Algebraic Numbers in Isabelle/HOL*

René Thiemann, Akihisa Yamada, and Sebastiaan Joosten

May 26, 2024

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

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

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 momemnt we are limited to determine the complex roots of a polynomial over $\mathbb{Q}$, and cannot determine the real or complex roots of an polynomial having arbitrary algebraic coefficients.

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

2 Auxiliary Algorithms

3 Algebraic Numbers – Excluding Addition and Multiplication

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

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

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

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

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

begin

lemma *primitive-imp-unit-iff*:
fixes $p :: 'a :: \{comm\text{-}semiring\text{-}1, semiring\text{-}no\text{-}zero\text{-}divisors\}\ poly$
assumes *pr*: *primitive p*
shows *p dvd 1* $\longleftrightarrow$ *degree p = 0*
⟨*proof*⟩

lemma *dvd-all-coeffs-imp-dvd*:
assumes $\forall a \in set\ (coeffs\ p).\ c\ dvd\ a$ **shows** $[:c:]\ dvd\ p$
⟨*proof*⟩

lemma *irreducible-content*:
fixes $p :: 'a :: \{comm\text{-}semiring\text{-}1, semiring\text{-}no\text{-}zero\text{-}divisors\}\ poly$
assumes *irreducible p* **shows** *degree p = 0* ∨ *primitive p*
⟨*proof*⟩

lemma *linear-irreducible-field*:
fixes $p :: 'a :: field\ poly$
assumes *deg*: *degree p = 1* **shows** *irreducible p*
⟨*proof*⟩

lemma *linear-irreducible-int*:
fixes $p :: int\ poly$
assumes *deg*: *degree p = 1* **and** *cp*: *content p dvd 1*
shows *irreducible p*
⟨*proof*⟩

lemma *irreducible-connect-rev*:
fixes $p :: 'a :: \{comm\text{-}semiring\text{-}1, semiring\text{-}no\text{-}zero\text{-}divisors\}\ poly$
assumes *irr*: *irreducible p* **and** *deg*: *degree p > 0*
shows $irreducible_d\ p$
⟨*proof*⟩

3.1 Polynomial Evaluation of Integer and Rational Polynomials in Fields.

abbreviation *ipoly* **where** *ipoly f x* ≡ *poly* (*of-int-poly f*) *x*

lemma *poly-map-poly-code*[*code-unfold*]: *poly* (*map-poly h p*) *x* = *fold-coeffs* (λ *a b*. *h a* + *x* ∗ *b*) *p 0*
⟨*proof*⟩

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

abbreviation *real-of-rat-poly* :: *rat poly* ⇒ *real poly* **where**

real-of-rat-poly ≡ *map-poly of-rat*

lemma *of-rat-of-int*[*simp*]: *of-rat* ∘ *of-int* = *of-int* ⟨*proof*⟩

lemma *ipoly-of-rat*[*simp*]: *ipoly p* (*of-rat y*) = *of-rat* (*ipoly p y*)
⟨*proof*⟩

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

lemma *finite-ipoly-roots*: **assumes** *p* ≠ *0*
shows *finite* {*x* :: *real*. *ipoly p x* = *0*}
⟨*proof*⟩

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 an field via $\mathbb{Z}$, we work with integer polynomials of type *int* and then use *ipoly* for evaluating the polynomial at a real or complex point.

lemma *algebraic-altdef-ipoly*:
shows *algebraic x* ⟷ (∃ *p*. *ipoly p x* = *0* ∧ *p* ≠ *0*)
⟨*proof*⟩

Definition of being algebraic with explicit witness polynomial.

definition *represents* :: *int poly* ⇒ *'a* :: *field-char-0* ⇒ *bool* (**infix** *represents 51*)
where *p represents x* = (*ipoly p x* = *0* ∧ *p* ≠ *0*)

lemma *representsI*[*intro*]: *ipoly p x* = *0* ⟹ *p* ≠ *0* ⟹ *p represents x*
⟨*proof*⟩

lemma *representsD*:
assumes *p represents x* **shows** *p* ≠ *0* **and** *ipoly p x* = *0* ⟨*proof*⟩

lemma *representsE*:
assumes *p represents x* **and** *p* ≠ *0* ⟹ *ipoly p x* = *0* ⟹ *thesis*
shows *thesis* ⟨*proof*⟩

lemma *represents-imp-degree*:
fixes *x* :: *'a* :: *field-char-0*
assumes *p represents x* **shows** *degree p* ≠ *0*
⟨*proof*⟩

lemma *representsE-full*[*elim*]:
assumes *rep*: *p represents x*
and *main*: *p* ≠ *0* ⟹ *ipoly p x* = *0* ⟹ *degree p* ≠ *0* ⟹ *thesis*
shows *thesis*
⟨*proof*⟩

lemma *represents-of-rat*[*simp*]: *p represents* (*of-rat x*) = *p represents x* ⟨*proof*⟩
lemma *represents-of-real*[*simp*]: *p represents* (*of-real x*) = *p represents x* ⟨*proof*⟩

lemma *algebraic-iff-represents*: *algebraic x* ⟷ (∃ *p*. *p represents x*)
⟨*proof*⟩

lemma *represents-irr-non-0*:
assumes *irr*: *irreducible p* **and** *ap*: *p represents x* **and** *x0*: *x* ≠ *0*
shows *poly p 0* ≠ *0*
⟨*proof*⟩

The polynomial encoding a rational number.

definition *poly-rat* :: *rat* ⇒ *int poly* **where**
poly-rat x = (*case quotient-of x of* (*n*,*d*) ⇒ [:−*n*,*d*:])

definition *abs-int-poly*:: *int poly* ⇒ *int poly* **where**
abs-int-poly p ≡ *if lead-coeff p* < *0 then* −*p else p*

lemma *pos-poly-abs-poly*[*simp*]:
shows *lead-coeff* (*abs-int-poly p*) > *0* ⟷ *p* ≠ *0*
⟨*proof*⟩

lemma *abs-int-poly-0*[*simp*]: *abs-int-poly 0* = *0*
⟨*proof*⟩

lemma *abs-int-poly-eq-0-iff*[*simp*]: *abs-int-poly p* = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *degree-abs-int-poly*[*simp*]: *degree* (*abs-int-poly p*) = *degree p*
⟨*proof*⟩

lemma *abs-int-poly-dvd*[*simp*]: *abs-int-poly p dvd q* ⟷ *p dvd q*
⟨*proof*⟩

lemma (**in** *idom*) *irreducible-uminus*[*simp*]: *irreducible* (−*x*) ⟷ *irreducible x*
⟨*proof*⟩

lemma *irreducible-abs-int-poly*[*simp*]:
irreducible (*abs-int-poly p*) ⟷ *irreducible p*
⟨*proof*⟩

lemma *coeff-abs-int-poly*[*simp*]:
coeff (*abs-int-poly p*) *n* = (*if lead-coeff p* < *0 then* − *coeff p n else coeff p n*)
⟨*proof*⟩

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

⟨*proof*⟩

lemma *ipoly-abs-int-poly-eq-zero-iff*[*simp*]:
ipoly (*abs-int-poly p*) (*x* :: *'a* :: *comm-ring-1*) = *0* ⟷ *ipoly p x* = *0*
⟨*proof*⟩

lemma *abs-int-poly-represents*[*simp*]:
abs-int-poly p represents x ⟷ *p represents x* ⟨*proof*⟩

lemma *content-pCons*[*simp*]: *content* (*pCons a p*) = *gcd a* (*content p*)
⟨*proof*⟩

lemma *content-uminus*[*simp*]:
fixes *p* :: *'a* :: *ring-gcd poly* **shows** *content* (−*p*) = *content p*
⟨*proof*⟩

lemma *primitive-abs-int-poly*[*simp*]:
primitive (*abs-int-poly p*) ⟷ *primitive p*
⟨*proof*⟩

lemma *abs-int-poly-inv*[*simp*]: *smult* (*sgn* (*lead-coeff p*)) (*abs-int-poly p*) = *p*
⟨*proof*⟩

definition *cf-pos* :: *int poly* ⇒ *bool* **where**
cf-pos p = (*content p* = *1* ∧ *lead-coeff p* > *0*)

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

lemma *sgn-is-unit*[*intro*!]:
fixes *x* :: *'a* :: *linordered-idom*
assumes *x* ≠ *0*
shows *sgn x dvd 1* ⟨*proof*⟩

lemma *cf-pos-poly-0*[*simp*]: *cf-pos-poly 0* = *0* ⟨*proof*⟩

lemma *cf-pos-poly-eq-0*[*simp*]: *cf-pos-poly f* = *0* ⟷ *f* = *0*
⟨*proof*⟩

lemma
shows *cf-pos-poly-main*: *smult* (*sgn* (*lead-coeff f*) ∗ *content f*) (*cf-pos-poly f*) =
f (**is** *?g1*)

and *content-cf-pos-poly*[*simp*]: *content* (*cf-pos-poly f*) = (*if f* = *0 then 0 else 1*) (**is** *?g2*)
and *lead-coeff-cf-pos-poly*[*simp*]: *lead-coeff* (*cf-pos-poly f*) > *0* ⟷ *f* ≠ *0* (**is** *?g3*)
and *cf-pos-poly-dvd*[*simp*]: *cf-pos-poly f dvd f* (**is** *?g4*)
⟨*proof*⟩

lemma *irreducible-connect-int*:
fixes *p* :: *int poly*
assumes *ir*: *irreducible*$_d$ *p* **and** *c*: *content p* = *1*
shows *irreducible p*
⟨*proof*⟩

lemma
fixes *x* :: *'a* :: {*idom*,*ring-char-0*}
shows *ipoly-cf-pos-poly-eq-0*[*simp*]: *ipoly* (*cf-pos-poly p*) *x* = *0* ⟷ *ipoly p x* = *0*
and *degree-cf-pos-poly*[*simp*]: *degree* (*cf-pos-poly p*) = *degree p*
and *cf-pos-cf-pos-poly*[*intro*]: *p* ≠ *0* ⟹ *cf-pos* (*cf-pos-poly p*)
⟨*proof*⟩

lemma *cf-pos-poly-eq-1*: *cf-pos-poly f* = *1* ⟷ *degree f* = *0* ∧ *f* ≠ *0* (**is** *?l* ⟷ *?r*)
⟨*proof*⟩

lemma *irr-cf-poly-rat*[*simp*]: *irreducible* (*poly-rat x*)
lead-coeff (*poly-rat x*) > *0 primitive* (*poly-rat x*)
⟨*proof*⟩

lemma *poly-rat*[*simp*]: *ipoly* (*poly-rat x*) (*of-rat x* :: *'a* :: *field-char-0*) = *0 ipoly* (*poly-rat x*) *x* = *0*
poly-rat x ≠ *0 ipoly* (*poly-rat x*) *y* = *0* ⟷ *y* = (*of-rat x* :: *'a*)
⟨*proof*⟩

lemma *poly-rat-represents-of-rat*: (*poly-rat x*) *represents* (*of-rat x*) ⟨*proof*⟩

lemma *ipoly-smult-0-iff*: **assumes** *c*: *c* ≠ *0*
shows (*ipoly* (*smult c p*) *x* = (*0* :: *real*)) = (*ipoly p x* = *0*)
⟨*proof*⟩

lemma *not-irreducibleD*:
assumes ¬ *irreducible x* **and** *x* ≠ *0* **and** ¬ *x dvd 1*
shows ∃ *y z*. *x* = *y* ∗ *z* ∧ ¬ *y dvd 1* ∧ ¬ *z dvd 1* ⟨*proof*⟩

lemma *cf-pos-poly-represents*[*simp*]: (*cf-pos-poly p*) *represents x* ⟷ *p represents x*
⟨*proof*⟩

lemma *coprime-prod*:
a ≠ *0* ⟹ *c* ≠ *0* ⟹ *coprime* (*a* ∗ *b*) (*c* ∗ *d*) ⟹ *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*) ≠ *0*
shows *degree* (*map-poly f p*) = *degree p*
⟨*proof*⟩

lemma *irreducible-cf-pos-poly*:
assumes *irr*: *irreducible p* **and** *deg*: *degree p* ≠ *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* ⟹ ∃ *z*. *hz* = *hom z* ∧ *x* = *y* ∗ *z*
begin

lemma *hom-dvd-hom-iff*[*simp*]: *hom x dvd hom y* ⟷ *x dvd y*
⟨*proof*⟩

sublocale *unit-preserving-hom*
⟨*proof*⟩

sublocale *zero-hom-0*
⟨*proof*⟩

end

lemma *smult-inverse-monom*:*p* ≠ *0* ⟹ *smult* (*inverse c*) (*p*::*rat poly*) = *1* ⟷ *p* = [: *c* :]
⟨*proof*⟩

lemma *of-int-monom*:*of-int-poly p* = [:*rat-of-int c*:] ⟷ *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*:: *ring-gcd*
shows *prime-elem x* ⟹ *gcd x y* = *normalize x* ∨ *gcd x y* = *1*
⟨*proof*⟩

lemma *irreducible-pos-gcd*:
fixes *p* :: *int poly*
assumes *ir*: *irreducible p* **and** *pos*: *lead-coeff p* > *0* **shows** *gcd p q* ∈ {*1*,*p*}
⟨*proof*⟩

lemma *irreducible-pos-gcd-twice*:
fixes *p q* :: *int poly*
assumes *p*: *irreducible p lead-coeff p* > *0*
and *q*: *irreducible q lead-coeff q* > *0*
shows *gcd p q* = *1* ∨ *p* = *q*
⟨*proof*⟩

interpretation *of-rat-hom*: *field-hom-0'* *of-rat*⟨*proof*⟩

lemma *poly-zero-imp-not-unit*:
assumes *poly p x* = *0* **shows** ¬ *p dvd 1*
⟨*proof*⟩

lemma *poly-prod-mset-zero-iff*:
fixes *x* :: *'a* :: *idom*
shows *poly* (*prod-mset F*) *x* = *0* ⟷ (∃*f* ∈# *F*. *poly f x* = *0*)
⟨*proof*⟩

lemma *algebraic-imp-represents-irreducible*:
fixes *x* :: *'a* :: *field-char-0*
assumes *algebraic x*
shows ∃*p*. *p represents x* ∧ *irreducible p*
⟨*proof*⟩

lemma *algebraic-imp-represents-irreducible-cf-pos*:
assumes *algebraic* (*x*::*'a*::*field-char-0*)
shows ∃*p*. *p represents x* ∧ *irreducible p* ∧ *lead-coeff p* > *0* ∧ *primitive p*
⟨*proof*⟩

lemma *gcd-of-int-poly*: *gcd* (*of-int-poly f*) (*of-int-poly g* :: *'a* :: {*field-char-0*,*field-gcd*}
poly) =
smult (*inverse* (*of-int* (*lead-coeff* (*gcd f g*)))) (*of-int-poly* (*gcd f g*))
⟨*proof*⟩

lemma *algebraic-imp-represents-unique*:
fixes *x* :: *'a* :: {*field-char-0*,*field-gcd*}
assumes *algebraic x*
shows ∃! *p*. *p represents x* ∧ *irreducible p* ∧ *lead-coeff p* > *0* (**is** *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* ⇒ $'a$ *poly* **where** [*code del*]:
poly-uminus $p \equiv \sum i{\le}$*degree* p. *monom* $((-1)\hat{}i * coeff\ p\ i)\ i$

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* ⇒ $'a$ *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* ⟷ *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 *content* (*poly-uminus-inner as*) = *content* (*Poly as*)
⟨*proof*⟩

Multiplicative inverse is represented by *reflect-poly*.

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

lemma (**in** *inj-idom-hom*) *reflect-poly-hom*:
reflect-poly (*map-poly hom p*) = *map-poly hom* (*reflect-poly p*)
⟨*proof*⟩

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

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

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

context
fixes $n :: nat$
begin

Polynomial for n-th root.

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

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

context
assumes n: $n \neq 0$
begin
lemma *poly-nth-root-0*[*simp*]: *poly-nth-root p* = $0 \longleftrightarrow p = 0$
⟨*proof*⟩

lemma *represents-nth-root*:
assumes y: $y \hat{} n = x$ **and** *alg*: *p represents x*
shows (*poly-nth-root p*) *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-csqrt*:
assumes *alg*: *p represents x* **shows** (*poly-nth-root 2 p*) *represents* (*csqrt x*)
⟨*proof*⟩

lemma *represents-sqrt*:
assumes *alg*: *p represents x* **and** *pos*: $x \geq 0$
shows (*poly-nth-root 2 p*) *represents* (*sqrt x*)
⟨*proof*⟩

lemma *represents-degree*:
assumes *p represents x* **shows** *degree p* $\neq 0$
⟨*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 *ipoly* (*poly-mult-rat-main n d p*) x = *of-int n* $\hat{}$ *degree p* $*$ *ipoly p* ($x *$ *of-int d* / *of-int n*)
⟨*proof*⟩

lemma *degree-poly-mult-rat*[*simp*]: **assumes** $r \neq 0$ **shows** *degree* (*poly-mult-rat r p*) = *degree p*
⟨*proof*⟩

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

lemma *poly-mult-rat-main-0*[*simp*]:
assumes $n \neq 0$ $d \neq 0$ **shows** *poly-mult-rat-main n d p* = 0 ⟷ p = 0
⟨*proof*⟩

lemma *poly-mult-rat-0*[*simp*]: **assumes** *r0*: $r \neq 0$ **shows** *poly-mult-rat r p* = 0 ⟷ p = 0
⟨*proof*⟩

lemma *represents-mult-rat*:
assumes *r*: $r \neq 0$ **and** *p represents x* **shows** (*poly-mult-rat r p*) *represents* (*of-rat r* $*$ x)
⟨*proof*⟩

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

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

lemma *poly-add-rat-code*[*code*]: *poly-add-rat r p* ≡ *case quotient-of r of* (*n*,*d*) ⇒
let p′ = (*let fs* = *coeffs p*; *k* = *length fs in poly-of-list* (*map* (λ(*fi*, *i*). *fi* $*$ *d* $\hat{}$ (*k* − *Suc i*)) (*zip fs* [*0*..<*k*]))); *p*″ = *p*′ ∘$_p$ [:−*n*,*d*:] *in p*″
⟨*proof*⟩

lemma *degree-poly-add-rat*[*simp*]: *degree* (*poly-add-rat r p*) = *degree p*
⟨*proof*⟩

lemma *ipoly-add-rat*: *ipoly* (*poly-add-rat r p*) x = (*of-int* (*snd* (*quotient-of r*)) $\hat{}$ *degree p*) $*$ *ipoly p* (x − *of-rat r*)
⟨*proof*⟩

lemma *poly-add-rat-0*[*simp*]: *poly-add-rat r p* = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *add-rat-roots*: *ipoly* (*poly-add-rat r p*) *x* = *0* ⟷ *ipoly p* (*x* − *of-rat r*) = *0*
⟨*proof*⟩

lemma *represents-add-rat*:
assumes *p represents x* **shows** (*poly-add-rat r p*) *represents* (*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*:
ipoly (*poly-add-rat r p*) (*x*::′*a*::*linordered-field*) < *0* ⟷ *ipoly p* (*x* − *of-rat r*) < *0*
ipoly (*poly-add-rat r p*) (*x*::′*a*::*linordered-field*) > *0* ⟷ *ipoly p* (*x* − *of-rat r*) > *0*
⟨*proof*⟩

lemma *sgn-ipoly-add-rat*[*simp*]:
sgn (*ipoly* (*poly-add-rat r p*) (*x*::′*a*::*linordered-field*)) = *sgn* (*ipoly p* (*x* − *of-rat r*)) (**is** *sgn ?l* = *sgn ?r*)
⟨*proof*⟩

lemma *deg-nonzero-represents*:
assumes *deg*: *degree p* ≠ *0* **shows** ∃ *x* :: *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 *poly2* :: *'a*::*comm-semiring-1 poly poly* ⇒ *'a* ⇒ *'a* ⇒ *'a*
  where *poly2 p x y* = *poly* (*poly p* [: *y* :]) *x*

lemma *poly2-by-map*: *poly2 p x* = *poly* (*map-poly* (λ*c*. *poly c x*) *p*)
  ⟨*proof*⟩

lemma *poly2-const*[*simp*]: *poly2* [:[:*a*:]:] *x y* = *a* ⟨*proof*⟩
lemma *poly2-smult*[*simp*,*hom-distribs*]: *poly2* (*smult a p*) *x y* = *poly a x* ∗ *poly2 p x y* ⟨*proof*⟩

interpretation *poly2-hom*: *comm-semiring-hom* λ*p*. *poly2 p x y* ⟨*proof*⟩
interpretation *poly2-hom*: *comm-ring-hom* λ*p*. *poly2 p x y*⟨*proof*⟩
interpretation *poly2-hom*: *idom-hom* λ*p*. *poly2 p x y*⟨*proof*⟩

lemma *poly2-pCons*[*simp*,*hom-distribs*]: *poly2* (*pCons a p*) *x y* = *poly a x* + *y* ∗ *poly2 p x y* ⟨*proof*⟩
lemma *poly2-monom*: *poly2* (*monom a n*) *x y* = *poly a x* ∗ *y* ^ *n* ⟨*proof*⟩

lemma *poly-poly-as-poly2*: *poly2 p x* (*poly q x*) = *poly* (*poly p q*) *x* ⟨*proof*⟩

