roots-interval } pp$
 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* :: $\text{int poly} \Rightarrow \text{nat}$ **where**
 [code del]: $count\text{-}roots\text{-}rat\ p = (\text{count-roots } (\text{real-of-int-poly } p))$

lemma *count-roots-interval-sf-rat*: **assumes** $p: p \neq 0$
shows $root\text{-}info\text{-}cond\ (\text{count-roots-interval-sf-rat } p)\ p$
 ⟨proof⟩

lemma *of-rat-of-int-poly*: $\text{map-poly } \text{of-rat } (\text{of-int-poly } p) = \text{of-int-poly } p$
 ⟨proof⟩

lemma *square-free-of-int-poly*: **assumes** *square-free* p
shows $\text{square-free } (\text{of-int-poly } p :: 'a :: \{\text{field-gcd, field-char-0}\} \text{poly})$
 ⟨proof⟩

lemma *count-roots-interval-rat*: **assumes** *sf*: square-free *p*
shows *root-info-cond* (*count-roots-interval-rat* *p*) *p*
 ⟨*proof*⟩

lemma *count-roots-rat*: *count-roots-rat* *p* = *card* {*x*. *ipoly* *p* *x* = (0 :: *real*)}
 ⟨*proof*⟩

6.2 Implementing Sturm on Rational Polynomials

function *sturm-aux-rat* **where**
sturm-aux-rat (*p* :: *rat poly*) *q* =
 (if *degree* *q* = 0 then [*p*,*q*] else *p* # *sturm-aux-rat* *q* (-(*p mod q*)))
 ⟨*proof*⟩
termination ⟨*proof*⟩

lemma *sturm-aux-rat*: *sturm-aux* (*real-of-rat-poly* *p*) (*real-of-rat-poly* *q*) =
map *real-of-rat-poly* (*sturm-aux-rat* *p* *q*)
 ⟨*proof*⟩

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*)
 ⟨*proof*⟩

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*)
 ⟨*proof*⟩

lemma *poly-neg-number-rootat*: *poly-neg-inf* (*real-of-rat-poly* *p*) = *real-of-rat* (*poly-neg-number-rootat* *p*)
 ⟨*proof*⟩

definition *sign-changes-rat* **where**
sign-changes-rat *ps* (*x*::*rat*) =
length (*remdups-adj* (*filter* ($\lambda x. x \neq 0$) (*map* ($\lambda p. \text{sgn} (\text{poly } p \ x)$) *ps*))) - 1

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

definition *sign-changes-neg-number-rootat* **where**

sign-changes-neg-number-rootat *ps* =
 $\text{length } (\text{remdups-adj } (\text{filter } (\lambda x. x \neq 0) (\text{map } \text{poly-neg-number-rootat } ps))) - 1$

lemma *real-of-rat-list-neq*: *list-neq* (*map* *real-of-rat* *xs*) 0

= *map* *real-of-rat* (*list-neq* *xs* 0)
 ⟨*proof*⟩

lemma *real-of-rat-remdups-adj*: *remdups-adj* (*map* *real-of-rat* *xs*) = *map* *real-of-rat* (*remdups-adj* *xs*)

⟨*proof*⟩

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*⟩

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*⟩

lemma *sign-changes-number-rootat*: *sign-changes-inf* (*map* *real-of-rat-poly* *ps*)

= *sign-changes-number-rootat* *ps* (**is** ?*l* = ?*r*)
 ⟨*proof*⟩

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*
 ($\lambda a b. \text{sign-changes-rat } ps a - \text{sign-changes-rat } ps b + (\text{if } \text{poly } rp a = 0 \text{ then } 1 \text{ else } 0)$)
 ($\lambda a. \text{sign-changes-neg-number-rootat } ps - \text{sign-changes-rat } ps a$)
 ⟨*proof*⟩

lemma *count-roots-rat-code*[*code*]:

count-roots-rat *p* = (*let* *rp* = *map-poly* *rat-of-int* *p* in if *p* = 0 then 0 else let *ps*
 = *sturm-rat* *rp*
 in *sign-changes-neg-number-rootat* *ps* - *sign-changes-number-rootat* *ps*)
 ⟨*proof*⟩

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

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

definition *root-info* :: *int* *poly* \Rightarrow *root-info* **where**

root-info *p* = (*if* *degree* *p* = 1 then
 (*let* *x* = *Rat.Fract* (- *coeff* *p* 0) (*coeff* *p* 1)
 in *Root-Info* ($\lambda l r. \text{if } l \leq x \wedge x \leq r \text{ then } 1 \text{ else } 0$) ($\lambda b. \text{if } x \leq b \text{ then } 1 \text{ else } 0$)) else

```

    (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)
      ( $\lambda$  a. sign-changes-neg-number-rootat ps - 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>

```

end

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)  $\neq$  0  $\longleftrightarrow$ 
  degree (gcd (of-int-poly q :: 'a :: {field-char-0, field-gcd} poly) (of-int-poly r))  $\neq$  0
  <proof>

```

```

definition factors-of-int-poly :: int poly  $\Rightarrow$  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>

```

```

lemma factors-of-int-poly:
  defines rp  $\equiv$  ipoly :: int poly  $\Rightarrow$  'a :: {field-gcd,field-char-0}  $\Rightarrow$  'a
  assumes factors-of-int-poly p = qs
  shows  $\bigwedge$  q. q  $\in$  set qs  $\implies$  irreducible q  $\wedge$  lead-coeff q > 0  $\wedge$  degree q  $\leq$  degree
  p  $\wedge$  degree q  $\neq$  0
  p  $\neq$  0  $\implies$  rp p x = 0  $\longleftrightarrow$  ( $\exists$  q  $\in$  set qs. rp q x = 0)
  p  $\neq$  0  $\implies$  rp p x = 0  $\implies$   $\exists!$  q  $\in$  set qs. rp q x = 0
  distinct qs
  <proof>

```

```

lemma factors-int-poly-represents:

```

fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes $p: p \text{ represents } x$
shows $\exists q \in \text{set } (\text{factors-of-int-poly } p).$
 $q \text{ represents } x \wedge \text{irreducible } q \wedge \text{lead-coeff } q > 0 \wedge \text{degree } q \leq \text{degree } p$
 $\langle \text{proof} \rangle$

corollary *irreducible-represents-imp-degree*:
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes *irreducible* f **and** $f \text{ represents } x$ **and** $g \text{ represents } x$
shows $\text{degree } f \leq \text{degree } g$
 $\langle \text{proof} \rangle$

lemma *irreducible-preservation*:
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes *irr*: *irreducible* p
and $x: p \text{ represents } x$
and $y: q \text{ represents } y$
and $\text{deg}: \text{degree } p \geq \text{degree } q$
and $f: \bigwedge q. q \text{ represents } y \implies (f \ q) \text{ represents } x \wedge \text{degree } (f \ q) \leq \text{degree } q$
and $\text{pr}: \text{primitive } q$
shows *irreducible* q
 $\langle \text{proof} \rangle$

declare *irreducible-const-poly-iff* [*simp*]

lemma *poly-uminus-irreducible*:
assumes $p: \text{irreducible } (p :: \text{int poly})$ **and** $\text{deg}: \text{degree } p \neq 0$
shows *irreducible* $(\text{poly-uminus } p)$
 $\langle \text{proof} \rangle$

lemma *reflect-poly-irreducible*:
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes $p: \text{irreducible } p$ **and** $x: p \text{ represents } x$ **and** $x0: x \neq 0$
shows *irreducible* $(\text{reflect-poly } p)$
 $\langle \text{proof} \rangle$

lemma *poly-add-rat-irreducible*:
assumes $p: \text{irreducible } p$ **and** $\text{deg}: \text{degree } p \neq 0$
shows *irreducible* $(\text{cf-pos-poly } (\text{poly-add-rat } r \ p))$
 $\langle \text{proof} \rangle$

lemma *poly-mult-rat-irreducible*:
assumes $p: \text{irreducible } p$ **and** $\text{deg}: \text{degree } p \neq 0$ **and** $r: r \neq 0$
shows *irreducible* $(\text{cf-pos-poly } (\text{poly-mult-rat } r \ p))$
 $\langle \text{proof} \rangle$

interpretation *coeff-lift-hom*:
 $\text{factor-preserving-hom } \text{coeff-lift} :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$
 $\implies -$

<proof>

end

8 The minimal polynomial of an algebraic number

theory *Min-Int-Poly*

imports

Algebraic-Numbers-Prelim

begin

Given an algebraic number x in a field, the minimal polynomial is the unique irreducible integer polynomial with positive leading coefficient that has x as a root.

Note that we assume characteristic 0 since the material upon which all of this builds also assumes it.

definition *min-int-poly* :: 'a :: field-char-0 \Rightarrow int poly **where**

min-int-poly $x =$

(if algebraic x then *THE* p . p represents $x \wedge$ irreducible $p \wedge$ lead-coeff $p > 0$
else $[:0, 1:]$)

lemma

fixes $x :: 'a :: \{field-char-0, field-gcd\}$

shows *min-int-poly-represents* [intro]: algebraic $x \implies$ *min-int-poly* x represents x

and *min-int-poly-irreducible* [intro]: irreducible (*min-int-poly* x)

and *lead-coeff-min-int-poly-pos*: lead-coeff (*min-int-poly* x) > 0

<proof>

lemma

fixes $x :: 'a :: \{field-char-0, field-gcd\}$

shows *degree-min-int-poly-pos* [intro]: degree (*min-int-poly* x) > 0

and *degree-min-int-poly-nonzero* [simp]: degree (*min-int-poly* x) $\neq 0$

<proof>

lemma *min-int-poly-primitive* [intro]:

fixes $x :: 'a :: \{field-char-0, field-gcd\}$

shows *primitive* (*min-int-poly* x)

<proof>

lemma *min-int-poly-content* [simp]:

fixes $x :: 'a :: \{field-char-0, field-gcd\}$

shows *content* (*min-int-poly* x) = 1

<proof>

lemma *ipoly-min-int-poly* [simp]:

algebraic $x \implies$ *ipoly* (*min-int-poly* x) ($x :: 'a :: \{field-gcd, field-char-0\}$) = 0

<proof>

lemma *min-int-poly-nonzero* [*simp*]:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 shows $\text{min-int-poly } x \neq 0$
 <proof>

lemma *min-int-poly-normalize* [*simp*]:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 shows $\text{normalize } (\text{min-int-poly } x) = \text{min-int-poly } x$
 <proof>

lemma *min-int-poly-prime-elem* [*intro*]:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 shows $\text{prime-elem } (\text{min-int-poly } x)$
 <proof>

lemma *min-int-poly-prime* [*intro*]:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 shows $\text{prime } (\text{min-int-poly } x)$
 <proof>

lemma *min-int-poly-unique*:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 assumes p represents x irreducible p lead-coeff $p > 0$
 shows $\text{min-int-poly } x = p$
 <proof>

lemma *min-int-poly-of-int* [*simp*]:
 $\text{min-int-poly } (\text{of-int } n :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}) = [:-\text{of-int } n, 1:]$
 <proof>

lemma *min-int-poly-of-nat* [*simp*]:
 $\text{min-int-poly } (\text{of-nat } n :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}) = [:-\text{of-nat } n, 1:]$
 <proof>

lemma *min-int-poly-0* [*simp*]: $\text{min-int-poly } (0 :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}) = [:-0, 1:]$
 <proof>

lemma *min-int-poly-1* [*simp*]: $\text{min-int-poly } (1 :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}) = [:-1, 1:]$
 <proof>

lemma *poly-min-int-poly-0-eq-0-iff* [*simp*]:
 fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
 assumes algebraic x
 shows $\text{poly } (\text{min-int-poly } x) 0 = 0 \longleftrightarrow x = 0$
 <proof>

lemma *min-int-poly-eqI*:
fixes $x :: 'a :: \{\text{field-char-0}, \text{field-gcd}\}$
assumes p represents x irreducible p lead-coeff $p \geq 0$
shows $\text{min-int-poly } x = p$
 $\langle \text{proof} \rangle$

Implementation for real and rational numbers

lemma *min-int-poly-of-rat*: $\text{min-int-poly } (\text{of-rat } r :: 'a :: \{\text{field-char-0}, \text{field-gcd}\})$
 $= \text{poly-rat } r$
 $\langle \text{proof} \rangle$

definition *min-int-poly-real* $:: \text{real} \Rightarrow \text{int poly}$ **where**
 $[\text{simp}]$: $\text{min-int-poly-real} = \text{min-int-poly}$

lemma *min-int-poly-real-code-unfold* $[\text{code-unfold}]$: $\text{min-int-poly} = \text{min-int-poly-real}$
 $\langle \text{proof} \rangle$

lemma *min-int-poly-real-basic-impl* $[\text{code}]$: $\text{min-int-poly-real } (\text{real-of-rat } x) = \text{poly-rat } x$
 $\langle \text{proof} \rangle$

lemma *min-int-poly-rat-code-unfold* $[\text{code-unfold}]$: $\text{min-int-poly} = \text{poly-rat}$
 $\langle \text{proof} \rangle$

end

9 Algebraic Numbers – Preliminary Implementation

This theory gathers some preliminary results to implement algebraic numbers, e.g., it defines an invariant to have unique representing polynomials and shows that polynomials for unary minus and inversion preserve this invariant.

theory *Algebraic-Numbers-Pre-Impl*

imports

Abstract-Rewriting.SN-Order-Carrier

Deriving.Compare-Rat

Deriving.Compare-Real

Jordan-Normal-Form.Gauss-Jordan-Array-Impl

Algebraic-Numbers

Sturm-Rat

Factors-of-Int-Poly

Min-Int-Poly

begin

For algebraic numbers, it turned out that *gcd-int-poly* is not preferable to the default implementation of *gcd*, which just implements Collin’s primitive

remainder sequence.

declare *gcd-int-poly-code*[*code-unfold del*]

lemma *ex1-imp-Collect-singleton*: $(\exists!x. P x) \wedge P x \longleftrightarrow \text{Collect } P = \{x\}$
<proof>

lemma *ex1-Collect-singleton*[*consumes 2*]:
assumes $\exists!x. P x$ **and** $P x$ **and** $\text{Collect } P = \{x\} \implies$ *thesis* **shows** *thesis*
<proof>

lemma *ex1-iff-Collect-singleton*: $P x \implies (\exists!x. P x) \longleftrightarrow \text{Collect } P = \{x\}$
<proof>

context

fixes *f*

assumes *bij*: *bij f*

begin

lemma *bij-imp-ex1-iff*: $(\exists!x. P (f x)) \longleftrightarrow (\exists!y. P y)$ (**is** *?l = ?r*)
<proof>

lemma *bij-ex1-imp-the-shift*:

assumes *ex1*: $\exists!y. P y$ **shows** $(\text{THE } x. P (f x)) = \text{Hilbert-Choice.inv } f (\text{THE } y. P y)$ (**is** *?l = ?r*)
<proof>

lemma *bij-imp-Collect-image*: $\{x. P (f x)\} = \text{Hilbert-Choice.inv } f \{y. P y\}$ (**is** *?l = ?g ' -*)
<proof>

lemma *bij-imp-card-image*: $\text{card } (f \text{ ' } X) = \text{card } X$
<proof>

end

definition *poly-cond* :: *int poly* \Rightarrow *bool* **where**

poly-cond p = $(\text{lead-coeff } p > 0 \wedge \text{irreducible } p)$

lemma *poly-condI*[*intro*]:

assumes $\text{lead-coeff } p > 0$ **and** *irreducible p* **shows** *poly-cond p* *<proof>*

lemma *poly-condD*:

assumes *poly-cond p*

shows *irreducible p* **and** $\text{lead-coeff } p > 0$ **and** *root-free p* **and** *square-free p* **and** $p \neq 0$
<proof>

lemma *poly-condE*[*elim*]:

assumes *poly-cond p*

and $\text{irreducible } p \implies \text{lead-coeff } p > 0 \implies \text{root-free } p \implies \text{square-free } p \implies p \neq 0 \implies \text{thesis}$
shows *thesis*
 ⟨*proof*⟩

lemma $\text{poly-cond-abs-int-poly[simp]}$: $\text{irreducible } p \implies \text{poly-cond } (\text{abs-int-poly } p)$
 ⟨*proof*⟩

definition $\text{poly-uminus-abs} :: \text{int poly} \Rightarrow \text{int poly}$ **where**
 $\text{poly-uminus-abs } p = \text{abs-int-poly } (\text{poly-uminus } p)$

lemma $\text{irreducible-poly-uminus[simp]}$: $\text{irreducible } p \implies \text{irreducible } (\text{poly-uminus } (p :: \text{int poly}))$
 ⟨*proof*⟩

lemma $\text{irreducible-poly-uminus-abs[simp]}$: $\text{irreducible } p \implies \text{irreducible } (\text{poly-uminus-abs } p)$
 ⟨*proof*⟩

lemma $\text{poly-cond-poly-uminus-abs[simp]}$: $\text{poly-cond } p \implies \text{poly-cond } (\text{poly-uminus-abs } p)$
 ⟨*proof*⟩

lemma $\text{ipoly-poly-uminus-abs-zero[simp]}$: $\text{ipoly } (\text{poly-uminus-abs } p) (x :: 'a :: \text{idom}) = 0 \iff \text{ipoly } p (-x) = 0$
 ⟨*proof*⟩

lemma $\text{degree-poly-uminus-abs[simp]}$: $\text{degree } (\text{poly-uminus-abs } p) = \text{degree } p$
 ⟨*proof*⟩

definition $\text{poly-inverse} :: \text{int poly} \Rightarrow \text{int poly}$ **where**
 $\text{poly-inverse } p = \text{abs-int-poly } (\text{reflect-poly } p)$

lemma $\text{irreducible-poly-inverse[simp]}$: $\text{coeff } p 0 \neq 0 \implies \text{irreducible } p \implies \text{irreducible } (\text{poly-inverse } p)$
 ⟨*proof*⟩

lemma $\text{degree-poly-inverse[simp]}$: $\text{coeff } p 0 \neq 0 \implies \text{degree } (\text{poly-inverse } p) = \text{degree } p$
 ⟨*proof*⟩

lemma $\text{ipoly-poly-inverse[simp]}$: **assumes** $\text{coeff } p 0 \neq 0$
shows $\text{ipoly } (\text{poly-inverse } p) (x :: 'a :: \text{field-char-0}) = 0 \iff \text{ipoly } p (\text{inverse } x) = 0$
 ⟨*proof*⟩

lemma $\text{ipoly-roots-finite}$: $p \neq 0 \implies \text{finite } \{x :: 'a :: \{\text{idom}, \text{ring-char-0}\}. \text{ipoly } p x = 0\}$

<proof>

lemma *root-sign-change*: **assumes**

p0: *poly* (*p*::*real poly*) *x* = 0 **and**

pd-ne0: *poly* (*pderiv p*) *x* ≠ 0

obtains *d* **where**

$0 < d$

$\text{sgn } (\text{poly } p \ (x - d)) \neq \text{sgn } (\text{poly } p \ (x + d))$

$\text{sgn } (\text{poly } p \ (x - d)) \neq 0$

$0 \neq \text{sgn } (\text{poly } p \ (x + d))$

$\forall d' > 0. d' \leq d \longrightarrow \text{sgn } (\text{poly } p \ (x + d')) = \text{sgn } (\text{poly } p \ (x + d)) \wedge \text{sgn } (\text{poly } p \ (x - d')) = \text{sgn } (\text{poly } p \ (x - d))$

<proof>

lemma *gt-rat-sign-change-square-free*:

assumes *ur*: $\exists! x. \text{root-cond } \text{plr } x$

and *plr*[*simp*]: *plr* = (*p*,*l*,*r*)

and *sf*: *square-free p* **and** *in-interval*: $l \leq y \leq r$

and *py0*: *ipoly p y* ≠ 0 **and** *pr0*: *ipoly p r* ≠ 0

shows $(\text{sgn } (\text{ipoly } p \ y) = \text{sgn } (\text{ipoly } p \ r)) = (\text{of-rat } y > (\text{THE } x. \text{root-cond } \text{plr } x))$ (**is** ?*gt* = -)

<proof>

definition *algebraic-real* :: *real* ⇒ *bool* **where**

[*simp*]: *algebraic-real* = *algebraic*

lemma *algebraic-real-iff*[*code-unfold*]: *algebraic* = *algebraic-real* *<proof>*

end

10 Cauchy's Root Bound

This theory contains a formalization of Cauchy's root bound, i.e., given an integer polynomial it determines a bound *b* such that all real or complex roots of the polynomials have a norm below *b*.

theory *Cauchy-Root-Bound*

imports

Algebraic-Numbers-Pre-Impl

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* \Rightarrow *rat* **where**

root-bound *p* \equiv *let*

n = *degree* *p*;

m = 1 + *div-ceiling* (*max-list-non-empty* (*map* ($\lambda i.$ *abs* (*coeff* *p* *i*)) [0..*n*]))
(abs (*lead-coeff* *p*))

— round to the next higher number $2^{\wedge}n$, so that bisection will

— stay on integers for as long as possible

in of-int ($2^{\wedge}(\log\text{-ceiling } 2\ m)$)

lemma *root-imp-deg-nonzero*: **assumes** *p* $\neq 0$ *poly* *p* *x* = 0

shows *degree* *p* $\neq 0$

<proof>

lemma *cauchy-root-bound*: **fixes** *x* :: 'a :: *real-normed-field*

assumes *x*: *poly* *p* *x* = 0 **and** *p*: *p* $\neq 0$

shows *norm* *x* $\leq 1 + \max\text{-list-non-empty}$ (*map* ($\lambda i.$ *norm* (*coeff* *p* *i*)) [0..*degree* *p*])

/ *norm* (*lead-coeff* *p*) (**is** - \leq - + ?*max* / ?*nlc*)

<proof>

lemma *div-le-div-ceiling*: *x* *div* *y* \leq *div-ceiling* *x* *y*

<proof>

lemma *div-ceiling*: **assumes** *q*: *q* $\neq 0$

shows (*of-int* *x* :: 'a :: *floor-ceiling*) / *of-int* *q* \leq *of-int* (*div-ceiling* *x* *q*)

<proof>

lemma *max-list-non-empty-map*: **assumes** *hom*: $\bigwedge x\ y.$ *max* (*f* *x*) (*f* *y*) = *f* (*max* *x* *y*)

shows *xs* $\neq [] \implies \max\text{-list-non-empty}$ (*map* *f* *xs*) = *f* (*max-list-non-empty* *xs*)

<proof>

lemma *root-bound*: **assumes** *root-bound* *p* = *B* **and** *deg*: *degree* *p* > 0

shows *ipoly* *p* (*x* :: 'a :: *real-normed-field*) = 0 $\implies \text{norm } x \leq \text{of-rat } B\ B \geq 0$

<proof>

end

11 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 equiva-

lence.

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
  Algebraic-Numbers-Pre-Impl
begin
```

```
lemma ex1-imp-Collect-singleton:  $(\exists!x. P x) \wedge P x \longleftrightarrow \text{Collect } P = \{x\}$ 
<proof>
```

```
lemma ex1-Collect-singleton[consumes 2]:
  assumes  $\exists!x. P x$  and  $P x$  and  $\text{Collect } P = \{x\} \implies \text{thesis}$  shows thesis
<proof>
```

```
lemma ex1-iff-Collect-singleton:  $P x \implies (\exists!x. P x) \longleftrightarrow \text{Collect } P = \{x\}$ 
<proof>
```

```
lemma bij-imp-card: assumes bij: bij f shows  $\text{card } \{x. P (f x)\} = \text{card } \{x. P x\}$ 
<proof>
```

```
lemma bij-add: bij  $(\lambda x. x + y :: 'a :: \text{group-add})$  (is ?g1)
  and bij-minus: bij  $(\lambda x. x - y :: 'a)$  (is ?g2)
  and inv-add[simp]: Hilbert-Choice.inv  $(\lambda x. x + y) = (\lambda x. x - y)$  (is ?g3)
  and inv-minus[simp]: Hilbert-Choice.inv  $(\lambda x. x - y) = (\lambda x. x + y)$  (is ?g4)
<proof>
```

```
lemmas ex1-shift[simp] = bij-imp-ex1-iff[OF bij-add] bij-imp-ex1-iff[OF bij-minus]
```

```
lemma ex1-the-shift:
  assumes ex1:  $\exists!y :: 'a :: \text{group-add}. P y$ 
  shows  $(\text{THE } x. P (x + d)) = (\text{THE } y. P y) - d$ 
  and  $(\text{THE } x. P (x - d)) = (\text{THE } y. P y) + d$ 
<proof>
```

```
lemma card-shift-image[simp]:
  shows  $\text{card } ((\lambda x :: 'a :: \text{group-add}. x + d) ` X) = \text{card } X$ 
  and  $\text{card } ((\lambda x. x - d) ` X) = \text{card } X$ 
<proof>
```

```
lemma irreducible-root-free:
  fixes p :: 'a :: {idom,comm-ring-1} poly
  assumes irr: irreducible p shows root-free p
<proof>
```