The following lemma is an extension rule for bivariate polynomials.

lemma *poly2-ext*:
  fixes *p q* :: *'a* :: {*ring-char-0*,*idom*} *poly poly*
  assumes ⋀*x y*. *poly2 p x y* = *poly2 q x y* **shows** *p* = *q*
⟨*proof*⟩

abbreviation (*input*) *coeff-lift2* == λ*a*. [:[: *a* :]:]

lemma *coeff-lift2-lift*: *coeff-lift2* = *coeff-lift* ∘ *coeff-lift* ⟨*proof*⟩

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

lemma *degree-poly-lift*[*simp*]: *degree* (*poly-lift p*) = *degree p*
  ⟨*proof*⟩

lemma *poly-lift-0*[*simp*]: *poly-lift 0* = *0* ⟨*proof*⟩

lemma *poly-lift-0-iff*[*simp*]: *poly-lift p* = *0* ⟷ *p* = *0*
  ⟨*proof*⟩

lemma *poly-lift-pCons*[*simp*]:
  poly-lift (*pCons a p*) = *pCons* [:*a*:] (*poly-lift p*)
  ⟨*proof*⟩

lemma *coeff-poly-lift*[*simp*]:
  fixes *p*:: *'a* :: *comm-monoid-add poly*
  shows *coeff* (*poly-lift p*) *i* = *coeff-lift* (*coeff p i*)
  ⟨*proof*⟩

lemma *pcompose-conv-poly*: *pcompose p q* = *poly* (*poly-lift p*) *q*
  ⟨*proof*⟩

interpretation *poly-lift-hom*: *inj-comm-monoid-add-hom poly-lift*
⟨*proof*⟩
interpretation *poly-lift-hom*: *inj-comm-semiring-hom poly-lift*
⟨*proof*⟩
interpretation *poly-lift-hom*: *inj-comm-ring-hom poly-lift*⟨*proof*⟩
interpretation *poly-lift-hom*: *inj-idom-hom poly-lift*⟨*proof*⟩

lemma (**in** *comm-monoid-add-hom*) *map-poly-hom-coeff-lift*[*simp*, *hom-distribs*]:
  map-poly hom (*coeff-lift a*) = *coeff-lift* (*hom a*) ⟨*proof*⟩

lemma (**in** *comm-ring-hom*) *map-poly-coeff-lift-hom*:
  map-poly (*coeff-lift* ∘ *hom*) *p* = *map-poly* (*map-poly hom*) (*map-poly coeff-lift p*)
⟨*proof*⟩

lemma *poly-poly-lift*[*simp*]:
  fixes *p* :: *'a* :: *comm-semiring-0 poly*
  shows *poly* (*poly-lift p*) [:*x*:] = [: *poly p x* :]
⟨*proof*⟩

lemma *degree-poly-lift2*[*simp*]:
  degree (*poly-lift2 p*) = *degree p* ⟨*proof*⟩

lemma *poly-lift2-0*[*simp*]: *poly-lift2 0* = *0* ⟨*proof*⟩

lemma *poly-lift2-0-iff*[*simp*]: *poly-lift2 p* = *0* ⟷ *p* = *0*
  ⟨*proof*⟩

lemma *poly-lift2-pCons*[*simp*]:
  poly-lift2 (*pCons a p*) = *pCons* [:[:*a*:]:] (*poly-lift2 p*)
  ⟨*proof*⟩

lemma *poly-lift2-lift*: *poly-lift2* = *poly-lift* ∘ *poly-lift* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *poly2-poly-lift*[*simp*]: *poly2* (*poly-lift p*) *x y* = *poly p y* ⟨*proof*⟩

lemma *poly-lift2-nonzero*:
  assumes *p* ≠ *0* **shows** *poly-lift2 p* ≠ *0*
  ⟨*proof*⟩

4.1.2 Swapping the Order of Variables

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

lemma *poly-y-x-fix-y-deg*:
assumes *ydeg*: ∀ *i*≤*degree p*. *degree* (*coeff p i*) ≤ *d*
shows *poly-y-x p* = (∑ *i*≤*degree p*. ∑ *j*≤*d*. *monom* (*monom* (*coeff* (*coeff p i*) *j*) *i*) *j*)
(**is** - = *sum* (λ*i*. *sum* (*?f i*) -) -)
⟨*proof*⟩

lemma *poly-y-x-fixed-deg*:
fixes *p* :: *'a* :: *comm-monoid-add poly poly*
defines *d* ≡ *Max* { *degree* (*coeff p i*) | *i*. *i* ≤ *degree p* }
shows *poly-y-x p* = (∑ *i*≤*degree p*. ∑ *j*≤*d*. *monom* (*monom* (*coeff* (*coeff p i*) *j*) *i*) *j*)
⟨*proof*⟩

lemma *poly-y-x-swapped*:
fixes *p* :: *'a* :: *comm-monoid-add poly poly*
defines *d* ≡ *Max* { *degree* (*coeff p i*) | *i*. *i* ≤ *degree p* }
shows *poly-y-x p* = (∑ *j*≤*d*. ∑ *i*≤*degree p*. *monom* (*monom* (*coeff* (*coeff p i*) *j*) *i*) *j*)
⟨*proof*⟩

lemma *poly2-poly-y-x*[*simp*]: *poly2* (*poly-y-x p*) *x y* = *poly2 p y x*
⟨*proof*⟩

context begin
private lemma *poly-monom-mult*:
fixes *p* :: *'a* :: *comm-semiring-1*
shows *poly* (*monom p i* ∗ *q* ^ *j*) *y* = *poly* (*monom p j* ∗ [:*y*:] ^ *i*) (*poly q y*)
⟨*proof*⟩

lemma *poly-poly-y-x*:
fixes *p* :: *'a* :: *comm-semiring-1 poly poly*
shows *poly* (*poly* (*poly-y-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* ∀ *p q*. *h* (*p* + *q*) = *h p* + *h q*
shows *sum* (λ*i*. *map-poly h* (*f i*)) *S* = *map-poly h* (*sum f S*)

⟨*proof*⟩

lemma *poly-y-x-const*: *poly-y-x* [:*p*:] = *poly-lift p* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *poly-y-x-pCons*:
shows *poly-y-x* (*pCons a p*) = *poly-lift a* + *map-poly* (*pCons 0*) (*poly-y-x p*)
⟨*proof*⟩

lemma *poly-y-x-pCons-0*: *poly-y-x* (*pCons 0 p*) = *map-poly* (*pCons 0*) (*poly-y-x p*)
⟨*proof*⟩

lemma *poly-y-x-map-poly-pCons-0*: *poly-y-x* (*map-poly* (*pCons 0*) *p*) = *pCons 0* (*poly-y-x p*)
⟨*proof*⟩

interpretation *poly-y-x-hom*: *comm-monoid-add-hom poly-y-x* :: *'a* :: *comm-monoid-add poly poly* ⇒ -
⟨*proof*⟩

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*:]
⟨*proof*⟩

lemma *poly-y-x-id*[*simp*]:
fixes *p*:: *'a* :: *comm-monoid-add poly poly*
shows *poly-y-x* (*poly-y-x p*) = *p*
⟨*proof*⟩

interpretation *poly-y-x-hom*:
bijective poly-y-x :: *'a* :: *comm-monoid-add poly poly* ⇒ -
⟨*proof*⟩

lemma *inv-poly-y-x*[*simp*]: *Hilbert-Choice.inv poly-y-x* = *poly-y-x* ⟨*proof*⟩

interpretation *poly-y-x-hom*: *comm-monoid-add-isom poly-y-x*
⟨*proof*⟩

lemma *pCons-as-add*:
fixes *p* :: *'a* :: *comm-semiring-1 poly*
shows *pCons a p* = [:*a*:] + *monom 1 1* ∗ *p* ⟨*proof*⟩

lemma *mult-pCons-0*: (∗) (*pCons 0 1*) = *pCons 0* ⟨*proof*⟩

lemma *pCons-0-as-mult*:
shows *pCons* (*0* :: *'a* :: *comm-semiring-1*) = (λ*p*. *pCons 0 1* ∗ *p*) ⟨*proof*⟩

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

lemma *poly-y-x-monom*:
fixes $a :: 'a :: comm\text{-}semiring\text{-}1\ poly$
shows $poly\text{-}y\text{-}x\ (monom\ a\ n) = smult\ (monom\ 1\ n)\ (poly\text{-}lift\ a)$
⟨*proof*⟩

lemma *poly-y-x-smult*:
fixes $c :: 'a :: comm\text{-}semiring\text{-}1\ poly$
shows $poly\text{-}y\text{-}x\ (smult\ c\ p) = poly\text{-}lift\ c * poly\text{-}y\text{-}x\ p$ (**is** $?l = ?r$)
⟨*proof*⟩

interpretation *poly-y-x-hom*:
$comm\text{-}semiring\text{-}isom\ poly\text{-}y\text{-}x :: 'a :: comm\text{-}semiring\text{-}1\ poly\ poly \Rightarrow$ -
⟨*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*:
$Max\ \{\ degree\ (coeff\ (pCons\ a\ p)\ i)\ |\ i.\ i \leq degree\ (pCons\ a\ p)\} =$
$max\ (degree\ a)\ (Max\ \{degree\ (coeff\ p\ x)\ |x.\ x \leq degree\ p\})$
⟨*proof*⟩

lemma *degree-poly-y-x*:
fixes $p :: 'a :: comm\text{-}ring\text{-}1\ poly\ poly$
assumes $p \neq 0$
shows $degree\ (poly\text{-}y\text{-}x\ p) = Max\ \{\ degree\ (coeff\ p\ i)\ |\ i.\ i \leq degree\ p\ \}$
(**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 ≡
vec n (λ*i*. *if i* ≤ *j* ∧ *j* ≤ *degree p* + *i then coeff p* (*degree p* + *i* − *j*) *else 0*)

lemma *vec-of-poly-rev-shifted-dim*[*simp*]: *dim-vec* (*vec-of-poly-rev-shifted p n j*) = *n*
⟨*proof*⟩

lemma *col-sylvester*:
fixes *p q*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *j*: *j* < *m*+*n*
shows *col* (*sylvester-mat p q*) *j* =
vec-of-poly-rev-shifted p n j @$_v$ *vec-of-poly-rev-shifted q m j* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *inj-on-diff-nat2*: *inj-on* (λ*i*. (*n*::*nat*) − *i*) {..*n*} ⟨*proof*⟩

lemma *image-diff-atMost*: (λ*i*. (*n*::*nat*) − *i*) ‘ {..*n*} = {..*n*} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *sylvester-sum-mat-upper*:
fixes *p q* :: *'a* :: *comm-semiring-1 poly*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *i*: *i* < *n*
shows (∑*j*<*m*+*n*. *monom* (*sylvester-mat p q* $$ (*i*,*j*)) (*m* + *n* − *Suc j*)) =
monom 1 (*n* − *Suc i*) ∗ *p* (**is** *sum ?f* - = *?r*)
⟨*proof*⟩

lemma *sylvester-sum-mat-lower*:
fixes *p q* :: *'a* :: *comm-semiring-1 poly*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *ni*: *n* ≤ *i* **and** *imn*: *i* < *m*+*n*
shows (∑*j*<*m*+*n*. *monom* (*sylvester-mat p q* $$ (*i*,*j*)) (*m* + *n* − *Suc j*)) =
monom 1 (*m* + *n* − *Suc i*) ∗ *q* (**is** *sum ?f* - = *?r*)
⟨*proof*⟩

definition *vec-of-poly p* ≡ *let m* = *degree p in vec* (*Suc m*) (λ*i*. *coeff p* (*m*−*i*))

definition *poly-of-vec v* ≡ *let d* = *dim-vec v in* ∑*i*<*d*. *monom* (*v* $ (*d* − *Suc i*))
i

lemma *poly-of-vec-of-poly*[*simp*]:
fixes *p* :: *'a* :: *comm-monoid-add poly*
shows *poly-of-vec* (*vec-of-poly p*) = *p*
⟨*proof*⟩

lemma *poly-of-vec-0*[*simp*]: *poly-of-vec* (0_v *n*) = *0* ⟨*proof*⟩

lemma *poly-of-vec-0-iff*[*simp*]:
fixes *v* :: *'a* :: *comm-monoid-add vec*
shows *poly-of-vec v* = *0* ⟷ *v* = 0_v (*dim-vec v*) (**is** *?v* = - ⟷ - = *?z*)
⟨*proof*⟩

lemma *degree-sum-smaller*:
assumes *n* > *0 finite A*
shows (⋀ *x*. *x* ∈*A* ⟹ *degree* (*f x*) < *n*) ⟹ *degree* (∑ *x*∈*A*. *f x*) < *n*
⟨*proof*⟩

lemma *degree-poly-of-vec-less*:
fixes *v* :: *'a* :: *comm-monoid-add vec*
assumes *dim*: *dim-vec v* > *0*
shows *degree* (*poly-of-vec v*) < *dim-vec v*
⟨*proof*⟩

lemma *coeff-poly-of-vec*:
coeff (*poly-of-vec v*) *i* = (*if i* < *dim-vec v then v* $ (*dim-vec v* − *Suc i*) *else 0*)
(**is** *?l* = *?r*)
⟨*proof*⟩

lemma *vec-of-poly-rev-shifted-scalar-prod*:
fixes *p v*
defines *q* ≡ *poly-of-vec v*
assumes *m*[*simp*]: *degree p* = *m* **and** *n*: *dim-vec v* = *n*
assumes *j*: *j* < *m*+*n*
shows *vec-of-poly-rev-shifted p n* (*n*+*m*−*Suc j*) ⋅ *v* = *coeff* (*p* ∗ *q*) *j* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *sylvester-vec-poly*:
fixes *p q* :: *'a* :: *comm-semiring-0 poly*
defines *m* ≡ *degree p*
and *n* ≡ *degree q*
assumes *v*: *v* ∈ *carrier-vec* (*m*+*n*)
shows *poly-of-vec* (*transpose-mat* (*sylvester-mat p q*) ∗$_v$ *v*) =
poly-of-vec (*vec-first v n*) ∗ *p* + *poly-of-vec* (*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 poly*
shows *hom* (*resultant-sub m n p q*) = *resultant-sub m n* (*map-poly hom p*)
(*map-poly hom q*)
(**is** *?l* = *?r'*)

⟨*proof*⟩

4.2.3 Resultant as Polynomial Expression

context begin

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

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

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

private lemma *mk-poly-sub-dim*[*simp*]:
dim-row (mk-poly-sub A l j) = dim-row A
dim-col (mk-poly-sub A l j) = dim-col A
⟨*proof*⟩ **lemma** *mk-poly-sub-carrier*:
assumes *A ∈ carrier-mat nr nc* **shows** *mk-poly-sub A l j ∈ 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 ≠ j′* **and** *i < dim-row A* **and** *j′ < dim-col A*
shows *mk-poly-sub A l j $$ (i,j′) = A $$ (i,j′)*
⟨*proof*⟩ **lemma** *mk-poly-others*[*simp*]:
assumes *i*: *i < dim-row A* **and** *j*: *j < dim-col A − 1*
shows *mk-poly A $$ (i,j) = [: A $$ (i,j) :]*
⟨*proof*⟩ **lemma** *mk-poly-delete*[*simp*]:
assumes *i*: *i < 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 ≠ j′* **and** *j′ < dim-col A*
shows *col (mk-poly-sub A l j) j′ = col A j′*
⟨*proof*⟩ **lemma** *det-mk-poly-sub*:
assumes *A*: *(A :: 'a :: comm-ring-1 poly mat) ∈ 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 :: comm-ring-1 mat*
shows *det (mk-poly A) = [: 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 ↦ pv $ n + 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-poly2-row A m (*dim-col A − Suc m*) (*mk-poly2-col A pv m*) (*dim-row A*)

private definition *mk-poly2 A* ≡ *mk-poly2-col A* (*0*$_v$ (*dim-row A*)) (*dim-col A*)

private lemma *mk-poly2-row-dim*[*simp*]: *dim-vec* (*mk-poly2-row A d j pv i*) = *dim-vec pv*
⟨*proof*⟩ **lemma** *mk-poly2-col-dim*[*simp*]: *dim-vec* (*mk-poly2-col A pv j*) = *dim-vec pv*
⟨*proof*⟩ **lemma** *mk-poly2-row*:
assumes *n*: *n* ≤ *dim-vec pv*
shows *mk-poly2-row A d j pv n* $ *i* =
(*if i* < *n then pv* $ *i* + *monom* (*A* $$ (*i*,*j*)) *d else pv* $ *i*)
⟨*proof*⟩ **lemma** *mk-poly2-row-col*:
assumes *dim*[*simp*]: *dim-vec pv* = *n dim-row A* = *n* **and** *j*: *j* < *dim-col A*
shows *mk-poly2-row A d j pv n* = *pv* + *map-vec* (λ*a*. *monom a d*) (*col A j*)
⟨*proof*⟩ **lemma** *mk-poly2-col*:
fixes *pv* :: *'a* :: *comm-semiring-1 poly vec* **and** *A* :: *'a mat*
assumes *i*: *i* < *dim-row A* **and** *dim*: *dim-row A* = *dim-vec pv*
shows *mk-poly2-col A pv j* $ *i* = *pv* $ *i* + (∑*j'*<*j*. *monom* (*A* $$ (*i*, *dim-col A* − *Suc j'*)) *j'*)
⟨*proof*⟩ **lemma** *mk-poly2-pre*:
fixes *A* :: *'a* :: *comm-semiring-1 mat*
assumes *i*: *i* < *dim-row A*
shows *mk-poly2 A* $ *i* = (∑*j'*<*dim-col A*. *monom* (*A* $$ (*i*, *dim-col A* − *Suc j'*)) *j'*)
⟨*proof*⟩ **lemma** *mk-poly2*:
fixes *A* :: *'a* :: *comm-semiring-1 mat*
assumes *i*: *i* < *dim-row A*
and *c*: *dim-col A* > *0*
shows *mk-poly2 A* $ *i* = (∑*j'*<*dim-col A*. *monom* (*A* $$ (*i*,*j'*)) (*dim-col A* − *Suc j'*))
(**is** *?l* = *sum ?f ?S*)
⟨*proof*⟩ **lemma** *mk-poly2-sylvester-upper*:
fixes *p q* :: *'a* :: *comm-semiring-1 poly*
assumes *i*: *i* < *degree q*
shows *mk-poly2* (*sylvester-mat p q*) $ *i* = *monom 1* (*degree q* − *Suc i*) ∗ *p*
⟨*proof*⟩ **lemma** *mk-poly2-sylvester-lower*:
fixes *p q* :: *'a* :: *comm-semiring-1 poly*
assumes *mi*: *i* ≥ *degree q* **and** *imn*: *i* < *degree p* + *degree q*
shows *mk-poly2* (*sylvester-mat p q*) $ *i* = *monom 1* (*degree p* + *degree q* − *Suc i*) ∗ *q*
⟨*proof*⟩ **lemma** *foo*:
fixes *v* :: *'a* :: *comm-semiring-1 vec*
shows *monom 1 d* ·$_v$ *map-vec coeff-lift v* = *map-vec* (λ*a*. *monom a d*) *v*
⟨*proof*⟩ **lemma** *mk-poly-sub-corresp*:
assumes *dimA*[*simp*]: *dim-col A* = *Suc l* **and** *dimpv*[*simp*]: *dim-vec pv* = *dim-row A*
and *j*: *j* < *dim-col A*
shows *pv* + *col* (*mk-poly-sub* (*map-mat coeff-lift A*) *l j*) *l* =

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

lemma *mk-poly-sylvester-upper*:
fixes *p q* :: *'a* :: *comm-ring-1 poly*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *i*: *i < n*
shows *mk-poly* (*sylvester-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-ring-1 poly*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *ni*: *n* ≤ *i* **and** *imn*: *i < m+n*
shows *mk-poly* (*sylvester-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-ring-1 poly*
assumes *degp*: *degree p > 0* **and** *degq*: *degree q > 0*
shows ∃ *p' q'*. *degree p' < degree q* ∧ *degree q' < degree p* ∧
[: *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-ring-1 poly*
assumes *deg*: *degree p > 0* ∨ *degree q > 0*
and *xp*: *poly p x = 0* **and** *xq*: *poly q x = 0*
shows *resultant p q = 0*
⟨*proof*⟩

lemma *poly-resultant-zero*:
fixes *p q* :: *'a* :: *comm-ring-1 poly poly*
assumes *deg*: *degree p > 0* ∨ *degree q > 0*
assumes *p0*: *poly2 p x y = 0* **and** *q0*: *poly2 q x y = 0*

shows *poly* (*resultant p q*) *x* = *0*
⟨*proof*⟩

lemma *resultant-as-nonzero-poly-weak*:
fixes *p q* :: *'a* :: *idom poly*
assumes *degp*: *degree p* > *0* **and** *degq*: *degree q* > *0*
and *r0*: *resultant p q* ≠ *0*
shows ∃ *p' q'*. *degree p'* < *degree q* ∧ *degree q'* < *degree p* ∧
[: *resultant p q* :] = *p'* ∗ *p* + *q'* ∗ *q* ∧ *p'* ≠ *0* ∧ *q'* ≠ *0*
⟨*proof*⟩

Next lemma corresponds to Lemma 7.2.2 of the textbook

lemma *resultant-as-nonzero-poly*:
fixes *p q* :: *'a* :: *idom poly*
defines *m* ≡ *degree p* **and** *n* ≡ *degree q*
assumes *degp*: *m* > *0* **and** *degq*: *n* > *0*
shows ∃ *p' q'*. *degree p'* < *n* ∧ *degree q'* < *m* ∧
[: *resultant p q* :] = *p'* ∗ *p* + *q'* ∗ *q* ∧ *p'* ≠ *0* ∧ *q'* ≠ *0*
⟨*proof*⟩

Corresponds to Lemma 7.2.3 of the textbook

lemma *resultant-zero-imp-common-factor*:
fixes *p q* :: *'a* :: *ufd poly*
assumes *deg*: *degree p* > *0* ∨ *degree q* > *0* **and** *r0*: *resultant p q* = *0*
shows ¬ *coprime p q*
⟨*proof*⟩

lemma *resultant-non-zero-imp-coprime*:
assumes *nz*: *resultant* (*f* :: *'a* :: *field poly*) *g* ≠ *0*
and *nz'*: *f* ≠ *0* ∨ *g* ≠ *0*
shows *coprime f g*
⟨*proof*⟩

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-Irreducibility
begin

interpretation *coeff-hom*: *monoid-add-hom* λ*p*. *coeff p i* ⟨*proof*⟩

interpretation *coeff-hom*: *comm-monoid-add-hom* $\lambda p.$ *coeff p i*⟨*proof*⟩
interpretation *coeff-hom*: *group-add-hom* $\lambda p.$ *coeff p i*⟨*proof*⟩
interpretation *coeff-hom*: *ab-group-add-hom* $\lambda p.$ *coeff p i*⟨*proof*⟩
interpretation *coeff-0-hom*: *monoid-mult-hom* $\lambda p.$ *coeff p 0* ⟨*proof*⟩
interpretation *coeff-0-hom*: *semiring-hom* $\lambda p.$ *coeff p 0*⟨*proof*⟩
interpretation *coeff-0-hom*: *comm-monoid-mult-hom* $\lambda p.$ *coeff p 0*⟨*proof*⟩
interpretation *coeff-0-hom*: *comm-semiring-hom* $\lambda p.$ *coeff p 0*⟨*proof*⟩

5.1 Addition of Algebraic Numbers

definition *x-y* ≡ [: [: *0*, *1* :], −*1* :]

definition *poly-x-minus-y p* = *poly-lift p* ∘$_p$ *x-y*

lemma *coeff-xy-power*:
assumes $k \leq n$
shows *coeff* (*x-y* ^ *n* :: *'a* :: *comm-ring-1 poly poly*) *k* =
monom (*of-nat* (*n choose* (*n* − *k*)) ∗ (− *1*) ^ *k*) (*n* − *k*)
⟨*proof*⟩

The following polynomial represents the sum of two algebraic numbers.

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

5.1.1 *poly-add* has desired root

interpretation *poly-x-minus-y-hom*:
comm-ring-hom poly-x-minus-y ⟨*proof*⟩

lemma *poly2-x-y*[*simp*]:
fixes *x* :: *'a* :: *comm-ring-1*
shows *poly2 x-y x y* = *x* − *y* ⟨*proof*⟩

lemma *degree-poly-x-minus-y*[*simp*]:
fixes *p* :: *'a*::*idom poly*
shows *degree* (*poly-x-minus-y p*) = *degree p* ⟨*proof*⟩

lemma *poly-x-minus-y-pCons*[*simp*]:
poly-x-minus-y (*pCons a p*) = [:[: *a* :]:] + *poly-x-minus-y p* ∗ *x-y*
⟨*proof*⟩

lemma *poly-poly-poly-x-minus-y*[*simp*]:
fixes *p* :: *'a* :: *comm-ring-1 poly*
shows *poly* (*poly* (*poly-x-minus-y p*) *q*) *x* = *poly p* (*x* − *poly q x*)
⟨*proof*⟩

lemma *poly2-poly-x-minus-y*[*simp*]:
fixes *p* :: *'a* :: *comm-ring-1 poly*
shows *poly2* (*poly-x-minus-y p*) *x y* = *poly p* (*x*−*y*) ⟨*proof*⟩

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

lemma *x-y-nonzero*[*simp*]: *x-y* $\neq$ *0* ⟨*proof*⟩

lemma *degree-x-y*[*simp*]: *degree x-y* = *1* ⟨*proof*⟩

interpretation *x-y-mult-hom*: *inj-comm-monoid-add-hom* $\lambda p :: 'a ::$ *idom poly poly*. *x-y* $*$ *p*
⟨*proof*⟩

interpretation *poly-x-minus-y-hom*: *inj-idom-hom poly-x-minus-y*
⟨*proof*⟩

lemma *poly-add*:
fixes *p q* :: $'a ::$*comm-ring-1 poly*
assumes *q0*: *q* $\neq$ *0* **and** *x*: *poly p x* = *0* **and** *y*: *poly q y* = *0*
shows *poly* (*poly-add p q*) (*x*+*y*) = *0*
⟨*proof*⟩

5.1.2 *poly-add* is nonzero

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

interpretation *poly-lift-hom*:
unit-preserving-hom poly-lift :: $'a ::$ {*comm-semiring-1*,*semiring-no-zero-divisors*} *poly* $\Rightarrow$ -
⟨*proof*⟩

interpretation *poly-lift-hom*:
factor-preserving-hom poly-lift::$'a$::*idom poly* $\Rightarrow$ $'a$ *poly poly*
⟨*proof*⟩

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*: *poly-x-minus-y* = (λp. *p* $\circ_p$ *x-y*) $\circ$ *poly-lift*
⟨*proof*⟩
context *idom-isom* **begin**
sublocale *comm-semiring-isom*⟨*proof*⟩
end

interpretation *poly-x-minus-y-hom*:
factor-preserving-hom poly-x-minus-y :: $'a ::$ *idom poly* $\Rightarrow$ $'a$ *poly poly*
⟨*proof*⟩

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

lemma *poly-y-x-const*[*simp*]: *poly-y-x* [:[:*a*:]:] = [:[:*a*:]:] ⟨*proof*⟩

context begin

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

lemma *poly-y-x-x-y*[*simp*]: *poly-y-x x-y* = *y-x* ⟨*proof*⟩ **lemma** *y-x*[*simp*]: **fixes** *x* :: *'a* :: *comm-ring-1* **shows** *poly2 y-x x y* = *y* − *x*
⟨*proof*⟩ **definition** *poly-y-minus-x p* ≡ *poly-lift p* ∘$_p$ *y-x*

private lemma *poly-y-minus-x-0*[*simp*]: *poly-y-minus-x 0* = *0* ⟨*proof*⟩ **lemma** *poly-y-minus-x-pCons*[*simp*]:
poly-y-minus-x (*pCons a p*) = [:[: *a* :]:] + *poly-y-minus-x p* ∗ *y-x* ⟨*proof*⟩ **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*
⟨*proof*⟩

lemma *degree-poly-y-minus-x*[*simp*]:
fixes *p* :: *'a* :: *idom poly*
shows *degree* (*poly-y-x* (*poly-x-minus-y p*)) = *degree p*
⟨*proof*⟩

end

lemma *dvd-all-coeffs-iff*:
fixes *x* :: *'a* :: *comm-semiring-1*
shows (∀ *pi* ∈ *set* (*coeffs p*). *x dvd pi*) ⟷ (∀ *i*. *x dvd coeff p i*) (**is** *?l* = *?r*)
⟨*proof*⟩

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* ∈# *F*
shows *degree f* ≠ *0*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *poly-add-nonzero*:
fixes *p q* :: *'a* :: *ufd poly*
assumes *p0*: *p* ≠ *0* **and** *q0*: *q* ≠ *0* **and** *x*: *poly p x* = *0* **and** *y*: *poly q y* = *0*
and *pr*: *primitive p*
shows *poly-add p q* ≠ *0*
⟨*proof*⟩

5.1.3 Summary for addition

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

lemma (**in** *comm-ring-hom*) *map-poly-x-minus-y*:
map-poly (*map-poly hom*) (*poly-x-minus-y p*) = *poly-x-minus-y* (*map-poly hom p*)
⟨*proof*⟩

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

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

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

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

lemma (**in** *idom-hom*) *poly-add-hom*:
assumes *p0*: *hom* (*lead-coeff p*) ≠ *0* **and** *q0*: *hom* (*lead-coeff q*) ≠ *0*
shows *map-poly hom* (*poly-add p q*) = *poly-add* (*map-poly hom p*) (*map-poly hom q*)
⟨*proof*⟩

lemma(**in** *zero-hom*) *hom-lead-coeff-nonzero-imp-map-poly-hom*:
assumes *hom* (*lead-coeff p*) ≠ *0*
shows *map-poly hom p* ≠ *0*
⟨*proof*⟩

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

lemma (**in** *comm-monoid-gcd*) *gcd-list-eq-0-iff*[*simp*]: *listgcd xs* = *0* ⟷ (∀ *x* ∈ *set xs*. *x* = *0*)
⟨*proof*⟩

lemma *primitive-field-poly*[*simp*]: *primitive* (*p* :: *'a* :: *field poly*) ⟷ *p* ≠ *0*
⟨*proof*⟩

lemma *ipoly-poly-add-nonzero*:
fixes *x y* :: *'a* :: *field*
assumes *p* ≠ *0* **and** *q* ≠ *0* **and** *ipoly p x* = *0* **and** *ipoly q y* = *0*
and (*of-int* (*lead-coeff p*) :: *'a*) ≠ *0* **and** (*of-int* (*lead-coeff q*) :: *'a*) ≠ *0*
shows *poly-add p q* ≠ *0*
⟨*proof*⟩

lemma *represents-add*:
assumes *x*: *p represents x* **and** *y*: *q represents y*
shows (*poly-add p q*) *represents* (*x* + *y*)
⟨*proof*⟩

5.2 Division of Algebraic Numbers

definition *poly-x-mult-y* **where**
[*code del*]: *poly-x-mult-y p* ≡ (∑ *i* ≤ *degree p*. *monom* (*monom* (*coeff p i*) *i*) *i*)

lemma *coeff-poly-x-mult-y*:
shows *coeff* (*poly-x-mult-y p*) *i* = *monom* (*coeff p i*) *i* (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *poly-x-mult-y-code*[*code*]: *poly-x-mult-y p* = (*let cs* = *coeffs p*
in poly-of-list (*map* (λ (*i*, *ai*). *monom ai i*) (*zip* [*0* ..< *length cs*] *cs*)))
⟨*proof*⟩

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

poly-div has desired roots.

lemma *poly2-poly-x-mult-y*:
fixes *p* :: *'a* :: *comm-ring-1 poly*
shows *poly2* (*poly-x-mult-y p*) *x y* = *poly p* (*x* ∗ *y*)
⟨*proof*⟩

lemma *poly-div*:
fixes *p q* :: *'a* ::*field poly*
assumes *q0*: *q* ≠ *0* **and** *x*: *poly p x* = *0* **and** *y*: *poly q y* = *0* **and** *y0*: *y* ≠ *0*
shows *poly* (*poly-div p q*) (*x*/*y*) = *0*
⟨*proof*⟩

poly-div is nonzero.

interpretation *poly-x-mult-y-hom*: *ring-hom poly-x-mult-y* :: *'a* :: {*idom*,*ring-char-0*}
poly ⇒ -
⟨*proof*⟩

interpretation *poly-x-mult-y-hom*: *inj-ring-hom poly-x-mult-y* :: *'a* :: {*idom*,*ring-char-0*} *poly* ⇒ -
⟨*proof*⟩

lemma *degree-poly-x-mult-y*[*simp*]:
  fixes *p* :: *'a* :: {*idom*, *ring-char-0*} *poly*
  shows *degree* (*poly-x-mult-y p*) = *degree p* (**is** *?l* = *?r*)
⟨*proof*⟩

interpretation *poly-x-mult-y-hom*: *unit-preserving-hom poly-x-mult-y* :: *'a* :: *field-char-0* *poly* ⇒ -
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *coprime-poly-x-mult-y-poly-lift*:
  fixes *p q* :: *'a* :: *field-char-0 poly*
  assumes *degp*: *degree p* > *0* **and** *degq*: *degree q* > *0*
    and *nz*: *poly p 0* ≠ *0* ∨ *poly q 0* ≠ *0*
  shows *coprime* (*poly-x-mult-y p*) (*poly-lift q*)
⟨*proof*⟩

lemma *poly-div-nonzero*:
  fixes *p q* :: *'a* :: *field-char-0 poly*
  assumes *p0*: *p* ≠ *0* **and** *q0*: *q* ≠ *0* **and** *x*: *poly p x* = *0* **and** *y*: *poly q y* = *0*
      and *p-0*: *poly p 0* ≠ *0* ∨ *poly q 0* ≠ *0*
  shows *poly-div p q* ≠ *0*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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

assumes $q \neq 0$ **and** *ipoly p x* $= 0$ **and** *ipoly q y* $= 0$ **and** $y \neq 0$
shows *ipoly* (*poly-div p q*) $(x/y) = 0$
⟨*proof*⟩

lemma *ipoly-poly-div-nonzero*:
fixes $x\ y :: 'a ::$ *field-char-0*
assumes $p \neq 0$ **and** $q \neq 0$ **and** *ipoly p x* $= 0$ **and** *ipoly q y* $= 0$ **and** *poly p 0* $\neq 0 \vee$ *poly q 0* $\neq 0$
shows *poly-div p q* $\neq 0$
⟨*proof*⟩

lemma *represents-div*:
fixes $x\ y :: 'a ::$ *field-char-0*
assumes *p represents x* **and** *q represents y* **and** *poly q 0* $\neq 0$
shows (*poly-div p q*) *represents* $(x\ /\ y)$
⟨*proof*⟩

5.3 Multiplication of Algebraic Numbers

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

lemma *represents-mult*:
assumes *px*: *p represents x* **and** *qy*: *q represents y* **and** *q-0*: *poly q 0* $\neq 0$
shows (*poly-mult p q*) *represents* $(x * y)$
⟨*proof*⟩

5.4 Summary: Closure Properties of Algebraic Numbers

lemma *algebraic-representsI*: *p represents x* $\Longrightarrow$ *algebraic x*
⟨*proof*⟩

lemma *algebraic-of-rat*: *algebraic* (*of-rat x*)
⟨*proof*⟩

lemma *algebraic-uminus*: *algebraic x* $\Longrightarrow$ *algebraic* $(-x)$
⟨*proof*⟩

lemma *algebraic-inverse*: *algebraic x* $\Longrightarrow$ *algebraic* (*inverse x*)
⟨*proof*⟩

lemma *algebraic-plus*: *algebraic x* $\Longrightarrow$ *algebraic y* $\Longrightarrow$ *algebraic* $(x + y)$
⟨*proof*⟩

lemma *algebraic-div*:
assumes *x*: *algebraic x* **and** *y*: *algebraic y* **shows** *algebraic* (x/y)
⟨*proof*⟩

lemma *algebraic-times*: *algebraic x* $\Longrightarrow$ *algebraic y* $\Longrightarrow$ *algebraic* $(x * y)$
⟨*proof*⟩

lemma *algebraic-root*: *algebraic x* $\Longrightarrow$ *algebraic* (*root n x*)
⟨*proof*⟩

lemma *algebraic-nth-root*: $n \neq 0 \Longrightarrow$ *algebraic x* $\Longrightarrow y \hat{} n = x \Longrightarrow$ *algebraic y*
⟨*proof*⟩