11.1 Real Algebraic Numbers – Innermost Layer

We represent a real algebraic number α by a tuple (p,l,r) : α 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. $(\dots,x-3, 3,3)$ is equivalent to $(\dots,x-3,2,10)$.

11.1.1 Basic Definitions

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

fun $poly\text{-}real\text{-}alg\text{-}1 :: real\text{-}alg\text{-}1 \Rightarrow int\ poly$ **where** $poly\text{-}real\text{-}alg\text{-}1\ (p,-,-) = p$

fun $rai\text{-}ub :: real\text{-}alg\text{-}1 \Rightarrow rat$ **where** $rai\text{-}ub\ (-,-,r) = r$

fun $rai\text{-}lb :: real\text{-}alg\text{-}1 \Rightarrow rat$ **where** $rai\text{-}lb\ (-,l,-) = l$

abbreviation $roots\text{-}below\ p\ x \equiv \{y :: real. y \leq x \wedge ipoly\ p\ y = 0\}$

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

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

abbreviation $real\text{-}of\text{-}1$ **where** $real\text{-}of\text{-}1 \equiv the\text{-}unique\text{-}root$

lemma $root\text{-}condI$ [*intro*]:

assumes $of\text{-}rat\ (rai\text{-}lb\ plr) \leq x$ **and** $x \leq of\text{-}rat\ (rai\text{-}ub\ plr)$ **and** $ipoly\ (poly\text{-}real\text{-}alg\text{-}1\ plr)\ x = 0$

shows $root\text{-}cond\ plr\ x$

<proof>

lemma $root\text{-}condE$ [*elim*]:

assumes $root\text{-}cond\ plr\ x$

and $of\text{-}rat\ (rai\text{-}lb\ plr) \leq x \Longrightarrow x \leq of\text{-}rat\ (rai\text{-}ub\ plr) \Longrightarrow ipoly\ (poly\text{-}real\text{-}alg\text{-}1\ plr)\ x = 0 \Longrightarrow thesis$

shows $thesis$

<proof>

lemma

assumes $ur: unique\text{-}root\ plr$

defines $x \equiv the\text{-}unique\text{-}root\ plr$ **and** $p \equiv poly\text{-}real\text{-}alg\text{-}1\ plr$ **and** $l \equiv rai\text{-}lb\ plr$
and $r \equiv rai\text{-}ub\ plr$

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

$x = y \iff root\text{-}cond\ plr\ y \wedge y = x \iff root\text{-}cond\ plr\ y$

and $the\text{-}unique\text{-}root\text{-}eqI: root\text{-}cond\ plr\ y \Longrightarrow y = x \wedge root\text{-}cond\ plr\ y \Longrightarrow x = y$
<proof>

lemma *unique-rootE*:

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 $r \equiv \text{rai-ub } plr$

assumes *main*: $\text{of-rat } l \leq x \implies x \leq \text{of-rat } r \implies \text{ipoly } p \ x = 0 \implies \text{root-cond } plr \ x \implies$

$(\bigwedge y. x = y \longleftrightarrow \text{root-cond } plr \ y) \implies (\bigwedge y. y = x \longleftrightarrow \text{root-cond } plr \ y) \implies$

thesis

shows *thesis* $\langle \text{proof} \rangle$

lemma *unique-rootI*:

assumes $\bigwedge y. \text{root-cond } plr \ y \implies x = y \text{ root-cond } plr \ x$

shows *unique-root plr* $\langle \text{proof} \rangle$

definition *invariant-1* :: *real-alg-1* \Rightarrow *bool* **where**

invariant-1 tup \equiv *case tup of* $(p,l,r) \Rightarrow$

unique-root $(p,l,r) \wedge \text{sgn } l = \text{sgn } r \wedge \text{poly-cond } p$

lemma *invariant-1I*:

assumes *unique-root plr* **and** $\text{sgn}(\text{rai-lb } plr) = \text{sgn}(\text{rai-ub } plr)$ **and** *poly-cond*
(poly-real-alg-1 plr)

shows *invariant-1 plr*

$\langle \text{proof} \rangle$

lemma

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 $r \equiv \text{rai-ub } plr$

shows *invariant-1D*: *root-cond plr x*

$\text{sgn } l = \text{sgn } r \ \text{sgn } x = \text{of-rat}(\text{sgn } r) \ \text{unique-root } plr \ \text{poly-cond } p \ \text{degree } p > 0$
primitive p

and *invariant-1-root-cond*: $\bigwedge y. \text{root-cond } plr \ y \longleftrightarrow y = x$

$\langle \text{proof} \rangle$

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 plr x* \implies

$\text{sgn } l = \text{sgn } r \implies \text{sgn } x = \text{of-rat}(\text{sgn } r) \implies \text{unique-root } plr \implies \text{poly-cond } p$
 $\implies \text{degree } p > 0 \implies$

primitive p \implies *thesis*

shows *thesis* $\langle \text{proof} \rangle$

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** $\text{sgn } l = \text{sgn } r$

and *ur: unique-root plr*
and *poly-cond p*
shows *invariant-1 plr \wedge real-of-1 plr = x*
<proof>

lemma *real-of-1-0:*

assumes *invariant-1 (p,l,r)*
shows [*simp*]: *the-unique-root (p,l,r) = 0 \longleftrightarrow r = 0*
and [*dest*]: *l = 0 \implies r = 0*
and [*intro*]: *r = 0 \implies l = 0*
<proof>

lemma *invariant-1-pos: assumes rc: invariant-1 (p,l,r)*

shows [*simp*]: *the-unique-root (p,l,r) > 0 \longleftrightarrow r > 0 (is ?x > 0 \longleftrightarrow -)*
and [*simp*]: *the-unique-root (p,l,r) < 0 \longleftrightarrow r < 0*
and [*simp*]: *the-unique-root (p,l,r) \leq 0 \longleftrightarrow r \leq 0*
and [*simp*]: *the-unique-root (p,l,r) \geq 0 \longleftrightarrow r \geq 0*
and [*intro*]: *r > 0 \implies l > 0*
and [*dest*]: *l > 0 \implies r > 0*
and [*intro*]: *r < 0 \implies l < 0*
and [*dest*]: *l < 0 \implies r < 0*
<proof>

definition *invariant-1-2 where*

invariant-1-2 rai \equiv invariant-1 rai \wedge degree (poly-real-alg-1 rai) > 1

definition *poly-cond2 where poly-cond2 p \equiv poly-cond p \wedge degree p > 1*

lemma *poly-cond2I[intro!]: poly-cond p \implies degree p > 1 \implies poly-cond2 p <proof>*

lemma *poly-cond2D:*

assumes *poly-cond2 p*
shows *poly-cond p and degree p > 1 <proof>*

lemma *poly-cond2E[elim!]:*

assumes *poly-cond2 p and poly-cond p \implies degree p > 1 \implies thesis shows thesis*
<proof>

lemma *invariant-1-2-poly-cond2: invariant-1-2 rai \implies poly-cond2 (poly-real-alg-1 rai)*

<proof>

lemma *invariant-1-2I[intro!]:*

assumes *invariant-1 rai and degree (poly-real-alg-1 rai) > 1 shows invariant-1-2 rai*
<proof>

lemma *invariant-1-2E[elim!]:*

assumes *invariant-1-2 rai*
and *invariant-1 rai* \implies *degree (poly-real-alg-1 rai) > 1* \implies *thesis*
shows *thesis* \langle *proof* \rangle

lemma *invariant-1-2-realI*:
fixes *plr :: real-alg-1*
defines *p* \equiv *poly-real-alg-1 plr* **and** *l* \equiv *rai-lb plr* **and** *r* \equiv *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* \wedge *real-of-1 plr = x*
 \langle *proof* \rangle

11.2 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 $x^2 - 2$.