5.5 More on algebraic integers

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

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

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

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

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

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

lemma *algebraic-int-diff* [*intro*]:
fixes *x y* :: *'a* :: *field-char-0*
assumes *algebraic-int x algebraic-int y*
shows *algebraic-int* (*x* − *y*)
⟨*proof*⟩

lemma *algebraic-int-sum* [*intro*]:
(⋀*x*. *x* ∈ *A* ⟹ *algebraic-int* (*f x* :: *'a* :: *field-char-0*))
⟹ *algebraic-int* (*sum f A*)
⟨*proof*⟩

lemma *algebraic-int-prod* [*intro*]:
(⋀*x*. *x* ∈ *A* ⟹ *algebraic-int* (*f x* :: *'a* :: *field-char-0*))
⟹ *algebraic-int* (*prod f A*)
⟨*proof*⟩

lemma *algebraic-int-nth-root-real-iff*:
algebraic-int (*root n x*) ⟷ *n* = *0* ∨ *algebraic-int x*
⟨*proof*⟩

lemma *algebraic-int-power-iff*:
algebraic-int (*x* ^ *n* :: *'a* :: *field-char-0*) ⟷ *n* = *0* ∨ *algebraic-int x*
⟨*proof*⟩

lemma *algebraic-int-power-iff'* [*simp*]:
n > *0* ⟹ *algebraic-int* (*x* ^ *n* :: *'a* :: *field-char-0*) ⟷ *algebraic-int x*
⟨*proof*⟩

lemma *algebraic-int-sqrt-iff* [*simp*]: *algebraic-int* (*sqrt x*) ⟷ *algebraic-int x*
⟨*proof*⟩

lemma *algebraic-int-csqrt-iff* [*simp*]: *algebraic-int* (*csqrt x*) ⟷ *algebraic-int x*
⟨*proof*⟩

lemma *algebraic-int-norm-complex* [*intro*]:
assumes *algebraic-int* (*z* :: *complex*)
shows *algebraic-int* (*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 of the algorithms w.r.t. Sturm and let them be executed on rational numbers. We then prove that corresponds to a homomorphism and therefore can transfer the existing soundness results.

theory *Sturm-Rat*

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

hide-const (**open**) *UnivPoly.coeff*

lemma *root-primitive-part* [*simp*]:
fixes $p :: 'a :: \{semiring\text{-}gcd, semiring\text{-}no\text{-}zero\text{-}divisors\}$ *poly*
shows *poly* (*primitive-part p*) $x = 0 \longleftrightarrow$ *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* ⇒ *rat* ⇒ *nat*) (*number-root*: *rat* ⇒ *nat*)
hide-const (**open**) *l-r*
hide-const (**open**) *number-root*

definition *count-roots-interval-sf* :: *real poly* ⇒ (*real* ⇒ *real* ⇒ *nat*) × (*real* ⇒ *nat*) **where**
count-roots-interval-sf p = (*let ps* = *sturm-squarefree p*
in ((λ *a b*. *sign-changes ps a* − *sign-changes ps b* + (*if poly p a* = *0 then 1 else 0*)),
(λ *a*. *sign-changes-neg-inf ps* − *sign-changes ps a*)))

definition *count-roots-interval* :: *real poly* ⇒ (*real* ⇒ *real* ⇒ *nat*) × (*real* ⇒ *nat*) **where**
count-roots-interval p = (*let ps* = *sturm p*
in ((λ *a b*. *sign-changes ps a* − *sign-changes ps b* + (*if poly p a* = *0 then 1 else 0*)),
(λ *a*. *sign-changes-neg-inf ps* − *sign-changes ps a*)))

lemma *count-roots-interval-iff*: *square-free p* ⟹ *count-roots-interval p* = *count-roots-interval-sf p*
⟨*proof*⟩

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

nr a = card {x. x ≤ a ∧ poly p x = 0}
⟨*proof*⟩

lemma *count-roots-interval*: **assumes** *cr*: *count-roots-interval p = (cr,nr)*
and *sf*: *square-free p*
shows *a ≤ b ⟹ cr a b = (card {x. a ≤ x ∧ x ≤ b ∧ poly p x = 0})*
nr a = card {x. x ≤ a ∧ poly p x = 0}
⟨*proof*⟩

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

definition *root-info-cond* :: *root-info ⇒ int poly ⇒ bool* **where**
root-info-cond ri p ≡ (∀ a b. a ≤ b ⟶ root-info.l-r ri a b = card {x. root-cond (p,a,b) x})
∧ (∀ a. root-info.number-root ri a = card {x. x ≤ real-of-rat a ∧ ipoly p x = 0})

lemma *root-info-condD*: *root-info-cond ri p ⟹ a ≤ b ⟹ root-info.l-r ri a b = card {x. root-cond (p,a,b) x}*
root-info-cond ri p ⟹ root-info.number-root ri a = card {x. x ≤ real-of-rat a ∧ ipoly p x = 0}
⟨*proof*⟩

definition *count-roots-interval-sf-rat* :: *int poly ⇒ root-info* **where**
count-roots-interval-sf-rat p = (let pp = real-of-int-poly p;
(cr,nr) = count-roots-interval-sf pp
in Root-Info (λ a b. cr (of-rat a) (of-rat b)) (λ a. nr (of-rat a)))

definition *count-roots-interval-rat* :: *int poly ⇒ root-info* **where**
[*code del*]: *count-roots-interval-rat p = (let pp = real-of-int-poly p;*
(cr,nr) = count-roots-interval pp
in Root-Info (λ a b. cr (of-rat a) (of-rat b)) (λ a. nr (of-rat a)))

definition *count-roots-rat* :: *int poly ⇒ nat* **where**
[*code del*]: *count-roots-rat p = (count-roots (real-of-int-poly p))*

lemma *count-roots-interval-sf-rat*: **assumes** *p*: *p ≠ 0*
shows *root-info-cond (count-roots-interval-sf-rat p) p*
⟨*proof*⟩

lemma *of-rat-of-int-poly*: *map-poly of-rat (of-int-poly p) = of-int-poly p*
⟨*proof*⟩

lemma *square-free-of-int-poly*: **assumes** *square-free p*
shows *square-free (of-int-poly p :: 'a :: {field-gcd, field-char-0} 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* (λ*x*. *x* ≠ *0*) (*map* (λ*p*. *sgn* (*poly p x*)) *ps*))) − *1*

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

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

lemma *real-of-rat-list-neq*: *list-neq (map real-of-rat xs) 0*
= map real-of-rat (list-neq xs 0)
⟨*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
(λ a b. sign-changes-rat ps a − sign-changes-rat ps b + (if poly rp a = 0 then 1 else 0))
(λ a. sign-changes-neg-number-rootat ps − 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* ⇒ *root-info* **where**
root-info p = (if degree p = 1 then
(let x = Rat.Fract (− coeff p 0) (coeff p 1)
in Root-Info (λ l r. if l ≤ x ∧ x ≤ r then 1 else 0) (λ b. if x ≤ b then 1 else 0)) else

(*let rp = map-poly rat-of-int p*; *ps = sturm-rat rp in*
Root-Info (λ *a b*. *sign-changes-rat ps a* − *sign-changes-rat ps b*)
(λ *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*) ≠ *0* ⟷
degree (*gcd* (*of-int-poly q* :: *'a* :: {*field-char-0*, *field-gcd*} *poly*) (*of-int-poly r*)) ≠ *0*
⟨*proof*⟩

definition *factors-of-int-poly* :: *int poly* ⇒ *int poly list* **where**
factors-of-int-poly p = *map* (*abs-int-poly o fst*) (*snd* (*factorize-int-poly p*))

lemma *factors-of-int-poly-const*: **assumes** *degree p* = *0*
shows *factors-of-int-poly p* = []
⟨*proof*⟩

lemma *factors-of-int-poly*:
defines *rp* ≡ *ipoly* :: *int poly* ⇒ *'a* :: {*field-gcd*,*field-char-0*} ⇒ *'a*
assumes *factors-of-int-poly p* = *qs*
shows ⋀ *q*. *q* ∈ *set qs* ⟹ *irreducible q* ∧ *lead-coeff q* > *0* ∧ *degree q* ≤ *degree p* ∧ *degree q* ≠ *0*
p ≠ *0* ⟹ *rp p x* = *0* ⟷ (∃ *q* ∈ *set qs*. *rp q x* = *0*)
p ≠ *0* ⟹ *rp p x* = *0* ⟹ ∃! *q* ∈ *set qs*. *rp q x* = *0*
distinct qs
⟨*proof*⟩

lemma *factors-int-poly-represents*:

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

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

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

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

lemma *poly-uminus-irreducible*:
assumes p: *irreducible* $(p :: int\ poly)$ **and** *deg*: *degree* $p \neq 0$
shows *irreducible* (*poly-uminus* p)
⟨*proof*⟩

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

lemma *poly-add-rat-irreducible*:
assumes p: *irreducible* p **and** *deg*: *degree* $p \neq 0$
shows *irreducible* (*cf-pos-poly* (*poly-add-rat* r p))
⟨*proof*⟩

lemma *poly-mult-rat-irreducible*:
assumes p: *irreducible* p **and** *deg*: *degree* $p \neq 0$ **and** r: $r \neq 0$
shows *irreducible* (*cf-pos-poly* (*poly-mult-rat* r p))
⟨*proof*⟩

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

⟨*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* ⇒ *int poly* **where**
min-int-poly x =
(*if algebraic x then THE p. p represents x* ∧ *irreducible p* ∧ *lead-coeff p* > *0*
else [:*0*, *1*:])

lemma
fixes *x* :: *'a* :: {*field-char-0*, *field-gcd*}
shows *min-int-poly-represents* [*intro*]: *algebraic x* ⟹ *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*) ≠ *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 ⟹ *ipoly* (*min-int-poly x*) (*x* :: *'a* :: {*field-gcd*, *field-char-0*}) = *0*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *min-int-poly-eqI*:

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

Implementation for real and rational numbers

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

definition *min-int-poly-real* :: *real* ⇒ *int poly* **where**
[*simp*]: *min-int-poly-real* = *min-int-poly*

lemma *min-int-poly-real-code-unfold* [*code-unfold*]: *min-int-poly* = *min-int-poly-real*
⟨*proof*⟩

lemma *min-int-poly-real-basic-impl*[*code*]: *min-int-poly-real* (*real-of-rat* x) = *poly-rat* x
⟨*proof*⟩

lemma *min-int-poly-rat-code-unfold* [*code-unfold*]: *min-int-poly* = *poly-rat*
⟨*proof*⟩

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-IArray-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*: $(\exists!x.\ P\ x) \wedge P\ x \longleftrightarrow Collect\ P = \{x\}$
⟨*proof*⟩

lemma *ex1-Collect-singleton*[*consumes 2*]:
assumes $\exists!x.\ P\ x$ **and** $P\ x$ **and** $Collect\ P = \{x\} \Longrightarrow thesis$ **shows** *thesis*
⟨*proof*⟩

lemma *ex1-iff-Collect-singleton*: $P\ x \Longrightarrow (\exists!x.\ P\ x) \longleftrightarrow Collect\ P = \{x\}$
⟨*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** $(THE\ x.\ P\ (f\ x)) = Hilbert\text{-}Choice.inv\ f\ (THE\ y.\ P\ y)$ (**is** *?l = ?r*)
⟨*proof*⟩

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

lemma *bij-imp-card-image*: $card\ (f\ `\ X) = card\ X$
⟨*proof*⟩

end

definition *poly-cond* :: *int poly* ⇒ *bool* **where**
poly-cond p = (*lead-coeff p* > *0* ∧ *irreducible p*)

lemma *poly-condI*[*intro*]:
assumes *lead-coeff p* > *0* **and** *irreducible p* **shows** *poly-cond p* ⟨*proof*⟩

lemma *poly-condD*:
assumes *poly-cond p*
shows *irreducible p* **and** *lead-coeff p* > *0* **and** *root-free p* **and** *square-free p* **and** *p* ≠ *0*
⟨*proof*⟩

lemma *poly-condE*[*elim*]:
assumes *poly-cond p*
and *irreducible p* ⟹ *lead-coeff p* > *0* ⟹ *root-free p* ⟹ *square-free p* ⟹

$p \neq 0 \Longrightarrow thesis$
shows *thesis*
⟨*proof*⟩

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

definition *poly-uminus-abs* :: *int poly* ⇒ *int poly* **where**
poly-uminus-abs p = *abs-int-poly* (*poly-uminus p*)

lemma *irreducible-poly-uminus*[*simp*]: *irreducible p* $\Longrightarrow$ *irreducible* (*poly-uminus* (*p* :: *int poly*))
⟨*proof*⟩

lemma *irreducible-poly-uminus-abs*[*simp*]: *irreducible p* $\Longrightarrow$ *irreducible* (*poly-uminus-abs p*)
⟨*proof*⟩

lemma *poly-cond-poly-uminus-abs*[*simp*]: *poly-cond p* $\Longrightarrow$ *poly-cond* (*poly-uminus-abs p*)
⟨*proof*⟩

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

lemma *degree-poly-uminus-abs*[*simp*]: *degree* (*poly-uminus-abs p*) = *degree p*
⟨*proof*⟩

definition *poly-inverse* :: *int poly* ⇒ *int poly* **where**
poly-inverse p = *abs-int-poly* (*reflect-poly p*)

lemma *irreducible-poly-inverse*[*simp*]: *coeff p 0* ≠ *0* $\Longrightarrow$ *irreducible p* $\Longrightarrow$ *irreducible* (*poly-inverse p*)
⟨*proof*⟩

lemma *degree-poly-inverse*[*simp*]: *coeff p 0* ≠ *0* $\Longrightarrow$ *degree* (*poly-inverse p*) = *degree p*
⟨*proof*⟩

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

lemma *ipoly-roots-finite*: *p* ≠ *0* $\Longrightarrow$ *finite* {*x* :: *'a* :: {*idom*, *ring-char-0*}. *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*
sgn (*poly p* (*x* − *d*)) ≠ *sgn* (*poly p* (*x* + *d*))
sgn (*poly p* (*x* − *d*)) ≠ *0*
0 ≠ *sgn* (*poly p* (*x* + *d*))
∀ *d'* > *0*. *d'* ≤ *d* ⟶ *sgn* (*poly p* (*x* + *d'*)) = *sgn* (*poly p* (*x* + *d*)) ∧ *sgn* (*poly p* (*x* − *d'*)) = *sgn* (*poly p* (*x* − *d*))
⟨*proof*⟩

lemma *gt-rat-sign-change-square-free*:
assumes *ur*: ∃! *x*. *root-cond plr x*
and *plr*[*simp*]: *plr* = (*p*,*l*,*r*)
and *sf*: *square-free p* **and** *in-interval*: *l* ≤ *y y* ≤ *r*
and *py0*: *ipoly p y* ≠ *0* **and** *pr0*: *ipoly p r* ≠ *0*
shows (*sgn* (*ipoly p y*) = *sgn* (*ipoly p r*)) = (*of-rat y* > (*THE x*. *root-cond 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* ⇒ *rat* **where**

```
  root-bound p ≡ let
    n = degree p;
    m = 1 + div-ceiling (max-list-non-empty (map (λi. abs (coeff p i)) [0..<n]))
         (abs (lead-coeff p))
    — round to the next higher number 2^n, so that bisection will
    — stay on integers for as long as possible
   in of-int (2 ^(log-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** $x :: 'a :: real\text{-}normed\text{-}field$
assumes *x*: *poly p x* $= 0$ **and** *p*: $p \neq 0$
shows *norm x* $\leq 1 +$ *max-list-non-empty* (*map* (λ *i*. *norm* (*coeff p i*)) [*0* ..< *degree p*])
/ *norm* (*lead-coeff p*) (**is** - ≤ - + *?max* / *?nlc*)
⟨*proof*⟩

lemma *div-le-div-ceiling*: *x div y* ≤ *div-ceiling x y*
⟨*proof*⟩

lemma *div-ceiling*: **assumes** *q*: $q \neq 0$
shows (*of-int x* :: $'a$ :: *floor-ceiling*) / *of-int q* ≤ *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* ≠ [] ⟹ *max-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* ⟹ *norm x* ≤ *of-rat B B* ≥ *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 equivalence.

The theory also contains a code-setup to implement real numbers via real algebraic numbers.

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

theory *Real-Algebraic-Numbers*
imports
  Algebraic-Numbers-Pre-Impl
begin

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

lemma *ex1-Collect-singleton*[*consumes 2*]:
  assumes $\exists!x.\ P\ x$ **and** $P\ x$ **and** $Collect\ P = \{x\} \Longrightarrow thesis$ **shows** *thesis*
  ⟨*proof*⟩

lemma *ex1-iff-Collect-singleton*: $P\ x \Longrightarrow (\exists!x.\ P\ x) \longleftrightarrow Collect\ P = \{x\}$
  ⟨*proof*⟩

lemma *bij-imp-card*: **assumes** *bij*: *bij f* **shows** *card* $\{x.\ P\ (f\ x)\} = card\ \{x.\ P\ x\}$
⟨*proof*⟩

lemma *bij-add*: *bij* $(\lambda x.\ x + y :: 'a :: group\text{-}add)$ (**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*: $\exists!y :: 'a :: group\text{-}add.\ P\ y$
  shows $(THE\ x.\ P\ (x + d)) = (THE\ y.\ P\ y) - d$
    and $(THE\ x.\ P\ (x - d)) = (THE\ y.\ P\ y) + d$
  ⟨*proof*⟩

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

lemma *irreducible-root-free*:
  fixes $p :: 'a :: \{idom, comm\text{-}ring\text{-}1\}$ *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. (...,x-3, 3,3) is equivalent to (...,x-3,2,10).

11.1.1 Basic Definitions

type-synonym *real-alg-1* = *int poly* × *rat* × *rat*

fun *poly-real-alg-1* :: *real-alg-1* ⇒ *int poly* **where** *poly-real-alg-1* (*p*,-,-) = *p*
fun *rai-ub* :: *real-alg-1* ⇒ *rat* **where** *rai-ub* (-,-,*r*) = *r*
fun *rai-lb* :: *real-alg-1* ⇒ *rat* **where** *rai-lb* (-,*l*,-) = *l*

abbreviation *roots-below p x* ≡ {*y* :: *real*. *y* ≤ *x* ∧ *ipoly p y* = *0*}

abbreviation(*input*) *unique-root* :: *real-alg-1* ⇒ *bool* **where**
unique-root plr ≡ (∃! *x*. *root-cond plr x*)

abbreviation *the-unique-root* :: *real-alg-1* ⇒ *real* **where**
the-unique-root plr ≡ (*THE x*. *root-cond plr x*)

abbreviation *real-of-1* **where** *real-of-1* ≡ *the-unique-root*

lemma *root-condI*[*intro*]:
assumes *of-rat* (*rai-lb plr*) ≤ *x* **and** *x* ≤ *of-rat* (*rai-ub plr*) **and** *ipoly* (*poly-real-alg-1 plr*) *x* = *0*
shows *root-cond plr x*
⟨*proof*⟩

lemma *root-condE*[*elim*]:
assumes *root-cond plr x*
and *of-rat* (*rai-lb plr*) ≤ *x* ⟹ *x* ≤ *of-rat* (*rai-ub plr*) ⟹ *ipoly* (*poly-real-alg-1 plr*) *x* = *0* ⟹ *thesis*
shows *thesis*
⟨*proof*⟩

lemma
assumes *ur*: *unique-root plr*
defines *x* ≡ *the-unique-root plr* **and** *p* ≡ *poly-real-alg-1 plr* **and** *l* ≡ *rai-lb plr* **and** *r* ≡ *rai-ub plr*
shows *unique-rootD*: *of-rat l* ≤ *x x* ≤ *of-rat r ipoly p x* = *0 root-cond plr x*
x = *y* ⟷ *root-cond plr y y* = *x* ⟷ *root-cond plr y*
and *the-unique-root-eqI*: *root-cond plr y* ⟹ *y* = *x root-cond plr y* ⟹ *x* = *y*
⟨*proof*⟩

lemma *unique-rootE*:
assumes *ur*: *unique-root plr*
defines $x \equiv$ *the-unique-root plr* **and** $p \equiv$ *poly-real-alg-1 plr* **and** $l \equiv$ *rai-lb plr* **and** $r \equiv$ *rai-ub plr*
assumes *main*: *of-rat* $l \leq x \Longrightarrow x \leq$ *of-rat* $r \Longrightarrow$ *ipoly* $p\ x = 0 \Longrightarrow$ *root-cond plr* $x \Longrightarrow$
$(\bigwedge y.\ x = y \longleftrightarrow$ *root-cond plr* $y) \Longrightarrow (\bigwedge y.\ y = x \longleftrightarrow$ *root-cond plr* $y) \Longrightarrow$ *thesis*
shows *thesis* ⟨*proof*⟩

lemma *unique-rootI*:
assumes $\bigwedge$ *y. root-cond plr y* $\Longrightarrow x = y$ *root-cond plr x*
shows *unique-root plr* ⟨*proof*⟩

definition *invariant-1* :: *real-alg-1* ⇒ *bool* **where**
invariant-1 tup ≡ *case tup of* (*p*,*l*,*r*) ⇒
unique-root (*p*,*l*,*r*) ∧ *sgn l* = *sgn r* ∧ *poly-cond p*

lemma *invariant-1I*:
assumes *unique-root plr* **and** *sgn* (*rai-lb plr*) = *sgn* (*rai-ub plr*) **and** *poly-cond* (*poly-real-alg-1 plr*)
shows *invariant-1 plr*
⟨*proof*⟩

lemma
assumes *invariant-1 plr*
defines $x \equiv$ *the-unique-root plr* **and** $p \equiv$ *poly-real-alg-1 plr* **and** $l \equiv$ *rai-lb plr* **and** $r \equiv$ *rai-ub plr*
shows *invariant-1D*: *root-cond plr x*
sgn l = *sgn r sgn x* = *of-rat* (*sgn r*) *unique-root plr poly-cond p degree p* > *0 primitive p*
and *invariant-1-root-cond*: $\bigwedge$ *y. root-cond plr y* $\longleftrightarrow y = x$
⟨*proof*⟩

lemma *invariant-1E*[*elim*]:
assumes *invariant-1 plr*
defines $x \equiv$ *the-unique-root plr* **and** $p \equiv$ *poly-real-alg-1 plr* **and** $l \equiv$ *rai-lb plr* **and** $r \equiv$ *rai-ub plr*
assumes *main*: *root-cond plr x* $\Longrightarrow$
sgn l = *sgn r* $\Longrightarrow$ *sgn x* = *of-rat* (*sgn r*) $\Longrightarrow$ *unique-root plr* $\Longrightarrow$ *poly-cond p* $\Longrightarrow$ *degree p* > *0* $\Longrightarrow$
primitive p $\Longrightarrow$ *thesis*
shows *thesis* ⟨*proof*⟩

lemma *invariant-1-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 l* = *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* $\Longrightarrow$ *r* = *0*
and [*intro*]: *r* = *0* $\Longrightarrow$ *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* $\Longrightarrow$ *l* > *0*
and [*dest*]: *l* > *0* $\Longrightarrow$ *r* > *0*
and [*intro*]: *r* < *0* $\Longrightarrow$ *l* < *0*
and [*dest*]: *l* < *0* $\Longrightarrow$ *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* $\Longrightarrow$ *degree p* > *1* $\Longrightarrow$ *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* $\Longrightarrow$ *degree p* > *1* $\Longrightarrow$ *thesis* **shows** *thesis*
⟨*proof*⟩

lemma *invariant-1-2-poly-cond2*: *invariant-1-2 rai* $\Longrightarrow$ *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* $\Longrightarrow$ *degree* (*poly-real-alg-1 rai*) $>$ *1* $\Longrightarrow$ *thesis*
shows *thesis* ⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

typedef *real-alg-3* = *Collect invariant-2*
morphisms *rep-real-alg-3 Real-Alg-Invariant*
⟨*proof*⟩

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

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

11.2.2 Definitions and Algorithms on Quotient Type

quotient-type *real-alg* = *real-alg-3* / λ *x y*. *real-of-3 x* = *real-of-3 y*
morphisms *rep-real-alg Real-Alg-Quotient*
⟨*proof*⟩

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

lemma *real-of-inj*: (*real-of x* = *real-of y*) = (*x* = *y*)
⟨*proof*⟩

11.2.3 Sign

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

lemma *sgn-1*: *invariant-1 x* ⟹ *real-of-rat* (*sgn-1 x*) = *sgn* (*real-of-1 x*)
⟨*proof*⟩

lemma *sgn-1-inj*: *invariant-1 x* ⟹ *invariant-1 y* ⟹ *real-of-1 x* = *real-of-1 y* ⟹ *sgn-1 x* = *sgn-1 y*
⟨*proof*⟩

11.2.4 Normalization: Bounds Close Together

lemma *unique-root-lr*: **assumes** *ur*: *unique-root plr* **shows** *rai-lb plr* ≤ *rai-ub plr* (**is** *?l* ≤ *?r*)
⟨*proof*⟩

locale *map-poly-zero-hom-0* = *base*: *zero-hom-0*
begin
sublocale *zero-hom-0 map-poly hom* ⟨*proof*⟩
end
interpretation *of-int-poly-hom*:
map-poly-zero-hom-0 of-int :: *int* ⇒ *'a* :: {*ring-1*, *ring-char-0*} ⟨*proof*⟩

lemma *roots-below-the-unique-root*:
assumes *ur*: *unique-root* (*p*,*l*,*r*)
shows *roots-below p* (*the-unique-root* (*p*,*l*,*r*)) = *roots-below p* (*of-rat r*) (**is** *roots-below p ?x* = -)
⟨*proof*⟩

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*) ≠ *0*
⟨*proof*⟩

context
fixes *p* :: *int poly*
and *x* :: *rat*
begin

lemma *gt-rat-sign-change*:
assumes *ur*: *unique-root plr*
defines *p* ≡ *poly-real-alg-1 plr* **and** *l* ≡ *rai-lb plr* **and** *r* ≡ *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* > *the-unique-root plr*)
⟨*proof*⟩

definition *tighten-poly-bounds* :: *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* × *rat* × *rat* **where**
tighten-poly-bounds l r sr = (*let m* = (*l* + *r*) / *2*; *sm* = *sgn* (*ipoly p m*) *in*
if sm = *sr*
then (*l*,*m*,*sm*) *else* (*m*,*r*,*sr*))

lemma *tighten-poly-bounds*: **assumes** *res*: *tighten-poly-bounds l r sr* = (*l'*,*r'*,*sr'*)
and *ur*: *unique-root* (*p*,*l*,*r*)
and *p*: *poly-cond2 p*
and *sr*: *sr* = *sgn* (*ipoly p r*)
shows *root-cond* (*p*,*l'*,*r'*) (*the-unique-root* (*p*,*l*,*r*)) *l* ≤ *l' l'* ≤ *r' r'* ≤ *r*
(*r'* − *l'*) = (*r* − *l*) / *2 sr'* = *sgn* (*ipoly p r'*)
⟨*proof*⟩

partial-function (*tailrec*) *tighten-poly-bounds-epsilon* :: *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* × *rat* × *rat* **where**
[*code*]: *tighten-poly-bounds-epsilon l r sr* = (*if r* − *l* ≤ *x then* (*l*,*r*,*sr*) *else*
(*case tighten-poly-bounds l r sr of* (*l'*,*r'*,*sr'*) ⇒ *tighten-poly-bounds-epsilon l' r' sr'*))

partial-function (*tailrec*) *tighten-poly-bounds-for-x* :: *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* × *rat* × *rat* **where**
[*code*]: *tighten-poly-bounds-for-x l r sr* = (*if x* < *l* ∨ *r* < *x then* (*l*, *r*, *sr*) *else*
(*case tighten-poly-bounds l r sr of* (*l'*,*r'*,*sr'*) ⇒ *tighten-poly-bounds-for-x l' r' sr'*))

lemma *tighten-poly-bounds-epsilon*:
assumes *ur*: *unique-root* (*p*,*l*,*r*)
defines *u*: *u* ≡ *the-unique-root* (*p*,*l*,*r*)
assumes *p*: *poly-cond2 p*
and *res*: *tighten-poly-bounds-epsilon l r sr* = (*l'*,*r'*,*sr'*)
and *sr*: *sr* = *sgn* (*ipoly p r*)
and *x*: *x* > *0*
shows *l* ≤ *l' r'* ≤ *r root-cond* (*p*,*l'*,*r'*) *u r'* − *l'* ≤ *x sr'* = *sgn* (*ipoly p r'*)
⟨*proof*⟩

lemma *tighten-poly-bounds-for-x*:
assumes *ur*: *unique-root* (*p*,*l*,*r*)
defines *u*: *u* ≡ *the-unique-root* (*p*,*l*,*r*)
assumes *p*: *poly-cond2 p*
and *res*: *tighten-poly-bounds-for-x l r sr* = (*l'*,*r'*,*sr'*)
and *sr*: *sr* = *sgn* (*ipoly p r*)
shows *l* ≤ *l' l'* ≤ *r' r'* ≤ *r root-cond* (*p*,*l'*,*r'*) *u* ¬ (*l'* ≤ *x* ∧ *x* ≤ *r'*) *sr'* = *sgn* (*ipoly p r'*) *unique-root* (*p*,*l'*,*r'*)
⟨*proof*⟩
end

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

lemma *real-alg-precision*: *real-alg-precision* > *0*
⟨*proof*⟩

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

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

definition normalize-bounds-1 :: real-alg-1 ⇒ real-alg-1 where
  normalize-bounds-1 = (normalize-bounds-1-main real-alg-precision)

context
  fixes p q and l r :: rat
  assumes cong: ⋀ x. real-of-rat l ≤ x ⟹ x ≤ of-rat r ⟹ (ipoly p x = (0 ::
real)) = (ipoly q x = 0)
begin
lemma root-cond-cong: root-cond (p,l,r) = root-cond (q,l,r)
  ⟨proof⟩

lemma the-unique-root-cong:
  the-unique-root (p,l,r) = the-unique-root (q,l,r)
  ⟨proof⟩

lemma unique-root-cong:
  unique-root (p,l,r) = unique-root (q,l,r)
  ⟨proof⟩
end

lemma normalize-bounds-1-main: assumes eps: eps > 0 and rc: invariant-1-2 x
  defines y: y ≡ normalize-bounds-1-main eps x
  shows invariant-1-2 y ∧ (real-of-1 y = real-of-1 x)
⟨proof⟩

lemma normalize-bounds-1: assumes x: invariant-1-2 x
  shows invariant-1-2 (normalize-bounds-1 x) ∧ (real-of-1 (normalize-bounds-1 x)
= real-of-1 x)
⟨proof⟩

lemma normalize-bound-1-poly: poly-real-alg-1 (normalize-bounds-1 rai) = poly-real-alg-1
rai
  ⟨proof⟩

definition real-alg-2-main :: root-info ⇒ real-alg-1 ⇒ real-alg-2 where
  real-alg-2-main ri rai ≡ let p = poly-real-alg-1 rai
     in (if degree p = 1 then Rational (Rat.Fract (− coeff p 0) (coeff p 1))
       else (case normalize-bounds-1 rai of (p',l,r) ⇒
       Irrational (root-info.number-root ri r) (p',l,r)))

definition real-alg-2 :: real-alg-1 ⇒ real-alg-2 where
  real-alg-2 rai ≡ let p = poly-real-alg-1 rai
     in (if degree p = 1 then Rational (Rat.Fract (− coeff p 0) (coeff p 1))
       else (case normalize-bounds-1 rai of (p',l,r) ⇒
```

Irrational (*root-info.number-root* (*root-info p*) *r*) (*p'*,*l*,*r*)))

lemma *degree-1-ipoly*: **assumes** *degree p* = *Suc 0*
shows *ipoly p x* = *0* ⟷ (*x* = *real-of-rat* (*Rat.Fract* (− *coeff p 0*) (*coeff p 1*)))
⟨*proof*⟩

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

lemma *real-alg-2-main*:
assumes *inv*: *invariant-1 rai*
defines [*simp*]: *p* ≡ *poly-real-alg-1 rai*
assumes *ric*: *irreducible* (*poly-real-alg-1 rai*) ⟹ *root-info-cond ri* (*poly-real-alg-1 rai*)
shows *invariant-2* (*real-alg-2-main ri rai*) *real-of-2* (*real-alg-2-main ri rai*) = *real-of-1 rai*
⟨*proof*⟩

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* ≡ *poly-real-alg-1 plr* **and** *l* ≡ *rai-lb plr* **and** *r* ≡ *rai-ub plr*
assumes *x*: *root-cond plr x* **and** *sgn*: *sgn l* = *sgn r*
and *ur*: *unique-root plr*
and *p*: *poly-cond p*
shows *invariant-2* (*real-alg-2 plr*) ∧ *real-of-2* (*real-alg-2 plr*) = *x*
⟨*proof*⟩

11.2.5 Comparisons

fun *compare-rat-1* :: *rat* ⇒ *real-alg-1* ⇒ *order* **where**
compare-rat-1 x (*p*,*l*,*r*) = (*if x* < *l then Lt else if x* > *r then Gt else*
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*]: ¬ *cf-pos 0*
⟨*proof*⟩

11.2.6 Negation

fun *uminus-1* :: *real-alg-1* ⇒ *real-alg-1* **where**
uminus-1 (*p*,*l*,*r*) = (*abs-int-poly* (*poly-uminus p*), −*r*, −*l*)

lemma *uminus-1*: **assumes** *x*: *invariant-1 x*
defines *y*: $y \equiv$ *uminus-1 x*
shows *invariant-1 y* $\wedge$ (*real-of-1 y* $= -$ *real-of-1 x*)
⟨*proof*⟩

lemma *uminus-1-2*:
assumes *x*: *invariant-1-2 x*
defines *y*: $y \equiv$ *uminus-1 x*
shows *invariant-1-2 y* $\wedge$ (*real-of-1 y* $= -$ *real-of-1 x*)
⟨*proof*⟩

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 *real-of-2* (*uminus-2 x*) = *uminus* (*real-of-2 x*)
invariant-2 (*uminus-2 x*)
⟨*proof*⟩

declare *uminus-1.simps*[*simp del*]

lift-definition *uminus-3* :: *real-alg-3* $\Rightarrow$ *real-alg-3* **is** *uminus-2*
⟨*proof*⟩

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

instantiation *real-alg* :: *uminus*
begin
lift-definition *uminus-real-alg* :: *real-alg* $\Rightarrow$ *real-alg* **is** *uminus-3*
⟨*proof*⟩
instance ⟨*proof*⟩
end

lemma *uminus-real-alg*: $-$ (*real-of x*) = *real-of* ($-$ *x*)
⟨*proof*⟩

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 *real-of-1 rai* $\neq$ *of-rat x*
⟨*proof*⟩

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 ∃! *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-to f x ≡ (∀ *n. in-interval* (*f n*) *x* ∧ *sub-interval* (*f* (*Suc n*)) (*f n*))
∧ (∀ (*eps* :: *real*) > *0*. ∃ *n l r*. *f n* = (*l*,*r*) ∧ *of-rat r* − *of-rat l* ≤ *eps*)

context
fixes *bnd-update* :: *'a* ⇒ *'a*
and *bnd-get* :: *'a* ⇒ *rat* × *rat*
begin

definition *at-step* :: (*nat* ⇒ *rat* × *rat*) ⇒ *nat* ⇒ *'a* ⇒ *bool* **where**
at-step f n a ≡ ∀ *i. bnd-get* ((*bnd-update* ^^ *i*) *a*) = *f* (*n* + *i*)

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

definition *select-correct-factor* :: *'a* ⇒ (*int poly* × *root-info*)*list* ⇒
(*int poly* × *root-info*) × *rat* × *rat* **where**
select-correct-factor init polys = (*case bnd-get init of* (*l*,*r*) ⇒
select-correct-factor-main init polys [] *l r 0*)

lemma *select-correct-factor-main*: **assumes** *conv*: *converges-to f x*
and *at*: *at-step f i a*
and *res*: *select-correct-factor-main a todo old l r n* = ((*q*,*ri-fin*),(*l-fin*,*r-fin*))
and *bnd*: *bnd-get a* = (*l*,*r*)
and *ri*: ⋀ *q ri*. (*q*,*ri*) ∈ *set todo* ∪ *set old* ⟹ *root-info-cond ri q*
and *q0*: ⋀ *q ri*. (*q*,*ri*) ∈ *set todo* ∪ *set old* ⟹ *q* ≠ *0*
and *ex*: ∃ *q*. *q* ∈ *fst* ‘ *set todo* ∪ *fst* ‘ *set old* ∧ *ipoly q x* = *0*
and *dist*: *distinct* (*map fst* (*todo* @ *old*))
and *old*: ⋀ *q ri*. (*q*,*ri*) ∈ *set old* ⟹ *root-info.l-r ri l r* ≠ *0*
and *un*: ⋀ *x* :: *real*. (∃ *q*. *q* ∈ *fst* ‘ *set todo* ∪ *fst* ‘ *set old* ∧ *ipoly q x* = *0*) ⟹
∃!*q*. *q* ∈ *fst* ‘ *set todo* ∪ *fst* ‘ *set old* ∧ *ipoly q x* = *0*
and *n*: *n* = *sum-list* (*map* (λ (*q*,*ri*). *root-info.l-r ri l r*) *old*)
shows *unique-root* (*q*,*l-fin*,*r-fin*) ∧ (*q*,*ri-fin*) ∈ *set todo* ∪ *set old* ∧ *x* = *the-unique-root*
(*q*,*l-fin*,*r-fin*)
⟨*proof*⟩

lemma *select-correct-factor*: **assumes**
conv: *converges-to* (λ *i. bnd-get* ((*bnd-update* ^^ *i*) *init*)) *x*
and *res*: *select-correct-factor init polys* = ((*q*,*ri*),(*l*,*r*))
and *ri*: ⋀ *q ri*. (*q*,*ri*) ∈ *set polys* ⟹ *root-info-cond ri q*
and *q0*: ⋀ *q ri*. (*q*,*ri*) ∈ *set polys* ⟹ *q* ≠ *0*
and *ex*: ∃ *q*. *q* ∈ *fst* ‘ *set polys* ∧ *ipoly q x* = *0*

and *dist*: *distinct* (*map fst polys*)
and *un*: $\bigwedge x :: real.\ (\exists q.\ q \in fst\ `\ set\ polys \wedge ipoly\ q\ x = 0) \Longrightarrow$
$\exists! q.\ q \in fst\ `\ set\ polys \wedge ipoly\ q\ x = 0$
shows *unique-root* (*q*,*l*,*r*) $\wedge$ (*q*,*ri*) $\in$ *set polys* $\wedge$ *x* = *the-unique-root* (*q*,*l*,*r*)
⟨*proof*⟩

definition *real-alg-2'* :: *root-info* ⇒ *int poly* ⇒ *rat* ⇒ *rat* ⇒ *real-alg-2* **where**
[*code del*]: *real-alg-2' ri p l r* = (
if degree p = *1 then Rational* (*Rat.Fract* (− *coeff p 0*) (*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'*) ⇒ (*p*, *l'*, *r'*)))

lemma *real-alg-2'-code*[*code*]: *real-alg-2' ri p l r* =
(*if degree p* = *1 then Rational* (*Rat.Fract* (− *coeff p 0*) (*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'*) ⇒ (*p*, *l'*, *r'*))
of (*p'*, *l*, *r*) ⇒ *Irrational* (*root-info.number-root ri r*) (*p'*, *l*, *r*))
⟨*proof*⟩

definition *real-alg-2''* :: *root-info* ⇒ *int poly* ⇒ *rat* ⇒ *rat* ⇒ *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'*) ⇒ (*p*, *l'*, *r'*))
of (*p'*, *l*, *r*) ⇒ *Irrational* (*root-info.number-root ri r*) (*p'*, *l*, *r*))

lemma *real-alg-2''*: *degree p* ≠ *1* ⟹ *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* ≠ *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* ⇒ *int poly* ⇒ *real-alg-2* **where**
select-correct-factor-int-poly init p ≡
let qs = *factors-of-int-poly p*;
polys = *map* (λ *q*. (*q*, *root-info q*)) *qs*;
((*q*,*ri*),(*l*,*r*)) = *select-correct-factor init polys*
in real-alg-2' ri q l r

lemma *select-correct-factor-int-poly*: **assumes**
conv: *converges-to* (λ *i*. *bnd-get* ((*bnd-update* ^^ *i*) *init*)) *x*

and *rai*: *select-correct-factor-int-poly init p = rai*
and *x*: *ipoly p x = 0*
and *p*: *p ≠ 0*
shows *invariant-2 rai ∧ real-of-2 rai = x*
⟨*proof*⟩
end