11.2.1 Definitions and Algorithms on Raw Type

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

fun *invariant-2* :: *real-alg-2* \Rightarrow *bool* **where**
invariant-2 (Irrational n rai) = (*invariant-1-2 rai*
 \wedge *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* \Rightarrow *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* \Rightarrow *real-alg-2* **where**
[*code-unfold*]: *of-rat-2 = Rational*

lemma *of-rat-2*: *real-of-2 (of-rat-2 x)* = *of-rat x invariant-2 (of-rat-2 x)*
 \langle *proof* \rangle

typedef *real-alg-3* = *Collect invariant-2*
morphisms *rep-real-alg-3 Real-Alg-Invariant*
 \langle *proof* \rangle

setup-lifting *type-definition-real-alg-3*

lift-definition *real-of-3* :: *real-alg-3* \Rightarrow *real* **is** *real-of-2* \langle *proof* \rangle

11.2.2 Definitions and Algorithms on Quotient Type

quotient-type *real-alg* = *real-alg-3* / $\lambda x y. \text{real-of-3 } x = \text{real-of-3 } y$
morphisms *rep-real-alg* *Real-Alg-Quotient*
 \langle *proof* \rangle

lift-definition *real-of* :: *real-alg* \Rightarrow *real* **is** *real-of-3* \langle *proof* \rangle

lemma *real-of-inj*: $(\text{real-of } x = \text{real-of } y) = (x = y)$
 \langle *proof* \rangle

11.2.3 Sign

definition *sgn-1* :: *real-alg-1* \Rightarrow *rat* **where**
sgn-1 *x* = *sgn* (*rai-ub* *x*)

lemma *sgn-1*: *invariant-1* *x* \Longrightarrow *real-of-rat* (*sgn-1* *x*) = *sgn* (*real-of-1* *x*)
 \langle *proof* \rangle

lemma *sgn-1-inj*: *invariant-1* *x* \Longrightarrow *invariant-1* *y* \Longrightarrow *real-of-1* *x* = *real-of-1* *y* \Longrightarrow
sgn-1 *x* = *sgn-1* *y*
 \langle *proof* \rangle

11.2.4 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*)
 \langle *proof* \rangle

locale *map-poly-zero-hom-0* = *base*: *zero-hom-0*

begin

sublocale *zero-hom-0* *map-poly* *hom* \langle *proof* \rangle

end

interpretation *of-int-poly-hom*:

map-poly-zero-hom-0 *of-int* :: *int* \Rightarrow 'a :: {*ring-1*, *ring-char-0*} \langle *proof* \rangle

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* = -)
 \langle *proof* \rangle

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*⟩

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, r)
⟨*proof*⟩

lemma *rational-root-free-degree-iff*: **assumes** *rf*: *root-free* (*map-poly rat-of-int* p)
and *rt*: *ipoly* $p \ x = 0$
shows ($x \in \mathbb{Q}$) = (*degree* $p = 1$)
⟨*proof*⟩

lemma *rational-poly-cond-iff*: **assumes** *poly-cond* p **and** *ipoly* $p \ x = 0$ **and** *degree* $p > 1$
shows ($x \in \mathbb{Q}$) = (*degree* $p = 1$)
⟨*proof*⟩

lemma *poly-cond-degree-gt-1*: **assumes** *poly-cond* p *degree* $p > 1$ *ipoly* $p \ x = 0$
shows $x \notin \mathbb{Q}$
⟨*proof*⟩

lemma *poly-cond2-no-rat-root*: **assumes** *poly-cond2* p
shows *ipoly* p (*real-of-rat* x) $\neq 0$
⟨*proof*⟩

context
fixes $p :: \text{int poly}$
and $x :: \text{rat}$
begin

lemma *gt-rat-sign-change*:
assumes *ur*: *unique-root* plr
defines $p \equiv \text{poly-real-alg-1 } plr$ **and** $l \equiv \text{rai-lb } plr$ **and** $r \equiv \text{rai-ub } plr$
assumes p : *poly-cond2* p **and** *in-interval*: $l \leq y \ y \leq r$
shows (*sgn* (*ipoly* $p \ y$) = *sgn* (*ipoly* $p \ r$)) = (*of-rat* $y > \text{the-unique-root } plr$)
⟨*proof*⟩

definition *tighten-poly-bounds* :: $\text{rat} \Rightarrow \text{rat} \Rightarrow \text{rat} \Rightarrow \text{rat} \times \text{rat} \times \text{rat}$ **where**
tighten-poly-bounds $l \ r \ sr = (\text{let } m = (l + r) / 2; \ sm = \text{sgn} (\text{ipoly } p \ m) \ \text{in}$
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*)) $l \leq l' \ l' \leq r' \ r' \leq r$
 $(r' - l') = (r - l) / 2 \ sr' = \text{sgn} \ (\text{ipoly } p \ r')$
<proof>

partial-function (*tailrec*) *tighten-poly-bounds-epsilon* :: *rat* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *rat* \times *rat* \times *rat* **where**
[code]: *tighten-poly-bounds-epsilon* *l r sr* = (*if* $r - l \leq x$ *then* (*l,r,sr*) *else*
(*case* *tighten-poly-bounds* *l r sr* *of* (*l',r',sr'*) \Rightarrow *tighten-poly-bounds-epsilon* *l' r' sr'*)
sr')

partial-function (*tailrec*) *tighten-poly-bounds-for-x* :: *rat* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *rat* \times *rat* \times *rat* **where**
[code]: *tighten-poly-bounds-for-x* *l r sr* = (*if* $x < l \vee r < x$ *then* (*l, r, sr*) *else*
(*case* *tighten-poly-bounds* *l r sr* *of* (*l',r',sr'*) \Rightarrow *tighten-poly-bounds-for-x* *l' r' sr'*)
sr')

lemma *tighten-poly-bounds-epsilon*:
assumes *ur*: *unique-root* (*p,l,r*)
defines *u*: *u* \equiv *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 $l \leq l' \ r' \leq r$ *root-cond* (*p,l',r'*) $u \ r' - l' \leq x \ sr' = \text{sgn} \ (\text{ipoly } p \ r')$
<proof>

lemma *tighten-poly-bounds-for-x*:
assumes *ur*: *unique-root* (*p,l,r*)
defines *u*: *u* \equiv *the-unique-root* (*p,l,r*)
assumes *p*: *poly-cond2* *p*
and *res*: *tighten-poly-bounds-for-x* *l r sr* = (*l',r',sr'*)
and *sr*: *sr* = *sgn* (*ipoly p r*)
shows $l \leq l' \ l' \leq r' \ r' \leq r$ *root-cond* (*p,l',r'*) $u \ \neg \ (l' \leq x \wedge x \leq r') \ sr' = \text{sgn} \ (\text{ipoly } p \ r')$ *unique-root* (*p,l',r'*)
<proof>
end

definition *real-alg-precision* :: *rat* **where**
real-alg-precision \equiv *Rat.Fract* 1 2

lemma *real-alg-precision*: *real-alg-precision* > 0
<proof>

definition *normalize-bounds-1-main* :: *rat* \Rightarrow *real-alg-1* \Rightarrow *real-alg-1* **where**
normalize-bounds-1-main *eps rai* = (*case* *rai* *of* (*p,l,r*) \Rightarrow

let $(l', r', sr') = \text{tighten-poly-bounds-epsilon } p \text{ eps } l \ r \ (\text{sgn } (\text{ipoly } p \ r));$
 $fr = \text{rat-of-int } (\text{floor } r');$
 $(l'', r'', -) = \text{tighten-poly-bounds-for-x } p \text{ fr } l' \ r' \ sr'$
 in (p, l'', r'')

definition *normalize-bounds-1* :: *real-alg-1* \Rightarrow *real-alg-1* **where**
normalize-bounds-1 = (*normalize-bounds-1-main* *real-alg-precision*)

context

fixes $p \ q$ **and** $l \ r :: \text{rat}$

assumes *cong*: $\bigwedge x. \text{real-of-rat } l \leq x \Rightarrow x \leq \text{of-rat } r \Rightarrow (\text{ipoly } p \ x = (0 :: \text{real})) = (\text{ipoly } q \ x = 0)$

begin

lemma *root-cond-cong*: $\text{root-cond } (p, l, r) = \text{root-cond } (q, l, r)$

<proof>

lemma *the-unique-root-cong*:

the-unique-root $(p, l, r) = \text{the-unique-root } (q, l, r)$

<proof>

lemma *unique-root-cong*:

unique-root $(p, l, r) = \text{unique-root } (q, l, r)$

<proof>

end

lemma *normalize-bounds-1-main*: **assumes** *eps*: $\text{eps} > 0$ **and** *rc*: *invariant-1-2* x

defines $y: y \equiv \text{normalize-bounds-1-main } \text{eps } x$

shows *invariant-1-2* $y \wedge (\text{real-of-1 } y = \text{real-of-1 } x)$

<proof>

lemma *normalize-bounds-1*: **assumes** $x: \text{invariant-1-2 } x$

shows *invariant-1-2* $(\text{normalize-bounds-1 } x) \wedge (\text{real-of-1 } (\text{normalize-bounds-1 } x) = \text{real-of-1 } x)$

<proof>

lemma *normalize-bound-1-poly*: $\text{poly-real-alg-1 } (\text{normalize-bounds-1 } \text{rai}) = \text{poly-real-alg-1 } \text{rai}$

<proof>

definition *real-alg-2-main* :: *root-info* \Rightarrow *real-alg-1* \Rightarrow *real-alg-2* **where**

real-alg-2-main $\text{ri } \text{rai} \equiv \text{let } p = \text{poly-real-alg-1 } \text{rai}$

in (if $\text{degree } p = 1$ then *Rational* $(\text{Rat.Fract } (- \text{coeff } p \ 0) (\text{coeff } p \ 1))$)

else (case *normalize-bounds-1* rai of $(p', l, r) \Rightarrow$

Irrational $(\text{root-info.number-root } \text{ri } r) (p', l, r))$)

definition *real-alg-2* :: *real-alg-1* \Rightarrow *real-alg-2* **where**

real-alg-2 $\text{rai} \equiv \text{let } p = \text{poly-real-alg-1 } \text{rai}$

in (if $\text{degree } p = 1$ then *Rational* $(\text{Rat.Fract } (- \text{coeff } p \ 0) (\text{coeff } p \ 1))$)

else (case *normalize-bounds-1* rai of $(p', l, r) \Rightarrow$

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 \longleftrightarrow (x = real-of-rat (Rat.Fract (- coeff p 0) (coeff p 1)))*
(*proof*)

lemma *invariant-1-degree-0*:
assumes *inv: invariant-1 rai*
shows *degree (poly-real-alg-1 rai) \neq 0 (is degree ?p \neq 0)*
(*proof*)

lemma *real-alg-2-main*:
assumes *inv: invariant-1 rai*
defines [*simp*]: *p \equiv poly-real-alg-1 rai*
assumes *ric: irreducible (poly-real-alg-1 rai) \implies 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*)

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*)

lemma *invariant-2-realI*:
fixes *plr :: real-alg-1*
defines *p \equiv poly-real-alg-1 plr and l \equiv rai-lb plr and r \equiv 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) \wedge real-of-2 (real-alg-2 plr) = x*
(*proof*)

11.2.5 Comparisons

fun *compare-rat-1 :: rat \Rightarrow real-alg-1 \Rightarrow order where*
compare-rat-1 x (p,l,r) = (if x < l then Lt else if x > r then Gt else
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*)

lemma *cf-pos-0[simp]*: \neg *cf-pos 0*
(*proof*)

11.2.6 Negation

fun *uminus-1 :: real-alg-1 \Rightarrow 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:  $y \equiv \text{uminus-1 } x$ 
  shows  $\text{invariant-1 } y \wedge (\text{real-of-1 } y = - \text{real-of-1 } x)$ 
   $\langle \text{proof} \rangle$ 

lemma uminus-1-2:
  assumes x: invariant-1-2 x
  defines y:  $y \equiv \text{uminus-1 } x$ 
  shows  $\text{invariant-1-2 } y \wedge (\text{real-of-1 } y = - \text{real-of-1 } x)$ 
   $\langle \text{proof} \rangle$ 

fun uminus-2 :: real-alg-2  $\Rightarrow$  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  $\text{real-of-2 } (\text{uminus-2 } x) = \text{uminus } (\text{real-of-2 } x)$ 
  invariant-2 (uminus-2 x)
   $\langle \text{proof} \rangle$ 

declare uminus-1.simps[simp del]

lift-definition uminus-3 :: real-alg-3  $\Rightarrow$  real-alg-3 is uminus-2
   $\langle \text{proof} \rangle$ 

lemma uminus-3:  $\text{real-of-3 } (\text{uminus-3 } x) = - \text{real-of-3 } x$ 
   $\langle \text{proof} \rangle$ 

instantiation real-alg :: uminus
begin
lift-definition uminus-real-alg :: real-alg  $\Rightarrow$  real-alg is uminus-3
   $\langle \text{proof} \rangle$ 
instance  $\langle \text{proof} \rangle$ 
end

lemma uminus-real-alg:  $- (\text{real-of } x) = \text{real-of } (- x)$ 
   $\langle \text{proof} \rangle$ 

```

11.2.7 Inverse