11.2.10 Addition

lemma *ipoly-0-0*[*simp*]: *ipoly f* (*0*::*'a*::{*comm-ring-1*,*ring-char-0*}) = *0* ⟷ *poly f 0 = 0*
⟨*proof*⟩

lemma *add-rat-roots-below*[*simp*]: *roots-below* (*poly-add-rat r p*) *x* = (λ*y*. *y* + *of-rat r*) ‘ *roots-below p* (*x* − *of-rat r*)
⟨*proof*⟩

lemma *add-rat-root-cond*:
shows *root-cond* (*cf-pos-poly* (*poly-add-rat m p*),*l*,*r*) *x* = *root-cond* (*p*, *l* − *m*, *r* − *m*) (*x* − *of-rat m*)
⟨*proof*⟩

lemma *add-rat-unique-root*: *unique-root* (*cf-pos-poly* (*poly-add-rat m p*), *l*, *r*) = *unique-root* (*p*, *l*−*m*, *r*−*m*)
⟨*proof*⟩

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

lemma *poly-real-alg-1-add-rat*[*simp*]:
poly-real-alg-1 (*add-rat-1 r y*) = *cf-pos-poly* (*poly-add-rat r* (*poly-real-alg-1 y*))
⟨*proof*⟩

lemma *sgn-cf-pos*:
assumes *lead-coeff p* > *0* **shows** *sgn* (*ipoly* (*cf-pos-poly p*) (*x*::*'a*::*linordered-field*)) = *sgn* (*ipoly p x*)
⟨*proof*⟩

lemma *add-rat-1*: **fixes** *r1* :: *rat* **assumes** *inv-y*: *invariant-1-2 y*
defines *z* ≡ *add-rat-1 r1 y*
shows *invariant-1-2 z* ∧ (*real-of-1 z* = *of-rat r1* + *real-of-1 y*)
⟨*proof*⟩

fun *tighten-poly-bounds-binary* :: *int poly* ⇒ *int poly* ⇒ (*rat* × *rat* × *rat*) × *rat* ×

rat × *rat* ⇒ (*rat* × *rat* × *rat*) × *rat* × *rat* × *rat* **where**
tighten-poly-bounds-binary cr1 cr2 ((*l1*,*r1*,*sr1*),(*l2*,*r2*,*sr2*)) =
(*tighten-poly-bounds cr1 l1 r1 sr1*, *tighten-poly-bounds cr2 l2 r2 sr2*)

lemma *tighten-poly-bounds-binary*:
assumes *ur*: *unique-root* (*p1*,*l1*,*r1*) *unique-root* (*p2*,*l2*,*r2*) **and** *pt*: *poly-cond2 p1 poly-cond2 p2*
defines *x* ≡ *the-unique-root* (*p1*,*l1*,*r1*) **and** *y* ≡ *the-unique-root* (*p2*,*l2*,*r2*)
assumes *bnd*: ⋀ *l1 r1 l2 r2 l r sr1 sr2*. *I l1* ⟹ *I l2* ⟹ *root-cond* (*p1*,*l1*,*r1*) *x* ⟹ *root-cond* (*p2*,*l2*,*r2*) *y* ⟹
bnd ((*l1*,*r1*,*sr1*),(*l2*,*r2*,*sr2*)) = (*l*,*r*) ⟹ *of-rat l* ≤ *f x y* ∧ *f x y* ≤ *of-rat r*
and *approx*: ⋀ *l1 r1 l2 r2 l1′ r1′ l2′ r2′ l l′ r r′ sr1 sr2 sr1′ sr2′*.
I l1 ⟹ *I l2* ⟹
l1 ≤ *r1* ⟹ *l2* ≤ *r2* ⟹
(*l*,*r*) = *bnd* ((*l1*,*r1*,*sr1*), (*l2*,*r2*,*sr2*)) ⟹
(*l′*,*r′*) = *bnd* ((*l1′*,*r1′*,*sr1′*), (*l2′*,*r2′*,*sr2′*)) ⟹
(*l1′*,*r1′*) ∈ {(*l1*,(*l1*+*r1*)/2),((*l1*+*r1*)/2,*r1*)} ⟹
(*l2′*,*r2′*) ∈ {(*l2*,(*l2*+*r2*)/2),((*l2*+*r2*)/2,*r2*)} ⟹
(*r′* − *l′*) ≤ 3/4 ∗ (*r* − *l*) ∧ *l* ≤ *l′* ∧ *r′* ≤ *r*
and *I-mono*: ⋀ *l l′*. *I l* ⟹ *l* ≤ *l′* ⟹ *I l′*
and *I*: *I l1 I l2*
and *sr*: *sr1* = *sgn* (*ipoly p1 r1*) *sr2* = *sgn* (*ipoly p2 r2*)
shows *converges-to* (λ *i*. *bnd* ((*tighten-poly-bounds-binary p1 p2* ^^ *i*) ((*l1*,*r1*,*sr1*),(*l2*,*r2*,*sr2*))))
(*f x y*)
⟨*proof*⟩

fun *add-1* :: *real-alg-1* ⇒ *real-alg-1* ⇒ *real-alg-2* **where**
add-1 (*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-add p1 p2*))

lemma *add-1*:
assumes *x*: *invariant-1-2 x* **and** *y*: *invariant-1-2 y*
defines *z*: *z* ≡ *add-1 x y*
shows *invariant-2 z* ∧ (*real-of-2 z* = *real-of-1 x* + *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* ⇒ *real-alg-1* ⇒ *real-alg-2* **where**
mult-rat-1-pos r1 (*p2*,*l2*,*r2*) = *real-alg-2* (*cf-pos-poly* (*poly-mult-rat r1 p2*), *l2*∗*r1*,

*r2*r1*)

private fun *mult-1-pos* :: *real-alg-1* ⇒ *real-alg-1* ⇒ *real-alg-2* **where**
mult-1-pos (*p1*,*l1*,*r1*) (*p2*,*l2*,*r2*) =
select-correct-factor-int-poly
(*tighten-poly-bounds-binary p1 p2*)
(λ ((*l1*,*r1*,*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* ⇒ *real-alg-1* ⇒ *real-alg-2* **where**
mult-rat-1 x y =
(*if x* < *0 then uminus-2* (*mult-rat-1-pos* (−*x*) *y*)
else if x = *0 then Rational 0 else* (*mult-rat-1-pos x y*))

fun *mult-1* :: *real-alg-1* ⇒ *real-alg-1* ⇒ *real-alg-2* **where**
mult-1 x y = (*case* (*x*,*y*) *of* ((*p1*,*l1*,*r1*),(*p2*,*l2*,*r2*)) ⇒
if r1 > *0 then*
if r2 > *0 then mult-1-pos x y*
else uminus-2 (*mult-1-pos 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* ≡ *mult-rat-1-pos r1 y*
shows *invariant-2 z* ∧ (*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* ≡ *mult-1-pos x y*
assumes *pos*: *real-of-1 x* > *0 real-of-1 y* > *0*
shows *invariant-2 z* ∧ (*real-of-2 z* = *real-of-1 x* ∗ *real-of-1 y*)
⟨*proof*⟩

lemma *mult-1*: **assumes** *x*: *invariant-1-2 x* **and** *y*: *invariant-1-2 y*
defines *z*[*simp*]: *z* ≡ *mult-1 x y*
shows *invariant-2 z* ∧ (*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* ≡ *mult-rat-1 x y*
shows *invariant-2 z* ∧ (*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* ⇒ *real* **where**
ipoly-root-delta p = *Min* (*insert 1* { *abs* (*x* − *y*) | *x y*. *ipoly p x* = *0* ∧ *ipoly p y* = *0* ∧ *x* ≠ *y*}) / *4*

lemma *ipoly-root-delta*: **assumes** *p* ≠ *0*
shows *ipoly-root-delta p* > *0*
2 ≤ *card* (*Collect* (*root-cond* (*p*, *l*, *r*))) ⟹ *ipoly-root-delta p* ≤ *real-of-rat* (*r* − *l*) / *4*
⟨*proof*⟩

lemma *sgn-less-eq-1-rat*: **fixes** *a b* :: *rat*
shows *sgn a* = *1* ⟹ *a* ≤ *b* ⟹ *sgn b* = *1*
⟨*proof*⟩

lemma *sgn-less-eq-1-real*: **fixes** *a b* :: *real*
shows *sgn a* = *1* ⟹ *a* ≤ *b* ⟹ *sgn b* = *1*
⟨*proof*⟩

definition *compare-1-rat* :: *real-alg-1* ⇒ *rat* ⇒ *order* **where**
compare-1-rat rai = (*let p* = *poly-real-alg-1 rai in*
if degree p = *1 then let x* = *Rat.Fract* (− *coeff p 0*) (*coeff p 1*)
in (λ *y*. *compare y x*)
else (λ *y*. *compare-rat-1 y rai*))

lemma *compare-real-of-rat*: *compare* (*real-of-rat x*) (*of-rat y*) = *compare x y*
⟨*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* ⇒ *rat* **where**
initial-lower-bound l = (*if l* ≤ *1 then l else of-int* (*root-rat-floor n l*))

private definition *initial-upper-bound* :: *rat* ⇒ *rat* **where**
initial-upper-bound r = (*of-int* (*root-rat-ceiling n r*))

context
fixes *cmpx* :: *rat* ⇒ *order*
begin
fun *tighten-bound-root* ::
rat × *rat* ⇒ *rat* × *rat* **where**
tighten-bound-root (*l*′,*r*′) = (*let*
m′ = (*l*′ + *r*′) / *2*;
m = *m*′ ⌢ *n*

```
    in case cmpx m of
      Eq ⇒ (m',m')
    | Lt ⇒ (m',r')
    | Gt ⇒ (l',m'))

lemma tighten-bound-root: assumes sgn: sgn il = 1 real-of-1 x ≥ 0 and
  il: real-of-rat il ≤ root n (real-of-1 x) and
  ir: root n (real-of-1 x) ≤ real-of-rat ir and
  rai: invariant-1 x and
  cmpx: cmpx = compare-1-rat x and
  n: n ≠ 0
shows converges-to (λ i. (tighten-bound-root ^^ i) (il, ir))
    (root n (real-of-1 x)) (is converges-to ?f ?x)
  ⟨proof⟩
end

private fun root-pos-1 :: real-alg-1 ⇒ real-alg-2 where
  root-pos-1 (p,l,r) = (
    (select-correct-factor-int-poly
      (tighten-bound-root (compare-1-rat (p,l,r)))
      (λ x. x)
      (initial-lower-bound l, initial-upper-bound r)
      (poly-nth-root n p)))

fun root-1 :: real-alg-1 ⇒ real-alg-2 where
root-1 (p,l,r) = (
  if n = 0 ∨ r = 0 then Rational 0
  else if r > 0 then root-pos-1 (p,l,r)
  else uminus-2 (root-pos-1 (uminus-1 (p,l,r))))

context
  assumes n: n ≠ 0
begin

lemma initial-upper-bound: assumes x: x > 0 and xr: x ≤ of-rat r
  shows sgn (initial-upper-bound r) = 1 root n x ≤ of-rat (initial-upper-bound r)
⟨proof⟩

lemma initial-lower-bound: assumes l: l > 0 and lx: of-rat l ≤ x
  shows sgn (initial-lower-bound l) = 1 of-rat (initial-lower-bound l) ≤ root n x
⟨proof⟩

lemma root-pos-1:
  assumes x: invariant-1 x and pos: rai-ub x > 0
  defines y: y ≡ root-pos-1 x
  shows invariant-2 y ∧ real-of-2 y = root n (real-of-1 x)
⟨proof⟩

end
```

lemma *root-1*: **assumes** *x*: *invariant-1 x*
defines *y*: *y* ≡ *root-1 x*
shows *invariant-2 y* ∧ (*real-of-2 y* = *root n* (*real-of-1 x*))
⟨*proof*⟩
end

declare *root-1.simps*[*simp del*]

11.2.13 Embedding of Rational Numbers

definition *of-rat-1* :: *rat* ⇒ *real-alg-1* **where**
of-rat-1 x ≡ (*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*
⟨*proof*⟩

fun *info-2* :: *real-alg-2* ⇒ *rat* + *int poly* × *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* = *Inr* (*p*,*n*) ⟹ *poly-cond p* ∧ *ipoly p* (*real-of-2 x*) = *0* ∧ *degree p* ≥ *2*
∧ *card* (*roots-below p* (*real-of-2 x*)) = *n*
info-2 x = *Inl y* ⟹ *real-of-2 x* = *of-rat y*
⟨*proof*⟩

lemma *real-of-2-Irrational*: *invariant-2* (*Irrational n rai*) ⟹ *real-of-2* (*Irrational n rai*) ≠ *of-rat x*
⟨*proof*⟩

lemma *info-2*: **assumes**
ix: *invariant-2 x* **and** *iy*: *invariant-2 y*
shows *info-2 x* = *info-2 y* ⟷ *real-of-2 x* = *real-of-2 y*
⟨*proof*⟩

lemma *info-2-unique*: *invariant-2 x* ⟹ *invariant-2 y* ⟹
real-of-2 x = *real-of-2 y* ⟹ *info-2 x* = *info-2 y*
⟨*proof*⟩

lemma *info-2-inj*: *invariant-2 x* ⟹ *invariant-2 y* ⟹ *info-2 x* = *info-2 y* ⟹
real-of-2 x = *real-of-2 y*
⟨*proof*⟩

context
fixes *cr1 cr2* :: *rat* ⇒ *rat* ⇒ *nat*
begin

partial-function (*tailrec*) *compare-1* :: *int poly* ⇒ *int poly* ⇒ *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* ⇒ *rat* ⇒ *order* **where**
[*code*]: *compare-1 p1 p2 l1 r1 sr1 l2 r2 sr2* = (*if r1* < *l2 then Lt else if r2* < *l1 then Gt*
else let
(*l1'*,*r1'*,*sr1'*) = *tighten-poly-bounds p1 l1 r1 sr1*;
(*l2'*,*r2'*,*sr2'*) = *tighten-poly-bounds p2 l2 r2 sr2*
in compare-1 p1 p2 l1' r1' sr1' l2' r2' sr2')

lemma *compare-1*:
assumes *ur1*: *unique-root* (*p1*,*l1*,*r1*)
and *ur2*: *unique-root* (*p2*,*l2*,*r2*)
and *pc*: *poly-cond2 p1 poly-cond2 p2*
and *diff*: *the-unique-root* (*p1*,*l1*,*r1*) ≠ *the-unique-root* (*p2*,*l2*,*r2*)
and *sr*: *sr1* = *sgn* (*ipoly p1 r1*) *sr2* = *sgn* (*ipoly p2 r2*)
shows *compare-1 p1 p2 l1 r1 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* ⇒ *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* ⇒ *real-alg-2* ⇒ *real-alg-2* **where**
root-2 n x = *root-1 n* (*real-alg-1 x*)

lemma *root-2*: **assumes** *invariant-2 x*
shows *real-of-2* (*root-2 n x*) = *root n* (*real-of-2 x*)
invariant-2 (*root-2 n x*)
⟨*proof*⟩

fun *add-2* :: *real-alg-2* ⇒ *real-alg-2* ⇒ *real-alg-2* **where**
add-2 (*Rational r*) (*Rational q*) = *Rational* (*r* + *q*)
| *add-2* (*Rational r*) (*Irrational n x*) = *Irrational n* (*add-rat-1 r x*)
| *add-2* (*Irrational n x*) (*Rational q*) = *Irrational n* (*add-rat-1 q x*)
| *add-2* (*Irrational n x*) (*Irrational m y*) = *add-1 x y*

lemma *add-2*: **assumes** *x*: *invariant-2 x* **and** *y*: *invariant-2 y*
shows *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* ⇒ *real-alg-2* ⇒ *real-alg-2* **where**
  mult-2 (*Rational r*) (*Rational q*) = *Rational* (*r* ∗ *q*)
| *mult-2* (*Rational r*) (*Irrational n y*) = *mult-rat-1 r y*
| *mult-2* (*Irrational n x*) (*Rational q*) = *mult-rat-1 q x*
| *mult-2* (*Irrational n x*) (*Irrational m y*) = *mult-1 x y*

lemma *mult-2*: **assumes** *invariant-2 x invariant-2 y*
  shows *real-of-2* (*mult-2 x y*) = *real-of-2 x* ∗ *real-of-2 y*
  invariant-2 (*mult-2 x y*)
  ⟨*proof*⟩

fun *to-rat-2* :: *real-alg-2* ⇒ *rat option* **where**
  to-rat-2 (*Rational r*) = *Some r*
| *to-rat-2* (*Irrational n rai*) = *None*

lemma *to-rat-2*: **assumes** *rc*: *invariant-2 x*
  shows *to-rat-2 x* = (*if real-of-2 x* ∈ ℚ *then Some* (*THE q*. *real-of-2 x* = *of-rat q*) *else None*)
⟨*proof*⟩

fun *equal-2* :: *real-alg-2* ⇒ *real-alg-2* ⇒ *bool* **where**
  equal-2 (*Rational r*) (*Rational q*) = (*r* = *q*)
| *equal-2* (*Irrational n* (*p*,-)) (*Irrational m* (*q*,-)) = (*p* = *q* ∧ *n* = *m*)
| *equal-2* (*Rational r*) (*Irrational* - *yy*) = *False*
| *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* ⇒ *real-alg-2* ⇒ *order* **where**
  compare-2 (*Rational r*) (*Rational q*) = (*compare r q*)
| *compare-2* (*Irrational n* (*p*,*l*,*r*)) (*Irrational m* (*q*,*l*′,*r*′)) = (*if p* = *q* ∧ *n* = *m then Eq*
    else compare-1 p q l r (*sgn* (*ipoly p r*)) *l*′ *r*′ (*sgn* (*ipoly q r*′)))
| *compare-2* (*Rational r*) (*Irrational* - *xx*) = (*compare-rat-1 r xx*)
| *compare-2* (*Irrational* - *xx*) (*Rational r*) = (*invert-order* (*compare-rat-1 r xx*))

lemma *compare-2*: **assumes** *rc*: *invariant-2 x invariant-2 y*
  shows *compare-2 x y* = *compare* (*real-of-2 x*) (*real-of-2 y*)
⟨*proof*⟩

fun *sgn-2* :: *real-alg-2* ⇒ *rat* **where**
  sgn-2 (*Rational r*) = *sgn r*
| *sgn-2* (*Irrational n rai*) = *sgn-1 rai*

lemma *sgn-2*: *invariant-2 x* ⟹ *real-of-rat* (*sgn-2 x*) = *sgn* (*real-of-2 x*)
  ⟨*proof*⟩

fun *floor-2* :: *real-alg-2* ⇒ *int* **where**
  floor-2 (*Rational r*) = *floor r*
| *floor-2* (*Irrational n rai*) = *floor-1 rai*

lemma *floor-2*: *invariant-2 x* ⟹ *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* ⇒ *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* ⇒ *real-alg-3* ⇒ *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* ⇒ *real-alg-3* ⇒ *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* ⇒ *real-alg-3* ⇒ *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* ⇒ *real-alg-3* ⇒ *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* ⇒ *real-alg-3* ⇒ *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* ⇒ *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* ⇒ *rat option* **is** *to-rat-2* ⟨*proof*⟩

lemma *to-rat-3*: *to-rat-3 x* =
(*if real-of-3 x* ∈ ℚ *then Some* (*THE q*. *real-of-3 x* = *of-rat q*) *else None*)
⟨*proof*⟩

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

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

lift-definition *info-3* :: *real-alg-3* ⇒ *rat* + *int poly* × *nat* **is** *info-2* ⟨*proof*⟩

lemma *info-3-fun*: *real-of-3 x* = *real-of-3 y* ⟹ *info-3 x* = *info-3 y*
⟨*proof*⟩

lift-definition *info-real-alg* :: *real-alg* ⇒ *rat* + *int poly* × *nat* **is** *info-3*
⟨*proof*⟩

lemma *info-real-alg*:
info-real-alg x = *Inr* (*p*,*n*) ⟹ *p represents* (*real-of x*) ∧ *card* {*y*. *y* ≤ *real-of x*
∧ *ipoly p y* = *0*} = *n* ∧ *irreducible p*
info-real-alg x = *Inl q* ⟹ *real-of x* = *of-rat q*
⟨*proof*⟩

instantiation *real-alg* :: *plus*
begin
lift-definition *plus-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *real-alg* **is** *add-3*
⟨*proof*⟩
instance ⟨*proof*⟩
end

lemma *plus-real-alg*: (*real-of x*) + (*real-of y*) = *real-of* (*x* + *y*)
⟨*proof*⟩

instantiation *real-alg* :: *minus*
begin
definition *minus-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *real-alg* **where**
minus-real-alg x y = *x* + (−*y*)
instance ⟨*proof*⟩
end

lemma *minus-real-alg*: (*real-of x*) − (*real-of y*) = *real-of* (*x* − *y*)

⟨*proof*⟩

lift-definition *of-rat-real-alg* :: *rat* ⇒ *real-alg* **is** *of-rat-3* ⟨*proof*⟩

lemma *of-rat-real-alg*: *real-of-rat x* = *real-of* (*of-rat-real-alg x*)
⟨*proof*⟩

instantiation *real-alg* :: *zero*
begin
definition *zero-real-alg* :: *real-alg* **where** *zero-real-alg* ≡ *of-rat-real-alg 0*
instance ⟨*proof*⟩
end

lemma *zero-real-alg*: *0* = *real-of 0*
⟨*proof*⟩

instantiation *real-alg* :: *one*
begin
definition *one-real-alg* :: *real-alg* **where** *one-real-alg* ≡ *of-rat-real-alg 1*
instance ⟨*proof*⟩
end

lemma *one-real-alg*: *1* = *real-of 1*
⟨*proof*⟩

instantiation *real-alg* :: *times*
begin
lift-definition *times-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *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* ⇒ *real-alg* **is** *inverse-3*
⟨*proof*⟩
definition *divide-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *real-alg* **where**
divide-real-alg x y = *x* ∗ *inverse y*
instance ⟨*proof*⟩
end

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

lemma *divide-real-alg*: (*real-of x*) / (*real-of y*) = *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* ⇒ *real-alg* ⇒ *real-alg* **is** *root-3*
⟨*proof*⟩

lemma *root-real-alg*: *root n* (*real-of x*) = *real-of* (*root-real-alg n x*)
⟨*proof*⟩

lift-definition *sgn-real-alg-rat* :: *real-alg* ⇒ *rat* **is** *sgn-3*
⟨*proof*⟩

lemma *sgn-real-alg-rat*: *real-of-rat* (*sgn-real-alg-rat x*) = *sgn* (*real-of x*)
⟨*proof*⟩

instantiation *real-alg* :: *sgn*
begin
definition *sgn-real-alg* :: *real-alg* ⇒ *real-alg* **where**
sgn-real-alg x = *of-rat-real-alg* (*sgn-real-alg-rat x*)
instance ⟨*proof*⟩
end

lemma *sgn-real-alg*: *sgn* (*real-of x*) = *real-of* (*sgn x*)
⟨*proof*⟩

instantiation *real-alg* :: *equal*
begin
lift-definition *equal-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *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* ⇒ *real-alg* ⇒ *bool* **where**
[*code del*]: *less-real-alg* x y = (*real-of* x < *real-of* y)

definition *less-eq-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *bool* **where**
[*code del*]: *less-eq-real-alg* x y = (*real-of* x ≤ *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 ≤ y) ⟨*proof*⟩

instantiation *real-alg* :: *compare-order*
begin

lift-definition *compare-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *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* ⇒ *real-alg* ⇒ *bool*) = *le-of-comp compare*
(*less* :: *real-alg* ⇒ *real-alg* ⇒ *bool*) = *lt-of-comp compare*
⟨*proof*⟩

instantiation *real-alg* :: *abs*
begin

definition *abs-real-alg* :: *real-alg* ⇒ *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*: *sgn x* = (*if x* = *0 then 0 else if 0* < *real-of x then 1 else* − *1*)
(**is** - = *?r*)
⟨*proof*⟩

lemma *real-of-of-int*: *real-of-rat* (*rat-of-int z*) = *real-of* (*of-int z*)
⟨*proof*⟩

instance *real-alg* :: *linordered-field*
⟨*proof*⟩

instantiation *real-alg* :: *floor-ceiling*
begin
lift-definition *floor-real-alg* :: *real-alg* ⇒ *int* **is** *floor-3*
⟨*proof*⟩

lemma *floor-real-alg*: *floor* (*real-of x*) = *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* ⇒ -)
definition [*simp*]: *unit-factor-real-alg* = (*unit-factor-field* :: *real-alg* ⇒ -)
definition [*simp*]: *modulo-real-alg* = (*mod-field* :: *real-alg* ⇒ -)
definition [*simp*]: *euclidean-size-real-alg* = (*euclidean-size-field* :: *real-alg* ⇒ -)
definition [*simp*]: *division-segment* (*x* :: *real-alg*) = *1*

instance
⟨*proof*⟩

end

instantiation *real-alg* :: *euclidean-ring-gcd*
begin

definition *gcd-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *real-alg* **where**
gcd-real-alg = *Euclidean-Algorithm.gcd*
definition *lcm-real-alg* :: *real-alg* ⇒ *real-alg* ⇒ *real-alg* **where**
lcm-real-alg = *Euclidean-Algorithm.lcm*
definition *Gcd-real-alg* :: *real-alg set* ⇒ *real-alg* **where**
Gcd-real-alg = *Euclidean-Algorithm.Gcd*
definition *Lcm-real-alg* :: *real-alg set* ⇒ *real-alg* **where**
Lcm-real-alg = *Euclidean-Algorithm.Lcm*

instance ⟨*proof*⟩

end

instance *real-alg* :: *field-gcd* ⟨*proof*⟩

definition *min-int-poly-real-alg* :: *real-alg* ⇒ *int poly* **where**
min-int-poly-real-alg x = (*case info-real-alg x of Inl r* ⇒ *poly-rat r* | *Inr* (*p*,-) ⇒ *p*)

lemma *min-int-poly-real-alg-real-of*: *min-int-poly-real-alg x* = *min-int-poly* (*real-of x*)
⟨*proof*⟩

lemma *min-int-poly-real-code*: *min-int-poly-real* (*real-of x*) = *min-int-poly-real-alg x*
⟨*proof*⟩

lemma *min-int-poly-real-of*: *min-int-poly* (*real-of x*) = *min-int-poly x*
⟨*proof*⟩

definition *real-alg-of-real* :: *real* ⇒ *real-alg* **where**
real-alg-of-real x = (*if* (∃ *y*. *x* = *real-of y*) *then* (*THE y*. *x* = *real-of y*) *else 0*)

lemma *real-alg-of-real-code*[*code*]: *real-alg-of-real* (*real-of x*) = *x*
⟨*proof*⟩

lift-definition *to-rat-real-alg-main* :: *real-alg* ⇒ *rat option* **is** *to-rat-3*
⟨*proof*⟩

lemma *to-rat-real-alg-main*: *to-rat-real-alg-main x* = (*if real-of x* ∈ ℚ *then*
Some (*THE q*. *real-of x* = *of-rat q*) *else None*)
⟨*proof*⟩

definition *to-rat-real-alg* :: *real-alg* ⇒ *rat* **where**
to-rat-real-alg x = (*case to-rat-real-alg-main x of Some q* ⇒ *q* | *None* ⇒ *0*)

definition *is-rat-real-alg* :: *real-alg* ⇒ *bool* **where**
is-rat-real-alg x = (*case to-rat-real-alg-main x of Some q* ⇒ *True* | *None* ⇒ *False*)

lemma *is-rat-real-alg*: *is-rat* (*real-of x*) = (*is-rat-real-alg x*)
⟨*proof*⟩

lemma *to-rat-real-alg*: *to-rat* (*real-of x*) = (*to-rat-real-alg x*)
⟨*proof*⟩

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* ∘ *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 ⇒ *root-info* ⇒ (*rat* ⇒ *rat* ⇒ *nat*) ⇒ (*rat* × *rat*)*list* ⇒ *real-alg-2 list* ⇒ *real-alg-2 list* **where**
[*code*]: *roots-of-2-main p ri cr lrs rais* = (*case lrs of Nil* ⇒ *rais*
| (*l*,*r*) # *lrs* ⇒ *let c* = *cr l r in*
if c = *0 then roots-of-2-main p ri cr lrs rais*
else if c = *1 then roots-of-2-main p ri cr lrs* (*real-alg-2''' ri p l r* # *rais*)
else let m = (*l* + *r*) / *2 in roots-of-2-main p ri cr* ((*m*,*r*) # (*l*,*m*) # *lrs*) *rais*)

definition *roots-of-2-irr* :: *int poly* ⇒ *real-alg-2 list* **where**
roots-of-2-irr p = (*if degree p* = *1*
then [*Rational* (*Rat.Fract* (− *coeff p 0*) (*coeff p 1*))] *else*
let ri = *root-info p*;
cr = *root-info.l-r ri*;
B = *root-bound p*
in (*roots-of-2-main p ri cr* [(−*B*,*B*)] []))

fun *pairwise-disjoint* :: *'a set list* ⇒ *bool* **where**
pairwise-disjoint [] = *True*
| *pairwise-disjoint* (*x* # *xs*) = ((*x* ∩ (⋃ *y* ∈ *set xs*. *y*) = {}) ∧ *pairwise-disjoint xs*)

lemma *roots-of-2-irr*: **assumes** *pc*: *poly-cond p* **and** *deg*: *degree p* > *0*

shows *real-of-2 ‘ set (roots-of-2-irr p) = {x. ipoly p x = 0}* (**is** *?one*)
Ball (set (roots-of-2-irr p)) invariant-2 (**is** *?two*)
distinct (map real-of-2 (roots-of-2-irr p)) (**is** *?three*)
⟨*proof*⟩

definition *roots-of-2 :: int poly ⇒ real-alg-2 list* **where**
roots-of-2 p = concat (map roots-of-2-irr
(factors-of-int-poly p))

lemma *roots-of-2*:
shows *p ≠ 0 ⟹ real-of-2 ‘ set (roots-of-2 p) = {x. ipoly p x = 0}*
Ball (set (roots-of-2 p)) invariant-2
distinct (map real-of-2 (roots-of-2 p))
⟨*proof*⟩

lift-definition (*code-dt*) *roots-of-3 :: int poly ⇒ real-alg-3 list* **is** *roots-of-2*
⟨*proof*⟩

lemma *roots-of-3*:
shows *p ≠ 0 ⟹ real-of-3 ‘ set (roots-of-3 p) = {x. ipoly p x = 0}*
distinct (map real-of-3 (roots-of-3 p))
⟨*proof*⟩

lift-definition *roots-of-real-alg :: int poly ⇒ real-alg list* **is** *roots-of-3* ⟨*proof*⟩

lemma *roots-of-real-alg*:
p ≠ 0 ⟹ real-of ‘ set (roots-of-real-alg p) = {x. ipoly p x = 0}
distinct (map real-of (roots-of-real-alg p))
⟨*proof*⟩

definition *real-roots-of-int-poly :: int poly ⇒ real list* **where**
real-roots-of-int-poly p = map real-of (roots-of-real-alg p)

definition *real-roots-of-rat-poly :: rat poly ⇒ real list* **where**
real-roots-of-rat-poly p = map real-of (roots-of-real-alg (snd (rat-to-int-poly p)))

abbreviation *rpoly :: rat poly ⇒ 'a :: field-char-0 ⇒ 'a*
where *rpoly f ≡ poly (map-poly of-rat f)*

lemma *real-roots-of-int-poly*: *p ≠ 0 ⟹ set (real-roots-of-int-poly p) = {x. ipoly p x = 0}*
distinct (real-roots-of-int-poly p)
⟨*proof*⟩

lemma *real-roots-of-rat-poly*: *p ≠ 0 ⟹ set (real-roots-of-rat-poly p) = {x. rpoly p x = 0}*
distinct (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* $\Longrightarrow$ *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* ^ *n*)) $\subseteq \mathbb{R}$
⟨*proof*⟩

lemma *real-poly-prod*: **fixes** *f* :: *'a* ⇒ *'b* :: {*idom*,*real-algebra-1*} *poly*
assumes $\bigwedge$ *x*. *x* $\in$ *S* $\Longrightarrow$ *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*: *set* (*coeffs p*) ⊆ ℝ **and** ∗: *set* (*coeffs q*) ⊆ ℝ
shows *real-poly-div*: *set* (*coeffs* (*q div p*)) ⊆ ℝ
and *real-poly-mod*: *set* (*coeffs* (*q mod p*)) ⊆ ℝ
⟨*proof*⟩

lemma *real-poly-factor*: **fixes** *p* :: *'a* :: *real-field poly*
assumes *set* (*coeffs* (*p* ∗ *q*)) ⊆ ℝ
set (*coeffs p*) ⊆ ℝ
p ≠ *0*
shows *set* (*coeffs q*) ⊆ ℝ
⟨*proof*⟩

lemma *complex-conjugate-order*: **assumes** *real*: *set* (*coeffs p*) ⊆ ℝ
p ≠ *0*
shows *order* (*cnj c*) *p* = *order c p*
⟨*proof*⟩

lemma *map-poly-of-real-Re*: **assumes** *set* (*coeffs p*) ⊆ ℝ
shows *map-poly of-real* (*map-poly Re p*) = *p*
⟨*proof*⟩

lemma *map-poly-Re-of-real*: *map-poly Re* (*map-poly of-real p*) = *p*
⟨*proof*⟩

lemma *map-poly-Re-mult*: **assumes** *p*: *set* (*coeffs p*) ⊆ ℝ
and *q*: *set* (*coeffs q*) ⊆ ℝ **shows** *map-poly Re* (*p* ∗ *q*) = *map-poly Re p* ∗ *map-poly Re q*
⟨*proof*⟩

lemma *map-poly-Re-power*: **assumes** *p*: *set* (*coeffs p*) ⊆ ℝ
shows *map-poly Re* (*p*^*n*) = (*map-poly Re p*)^*n*
⟨*proof*⟩

lemma *real-degree-2-factorization-exists-complex*: **fixes** *p* :: *complex poly*
assumes *pR*: *set* (*coeffs p*) ⊆ ℝ
shows ∃ *qs*. *p* = *prod-list qs* ∧ (∀ *q* ∈ *set qs*. *set* (*coeffs q*) ⊆ ℝ ∧ *degree q* ≤ *2*)
⟨*proof*⟩

lemma *real-degree-2-factorization-exists*: **fixes** *p* :: *real poly*
shows ∃ *qs*. *p* = *prod-list qs* ∧ (∀ *q* ∈ *set qs*. *degree q* ≤ *2*)
⟨*proof*⟩

lemma *odd-degree-imp-real-root*: **assumes** *odd* (*degree p*)
shows ∃ *x*. *poly p x* = (*0* :: *real*)
⟨*proof*⟩

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-fin*]]
declare [[*code drop*: *Lcm-fin*]]

definition *gcds* :: *'a*::*semiring-gcd list* ⇒ *'a*
where [*simp*, *code-abbrev*]: *gcds xs* = *gcd-list xs*

lemma [*code*]:
gcds xs = *fold gcd xs 0*
⟨*proof*⟩

definition *lcms* :: *'a*::*semiring-gcd list* ⇒ *'a*
where [*simp*, *code-abbrev*]: *lcms xs* = *lcm-list xs*

lemma [*code*]:
lcms xs = *fold lcm xs 1*
⟨*proof*⟩

lemma *in-reals-code* [*code-unfold*]:
x ∈ ℝ ⟷ *Im x* = *0*
⟨*proof*⟩

definition *is-norm-1* :: *complex* ⇒ *bool* **where**
is-norm-1 z = ((*Re z*)^2 + (*Im z*)^2 = *1*)

lemma *is-norm-1*[*simp*]: *is-norm-1 x* = (*norm x* = *1*)
⟨*proof*⟩

definition *is-norm-le-1* :: *complex* ⇒ *bool* **where**
is-norm-le-1 z = ((*Re z*)^2 + (*Im z*)^2 ≤ *1*)

lemma *is-norm-le-1*[*simp*]: *is-norm-le-1 x* = (*norm x* ≤ *1*)
⟨*proof*⟩

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* ⟨*proof*⟩

Intervals do not form an additive group, but satisfy some properties.

lemma *interval-uminus-zero*[*simp*]:
shows $-(0 :: 'a :: \textit{group-add interval}) = 0$
⟨*proof*⟩

lemma *interval-diff-zero*[*simp*]:
fixes $a :: 'a :: \textit{cancel-comm-monoid-add interval}$
shows $a - 0 = a$ ⟨*proof*⟩

Without type invariant, intervals do not form a multiplicative monoid, but satisfy some properties.

instance *interval* :: ({*linorder*,*mult-zero*}) *mult-zero*
⟨*proof*⟩

14.3 Membership

fun *in-interval* :: $'a :: \textit{order} \Rightarrow 'a$ *interval* ⇒ *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*!]: $a \in_i$ *to-interval a*
⟨*proof*⟩

lemma *plus-in-interval*:
fixes $x\ y :: 'a :: \textit{ordered-comm-monoid-add}$
shows $x \in_i X \Longrightarrow y \in_i Y \Longrightarrow x + y \in_i X + Y$
⟨*proof*⟩

lemma *uminus-in-interval*:
fixes $x :: 'a :: \textit{ordered-ab-group-add}$
shows $x \in_i X \Longrightarrow -x \in_i -X$
⟨*proof*⟩

lemma *minus-in-interval*:
fixes $x\ y :: 'a :: \textit{ordered-ab-group-add}$
shows $x \in_i X \Longrightarrow y \in_i Y \Longrightarrow x - y \in_i X - Y$
⟨*proof*⟩

lemma *times-in-interval*:
fixes $x\ y :: 'a :: \textit{linordered-ring}$
assumes $x \in_i X\ y \in_i Y$
shows $x * y \in_i X * Y$
⟨*proof*⟩

14.4 Convergence

definition *interval-tendsto* :: $(\textit{nat} \Rightarrow 'a :: \textit{topological-space interval}) \Rightarrow 'a \Rightarrow \textit{bool}$