```

fun inverse-1 :: real-alg-1  $\Rightarrow$  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  $\text{real-of-1 } \text{rai} \neq \text{of-rat } x$ 
   $\langle \text{proof} \rangle$ 

```

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*⟩

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)*
invariant-2 (inverse-2 x)
⟨*proof*⟩

lift-definition *inverse-3* :: *real-alg-3 ⇒ real-alg-3* **is** *inverse-2*
⟨*proof*⟩

lemma *inverse-3*: *real-of-3 (inverse-3 x) = inverse (real-of-3 x)*
⟨*proof*⟩

11.2.8 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*⟩

11.2.9 Generic Factorization and Bisection Framework

lemma *card-1-Collect-ex1*: **assumes** *card (Collect P) = 1*
shows $\exists! x. P x$
⟨*proof*⟩

fun *sub-interval* :: *rat × rat ⇒ rat × rat ⇒ bool* **where**
sub-interval (l,r) (l',r') = (l' ≤ l ∧ r ≤ r')

fun *in-interval* :: *rat × rat ⇒ real ⇒ bool* **where**
in-interval (l,r) x = (of-rat l ≤ x ∧ x ≤ of-rat r)

definition *converges-to* :: *(nat ⇒ rat × rat) ⇒ real ⇒ bool* **where**

$converges\text{-}to\ f\ x \equiv (\forall\ n.\ in\text{-}interval\ (f\ n)\ x \wedge\ sub\text{-}interval\ (f\ (Suc\ n))\ (f\ n))$
 $\wedge\ (\forall\ (eps :: real) > 0.\ \exists\ n\ l\ r.\ f\ n = (l,r) \wedge\ of\text{-}rat\ r - of\text{-}rat\ l \leq eps)$

context

fixes $bnd\text{-}update :: 'a \Rightarrow 'a$
and $bnd\text{-}get :: 'a \Rightarrow rat \times rat$
begin

definition $at\text{-}step :: (nat \Rightarrow rat \times rat) \Rightarrow nat \Rightarrow 'a \Rightarrow bool$ **where**

$at\text{-}step\ f\ n\ a \equiv \forall\ i.\ bnd\text{-}get\ ((bnd\text{-}update\ \overset{\sim}{\sim} i)\ a) = f\ (n + i)$

partial-function (*tailrec*) $select\text{-}correct\text{-}factor\text{-}main$

$:: 'a \Rightarrow (int\ poly \times root\text{-}info)list \Rightarrow (int\ poly \times root\text{-}info)list$
 $\Rightarrow rat \Rightarrow rat \Rightarrow nat \Rightarrow (int\ poly \times root\text{-}info) \times rat \times rat$ **where**
[*code*]: $select\text{-}correct\text{-}factor\text{-}main\ bnd\ todo\ old\ l\ r\ n = (case\ todo\ of\ Nil$
 $\Rightarrow\ if\ n = 1\ then\ (hd\ old,\ l,\ r)\ else\ let\ bnd' = bnd\text{-}update\ bnd\ in\ (case\ bnd\text{-}get$
 $bnd'\ of\ (l,r) \Rightarrow$
 $select\text{-}correct\text{-}factor\text{-}main\ bnd'\ old\ []\ l\ r\ 0)$
 $| Cons\ (p,ri)\ todo \Rightarrow let\ m = root\text{-}info.l\text{-}r\ ri\ l\ r\ in$
 $if\ m = 0\ then\ select\text{-}correct\text{-}factor\text{-}main\ bnd\ todo\ old\ l\ r\ n$
 $else\ select\text{-}correct\text{-}factor\text{-}main\ bnd\ todo\ ((p,ri) \# old)\ l\ r\ (n + m))$

definition $select\text{-}correct\text{-}factor :: 'a \Rightarrow (int\ poly \times root\text{-}info)list \Rightarrow$

$(int\ poly \times root\text{-}info) \times rat \times rat$ **where**

$select\text{-}correct\text{-}factor\ init\ polys = (case\ bnd\text{-}get\ init\ of\ (l,r) \Rightarrow$
 $select\text{-}correct\text{-}factor\text{-}main\ init\ polys\ []\ l\ r\ 0)$

lemma $select\text{-}correct\text{-}factor\text{-}main$: **assumes** $conv$: $converges\text{-}to\ f\ x$

and at : $at\text{-}step\ f\ i\ a$

and res : $select\text{-}correct\text{-}factor\text{-}main\ a\ todo\ old\ l\ r\ n = ((q,ri\text{-}fin),(l\text{-}fin,r\text{-}fin))$

and bnd : $bnd\text{-}get\ a = (l,r)$

and ri : $\bigwedge\ q\ ri.\ (q,ri) \in set\ todo \cup set\ old \implies root\text{-}info\text{-}cond\ ri\ q$

and $q0$: $\bigwedge\ q\ ri.\ (q,ri) \in set\ todo \cup set\ old \implies q \neq 0$

and ex : $\exists\ q.\ q \in fst\ 'set\ todo \cup fst\ 'set\ old \wedge\ ipoly\ q\ x = 0$

and $dist$: $distinct\ (map\ fst\ (todo\ @\ old))$

and old : $\bigwedge\ q\ ri.\ (q,ri) \in set\ old \implies root\text{-}info.l\text{-}r\ ri\ l\ r \neq 0$

and un : $\bigwedge\ x :: real.\ (\exists\ q.\ q \in fst\ 'set\ todo \cup fst\ 'set\ old \wedge\ ipoly\ q\ x = 0) \implies$

$\exists!\ q.\ q \in fst\ 'set\ todo \cup fst\ 'set\ old \wedge\ ipoly\ q\ x = 0$

and n : $n = sum\text{-}list\ (map\ (\lambda\ (q,ri).\ root\text{-}info.l\text{-}r\ ri\ l\ r)\ old)$

shows $unique\text{-}root\ (q,l\text{-}fin,r\text{-}fin) \wedge (q,ri\text{-}fin) \in set\ todo \cup set\ old \wedge x = the\text{-}unique\text{-}root$
 $(q,l\text{-}fin,r\text{-}fin)$

<proof>

lemma $select\text{-}correct\text{-}factor$: **assumes**

$conv$: $converges\text{-}to\ (\lambda\ i.\ bnd\text{-}get\ ((bnd\text{-}update\ \overset{\sim}{\sim} i)\ init))\ x$

and res : $select\text{-}correct\text{-}factor\ init\ polys = ((q,ri),(l,r))$

and ri : $\bigwedge\ q\ ri.\ (q,ri) \in set\ polys \implies root\text{-}info\text{-}cond\ ri\ q$

and $q0$: $\bigwedge\ q\ ri.\ (q,ri) \in set\ polys \implies q \neq 0$

and ex : $\exists\ q.\ q \in fst\ 'set\ polys \wedge\ ipoly\ q\ x = 0$

and *dist*: *distinct* (map *fst polys*)
and *un*: $\bigwedge x :: \text{real}. (\exists q. q \in \text{fst } ' \text{ set polys } \wedge \text{ipoly } q x = 0) \implies$
 $\exists ! q. q \in \text{fst } ' \text{ set polys } \wedge \text{ipoly } q x = 0$
shows *unique-root* $(q,l,r) \wedge (q,ri) \in \text{set polys } \wedge x = \text{the-unique-root } (q,l,r)$
<proof>

definition *real-alg-2'* :: *root-info* \Rightarrow *int poly* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *real-alg-2* **where**
`[code del]`: *real-alg-2'* *ri p l r* = (
 if *degree p* = 1 then *Rational* (*Rat.Fract* ($- \text{coeff } p 0$) ($\text{coeff } p 1$)) else
real-alg-2-main ri (*case tighten-poly-bounds-for-x p 0 l r* (*sgn* (*ipoly p r*)) of
 (l',r',sr') \Rightarrow (p, l', r'))

lemma *real-alg-2'-code*[code]: *real-alg-2'* *ri p l r* =
 (if *degree p* = 1 then *Rational* (*Rat.Fract* ($- \text{coeff } p 0$) ($\text{coeff } p 1$))
 else *case normalize-bounds-1*
 (*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 *Irrational* (*root-info.number-root ri r*) (p', l, r))
<proof>

definition *real-alg-2''* :: *root-info* \Rightarrow *int poly* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *real-alg-2* **where**
real-alg-2'' ri p l r = (*case normalize-bounds-1*
 (*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 *Irrational* (*root-info.number-root ri r*) (p', l, r))

lemma *real-alg-2''*: *degree p* \neq 1 \implies *real-alg-2'' ri p l r* = *real-alg-2' ri p l r*
<proof>

lemma *poly-cond-degree-0-imp-no-root*:
fixes *x* :: 'b :: {*comm-ring-1,ring-char-0*}
assumes *pc*: *poly-cond p* **and** *deg*: *degree p* = 0 **shows** *ipoly p x* \neq 0
<proof>

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*) \wedge *real-of-2* (*real-alg-2' ri q l r*) =
the-unique-root (q,l,r) (**is** - \wedge - = ?*x*)
<proof>

definition *select-correct-factor-int-poly* :: 'a \Rightarrow *int poly* \Rightarrow *real-alg-2* **where**
select-correct-factor-int-poly init p \equiv
 let *qs* = *factors-of-int-poly p*;
 polys = map ($\lambda q. (q, \text{root-info } q)$) *qs*;
 $((q,ri),(l,r)) = \text{select-correct-factor init polys}$
 in *real-alg-2' ri q l r*

lemma *select-correct-factor-int-poly*: **assumes**
conv: *converges-to* ($\lambda i. \text{bnd-get } ((\text{bnd-update } \sim i) \text{init})) x$

and *rai*: *select-correct-factor-int-poly* *init* $p = \text{rai}$
and x : *ipoly* $p \ x = 0$
and p : $p \neq 0$
shows *invariant-2* *rai* \wedge *real-of-2* *rai* $= x$
 \langle *proof* \rangle
end

11.2.10 Addition

lemma *ipoly-0-0*[*simp*]: *ipoly* $f \ (0::'a::\{\text{comm-ring-1}, \text{ring-char-0}\}) = 0 \longleftrightarrow \text{poly}$
 $f \ 0 = 0$
 \langle *proof* \rangle

lemma *add-rat-roots-below*[*simp*]: *roots-below* (*poly-add-rat* $r \ p$) $x = (\lambda y. y + \text{of-rat}$
 $r) \ \text{'roots-below } p \ (x - \text{of-rat } r)$
 \langle *proof* \rangle

lemma *add-rat-root-cond*:
shows *root-cond* (*cf-pos-poly* (*poly-add-rat* $m \ p$), l, r) $x = \text{root-cond} \ (p, l - m, r$
 $- m) \ (x - \text{of-rat } m)$
 \langle *proof* \rangle

lemma *add-rat-unique-root*: *unique-root* (*cf-pos-poly* (*poly-add-rat* $m \ p$), l, r) $=$
unique-root ($p, l - m, r - m$)
 \langle *proof* \rangle

fun *add-rat-1* :: *rat* \Rightarrow *real-alg-1* \Rightarrow *real-alg-1* **where**
add-rat-1 $r1 \ (p2, l2, r2) =$
 $\text{let } p = \text{cf-pos-poly} \ (\text{poly-add-rat } r1 \ p2);$
 $(l, r, sr) = \text{tighten-poly-bounds-for-}x \ p \ 0 \ (l2+r1) \ (r2+r1) \ (\text{sgn} \ (\text{ipoly } p$
 $(r2+r1)))$
 in
 (p, l, r)

lemma *poly-real-alg-1-add-rat*[*simp*]:
poly-real-alg-1 (*add-rat-1* $r \ y$) $= \text{cf-pos-poly} \ (\text{poly-add-rat } r \ (\text{poly-real-alg-1 } y))$
 \langle *proof* \rangle

lemma *sgn-cf-pos*:
assumes *lead-coeff* $p > 0$ **shows** *sgn* (*ipoly* (*cf-pos-poly* p) ($x::'a::\text{linordered-field}$))
 $= \text{sgn} \ (\text{ipoly } p \ x)$
 \langle *proof* \rangle

lemma *add-rat-1*: **fixes** $r1$:: *rat* **assumes** *inv-y*: *invariant-1-2* y
defines $z \equiv \text{add-rat-1 } r1 \ y$
shows *invariant-1-2* $z \wedge (\text{real-of-1 } z = \text{of-rat } r1 + \text{real-of-1 } y)$
 \langle *proof* \rangle

fun *tighten-poly-bounds-binary* :: *int poly* \Rightarrow *int poly* \Rightarrow (*rat* \times *rat* \times *rat*) \times *rat* \times

$\text{rat} \times \text{rat} \Rightarrow (\text{rat} \times \text{rat} \times \text{rat}) \times \text{rat} \times \text{rat} \times \text{rat}$ **where**
tighten-poly-bounds-binary cr1 cr2 $((l1, r1, sr1), (l2, r2, sr2)) =$
(tighten-poly-bounds cr1 $l1$ $r1$ $sr1, \text{tighten-poly-bounds}$ 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 $x \equiv \text{the-unique-root}$ $(p1, l1, r1)$ **and** $y \equiv \text{the-unique-root}$ $(p2, l2, r2)$

assumes bnd : $\bigwedge l1\ r1\ l2\ r2\ l\ r\ sr1\ sr2. I\ l1 \Rightarrow I\ l2 \Rightarrow \text{root-cond}$ $(p1, l1, r1)$ x
 $\Rightarrow \text{root-cond}$ $(p2, l2, r2)$ $y \Rightarrow$

bnd $((l1, r1, sr1), (l2, r2, sr2)) = (l, r) \Rightarrow \text{of-rat}$ $l \leq f\ x\ y \wedge f\ x\ y \leq \text{of-rat}$ r

and approx : $\bigwedge l1\ r1\ l2\ r2\ l1'\ r1'\ l2'\ r2'\ l\ l'\ r\ r'\ sr1\ sr2\ sr1'\ sr2'$.

$I\ l1 \Rightarrow I\ l2 \Rightarrow$

$l1 \leq r1 \Rightarrow l2 \leq r2 \Rightarrow$

$(l, r) = \text{bnd}$ $((l1, r1, sr1), (l2, r2, sr2)) \Rightarrow$

$(l', r') = \text{bnd}$ $((l1', r1', sr1'), (l2', r2', sr2')) \Rightarrow$

$(l1', r1') \in \{(l1, (l1+r1)/2), ((l1+r1)/2, r1)\} \Rightarrow$

$(l2', r2') \in \{(l2, (l2+r2)/2), ((l2+r2)/2, r2)\} \Rightarrow$

$(r' - l') \leq 3/4 * (r - l) \wedge l \leq l' \wedge r' \leq r$

and $I\text{-mono}$: $\bigwedge l\ l'. I\ l \Rightarrow l \leq l' \Rightarrow I\ l'$

and I : $I\ l1\ I\ l2$

and sr : $\text{sr1} = \text{sgn}$ $(\text{ipoly}$ $p1$ $r1)$ $\text{sr2} = \text{sgn}$ $(\text{ipoly}$ $p2$ $r2)$

shows *converges-to* $(\lambda\ i. \text{bnd}$ $((\text{tighten-poly-bounds-binary}$ $p1$ $p2$ $\sim i)$ $((l1, r1, sr1), (l2, r2, sr2))))$
 $(f\ x\ y)$

<proof>

fun *add-1* :: *real-alg-1* \Rightarrow *real-alg-1* \Rightarrow *real-alg-2* **where**

add-1 $(p1, l1, r1)$ $(p2, l2, r2) =$ $($

select-correct-factor-int-poly

(tighten-poly-bounds-binary $p1$ $p2)$

$(\lambda\ ((l1, r1, sr1), (l2, r2, sr2)). (l1 + l2, r1 + r2))$

$((l1, r1, \text{sgn}$ $(\text{ipoly}$ $p1$ $r1)), (l2, r2, \text{sgn}$ $(\text{ipoly}$ $p2$ $r2)))$

(poly-add $p1$ $p2))$

lemma *add-1*:

assumes x : *invariant-1-2* x **and** y : *invariant-1-2* y

defines z : $z \equiv \text{add-1}$ $x\ y$

shows *invariant-2* $z \wedge (\text{real-of-2}$ $z = \text{real-of-1}$ $x + \text{real-of-1}$ $y)$

<proof>

declare *add-rat-1.simps*[*simp del*]

declare *add-1.simps*[*simp del*]

11.2.11 Multiplication

context

begin

private fun *mult-rat-1-pos* :: *rat* \Rightarrow *real-alg-1* \Rightarrow *real-alg-2* **where**

mult-rat-1-pos $r1$ $(p2, l2, r2) = \text{real-alg-2}$ $(\text{cf-pos-poly}$ $(\text{poly-mult-rat}$ $r1$ $p2), l2 * r1,$

$r2 * r1$)

private fun *mult-1-pos* :: *real-alg-1* \Rightarrow *real-alg-1* \Rightarrow *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*,*sr1*),(*l2*,*r2*,*sr2*)). (*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* \Rightarrow *real-alg-1* \Rightarrow *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* \Rightarrow *real-alg-1* \Rightarrow *real-alg-2* **where**
 mult-1 *x* *y* = (*case* (*x*,*y*) *of* ((*p1*,*l1*,*r1*),(*p2*,*l2*,*r2*)) \Rightarrow
 if *r1* > 0 *then*
 if *r2* > 0 *then* *mult-1-pos* *x* *y*
 else *uminus-2* (*mult-1-pos* *x* (*uminus-1* *y*))
 else if *r2* > 0 *then* *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* \equiv *mult-rat-1-pos* *r1* *y*
 shows *invariant-2* *z* \wedge (*real-of-2* *z* = *of-rat* *r1* * *real-of-1* *y*)
 <*proof*>

lemma *mult-1-pos*: **assumes** *x*: *invariant-1-2* *x* **and** *y*: *invariant-1-2* *y*
 defines *z*: *z* \equiv *mult-1-pos* *x* *y*
 assumes *pos*: *real-of-1* *x* > 0 *real-of-1* *y* > 0
 shows *invariant-2* *z* \wedge (*real-of-2* *z* = *real-of-1* *x* * *real-of-1* *y*)
 <*proof*>

lemma *mult-1*: **assumes** *x*: *invariant-1-2* *x* **and** *y*: *invariant-1-2* *y*
 defines *z*[*simp*]: *z* \equiv *mult-1* *x* *y*
 shows *invariant-2* *z* \wedge (*real-of-2* *z* = *real-of-1* *x* * *real-of-1* *y*)
 <*proof*>

lemma *mult-rat-1*: **fixes** *x* **assumes** *y*: *invariant-1* *y*
 defines *z*: *z* \equiv *mult-rat-1* *x* *y*
 shows *invariant-2* *z* \wedge (*real-of-2* *z* = *of-rat* *x* * *real-of-1* *y*)
 <*proof*>
end

declare *mult-1.simps*[*simp del*]
declare *mult-rat-1.simps*[*simp del*]

11.2.12 Root

definition *ipoly-root-delta* :: *int poly* \Rightarrow *real* **where**

ipoly-root-delta *p* = *Min* (*insert* 1 { *abs* (*x* - *y*) | *x y. ipoly p x = 0* \wedge *ipoly p y = 0* \wedge *x* \neq *y*}) / 4

lemma *ipoly-root-delta*: **assumes** *p* \neq 0

shows *ipoly-root-delta* *p* > 0

$2 \leq \text{card } (\text{Collect } (\text{root-cond } (p, l, r))) \implies \text{ipoly-root-delta } p \leq \text{real-of-rat } (r - l) / 4$

<proof>

lemma *sgn-less-eq-1-rat*: **fixes** *a b* :: *rat*

shows *sgn a = 1* $\implies a \leq b \implies \text{sgn } b = 1$

<proof>

lemma *sgn-less-eq-1-real*: **fixes** *a b* :: *real*

shows *sgn a = 1* $\implies a \leq b \implies \text{sgn } b = 1$

<proof>

definition *compare-1-rat* :: *real-alg-1* \Rightarrow *rat* \Rightarrow *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 ($\lambda y. \text{compare } y x$)
else ($\lambda y. \text{compare-rat-1 } y \text{ rai}$))

lemma *compare-real-of-rat*: *compare* (*real-of-rat* *x*) (*of-rat* *y*) = *compare* *x y*

<proof>

lemma *compare-1-rat*: **assumes** *rc*: *invariant-1 y*

shows *compare-1-rat* *y x* = *compare* (*of-rat* *x*) (*real-of-1 y*)

<proof>

context

fixes *n* :: *nat*

begin

private definition *initial-lower-bound* :: *rat* \Rightarrow *rat* **where**

initial-lower-bound *l* = (*if* *l* \leq 1 then *l* else *of-int* (*root-rat-floor* *n l*))

private definition *initial-upper-bound* :: *rat* \Rightarrow *rat* **where**

initial-upper-bound *r* = (*of-int* (*root-rat-ceiling* *n r*))

context

fixes *cmpx* :: *rat* \Rightarrow *order*

begin

fun *tighten-bound-root* ::

rat \times *rat* \Rightarrow *rat* \times *rat* **where**

tighten-bound-root (*l',r'*) = (*let*

m' = (*l' + r'*) / 2;

m = *m'* \wedge *n*

in case *cmpx* *m* of
 Eq $\Rightarrow (m', m')$
 | Lt $\Rightarrow (m', r')$
 | Gt $\Rightarrow (l', m')$

lemma *tighten-bound-root*: **assumes** *sgn*: $\text{sgn } il = 1$ *real-of-1* $x \geq 0$ **and**
il: *real-of-rat* $il \leq \text{root } n$ (*real-of-1* x) **and**
ir: $\text{root } n$ (*real-of-1* x) \leq *real-of-rat* *ir* **and**
rai: *invariant-1* x **and**
cmpx: *cmpx* = *compare-1-rat* x **and**
n: $n \neq 0$
shows *converges-to* $(\lambda i. (\text{tighten-bound-root } \sim i) (il, ir))$
 ($\text{root } n$ (*real-of-1* x)) (**is** *converges-to* ?f ?x)
 <proof>
end

private fun *root-pos-1* :: *real-alg-1* \Rightarrow *real-alg-2* **where**
root-pos-1 (*p*,*l*,*r*) = (
 (*select-correct-factor-int-poly*
 (*tighten-bound-root* (*compare-1-rat* (*p*,*l*,*r*)))
 ($\lambda x. x$)
 (*initial-lower-bound* *l*, *initial-upper-bound* *r*)
 (*poly-nth-root* n *p*)))

fun *root-1* :: *real-alg-1* \Rightarrow *real-alg-2* **where**
root-1 (*p*,*l*,*r*) = (
 if $n = 0 \vee r = 0$ then *Rational* 0
 else if $r > 0$ then *root-pos-1* (*p*,*l*,*r*)
 else *uminus-2* (*root-pos-1* (*uminus-1* (*p*,*l*,*r*))))

context
assumes *n*: $n \neq 0$
begin

lemma *initial-upper-bound*: **assumes** *x*: $x > 0$ **and** *xr*: $x \leq \text{of-rat } r$
shows $\text{sgn } (\text{initial-upper-bound } r) = 1$ $\text{root } n$ $x \leq \text{of-rat } (\text{initial-upper-bound } r)$
 <proof>

lemma *initial-lower-bound*: **assumes** *l*: $l > 0$ **and** *lx*: $\text{of-rat } l \leq x$
shows $\text{sgn } (\text{initial-lower-bound } l) = 1$ $\text{of-rat } (\text{initial-lower-bound } l) \leq \text{root } n$ x
 <proof>

lemma *root-pos-1*:
assumes *x*: *invariant-1* x **and** *pos*: *rai-ub* $x > 0$
defines *y*: $y \equiv \text{root-pos-1 } x$
shows *invariant-2* $y \wedge \text{real-of-2 } y = \text{root } n$ (*real-of-1* x)
 <proof>

end

lemma *root-1*: **assumes** x : *invariant-1* x
defines y : $y \equiv \text{root-1 } x$
shows *invariant-2* $y \wedge (\text{real-of-2 } y = \text{root } n (\text{real-of-1 } x))$
 $\langle \text{proof} \rangle$
end

declare *root-1.simps*[*simp del*]

11.2.13 Embedding of Rational Numbers

definition *of-rat-1* :: $\text{rat} \Rightarrow \text{real-alg-1}$ **where**
of-rat-1 $x \equiv (\text{poly-rat } x, x, x)$

lemma *of-rat-1*:
shows *invariant-1* (*of-rat-1* x) **and** *real-of-1* (*of-rat-1* x) = *of-rat* x
 $\langle \text{proof} \rangle$

fun *info-2* :: $\text{real-alg-2} \Rightarrow \text{rat} + \text{int poly} \times \text{nat}$ **where**
info-2 (*Rational* x) = *Inl* x
 $|$ *info-2* (*Irrational* $n (p, l, r)$) = *Inr* (p, n)

lemma *info-2-card*: **assumes** rc : *invariant-2* x
shows *info-2* $x = \text{Inr } (p, n) \Longrightarrow \text{poly-cond } p \wedge \text{ipoly } p (\text{real-of-2 } x) = 0 \wedge \text{degree } p \geq 2$
 $\wedge \text{card } (\text{roots-below } p (\text{real-of-2 } x)) = n$
info-2 $x = \text{Inl } y \Longrightarrow \text{real-of-2 } x = \text{of-rat } y$
 $\langle \text{proof} \rangle$

lemma *real-of-2-Irrational*: *invariant-2* (*Irrational* $n \text{rai}$) \Longrightarrow *real-of-2* (*Irrational* $n \text{rai}$) \neq *of-rat* x
 $\langle \text{proof} \rangle$

lemma *info-2*: **assumes**
 ix : *invariant-2* x **and** iy : *invariant-2* y
shows *info-2* $x = \text{info-2 } y \longleftrightarrow \text{real-of-2 } x = \text{real-of-2 } y$
 $\langle \text{proof} \rangle$

lemma *info-2-unique*: *invariant-2* $x \Longrightarrow \text{invariant-2 } y \Longrightarrow$
 $\text{real-of-2 } x = \text{real-of-2 } y \Longrightarrow \text{info-2 } x = \text{info-2 } y$
 $\langle \text{proof} \rangle$

lemma *info-2-inj*: *invariant-2* $x \Longrightarrow \text{invariant-2 } y \Longrightarrow \text{info-2 } x = \text{info-2 } y \Longrightarrow$
 $\text{real-of-2 } x = \text{real-of-2 } y$
 $\langle \text{proof} \rangle$

context
fixes $cr1$ $cr2$:: $\text{rat} \Rightarrow \text{rat} \Rightarrow \text{nat}$
begin

partial-function (*tailrec*) *compare-1* :: *int poly* \Rightarrow *int poly* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *rat* \Rightarrow *order* **where**
 [code]: *compare-1* *p1 p2 l1 r1 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*) \neq *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 sr1 l2 r2 sr2* = *compare* (*the-unique-root* (*p1,l1,r1*))
(*the-unique-root* (*p2,l2,r2*))
<*proof*>
end

fun *real-alg-1* :: *real-alg-2* \Rightarrow *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*
 <*proof*>

definition *root-2* :: *nat* \Rightarrow *real-alg-2* \Rightarrow *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*>

fun *add-2* :: *real-alg-2* \Rightarrow *real-alg-2* \Rightarrow *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 *invariant-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*)
 <*proof*>

fun *mult-2* :: *real-alg-2* \Rightarrow *real-alg-2* \Rightarrow *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*)
⟨*proof*⟩

fun *to-rat-2* :: *real-alg-2* \Rightarrow *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* \in \mathbb{Q} then *Some* (*THE* *q*. *real-of-2* *x* = *of-rat* *q*) else *None*)
⟨*proof*⟩

fun *equal-2* :: *real-alg-2* \Rightarrow *real-alg-2* \Rightarrow *bool* **where**
equal-2 (*Rational* *r*) (*Rational* *q*) = (*r* = *q*)
| *equal-2* (*Irrational* *n* (*p*, -)) (*Irrational* *m* (*q*, -)) = (*p* = *q* \wedge *n* = *m*)
| *equal-2* (*Rational* *r*) (*Irrational* - *yy*) = *False*
| *equal-2* (*Irrational* - *xx*) (*Rational* *q*) = *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*)
⟨*proof*⟩

fun *compare-2* :: *real-alg-2* \Rightarrow *real-alg-2* \Rightarrow *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* \wedge *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*⟩

fun *sgn-2* :: *real-alg-2* \Rightarrow *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*)
⟨*proof*⟩

fun *floor-2* :: *real-alg-2* \Rightarrow *int* **where**
floor-2 (*Rational* *r*) = *floor* *r*
| *floor-2* (*Irrational* *n* *rai*) = *floor-1* *rai*

lemma *floor-2: invariant-2* *x* \Longrightarrow *floor-2* *x* = *floor* (*real-of-2* *x*)
<proof>

11.2.14 Definitions and Algorithms on Type with Invariant

lift-definition *of-rat-3* :: *rat* \Rightarrow *real-alg-3* **is** *of-rat-2*
<proof>

lemma *of-rat-3: real-of-3* (*of-rat-3* *x*) = *of-rat* *x*
<proof>

lift-definition *root-3* :: *nat* \Rightarrow *real-alg-3* \Rightarrow *real-alg-3* **is** *root-2*
<proof>

lemma *root-3: real-of-3* (*root-3* *n* *x*) = *root* *n* (*real-of-3* *x*)
<proof>

lift-definition *equal-3* :: *real-alg-3* \Rightarrow *real-alg-3* \Rightarrow *bool* **is** *equal-2* *<proof>*

lemma *equal-3: equal-3* *x* *y* = (*real-of-3* *x* = *real-of-3* *y*)
<proof>

lift-definition *compare-3* :: *real-alg-3* \Rightarrow *real-alg-3* \Rightarrow *order* **is** *compare-2* *<proof>*

lemma *compare-3: compare-3* *x* *y* = (*compare* (*real-of-3* *x*) (*real-of-3* *y*))
<proof>

lift-definition *add-3* :: *real-alg-3* \Rightarrow *real-alg-3* \Rightarrow *real-alg-3* **is** *add-2*
<proof>

lemma *add-3: real-of-3* (*add-3* *x* *y*) = *real-of-3* *x* + *real-of-3* *y*
<proof>

lift-definition *mult-3* :: *real-alg-3* \Rightarrow *real-alg-3* \Rightarrow *real-alg-3* **is** *mult-2*
<proof>

lemma *mult-3: real-of-3* (*mult-3* *x* *y*) = *real-of-3* *x* * *real-of-3* *y*
<proof>

lift-definition *sgn-3* :: *real-alg-3* \Rightarrow *rat* **is** *sgn-2* *<proof>*

lemma *sgn-3: real-of-rat* (*sgn-3* *x*) = *sgn* (*real-of-3* *x*)
<proof>

lift-definition *to-rat-3* :: *real-alg-3* \Rightarrow *rat option* **is** *to-rat-2* \langle *proof* \rangle

lemma *to-rat-3*: *to-rat-3* $x =$
(if *real-of-3* $x \in \mathbf{Q}$ then *Some* (*THE* q . *real-of-3* $x =$ *of-rat* q) else *None*)
 \langle *proof* \rangle

lift-definition *floor-3* :: *real-alg-3* \Rightarrow *int* **is** *floor-2* \langle *proof* \rangle

lemma *floor-3*: *floor-3* $x =$ *floor* (*real-of-3* x)
 \langle *proof* \rangle

lift-definition *info-3* :: *real-alg-3* \Rightarrow *rat + int poly \times nat* **is** *info-2* \langle *proof* \rangle

lemma *info-3-fun*: *real-of-3* $x =$ *real-of-3* $y \implies$ *info-3* $x =$ *info-3* y
 \langle *proof* \rangle

lift-definition *info-real-alg* :: *real-alg* \Rightarrow *rat + int poly \times nat* **is** *info-3*
 \langle *proof* \rangle

lemma *info-real-alg*:
info-real-alg $x =$ *Inr* (p, n) \implies p represents (*real-of* x) \wedge *card* $\{y. y \leq$ *real-of* x
 \wedge *ipoly* p $y = 0\} = n$ \wedge *irreducible* p
info-real-alg $x =$ *Inl* $q \implies$ *real-of* $x =$ *of-rat* q
 \langle *proof* \rangle

instantiation *real-alg* :: *plus*

begin

lift-definition *plus-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **is** *add-3*
 \langle *proof* \rangle

instance \langle *proof* \rangle

end

lemma *plus-real-alg*: (*real-of* x) + (*real-of* y) = *real-of* ($x + y$)
 \langle *proof* \rangle

instantiation *real-alg* :: *minus*

begin

definition *minus-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **where**
minus-real-alg x $y = x + (-y)$

instance \langle *proof* \rangle

end

lemma *minus-real-alg*: (*real-of* x) - (*real-of* y) = *real-of* ($x - y$)

<proof>

lift-definition *of-rat-real-alg* :: *rat* \Rightarrow *real-alg* **is** *of-rat-3* *<proof>*

lemma *of-rat-real-alg*: *real-of-rat* $x = \text{real-of}$ (*of-rat-real-alg* x)
<proof>

instantiation *real-alg* :: *zero*

begin

definition *zero-real-alg* :: *real-alg* **where** *zero-real-alg* \equiv *of-rat-real-alg* 0

instance *<proof>*

end

lemma *zero-real-alg*: $0 = \text{real-of}$ 0

<proof>

instantiation *real-alg* :: *one*

begin

definition *one-real-alg* :: *real-alg* **where** *one-real-alg* \equiv *of-rat-real-alg* 1

instance *<proof>*

end

lemma *one-real-alg*: $1 = \text{real-of}$ 1

<proof>

instantiation *real-alg* :: *times*

begin

lift-definition *times-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **is** *mult-3*

<proof>

instance *<proof>*

end

lemma *times-real-alg*: (*real-of* x) * (*real-of* y) = *real-of* ($x * y$)

<proof>

instantiation *real-alg* :: *inverse*

begin

lift-definition *inverse-real-alg* :: *real-alg* \Rightarrow *real-alg* **is** *inverse-3*

<proof>

definition *divide-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **where**

divide-real-alg $x y = x * \text{inverse}$ y

instance *<proof>*

end

lemma *inverse-real-alg*: $\text{inverse} (\text{real-of } x) = \text{real-of} (\text{inverse } x)$
⟨*proof*⟩

lemma *divide-real-alg*: $(\text{real-of } x) / (\text{real-of } y) = \text{real-of} (x / y)$
⟨*proof*⟩

instance *real-alg* :: *ab-group-add*
⟨*proof*⟩

instance *real-alg* :: *field*
⟨*proof*⟩

instance *real-alg* :: *numeral* ⟨*proof*⟩

lift-definition *root-real-alg* :: *nat* \Rightarrow *real-alg* \Rightarrow *real-alg* **is** *root-3*
⟨*proof*⟩

lemma *root-real-alg*: $\text{root } n (\text{real-of } x) = \text{real-of} (\text{root-real-alg } n x)$
⟨*proof*⟩

lift-definition *sgn-real-alg-rat* :: *real-alg* \Rightarrow *rat* **is** *sgn-3*
⟨*proof*⟩

lemma *sgn-real-alg-rat*: $\text{real-of-rat} (\text{sgn-real-alg-rat } x) = \text{sgn} (\text{real-of } x)$
⟨*proof*⟩

instantiation *real-alg* :: *sgn*

begin

definition *sgn-real-alg* :: *real-alg* \Rightarrow *real-alg* **where**

sgn-real-alg $x = \text{of-rat-real-alg} (\text{sgn-real-alg-rat } x)$

instance ⟨*proof*⟩

end

lemma *sgn-real-alg*: $\text{sgn} (\text{real-of } x) = \text{real-of} (\text{sgn } x)$
⟨*proof*⟩

instantiation *real-alg* :: *equal*

begin

lift-definition *equal-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *bool* **is** *equal-3*

⟨*proof*⟩

instance

⟨*proof*⟩

end

lemma *equal-real-alg*: $HOL.equal (real-of\ x) (real-of\ y) = (x = y)$
<proof>

instantiation *real-alg* :: *ord*
begin

definition *less-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *bool* **where**
[code del]: *less-real-alg* *x* *y* = (*real-of* *x* < *real-of* *y*)

definition *less-eq-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *bool* **where**
[code del]: *less-eq-real-alg* *x* *y* = (*real-of* *x* \leq *real-of* *y*)

instance *<proof>*
end

lemma *less-real-alg*: *less* (*real-of* *x*) (*real-of* *y*) = (*x* < *y*) *<proof>*

lemma *less-eq-real-alg*: *less-eq* (*real-of* *x*) (*real-of* *y*) = (*x* \leq *y*) *<proof>*

instantiation *real-alg* :: *compare-order*
begin

lift-definition *compare-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *order* **is** *compare-3*
<proof>

lemma *compare-real-alg*: *compare* (*real-of* *x*) (*real-of* *y*) = (*compare* *x* *y*)
<proof>

instance
<proof>
end

lemma *less-eq-real-alg-code*[*code*]:
(*less-eq* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *bool*) = *le-of-comp* *compare*
(*less* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *bool*) = *lt-of-comp* *compare*
<proof>

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 *<proof>*
end

lemma *abs-real-alg*: *abs* (*real-of* *x*) = *real-of* (*abs* *x*)
<proof>

lemma *sgn-real-alg-sound*: $\text{sgn } x = (\text{if } x = 0 \text{ then } 0 \text{ else if } 0 < \text{real-of } x \text{ then } 1 \text{ else } -1)$

(**is** - = ?*r*)
<*proof*>

lemma *real-of-of-int*: $\text{real-of-rat } (\text{rat-of-int } z) = \text{real-of } (\text{of-int } z)$