(**infixr** $\longrightarrow_i$ *55*) **where**
$(X \longrightarrow_i x) \equiv ((interval.upper \circ X) \longrightarrow x) \wedge ((interval.lower \circ X) \longrightarrow x)$

lemma *interval-tendstoI*[*intro*]:
assumes $(interval.upper \circ X) \longrightarrow x$ **and** $(interval.lower \circ X) \longrightarrow x$
shows $X \longrightarrow_i x$
⟨*proof*⟩

lemma *const-interval-tendsto*: $(\lambda i.\ to\text{-}interval\ a) \longrightarrow_i a$
⟨*proof*⟩

lemma *interval-tendsto-0*: $(\lambda i.\ 0) \longrightarrow_i 0$
⟨*proof*⟩

lemma *plus-interval-tendsto*:
fixes $x\ y :: 'a :: topological\text{-}monoid\text{-}add$
assumes $X \longrightarrow_i x\ Y \longrightarrow_i y$
shows $(\lambda\ i.\ X\ i + Y\ i) \longrightarrow_i x + y$
⟨*proof*⟩

lemma *uminus-interval-tendsto*:
fixes $x :: 'a :: topological\text{-}group\text{-}add$
assumes $X \longrightarrow_i x$
shows $(\lambda i.\ -\ X\ i) \longrightarrow_i -x$
⟨*proof*⟩

lemma *minus-interval-tendsto*:
fixes $x\ y :: 'a :: topological\text{-}group\text{-}add$
assumes $X \longrightarrow_i x\ Y \longrightarrow_i y$
shows $(\lambda\ i.\ X\ i - Y\ i) \longrightarrow_i x - y$
⟨*proof*⟩

lemma *times-interval-tendsto*:
fixes $x\ y :: 'a :: \{linorder\text{-}topology, real\text{-}normed\text{-}algebra\}$
assumes $X \longrightarrow_i x\ Y \longrightarrow_i y$
shows $(\lambda\ i.\ X\ i * Y\ i) \longrightarrow_i x * y$
⟨*proof*⟩

lemma *interval-tendsto-neq*:
fixes $a\ b :: real$
assumes $(\lambda\ i.\ f\ i) \longrightarrow_i a$ **and** $a \neq b$
shows $\exists\ n.\ \neg\ b \in_i f\ n$
⟨*proof*⟩

14.5 Complex Intervals

datatype *complex-interval* = *Complex-Interval* (*Re-interval*: *real interval*) (*Im-interval*: *real interval*)

definition *in-complex-interval* :: *complex* ⇒ *complex-interval* ⇒ *bool* ((-/ ∈$_c$ -) [51, 51] 50) **where**
y ∈$_c$ *x* ≡ (*case x of Complex-Interval r i* ⇒ *Re y* ∈$_i$ *r* ∧ *Im y* ∈$_i$ *i*)

instantiation *complex-interval* :: *comm-monoid-add* **begin**

definition *0* ≡ *Complex-Interval 0 0*

fun *plus-complex-interval* :: *complex-interval* ⇒ *complex-interval* ⇒ *complex-interval* **where**
Complex-Interval rx ix + *Complex-Interval ry iy* = *Complex-Interval* (*rx* + *ry*) (*ix* + *iy*)

instance
⟨*proof*⟩
end

lemma *plus-complex-interval*: *x* ∈$_c$ *X* ⟹ *y* ∈$_c$ *Y* ⟹ *x* + *y* ∈$_c$ *X* + *Y*
⟨*proof*⟩

definition *of-int-complex-interval* :: *int* ⇒ *complex-interval* **where**
of-int-complex-interval x = *Complex-Interval* (*of-int-interval x*) *0*

lemma *of-int-complex-interval-0*[*simp*]: *of-int-complex-interval 0* = *0*
⟨*proof*⟩

lemma *of-int-complex-interval*: *of-int i* ∈$_c$ *of-int-complex-interval i*
⟨*proof*⟩

instantiation *complex-interval* :: *mult-zero* **begin**

fun *times-complex-interval* **where**
Complex-Interval rx ix ∗ *Complex-Interval ry iy* =
Complex-Interval (*rx* ∗ *ry* − *ix* ∗ *iy*) (*rx* ∗ *iy* + *ix* ∗ *ry*)

instance
⟨*proof*⟩
end

instantiation *complex-interval* :: *minus* **begin**

fun *minus-complex-interval* **where**
Complex-Interval R I − *Complex-Interval R′ I′* = *Complex-Interval* (*R*−*R′*) (*I*−*I′*)

instance⟨*proof*⟩

end

lemma *times-complex-interval*: $x \in_c X \Longrightarrow y \in_c Y \Longrightarrow x * y \in_c X * Y$
⟨*proof*⟩

definition *ipoly-complex-interval* :: *int poly* ⇒ *complex-interval* ⇒ *complex-interval*
where
ipoly-complex-interval p x = *fold-coeffs* (λ*a b*. *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 (*pCons a p*) *x* = *of-int-complex-interval a* + *x* ∗ (*ipoly-complex-interval p x*)
⟨*proof*⟩

lemma *ipoly-complex-interval*: **assumes** *x*: $x \in_c X$
shows *ipoly p x* $\in_c$ *ipoly-complex-interval p X*
⟨*proof*⟩

definition *complex-interval-tendsto* (**infix** $\longrightarrow_c$ *55*) **where**
$C \longrightarrow_c c \equiv ((Re\text{-}interval \circ C) \longrightarrow_i Re\ c) \wedge ((Im\text{-}interval \circ C) \longrightarrow_i Im\ c)$

lemma *complex-interval-tendstoI*[*intro*!]:
$(Re\text{-}interval \circ C) \longrightarrow_i Re\ c \Longrightarrow (Im\text{-}interval \circ C) \longrightarrow_i Im\ c \Longrightarrow C \longrightarrow_c c$
⟨*proof*⟩

lemma *of-int-complex-interval-tendsto*: $(\lambda i.\ of\text{-}int\text{-}complex\text{-}interval\ n) \longrightarrow_c of\text{-}int\ n$
⟨*proof*⟩

lemma *Im-interval-plus*: *Im-interval* (*A* + *B*) = *Im-interval A* + *Im-interval B*
⟨*proof*⟩

lemma *Re-interval-plus*: *Re-interval* (*A* + *B*) = *Re-interval A* + *Re-interval B*
⟨*proof*⟩

lemma *Im-interval-minus*: *Im-interval* (*A* − *B*) = *Im-interval A* − *Im-interval B*

⟨*proof*⟩

lemma *Re-interval-minus*: *Re-interval* (*A* − *B*) = *Re-interval A* − *Re-interval B*
⟨*proof*⟩

lemma *Re-interval-times*: *Re-interval* (*A* ∗ *B*) = *Re-interval A* ∗ *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*:
$A \longrightarrow_c a \Longrightarrow B \longrightarrow_c b \Longrightarrow (\lambda i.\ A\ i + B\ i) \longrightarrow_c a + b$
⟨*proof*⟩

lemma *minus-complex-interval-tendsto*:
$A \longrightarrow_c a \Longrightarrow B \longrightarrow_c b \Longrightarrow (\lambda i.\ A\ i - B\ i) \longrightarrow_c a - b$
⟨*proof*⟩

lemma *times-complex-interval-tendsto*:
$A \longrightarrow_c a \Longrightarrow B \longrightarrow_c b \Longrightarrow (\lambda i.\ A\ i * B\ i) \longrightarrow_c a * b$
⟨*proof*⟩

lemma *ipoly-complex-interval-tendsto*:
assumes $C \longrightarrow_c c$
shows $(\lambda i.$ *ipoly-complex-interval p* $(C\ i)) \longrightarrow_c$ *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* ⇒ *complex poly* **where**
complex-of-int-poly ≡ *map-poly of-int*

abbreviation *complex-of-rat-poly* :: *rat poly* ⇒ *complex poly* **where**
complex-of-rat-poly ≡ *map-poly of-rat*

lemma *poly-complex-to-real*: (*poly* (*complex-of-int-poly p*) (*complex-of-real x*) = *0*)
= (*poly* (*real-of-int-poly p*) *x* = *0*)
⟨*proof*⟩

lemma *represents-cnj*: **assumes** *p represents x* **shows** *p represents* (*cnj x*)
⟨*proof*⟩

definition *poly-2i* :: *int poly* **where**
poly-2i ≡ [: *4*, *0*, *1*:]

lemma *represents-2i*: *poly-2i represents* (*2* ∗ i)
⟨*proof*⟩

definition *root-poly-Re* :: *int poly* ⇒ *int poly* **where**
root-poly-Re p = *cf-pos-poly* (*poly-mult-rat* (*inverse 2*) (*poly-add p p*))

lemma *root-poly-Re-code*[*code*]:
root-poly-Re p = (*let fs* = *coeffs* (*poly-add p p*); *k* = *length fs*
in cf-pos-poly (*poly-of-list* (*map* (λ(*fi*, *i*). *fi* ∗ *2* ^ *i*) (*zip fs* [*0*..<*k*]))))
⟨*proof*⟩

definition *root-poly-Im* :: *int poly* ⇒ *int poly list* **where**
root-poly-Im p = (*let fs* = *factors-of-int-poly*
(*poly-add p* (*poly-uminus p*))
in remdups ((*if* (∃ *f* ∈ *set fs*. *coeff f 0* = *0*) *then* [[:*0*,*1*:]] *else* [])) @
[*cf-pos-poly* (*poly-div f poly-2i*) . *f* ← *fs*, *coeff f 0* ≠ *0*])

lemma *represents-root-poly*:
assumes *ipoly p x* = *0* **and** *p*: *p* ≠ *0*
shows (*root-poly-Re p*) *represents* (*Re x*)
and ∃ *q* ∈ *set* (*root-poly-Im p*). *q represents* (*Im x*)
⟨*proof*⟩

definition *complex-poly* :: *int poly* ⇒ *int poly* ⇒ *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 ∃ *f* ∈ *set* (*complex-poly re im*). *f represents x* ⋀ *f*. *f* ∈ *set* (*complex-poly re im*) ⟹ *poly-cond f*
⟨*proof*⟩

lemma *algebraic-complex-iff*: *algebraic x* = (*algebraic* (*Re x*) ∧ *algebraic* (*Im x*))
⟨*proof*⟩

definition *algebraic-complex* :: *complex* ⇒ *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*) ∧ *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 *croots2*

hide-const (**open**) *eq*

primrec *remdups-gen* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ '*a list* ⇒ '*a list* **where**
remdups-gen eq [] = []
| *remdups-gen eq* (*x* # *xs*) = (*if* (∃ *y* ∈ *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* (λ *x*. *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 *Re x* ∈ *real-of-3* ‘ *set* (*real-parts-3 p*)
⟨*proof*⟩

lemma *distinct-real-parts-3*: *distinct* (*map real-of-3* (*real-parts-3 p*))
⟨*proof*⟩

lemma *pos-imaginary-parts-3*: **assumes** *p*: $p \neq 0$ **and** *ipoly p x* = *0* **and** *Im x* > *0*
shows *Im x* ∈ *real-of-3* ‘ *set* (*pos-imaginary-parts-3 p*)
⟨*proof*⟩

lemma *distinct-pos-imaginary-parts-3*: *distinct* (*map real-of-3* (*pos-imaginary-parts-3 p*))
⟨*proof*⟩

lemma *remdups-gen-subset*: *set* (*remdups-gen eq xs*) ⊆ *set xs*
⟨*proof*⟩

lemma *positive-pos-imaginary-parts-3*: **assumes** *x* ∈ *set* (*pos-imaginary-parts-3 p*)
shows *0* < *real-of-3 x*
⟨*proof*⟩

definition *pair-to-complex ri* ≡ *case ri of* (*r*,*i*) ⇒ *Complex* (*real-of-3 r*) (*real-of-3 i*)

fun *get-itvl-2* :: *real-alg-2* ⇒ *real interval* **where**
get-itvl-2 (*Irrational n* (*p*,*l*,*r*)) = *Interval* (*of-rat l*) (*of-rat r*)
| *get-itvl-2* (*Rational r*) = (*let rr* = *of-rat r in Interval rr rr*)

lemma *get-bounds-2*: **assumes** *invariant-2 x*
shows *real-of-2 x* ∈$_i$ *get-itvl-2 x*
⟨*proof*⟩

lift-definition *get-itvl-3* :: *real-alg-3* ⇒ *real interval* **is** *get-itvl-2* ⟨*proof*⟩

lemma *get-itvl-3*: *real-of-3 x* ∈$_i$ *get-itvl-3 x*
⟨*proof*⟩

fun *tighten-bounds-2* :: *real-alg-2* ⇒ *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*′,-) ⇒ *Irrational n* (*p*,*l*′,*r*′))
| *tighten-bounds-2* (*Rational r*) = *Rational r*

lemma *tighten-bounds-2*: **assumes** *inv*: *invariant-2 x*
shows *real-of-2* (*tighten-bounds-2 x*) = *real-of-2 x invariant-2* (*tighten-bounds-2*

x)
  get-itvl-2 x = Interval l r $\Longrightarrow$
   get-itvl-2 (tighten-bounds-2 x) = Interval l′ r′ $\Longrightarrow$ $r' - l' = (r - l) \;/\; 2$
⟨*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 $\Longrightarrow$
   get-itvl-3 (tighten-bounds-3 x) = Interval l′ r′ $\Longrightarrow$ $r' - l' = (r - l) \;/\; 2$
  ⟨*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 ∈ set xs* $\Longrightarrow$ *P x* $\Longrightarrow$ *p ((f ^^ i) x)*
  and $\bigwedge$ *x. x ∈ set xs* $\Longrightarrow$ $\neg$ *P x* $\Longrightarrow$ $\exists$ *i.* $\neg$ *p ((f ^^ 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 (λ 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 (λ x. [:−x,1:]) rrts));
        cpp = map-poly (λ r. Complex r 0) pp
      in crts @ croots2 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)
            (λ (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, cnj 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-itvl-get-bounds-tighten*: *real-of-3 x ∈ᵢ get-itvl-3 ((tighten-bounds-3 ^^ n) x)*
⟨*proof*⟩

lemma *sandwitch-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-itvl-3 ((tighten-bounds-3 ^^ i) x))*
definition *upper-3* **where** *upper-3 x i ≡ interval.upper (get-itvl-3 ((tighten-bounds-3 ^^ i) x))*

lemma *interval-size-3*: *upper-3 x i − lower-3 x i = (upper-3 x 0 − lower-3 x 0)/2^i*
⟨*proof*⟩

lemma *interval-size-3-tendsto-0*: (λ*i*. (*upper-3 x i − lower-3 x i*)) ⟶ *0*
⟨*proof*⟩

lemma *dist-tendsto-0-imp-tendsto*: (λ*i*. |*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*: (λ*x*. *get-itvl-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*)
⟨*proof*⟩

lemma *complex-roots-of-int-poly-all*: **assumes** *sf*: *degree p* ≥ *3* ⟹ *square-free p*
shows $p \neq 0 \Longrightarrow$ *set* (*complex-roots-of-int-poly-all p*) = {*x*. *ipoly p x* = *0*} (**is** - ⟹ *set ?l* = *?r*)
and *distinct* (*complex-roots-of-int-poly-all p*)
⟨*proof*⟩

It now comes the preferred function to compute complex roots of an integer polynomial.

definition *complex-roots-of-int-poly* :: *int poly* ⇒ *complex list* **where**
complex-roots-of-int-poly p = (
let ps = (*if degree p* ≥ *3 then factors-of-int-poly p else* [*p*])
in concat (*map complex-roots-of-int-poly-all ps*))

definition *complex-roots-of-rat-poly* :: *rat poly* ⇒ *complex list* **where**
complex-roots-of-rat-poly p = *complex-roots-of-int-poly* (*snd* (*rat-to-int-poly p*))

lemma *complex-roots-of-int-poly*:
shows $p \neq 0 \Longrightarrow$ *set* (*complex-roots-of-int-poly p*) = {*x*. *ipoly p x* = *0*} (**is** - ⟹ *?l* = *?r*)
and *distinct* (*complex-roots-of-int-poly p*)
⟨*proof*⟩

lemma *complex-roots-of-rat-poly*:
$p \neq 0 \Longrightarrow$ *set* (*complex-roots-of-rat-poly p*) = {*x*. *rpoly p x* = *0*} (**is** - ⟹ *?l* = *?r*)
distinct (*complex-roots-of-rat-poly p*)
⟨*proof*⟩

lemma *min-int-poly-complex-of-real*[*simp*]: *min-int-poly* (*complex-of-real x*) = *min-int-poly x*
⟨*proof*⟩

TODO: the implemention 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* ∈ *set* (*complex-roots-of-int-poly f*)

definition *min-int-poly-complex* :: *complex* ⇒ *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* (λ *f*. *x* ∈ *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* ⇒ *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* ≡ *show-real*
begin
definition *show-real* ≡ *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* ≡ *show-real-alg*
begin

definition *show-real-alg*[*code*]: *show-real-alg x* ≡ *let*
x1000′ = *floor* (*1000* ∗ *x*);
(*x1000*,*s*) = (*if x1000′* < *0 then* (−*x1000′*, *′′−′′*) *else* (*x1000′*, *′′′′*));
(*bef*,*aft*) = *divmod-int x1000 1000*;
a′ = *show aft*;
a = *replicate* (*3*−*length a′*) (*CHR ′′0′′*) @ *a′*
in
′′ ∼′′ @ *s* @ *show bef* @ *′′.′′* @ *a*

end

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* × *nat* ⇒ *real-alg-show-info* **where**
convert-info (*Inl q*) = *Rat-Info q*
| *convert-info* (*Inr* (*f*,*n*)) = (*if degree f* = *2 then* (*let a* = *coeff f 2*; *b* = *coeff f 1*; *c* = *coeff f 0*;

```
    b2a = Rat.Fract (−b) (2 ∗ a);
    below = Rat.Fract (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 ⇒ real-alg-show-info where
  real-alg-show-info x = convert-info (info-real-alg x)
```

We prove that the extracted information for showing an algebraic real number is correct.

```
lemma real-alg-show-info: real-alg-show-info x = Rat-Info r ⟹ real-of x = of-rat
r
  real-alg-show-info x = Sqrt-Info r sq ⟹ real-of x = of-rat r + sqrt (of-rat sq)
  real-alg-show-info x = Real-Alg-Info p n ⟹ p represents (real-of x) ∧ n = card
{y. y ≤ real-of x ∧ ipoly p y = 0}
  (is ?l ⟹ ?r)
⟨proof⟩

fun show-rai-info :: int ⇒ real-alg-show-info ⇒ string where
  show-rai-info fl (Rat-Info r) = show r
| show-rai-info fl (Sqrt-Info r sq) = (let sqrt = ''sqrt('' @ show (abs sq) @ '')''
     in if r = 0 then (if sq < 0 then '' −'' else []) @ sqrt
           else (''('' @ show r @ (if sq < 0 then ''−'' else ''+'') @ sqrt @ '')''))
| show-rai-info fl (Real-Alg-Info p n) =
   ''(root #'' @ show n @ '' of '' @ show p @ '', in ('' @ show fl @ '','' @ show (fl
+ 1) @ ''))''

overloading show-real-alg ≡ show-real-alg
begin
  definition show-real-alg[code]:
    show-real-alg x ≡ show-rai-info (floor x) (real-alg-show-info x)
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* ≡ *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 *root 3 4* > *sqrt* (*root 4 3*) + ⌊*1/10* ∗ *root 3 7*⌋ ⟨*proof*⟩
lemma *csqrt* (*4* + *3* ∗ i) ∉ ℝ ⟨*proof*⟩
value [*code*] *show* (*csqrt* (*4* + *3* ∗ i))
value [*code*] *show* (*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.