<*proof*>

instance *real-alg* :: *linordered-field*
<*proof*>

instantiation *real-alg* :: *floor-ceiling*

begin

lift-definition *floor-real-alg* :: *real-alg* \Rightarrow *int* **is** *floor-3*
<*proof*>

lemma *floor-real-alg*: $\text{floor } (\text{real-of } x) = \text{floor } x$
<*proof*>

instance
<*proof*>
end

instantiation *real-alg* ::
{*unique-euclidean-ring*, *normalization-euclidean-semiring*, *normalization-semidom-multiplicative*}
begin

definition [*simp*]: *normalize-real-alg* = (*normalize-field* :: *real-alg* \Rightarrow -)

definition [*simp*]: *unit-factor-real-alg* = (*unit-factor-field* :: *real-alg* \Rightarrow -)

definition [*simp*]: *modulo-real-alg* = (*mod-field* :: *real-alg* \Rightarrow -)

definition [*simp*]: *euclidean-size-real-alg* = (*euclidean-size-field* :: *real-alg* \Rightarrow -)

definition [*simp*]: *division-segment* (*x* :: *real-alg*) = 1

instance
<*proof*>

end

instantiation *real-alg* :: *euclidean-ring-gcd*
begin

definition *gcd-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **where**
gcd-real-alg = *Euclidean-Algorithm.gcd*

definition *lcm-real-alg* :: *real-alg* \Rightarrow *real-alg* \Rightarrow *real-alg* **where**
lcm-real-alg = *Euclidean-Algorithm.lcm*

definition *Gcd-real-alg* :: *real-alg set* \Rightarrow *real-alg* **where**
Gcd-real-alg = *Euclidean-Algorithm.Gcd*

definition *Lcm-real-alg* :: *real-alg set* \Rightarrow *real-alg* **where**
Lcm-real-alg = *Euclidean-Algorithm.Lcm*

instance $\langle proof \rangle$

end

instance $real\text{-}alg :: field\text{-}gcd \langle proof \rangle$

definition $min\text{-}int\text{-}poly\text{-}real\text{-}alg :: real\text{-}alg \Rightarrow int\text{-}poly$ **where**

$min\text{-}int\text{-}poly\text{-}real\text{-}alg\ x = (case\ info\text{-}real\text{-}alg\ x\ of\ Inl\ r \Rightarrow poly\text{-}rat\ r \mid Inr\ (p,-) \Rightarrow p)$

lemma $min\text{-}int\text{-}poly\text{-}real\text{-}alg\text{-}real\text{-}of: min\text{-}int\text{-}poly\text{-}real\text{-}alg\ x = min\text{-}int\text{-}poly\ (real\text{-}of\ x)$
 $\langle proof \rangle$

lemma $min\text{-}int\text{-}poly\text{-}real\text{-}code: min\text{-}int\text{-}poly\text{-}real\ (real\text{-}of\ x) = min\text{-}int\text{-}poly\text{-}real\text{-}alg\ x$
 $\langle proof \rangle$

lemma $min\text{-}int\text{-}poly\text{-}real\text{-}of: min\text{-}int\text{-}poly\ (real\text{-}of\ x) = min\text{-}int\text{-}poly\ x$
 $\langle proof \rangle$

definition $real\text{-}alg\text{-}of\text{-}real :: real \Rightarrow real\text{-}alg$ **where**

$real\text{-}alg\text{-}of\text{-}real\ x = (if\ (\exists\ y.\ x = real\text{-}of\ y)\ then\ (THE\ y.\ x = real\text{-}of\ y)\ else\ 0)$

lemma $real\text{-}alg\text{-}of\text{-}real\text{-}code[code]: real\text{-}alg\text{-}of\text{-}real\ (real\text{-}of\ x) = x$
 $\langle proof \rangle$

lift-definition $to\text{-}rat\text{-}real\text{-}alg\text{-}main :: real\text{-}alg \Rightarrow rat\ option$ **is** $to\text{-}rat\text{-}3$
 $\langle proof \rangle$

lemma $to\text{-}rat\text{-}real\text{-}alg\text{-}main: to\text{-}rat\text{-}real\text{-}alg\text{-}main\ x = (if\ real\text{-}of\ x \in \mathbb{Q}\ then\ Some\ (THE\ q.\ real\text{-}of\ x = of\text{-}rat\ q)\ else\ None)$
 $\langle proof \rangle$

definition $to\text{-}rat\text{-}real\text{-}alg :: real\text{-}alg \Rightarrow rat$ **where**

$to\text{-}rat\text{-}real\text{-}alg\ x = (case\ to\text{-}rat\text{-}real\text{-}alg\text{-}main\ x\ of\ Some\ q \Rightarrow q \mid None \Rightarrow 0)$

definition $is\text{-}rat\text{-}real\text{-}alg :: real\text{-}alg \Rightarrow bool$ **where**

$is\text{-}rat\text{-}real\text{-}alg\ x = (case\ to\text{-}rat\text{-}real\text{-}alg\text{-}main\ x\ of\ Some\ q \Rightarrow True \mid None \Rightarrow False)$

lemma $is\text{-}rat\text{-}real\text{-}alg: is\text{-}rat\ (real\text{-}of\ x) = (is\text{-}rat\text{-}real\text{-}alg\ x)$
 $\langle proof \rangle$

lemma $to\text{-}rat\text{-}real\text{-}alg: to\text{-}rat\ (real\text{-}of\ x) = (to\text{-}rat\text{-}real\text{-}alg\ x)$
 $\langle proof \rangle$

lemma *algebraic-real-code*[code]: *algebraic-real (real-of x) = True*
(*proof*)

11.3 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
min-int-poly-real-code

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*
min-int-poly-real
root]]

declare *real-alg-code-eqns* [*code equation*]

lemma *Ratreal-code*[code]:
Ratreal = *real-of* \circ *of-rat-real-alg*
 ⟨*proof*⟩

lemma *real-of-post*[code-post]: *real-of* (*Real-Alg-Quotient* (*Real-Alg-Invariant* (*Rational* *x*))) = *of-rat x*
 ⟨*proof*⟩

end

12 Real Roots

This theory contains an algorithm to determine the set of real roots of a rational polynomial. For polynomials with real coefficients, we refer to the AFP entry "Factor-Algebraic-Polynomial".

theory *Real-Roots*

imports

Cauchy-Root-Bound

Real-Algebraic-Numbers

begin

hide-const (**open**) *UnivPoly.coeff*

hide-const (**open**) *Module.smult*

partial-function (*tailrec*) *roots-of-2-main* ::

int poly \Rightarrow *root-info* \Rightarrow (*rat* \Rightarrow *rat* \Rightarrow *nat*) \Rightarrow (*rat* \times *rat*)*list* \Rightarrow *real-alg-2 list* \Rightarrow *real-alg-2 list* **where**

[*code*]: *roots-of-2-main p ri cr lrs rais* = (*case lrs of Nil* \Rightarrow *rais*

| (*l,r*) $\#$ *lrs* \Rightarrow *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* \Rightarrow *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.l-r ri;

B = root-bound p

in (roots-of-2-main p ri cr [(-B,B)] []))

fun *pairwise-disjoint* :: '*a set list* \Rightarrow *bool* **where**

pairwise-disjoint [] = *True*

| *pairwise-disjoint (x $\#$ xs)* = (*(x* \cap (\bigcup *y* \in *set xs. y*) = *{}*) \wedge *pairwise-disjoint xs*)

lemma *roots-of-2-irr*: **assumes** *pc*: *poly-cond p* **and** *deg*: *degree p > 0*

shows $\text{real-of-2} \text{ ' set } (\text{roots-of-2-irr } p) = \{x. \text{ipoly } p \ x = 0\}$ (**is** ?one)
Ball ($\text{set } (\text{roots-of-2-irr } p)$) *invariant-2* (**is** ?two)
distinct ($\text{map } \text{real-of-2} (\text{roots-of-2-irr } p)$) (**is** ?three)
 ⟨proof⟩

definition $\text{roots-of-2} :: \text{int poly} \Rightarrow \text{real-alg-2 list}$ **where**
 $\text{roots-of-2 } p = \text{concat } (\text{map } \text{roots-of-2-irr}$
 $\quad (\text{factors-of-int-poly } p))$

lemma roots-of-2 :
shows $p \neq 0 \Longrightarrow \text{real-of-2} \text{ ' set } (\text{roots-of-2 } p) = \{x. \text{ipoly } p \ x = 0\}$
Ball ($\text{set } (\text{roots-of-2 } p)$) *invariant-2*
distinct ($\text{map } \text{real-of-2} (\text{roots-of-2 } p)$)
 ⟨proof⟩

lift-definition (*code-dt*) $\text{roots-of-3} :: \text{int poly} \Rightarrow \text{real-alg-3 list}$ **is** roots-of-2
 ⟨proof⟩

lemma roots-of-3 :
shows $p \neq 0 \Longrightarrow \text{real-of-3} \text{ ' set } (\text{roots-of-3 } p) = \{x. \text{ipoly } p \ x = 0\}$
distinct ($\text{map } \text{real-of-3} (\text{roots-of-3 } p)$)
 ⟨proof⟩

lift-definition $\text{roots-of-real-alg} :: \text{int poly} \Rightarrow \text{real-alg list}$ **is** roots-of-3 ⟨proof⟩

lemma roots-of-real-alg :
 $p \neq 0 \Longrightarrow \text{real-of} \text{ ' set } (\text{roots-of-real-alg } p) = \{x. \text{ipoly } p \ x = 0\}$
distinct ($\text{map } \text{real-of} (\text{roots-of-real-alg } p)$)
 ⟨proof⟩

definition $\text{real-roots-of-int-poly} :: \text{int poly} \Rightarrow \text{real list}$ **where**
 $\text{real-roots-of-int-poly } p = \text{map } \text{real-of} (\text{roots-of-real-alg } p)$

definition $\text{real-roots-of-rat-poly} :: \text{rat poly} \Rightarrow \text{real list}$ **where**
 $\text{real-roots-of-rat-poly } p = \text{map } \text{real-of} (\text{roots-of-real-alg } (\text{snd } (\text{rat-to-int-poly } p)))$

abbreviation $\text{rpoly} :: \text{rat poly} \Rightarrow 'a :: \text{field-char-0} \Rightarrow 'a$
where $\text{rpoly } f \equiv \text{poly } (\text{map-poly } \text{of-rat } f)$

lemma $\text{real-roots-of-int-poly}$: $p \neq 0 \Longrightarrow \text{set } (\text{real-roots-of-int-poly } p) = \{x. \text{ipoly } p \ x = 0\}$
distinct ($\text{real-roots-of-int-poly } p$)
 ⟨proof⟩

lemma $\text{real-roots-of-rat-poly}$: $p \neq 0 \Longrightarrow \text{set } (\text{real-roots-of-rat-poly } p) = \{x. \text{rpoly } p \ x = 0\}$
distinct ($\text{real-roots-of-rat-poly } p$)
 ⟨proof⟩

end

13 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-idom-hom complex-of-real*⟨*proof*⟩

lemma *real-poly-real-coeff*: **assumes** *set (coeffs p) ⊆ ℝ*

shows *coeff p x ∈ ℝ*

⟨*proof*⟩

lemma *complex-conjugate-root*:

assumes *real: set (coeffs p) ⊆ ℝ* **and** *rt: poly p c = 0*

shows *poly p (cnj c) = 0*

⟨*proof*⟩

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*⟩

lemma *map-poly-Re-coeffs*:

coeffs (map-poly Re p) = map Re (coeffs p)

⟨*proof*⟩

lemma *map-poly-Re-0*: *map-poly Re p = 0 ⇒ p = 0*

⟨*proof*⟩

end

lemma *real-poly-add*:

assumes $set\ (coeffs\ p) \subseteq \mathbb{R}$ $set\ (coeffs\ q) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (p + q)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-sum*:

assumes $\bigwedge x. x \in S \implies set\ (coeffs\ (f\ x)) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (sum\ f\ S)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-smult*: **fixes** $p :: 'a :: \{idom, real-algebra-1\}$ *poly*

assumes $c \in \mathbb{R}$ $set\ (coeffs\ p) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (smult\ c\ p)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-pCons*:

assumes $c \in \mathbb{R}$ $set\ (coeffs\ p) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (pCons\ c\ p)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-mult*: **fixes** $p :: 'a :: \{idom, real-algebra-1\}$ *poly*

assumes $p: set\ (coeffs\ p) \subseteq \mathbb{R}$ **and** $q: set\ (coeffs\ q) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (p * q)) \subseteq \mathbb{R}$ *<proof>*

lemma *real-poly-power*: **fixes** $p :: 'a :: \{idom, real-algebra-1\}$ *poly*

assumes $p: set\ (coeffs\ p) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (p \wedge n)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-prod*: **fixes** $f :: 'a \Rightarrow 'b :: \{idom, real-algebra-1\}$ *poly*

assumes $\bigwedge x. x \in S \implies set\ (coeffs\ (f\ x)) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (prod\ f\ S)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-uminus*:

assumes $set\ (coeffs\ p) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (-p)) \subseteq \mathbb{R}$

<proof>

lemma *real-poly-minus*:

assumes $set\ (coeffs\ p) \subseteq \mathbb{R}$ $set\ (coeffs\ q) \subseteq \mathbb{R}$

shows $set\ (coeffs\ (p - q)) \subseteq \mathbb{R}$

<proof>

lemma **fixes** $p :: 'a :: real-field$ *poly*

assumes p : $\text{set } (\text{coeffs } p) \subseteq \mathbb{R}$ **and** $*$: $\text{set } (\text{coeffs } q) \subseteq \mathbb{R}$
shows *real-poly-div*: $\text{set } (\text{coeffs } (q \text{ div } p)) \subseteq \mathbb{R}$
and *real-poly-mod*: $\text{set } (\text{coeffs } (q \text{ mod } p)) \subseteq \mathbb{R}$
 $\langle \text{proof} \rangle$

lemma *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 $\text{set } (\text{coeffs } q) \subseteq \mathbb{R}$
 $\langle \text{proof} \rangle$

lemma *complex-conjugate-order*: **assumes** *real*: $\text{set } (\text{coeffs } p) \subseteq \mathbb{R}$
 $p \neq 0$
shows $\text{order } (\text{cnj } c) p = \text{order } c p$
 $\langle \text{proof} \rangle$

lemma *map-poly-of-real-Re*: **assumes** $\text{set } (\text{coeffs } p) \subseteq \mathbb{R}$
shows $\text{map-poly of-real } (\text{map-poly Re } p) = p$
 $\langle \text{proof} \rangle$

lemma *map-poly-Re-of-real*: $\text{map-poly Re } (\text{map-poly of-real } p) = p$
 $\langle \text{proof} \rangle$

lemma *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** $\text{map-poly Re } (p * q) = \text{map-poly Re } p * \text{map-poly Re } q$
 $\langle \text{proof} \rangle$

lemma *map-poly-Re-power*: **assumes** p : $\text{set } (\text{coeffs } p) \subseteq \mathbb{R}$
shows $\text{map-poly Re } (p^{\hat{n}}) = (\text{map-poly Re } p)^{\hat{n}}$
 $\langle \text{proof} \rangle$

lemma *real-degree-2-factorization-exists-complex*: **fixes** $p :: \text{complex poly}$
assumes pR : $\text{set } (\text{coeffs } p) \subseteq \mathbb{R}$
shows $\exists qs. p = \text{prod-list } qs \wedge (\forall q \in \text{set } qs. \text{set } (\text{coeffs } q) \subseteq \mathbb{R} \wedge \text{degree } q \leq 2)$
 $\langle \text{proof} \rangle$

lemma *real-degree-2-factorization-exists*: **fixes** $p :: \text{real poly}$
shows $\exists qs. p = \text{prod-list } qs \wedge (\forall q \in \text{set } qs. \text{degree } q \leq 2)$
 $\langle \text{proof} \rangle$

lemma *odd-degree-imp-real-root*: **assumes** *odd* ($\text{degree } p$)
shows $\exists x. \text{poly } p x = (0 :: \text{real})$
 $\langle \text{proof} \rangle$

end

13.1 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 $[[code\ drop:\ Gcd\text{-}fn]]$

declare $[[code\ drop:\ Lcm\text{-}fn]]$

definition *gcds* :: $'a::semiring\text{-}gcd\ list \Rightarrow 'a$
where $[simp, code\text{-}abbrev]: gcds\ xs = gcd\text{-}list\ xs$

lemma $[code]:$
 $gcds\ xs = fold\ gcd\ xs\ 0$
 $\langle proof \rangle$

definition *lcms* :: $'a::semiring\text{-}gcd\ list \Rightarrow 'a$
where $[simp, code\text{-}abbrev]: lcms\ xs = lcm\text{-}list\ xs$

lemma $[code]:$
 $lcms\ xs = fold\ lcm\ xs\ 1$
 $\langle proof \rangle$

lemma *in-reals-code* $[code\text{-}unfold]:$
 $x \in \mathbb{R} \iff Im\ x = 0$
 $\langle proof \rangle$

definition *is-norm-1* :: $complex \Rightarrow bool$ **where**
 $is\text{-}norm\text{-}1\ z = ((Re\ z)^2 + (Im\ z)^2 = 1)$

lemma *is-norm-1* $[simp]: is\text{-}norm\text{-}1\ x = (norm\ x = 1)$
 $\langle proof \rangle$

definition *is-norm-le-1* :: $complex \Rightarrow bool$ **where**
 $is\text{-}norm\text{-}le\text{-}1\ z = ((Re\ z)^2 + (Im\ z)^2 \leq 1)$

lemma *is-norm-le-1* $[simp]: is\text{-}norm\text{-}le\text{-}1\ x = (norm\ x \leq 1)$
 $\langle proof \rangle$

instantiation *complex* :: *finite-UNIV*

begin

definition *finite-UNIV* = *Phantom*(*complex*) *False*

```

instance
  ⟨proof⟩
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⟩
end

derive (eq) ceq complex real
derive (compare) ccompare complex
derive (compare) ccompare real
derive (dlist) set-impl complex real

end

```

14 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

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)

```

14.1 Syntactic Class Instantiations

```

instantiation interval :: (zero) zero begin
  definition zero-interval where 0 ≡ Interval 0 0
  instance⟨proof⟩
end

```

```

instantiation interval :: (one) one begin
  definition 1 = Interval 1 1
  instance<proof>
end

instantiation interval :: (plus) plus begin
  fun plus-interval where Interval lx ux + Interval ly uy = Interval (lx + ly) (ux
  + uy)
  instance<proof>
end

instantiation interval :: (uminus) uminus begin
  fun uminus-interval where - Interval l u = Interval (-u) (-l)
  instance<proof>
end

instantiation interval :: (minus) minus begin
  fun minus-interval where Interval lx ux - Interval ly uy = Interval (lx - uy)
  (ux - ly)
  instance<proof>
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<proof>
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<proof>
end

```

14.2 Class Instantiations

```

instance interval :: (semigroup-add) semigroup-add
<proof>

```

```

instance interval :: (monoid-add) monoid-add
<proof>

```

```

instance interval :: (ab-semigroup-add) ab-semigroup-add
<proof>

```


instance *interval* :: (*comm-monoid-add*) *comm-monoid-add* \langle *proof* \rangle

Intervals do not form an additive group, but satisfy some properties.

lemma *interval-uminus-zero*[*simp*]:
shows $-(0 :: 'a :: \text{group-add } \textit{interval}) = 0$
 \langle *proof* \rangle

lemma *interval-diff-zero*[*simp*]:
fixes $a :: 'a :: \text{cancel-comm-monoid-add } \textit{interval}$
shows $a - 0 = a$ \langle *proof* \rangle

Without type invariant, intervals do not form a multiplicative monoid, but satisfy some properties.

instance *interval* :: (*linorder,mult-zero*) *mult-zero*
 \langle *proof* \rangle

14.3 Membership

fun *in-interval* :: '*a* :: *order* \Rightarrow '*a* *interval* \Rightarrow *bool* ((-/ \in_i -) [51, 51] 50) **where**
 $y \in_i \textit{Interval } lx \ ux = (lx \leq y \wedge y \leq ux)$

lemma *in-interval-to-interval*[*intro!*]: $a \in_i \textit{to-interval } a$
 \langle *proof* \rangle

lemma *plus-in-interval*:
fixes $x \ y :: 'a :: \text{ordered-comm-monoid-add}$
shows $x \in_i X \Longrightarrow y \in_i Y \Longrightarrow x + y \in_i X + Y$
 \langle *proof* \rangle

lemma *uminus-in-interval*:
fixes $x :: 'a :: \text{ordered-ab-group-add}$
shows $x \in_i X \Longrightarrow -x \in_i -X$
 \langle *proof* \rangle

lemma *minus-in-interval*:
fixes $x \ y :: 'a :: \text{ordered-ab-group-add}$
shows $x \in_i X \Longrightarrow y \in_i Y \Longrightarrow x - y \in_i X - Y$
 \langle *proof* \rangle

lemma *times-in-interval*:
fixes $x \ y :: 'a :: \text{linordered-ring}$
assumes $x \in_i X \ y \in_i Y$
shows $x * y \in_i X * Y$
 \langle *proof* \rangle

14.4 Convergence

definition *interval-tendsto* :: (*nat* \Rightarrow '*a* :: *topological-space interval*) \Rightarrow '*a* \Rightarrow *bool*

(**infix** \longrightarrow_i 55) **where**
 $(X \longrightarrow_i x) \equiv ((\text{interval.upper} \circ X) \longrightarrow x) \wedge ((\text{interval.lower} \circ X) \longrightarrow x)$

lemma *interval-tendstoI*[intro]:
assumes $(\text{interval.upper} \circ X) \longrightarrow x$ **and** $(\text{interval.lower} \circ X) \longrightarrow x$
shows $X \longrightarrow_i x$
 $\langle \text{proof} \rangle$

lemma *const-interval-tendsto*: $(\lambda i. \text{to-interval } a) \longrightarrow_i a$
 $\langle \text{proof} \rangle$

lemma *interval-tendsto-0*: $(\lambda i. 0) \longrightarrow_i 0$
 $\langle \text{proof} \rangle$

lemma *plus-interval-tendsto*:
fixes $x y :: 'a :: \text{topological-monoid-add}$
assumes $X \longrightarrow_i x$ $Y \longrightarrow_i y$
shows $(\lambda i. X i + Y i) \longrightarrow_i x + y$
 $\langle \text{proof} \rangle$

lemma *uminus-interval-tendsto*:
fixes $x :: 'a :: \text{topological-group-add}$
assumes $X \longrightarrow_i x$
shows $(\lambda i. - X i) \longrightarrow_i -x$
 $\langle \text{proof} \rangle$

lemma *minus-interval-tendsto*:
fixes $x y :: 'a :: \text{topological-group-add}$
assumes $X \longrightarrow_i x$ $Y \longrightarrow_i y$
shows $(\lambda i. X i - Y i) \longrightarrow_i x - y$
 $\langle \text{proof} \rangle$

lemma *times-interval-tendsto*:
fixes $x y :: 'a :: \{\text{linorder-topology, real-normed-algebra}\}$
assumes $X \longrightarrow_i x$ $Y \longrightarrow_i y$
shows $(\lambda i. X i * Y i) \longrightarrow_i x * y$
 $\langle \text{proof} \rangle$

lemma *interval-tendsto-neq*:
fixes $a b :: \text{real}$
assumes $(\lambda i. f i) \longrightarrow_i a$ **and** $a \neq b$
shows $\exists n. \neg b \in_i f n$
 $\langle \text{proof} \rangle$

14.5 Complex Intervals

datatype *complex-interval* = *Complex-Interval* (*Re-interval*: *real interval*) (*Im-interval*: *real interval*)

definition *in-complex-interval* :: *complex* \Rightarrow *complex-interval* \Rightarrow *bool* ((-/ \in_c -)
[51, 51] 50) **where**
 $y \in_c x \equiv (\text{case } x \text{ of } \text{Complex-Interval } r \ i \Rightarrow \text{Re } y \in_i r \wedge \text{Im } y \in_i i)$

instantiation *complex-interval* :: *comm-monoid-add* **begin**

definition $0 \equiv \text{Complex-Interval } 0 \ 0$

fun *plus-complex-interval* :: *complex-interval* \Rightarrow *complex-interval* \Rightarrow *complex-interval*
where
 $\text{Complex-Interval } rx \ ix + \text{Complex-Interval } ry \ iy = \text{Complex-Interval } (rx + ry)$
 $(ix + iy)$

instance
 $\langle \text{proof} \rangle$
end

lemma *plus-complex-interval*: $x \in_c X \Rightarrow y \in_c Y \Rightarrow x + y \in_c X + Y$
 $\langle \text{proof} \rangle$

definition *of-int-complex-interval* :: *int* \Rightarrow *complex-interval* **where**
of-int-complex-interval $x = \text{Complex-Interval } (\text{of-int-interval } x) \ 0$

lemma *of-int-complex-interval-0[simp]*: *of-int-complex-interval* $0 = 0$
 $\langle \text{proof} \rangle$

lemma *of-int-complex-interval*: *of-int* $i \in_c \text{of-int-complex-interval } i$
 $\langle \text{proof} \rangle$

instantiation *complex-interval* :: *mult-zero* **begin**

fun *times-complex-interval* **where**
 $\text{Complex-Interval } rx \ ix * \text{Complex-Interval } ry \ iy =$
 $\text{Complex-Interval } (rx * ry - ix * iy) (rx * iy + ix * ry)$

instance
 $\langle \text{proof} \rangle$
end

instantiation *complex-interval* :: *minus* **begin**

fun *minus-complex-interval* **where**
 $\text{Complex-Interval } R \ I - \text{Complex-Interval } R' \ I' = \text{Complex-Interval } (R - R')$
 $(I - I')$

instance $\langle \text{proof} \rangle$
end

lemma *times-complex-interval*: $x \in_c X \implies y \in_c Y \implies x * y \in_c X * Y$
 ⟨proof⟩

definition *ipoly-complex-interval* :: *int poly* \Rightarrow *complex-interval* \Rightarrow *complex-interval*
where
ipoly-complex-interval p $x = \text{fold-coeffs } (\lambda a \ b. \text{of-int-complex-interval } a + x * b)$
 $p \ 0$

lemma *ipoly-complex-interval-0*[*simp*]:
ipoly-complex-interval 0 $x = 0$
 ⟨proof⟩

lemma *ipoly-complex-interval-pCons*[*simp*]:
ipoly-complex-interval ($p\text{Cons } a \ p$) $x = \text{of-int-complex-interval } a + x * (\text{ipoly-complex-interval } p \ x)$
 ⟨proof⟩

lemma *ipoly-complex-interval*: **assumes** $x: x \in_c X$
shows *ipoly* $p \ x \in_c \text{ipoly-complex-interval } p \ X$
 ⟨proof⟩

definition *complex-interval-tendsto* (**infix** \longrightarrow_c 55) **where**
 $C \longrightarrow_c c \equiv ((\text{Re-interval} \circ C) \longrightarrow_i \text{Re } c) \wedge ((\text{Im-interval} \circ C) \longrightarrow_i \text{Im } c)$

lemma *complex-interval-tendstoI*[*intro!*]:
 $(\text{Re-interval} \circ C) \longrightarrow_i \text{Re } c \implies (\text{Im-interval} \circ C) \longrightarrow_i \text{Im } c \implies C \longrightarrow_c c$
 ⟨proof⟩

lemma *of-int-complex-interval-tendsto*: $(\lambda i. \text{of-int-complex-interval } n) \longrightarrow_c \text{of-int } n$
 ⟨proof⟩

lemma *Im-interval-plus*: *Im-interval* $(A + B) = \text{Im-interval } A + \text{Im-interval } B$
 ⟨proof⟩

lemma *Re-interval-plus*: *Re-interval* $(A + B) = \text{Re-interval } A + \text{Re-interval } B$
 ⟨proof⟩

lemma *Im-interval-minus*: *Im-interval* $(A - B) = \text{Im-interval } A - \text{Im-interval } B$
 ⟨proof⟩

lemma *Re-interval-minus*: *Re-interval* $(A - B) = \text{Re-interval } A - \text{Re-interval } B$
 ⟨proof⟩

lemma *Re-interval-times*: *Re-interval* $(A * B) = \text{Re-interval } A * \text{Re-interval } B -$

*Im-interval A * Im-interval B*
 ⟨proof⟩

lemma *Im-interval-times: Im-interval (A * B) = Re-interval A * Im-interval B + Im-interval A * Re-interval B*
 ⟨proof⟩

lemma *plus-complex-interval-tendsto:*
 $A \longrightarrow_c a \implies B \longrightarrow_c b \implies (\lambda i. A i + B i) \longrightarrow_c a + b$
 ⟨proof⟩

lemma *minus-complex-interval-tendsto:*
 $A \longrightarrow_c a \implies B \longrightarrow_c b \implies (\lambda i. A i - B i) \longrightarrow_c a - b$
 ⟨proof⟩

lemma *times-complex-interval-tendsto:*
 $A \longrightarrow_c a \implies B \longrightarrow_c b \implies (\lambda i. A i * B i) \longrightarrow_c a * b$
 ⟨proof⟩

lemma *ipoly-complex-interval-tendsto:*
assumes $C \longrightarrow_c c$
shows $(\lambda i. \text{ipoly-complex-interval } p (C i)) \longrightarrow_c \text{ipoly } p c$
 ⟨proof⟩

lemma *complex-interval-tendsto-neq: assumes* $(\lambda i. f i) \longrightarrow_c a$
and $a \neq b$
shows $\exists n. \neg b \in_c f n$
 ⟨proof⟩

end

15 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.

For factorization of polynomials with complex algebraic coefficients, there is a separate AFP entry "Factor-Algebraic-Polynomial".

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

15.1 Complex Roots

hide-const (**open**) *UnivPoly.coeff*

hide-const (**open**) *Module.smult*

hide-const (**open**) *Coset.order*

abbreviation *complex-of-int-poly* :: *int poly* \Rightarrow *complex poly* **where**
complex-of-int-poly \equiv *map-poly of-int*

abbreviation *complex-of-rat-poly* :: *rat poly* \Rightarrow *complex poly* **where**
complex-of-rat-poly \equiv *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*)

lemma *represents-cnj*: **assumes** *p* *represents* *x* **shows** *p* *represents* (*cnj* *x*)
(*proof*)

definition *poly-2i* :: *int poly* **where**
poly-2i \equiv [: 4, 0, 1:]

lemma *represents-2i*: *poly-2i* *represents* (2 * i)
(*proof*)

definition *root-poly-Re* :: *int poly* \Rightarrow *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 (*fi*, *i*). *fi* * 2 ^ *i*) (*zip* *fs* [0..*k*]))))
(*proof*)

definition *root-poly-Im* :: *int poly* \Rightarrow *int poly list* **where**
root-poly-Im *p* = (*let* *fs* = *factors-of-int-poly*
(*poly-add* *p* (*poly-uminus* *p*))
in *remdups* ((*if* (\exists *f* \in *set* *fs*. *coeff* *f* 0 = 0) then [[:0,1:]] else [])) @
[*cf-pos-poly* (*poly-div* *f* *poly-2i*) . *f* \leftarrow *fs*, *coeff* *f* 0 \neq 0])

lemma *represents-root-poly*:
assumes *ipoly* *p* *x* = 0 **and** *p*: *p* \neq 0
shows (*root-poly-Re* *p*) *represents* (*Re* *x*)
and \exists *q* \in *set* (*root-poly-Im* *p*). *q* *represents* (*Im* *x*)
(*proof*)

definition *complex-poly* :: *int poly* \Rightarrow *int poly* \Rightarrow *int poly list* **where**
complex-poly *re im* = (let *i* = [:1,0,1:]
in *factors-of-int-poly* (*poly-add* *re* (*poly-mult* *im* *i*)))

lemma *complex-poly*: **assumes** *re*: *re* represents (*Re* *x*)
and *im*: *im* represents (*Im* *x*)
shows $\exists f \in \text{set } (\text{complex-poly } re \text{ im}). f \text{ represents } x \wedge f. f \in \text{set } (\text{complex-poly } re \text{ im}) \implies \text{poly-cond } f$
<proof>

lemma *algebraic-complex-iff*: *algebraic* *x* = (*algebraic* (*Re* *x*) \wedge *algebraic* (*Im* *x*))
<proof>

definition *algebraic-complex* :: *complex* \Rightarrow *bool* **where**
[*simp*]: *algebraic-complex* = *algebraic*

lemma *algebraic-complex-code-unfold*[*code-unfold*]: *algebraic* = *algebraic-complex*
<proof>

lemma *algebraic-complex-code*[*code*]:
algebraic-complex *x* = (*algebraic* (*Re* *x*) \wedge *algebraic* (*Im* *x*))
<proof>

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* \Rightarrow '*a* \Rightarrow *bool*) \Rightarrow '*a* list \Rightarrow '*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*
<proof>

lemma *distinct-remdups-equal-3*: *distinct* (*map* *real-of-3* (*remdups-gen* *equal-3* *xs*))
<proof>

lemma *real-of-3-code* [*code*]: *real-of-3* *x* = *real-of* (*Real-Alg-Quotient* *x*)
<proof>

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** $p: p \neq 0$ **and** *ipoly* $p\ x = 0$
shows $\text{Re } x \in \text{real-of-3 ' set (real-parts-3 } p)$
 $\langle \text{proof} \rangle$

lemma *distinct-real-parts-3*: *distinct (map real-of-3 (real-parts-3 p))*
 $\langle \text{proof} \rangle$

lemma *pos-imaginary-parts-3*: **assumes** $p: p \neq 0$ **and** *ipoly* $p\ x = 0$ **and** $\text{Im } x > 0$
shows $\text{Im } x \in \text{real-of-3 ' set (pos-imaginary-parts-3 } p)$
 $\langle \text{proof} \rangle$

lemma *distinct-pos-imaginary-parts-3*: *distinct (map real-of-3 (pos-imaginary-parts-3 p))*
 $\langle \text{proof} \rangle$

lemma *remdups-gen-subset*: $\text{set (remdups-gen eq } xs) \subseteq \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *positive-pos-imaginary-parts-3*: **assumes** $x \in \text{set (pos-imaginary-parts-3 } p)$
shows $0 < \text{real-of-3 } x$
 $\langle \text{proof} \rangle$

definition *pair-to-complex* $ri \equiv \text{case } ri \text{ of } (r,i) \Rightarrow \text{Complex (real-of-3 } r) (\text{real-of-3 } i)$

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)
 $| \text{get-itvl-2 (Rational } r) = (\text{let } rr = \text{of-rat } r \text{ in Interval } rr\ rr)$

lemma *get-bounds-2*: **assumes** *invariant-2* x
shows $\text{real-of-2 } x \in_i \text{get-itvl-2 } x$
 $\langle \text{proof} \rangle$

lift-definition *get-itvl-3* :: *real-alg-3* \Rightarrow *real interval* **is** *get-itvl-2* $\langle \text{proof} \rangle$

lemma *get-itvl-3*: $\text{real-of-3 } x \in_i \text{get-itvl-3 } x$
 $\langle \text{proof} \rangle$

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')
 $| \text{tighten-bounds-2 (Rational } r) = \text{Rational } r$

lemma *tighten-bounds-2*: **assumes** *inv*: *invariant-2* x
shows $\text{real-of-2 (tighten-bounds-2 } x) = \text{real-of-2 } x \text{ invariant-2 (tighten-bounds-2 } x)$

x)
 $get-itvl-2\ x = Interval\ l\ r \implies$
 $get-itvl-2\ (tighten-bounds-2\ x) = Interval\ l'\ r' \implies r' - l' = (r-l) / 2$
 ⟨proof⟩

lift-definition $tighten-bounds-3 :: real-alg-3 \Rightarrow real-alg-3$ is $tighten-bounds-2$
 ⟨proof⟩

lemma $tighten-bounds-3$:
 $real-of-3\ (tighten-bounds-3\ x) = real-of-3\ x$
 $get-itvl-3\ x = Interval\ l\ r \implies$
 $get-itvl-3\ (tighten-bounds-3\ x) = Interval\ l'\ r' \implies r' - l' = (r-l) / 2$
 ⟨proof⟩

partial-function $(tailrec)\ filter-list-length$
 $:: ('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow bool) \Rightarrow nat \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
 [code]: $filter-list-length\ f\ p\ n\ xs = (let\ ys = filter\ p\ xs$
 $in\ if\ length\ ys = n\ then\ ys\ else$
 $filter-list-length\ f\ p\ n\ (map\ f\ ys))$

lemma $filter-list-length$: **assumes** $length\ (filter\ P\ xs) = n$
and $\bigwedge i\ x. x \in set\ xs \implies P\ x \implies p\ ((f \sim i)\ x)$
and $\bigwedge x. x \in set\ xs \implies \neg P\ x \implies \exists i. \neg p\ ((f \sim i)\ x)$
and $g: \bigwedge x. g\ (f\ x) = g\ x$
and $P: \bigwedge x. P\ (f\ x) = P\ x$
shows $map\ g\ (filter-list-length\ f\ p\ n\ xs) = map\ g\ (filter\ P\ xs)$
 ⟨proof⟩

definition $complex-roots-of-int-poly3 :: int\ poly \Rightarrow complex\ list$ **where**
 $complex-roots-of-int-poly3\ p \equiv let\ n = degree\ p;$
 $rrts = real-roots-of-int-poly\ p;$
 $nr = length\ rrts;$
 $crts = map\ (\lambda r. Complex\ r\ 0)\ rrts$
 in
 $if\ n = nr\ then\ crts$
 $else\ let\ nr-crts = n - nr\ in\ if\ nr-crts = 2\ then$
 $let\ pp = real-of-int-poly\ p\ div\ (prod-list\ (map\ (\lambda x. [:-x,1:])\ rrts));$
 $cpp = map-poly\ (\lambda r. Complex\ r\ 0)\ pp$
 $in\ crts @ roots2\ cpp\ else$
 let
 $nr-pos-crts = nr-crts\ div\ 2;$
 $rxs = real-parts-3\ p;$
 $ixs = pos-imaginary-parts-3\ p;$
 $rts = [(rx, ix). rx <- rxs, ix <- ixs];$
 $crts' = map\ pair-to-complex$
 $(filter-list-length\ (map-prod\ tighten-bounds-3\ tighten-bounds-3)$
 $(\lambda (r, i). 0 \in_c\ ipoly-complex-interval\ p\ (Complex-Interval\ (get-itvl-3\ r)$
 $(get-itvl-3\ i)))\ nr-pos-crts\ rts)$

in crts @ (concat (map (λ x. [x, conj x]) crts')

definition *complex-roots-of-int-poly-all* :: int poly ⇒ complex list **where**
complex-roots-of-int-poly-all p = (let n = degree p in
 if n ≥ 3 then *complex-roots-of-int-poly3* p
 else if n = 1 then [roots1 (map-poly of-int p)] else if n = 2 then *croots2* (map-poly
 of-int p)
 else [])

lemma *in-real-itol-get-bounds-tighten*: real-of-3 x ∈_i get-itol-3 ((tighten-bounds-3
 ~ n) x)
 ⟨proof⟩

lemma *sandwich-real*:
 fixes l r :: nat ⇒ real
 assumes la: l → a and ra: r → a
 and lm: ∧i. l i ≤ m i and mr: ∧i. m i ≤ r i
 shows m → a
 ⟨proof⟩

lemma *real-of-tighten-bounds-many[simp]*: real-of-3 ((tighten-bounds-3 ~ i) x) =
 real-of-3 x
 ⟨proof⟩

definition *lower-3* **where** lower-3 x i ≡ interval.lower (get-itol-3 ((tighten-bounds-3
 ~ i) x))

definition *upper-3* **where** upper-3 x i ≡ interval.upper (get-itol-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) / 2ⁱ
 ⟨proof⟩

lemma *interval-size-3-tendsto-0*: (λi. (upper-3 x i - lower-3 x i)) → 0
 ⟨proof⟩

lemma *dist-tendsto-0-imp-tendsto*: (λi. |f i - a| :: real) → 0 ⇒ f → a
 ⟨proof⟩

lemma *upper-3-tendsto*: upper-3 x → real-of-3 x
 ⟨proof⟩

lemma *lower-3-tendsto*: lower-3 x → real-of-3 x
 ⟨proof⟩

lemma *tends-to-tight-bounds-3*: (λx. get-itol-3 ((tighten-bounds-3 ~ x) y)) →_i
 real-of-3 y
 ⟨proof⟩

lemma *complex-roots-of-int-poly3*: **assumes** $p: p \neq 0$ **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)
 $\langle proof \rangle$

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)
 $\langle proof \rangle$

It now comes the preferred function to compute complex roots of an integer polynomial.

definition *complex-roots-of-int-poly* :: *int poly* \Rightarrow *complex list* **where**
complex-roots-of-int-poly $p =$ (
let $ps = (if\ degree\ p \geq 3\ then\ factors-of-int-poly\ p\ else\ [p])$
in $concat (map\ complex-roots-of-int-poly-all\ ps)$)

definition *complex-roots-of-rat-poly* :: *rat poly* \Rightarrow *complex list* **where**
complex-roots-of-rat-poly $p = complex-roots-of-int-poly (snd (rat-to-int-poly\ p))$

lemma *complex-roots-of-int-poly*:
shows $p \neq 0 \implies set (complex-roots-of-int-poly\ p) = \{x. ipoly\ p\ x = 0\}$ (**is** $- \implies ?l = ?r$)
and *distinct* (*complex-roots-of-int-poly* p)
 $\langle proof \rangle$

lemma *complex-roots-of-rat-poly*:
 $p \neq 0 \implies set (complex-roots-of-rat-poly\ p) = \{x. rpoly\ p\ x = 0\}$ (**is** $- \implies ?l = ?r$)
distinct (*complex-roots-of-rat-poly* p)
 $\langle proof \rangle$

lemma *min-int-poly-complex-of-real[simp]*: *min-int-poly* (*complex-of-real* x) = *min-int-poly* x
 $\langle proof \rangle$

TODO: the implementation might be tuned, since the search process should be faster when using interval arithmetic to figure out the correct factor. (One might also implement the search via checking $ipoly\ f\ x = (0::'a)$, but because of complex-algebraic-number arithmetic, I think that search would be slower than the current one via $x \in set (complex-roots-of-int-poly\ f)$)

definition *min-int-poly-complex* :: *complex* \Rightarrow *int poly* **where**
min-int-poly-complex $x = (if\ algebraic\ x\ then\ if\ Im\ x = 0\ then\ min-int-poly-real (Re\ x)$
else\ the (find (\lambda\ f. x \in set (complex-roots-of-int-poly\ f)) (complex-poly (min-int-poly (Re\ x)) (min-int-poly (Im\ x)))))

else [:0,1:])

lemma *min-int-poly-complex*[code-unfold]: *min-int-poly* = *min-int-poly-complex*
<proof>

end

16 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 *showsp-real-alg* :: *real-alg* *showsp* **where**

showsp-real-alg *p* *x* *y* = (*show-real-alg* *x* @ *y*)

lemma *show-law-real-alg* [*show-law-intros*]:

show-law *showsp-real-alg* *r*

<proof>

lemma *showsp-real-alg-append* [*show-law-simps*]:

showsp-real-alg *p* *r* (*x* @ *y*) = *showsp-real-alg* *p* *r* *x* @ *y*

<proof>

<ML>

derive *show real-alg*

We now define *show-real*.

overloading *show-real* \equiv *show-real*

begin

definition *show-real* \equiv *show-real-alg* *o* *real-alg-of-real*

end

end

17 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  $\equiv$  show-real-alg
begin

definition show-real-alg[code]: show-real-alg x  $\equiv$  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

end

```

18 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  $\times$  nat  $\Rightarrow$  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 (b*b - 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* \Rightarrow *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 \implies *real-of* x = *of-rat* r

```

    real-alg-show-info x = Sqrt-Info r sq  $\implies$  real-of x = of-rat r + sqrt (of-rat sq)
    real-alg-show-info x = Real-Alg-Info p n  $\implies$  p represents (real-of x)  $\wedge$  n = card
    {y. y  $\leq$  real-of x  $\wedge$  ipoly p y = 0}
    (is ?l  $\implies$  ?r)
    <proof>

```

fun *show-rai-info* :: *int* \Rightarrow *real-alg-show-info* \Rightarrow *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* \equiv *show-real-alg*

begin

definition *show-real-alg*[code]:

```

    show-real-alg x  $\equiv$  show-rai-info (floor x) (real-alg-show-info x)

```

end

end

19 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
    Complex-Algebraic-Numbers
    Show-Real-Precise

```

begin

19.1 Stand-Alone Examples

abbreviation $(input)$ *show-lines* $x \equiv shows-lines\ x\ Nil$

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*)

Compute real and complex roots of a polynomial with rational coefficients.

value [code] *show* (*complex-roots-of-rat-poly testpoly*)

value [code] *show* (*real-roots-of-rat-poly testpoly*)

A sequence of calculations.

value [code] *show* ($- \sqrt{2} - \sqrt{3}$)

lemma $\text{root } 3\ 4 > \sqrt{\text{root } 4\ 3} + [1/10 * \text{root } 3\ 7]$ <proof>

lemma $\text{csqrt } (4 + 3 * i) \notin \mathbf{R}$ <proof>

value [code] *show* ($\text{csqrt } (4 + 3 * i)$)

value [code] *show* ($\text{csqrt } (1 + i)$)

19.2 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.

definition *eigenvalues* :: rat mat ⇒ complex list **where**

```
  eigenvalues A = 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 ← eigenvalues
testmat]))

```

```

value [code] char-poly testmat

```

```

value [code] spectral-radius-test

```

```

end

```

20 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 *int/nat*, *rat*, *string*, and *int poly*, resp.

```

definition decompose-rat = map-prod integer-of-int integer-of-int o quotient-of

```

20.1 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 of-rational-ra = ((λ (num, denom). of-rat-real-alg (Rat.Fract (int-of-integer
num) (int-of-integer denom)))

```

```

:: 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 *sgn* :: *real-alg* ⇒ *integer*)
definition *decompose-ra* = (*map-sum* *decompose-rat* (*map-prod* (*map integer-of-int* o *coeffs*) *integer-of-nat*) o *info-real-alg* :: *real-alg* ⇒ *integer* × *integer* + *integer list* × *integer*)

20.2 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* = ((λ (*num*, *denom*). *of-rat* (*Rat.Fract* (*int-of-integer num*) (*int-of-integer denom*))) :: *integer* × *integer* ⇒ *complex*)
definition *of-real-imag-ca* = ((λ (*real*, *imag*). *Complex* (*real-of-real*) (*real-of-imag*)) :: *real-alg* × *real-alg* ⇒ *complex*)
definition *plus-ca* = ((+) :: *complex* ⇒ *complex* ⇒ *complex*)
definition *minus-ca* = ((-) :: *complex* ⇒ *complex* ⇒ *complex*)
definition *uminus-ca* = (*uminus* :: *complex* ⇒ *complex*)
definition *times-ca* = ((* :: *complex* ⇒ *complex* ⇒ *complex*)
definition *divide-ca* = ((/) :: *complex* ⇒ *complex* ⇒ *complex*)
definition *inverse-ca* = (*inverse* :: *complex* ⇒ *complex*)
definition *equals-ca* = ((=) :: *complex* ⇒ *complex* ⇒ *bool*)
definition *roots-of-poly-ca* = (*complex-roots-of-int-poly* o *poly-of-list* o *map int-of-integer* :: *integer list* ⇒ *complex list*)
definition *csqrt-ca* = (*csqrt* :: *complex* ⇒ *complex*)
definition *show-ca* = ((*String.implode* o *show*) :: *complex* ⇒ *String.literal*)
definition *real-of-ca* = (*real-alg-of-real* o *Re* :: *complex* ⇒ *real-alg*)
definition *imag-of-ca* = (*real-alg-of-real* o *Im* :: *complex* ⇒ *real-alg*)

20.3 Export Constants in Haskell

export-code

order.Eq order.Lt order.Gt — for comparison
Inl Inr — make disjoint sums available for decomposition information

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
is-rational-ra
to-rational-ra
sign-ra
decompose-ra

zero-ca
one-ca
imag-unit-ca
of-integer-ca
of-rational-ca
of-real-imag-ca
plus-ca
minus-ca
uminus-ca
times-ca
divide-ca
inverse-ca
equals-ca
roots-of-poly-ca
csqrt-ca
show-ca
real-of-ca
imag-of-ca

in *Haskell* **module-name** *Algebraic-Numbers*

end

References

- [1] M. Eberl. A decision procedure for univariate real polynomials in Isabelle/HOL. In *Proc. CPP 2015*, pages 75–83. ACM, 2015.
- [2] B. Mishra. *Algorithmic Algebra*. Texts and Monographs in Computer Science. Springer, 1993.
- [3] R. Thiemann. Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]$. *Archive of Formal Proofs*, 2014, 2014.
- [4] R. Thiemann and A. Yamada. Matrices, Jordan normal forms, and spectral radius theory. *Archive of Formal Proofs*, 2015, 2015.