# Algebraic Numbers in Isabelle/HOL* 

René Thiemann, Akihisa Yamada, and Sebastiaan Joosten

September 13, 2023


#### Abstract

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

To this end, we mechanized several results on resultants.


## Contents

1 Introduction ..... 3
2 Auxiliary Algorithms ..... 5
3 Algebraic Numbers - Excluding Addition and Multiplica- tion ..... 5
3.1 Polynomial Evaluation of Integer and Rational Polynomials in Fields. ..... 8
3.2 Algebraic Numbers - Definition, Inverse, and Roots ..... 8
4 Resultants ..... 28
4.1 Bivariate Polynomials ..... 28
4.1.1 Evaluation of Bivariate Polynomials ..... 29
4.1.2 Swapping the Order of Variables ..... 31
4.2 Resultant ..... 39
4.2.1 Sylvester matrices and vector representation of poly- nomials ..... 39
4.2.2 Homomorphism and Resultant ..... 47

[^0]4.2.3 Resultant as Polynomial Expression ..... 47
4.2.4 Resultant as Nonzero Polynomial Expression ..... 56
5 Algebraic Numbers: Addition and Multiplication ..... 61
5.1 Addition of Algebraic Numbers ..... 61
5.1.1 poly-add has desired root ..... 62
5.1.2 poly-add is nonzero ..... 64
5.1.3 Summary for addition ..... 69
5.2 Division of Algebraic Numbers ..... 72
5.2.1 Summary for division ..... 76
5.3 Multiplication of Algebraic Numbers ..... 77
5.4 Summary: Closure Properties of Algebraic Numbers ..... 77
5.5 More on algebraic integers ..... 78
6 Separation of Roots: Sturm ..... 89
6.1 Interface for Separating Roots ..... 90
6.2 Implementing Sturm on Rational Polynomials ..... 93
7 Getting Small Representative Polynomials via Factorization ..... 96
8 The minimal polynomial of an algebraic number ..... 103
9 Algebraic Numbers - Preliminary Implementation ..... 106
10 Cauchy's Root Bound ..... 114
11 Real Algebraic Numbers ..... 118
11.1 Real Algebraic Numbers - Innermost Layer ..... 120
11.1.1 Basic Definitions ..... 120
11.2 Real Algebraic Numbers $=$ Rational + Irrational Real Alge- braic Numbers ..... 124
11.2.1 Definitions and Algorithms on Raw Type ..... 124
11.2.2 Definitions and Algorithms on Quotient Type ..... 125
11.2.3 Sign ..... 125
11.2.4 Normalization: Bounds Close Together ..... 126
11.2.5 Comparisons ..... 137
11.2.6 Negation ..... 138
11.2.7 Inverse ..... 140
11.2.8 Floor ..... 141
11.2.9 Generic Factorization and Bisection Framework ..... 144
11.2.10 Addition ..... 153
11.2.11 Multiplication ..... 159
11.2.12 Root ..... 165
11.2.13 Embedding of Rational Numbers ..... 173
11.2.14 Definitions and Algorithms on Type with Invariant ..... 181

### 11.3 Real Algebraic Numbers as Implementation for Real Numbers 191

12 Real Roots ..... 192
13 Complex Roots of Real Valued Polynomials ..... 201
13.1 Compare Instance for Complex Numbers ..... 211
14 Interval Arithmetic ..... 212
14.1 Syntactic Class Instantiations ..... 213
14.2 Class Instantiations ..... 214
14.3 Membership ..... 215
14.4 Convergence ..... 217
14.5 Complex Intervals ..... 219
15 Complex Algebraic Numbers ..... 223
15.1 Complex Roots ..... 224
16 Show for Real Algebraic Numbers - Interface ..... 245
17 Show for Real (Algebraic) Numbers - Approximate Repre- sentation ..... 246
18 Show for Real (Algebraic) Numbers - Unique Representa- tion ..... 247
19 Algebraic Number Tests ..... 250
19.1 Stand-Alone Examples ..... 250
19.2 Example Application: Compute Norms of Eigenvalues ..... 251
20 Explicit Constants for External Code ..... 251
20.1 Operations on Real Algebraic Numbers ..... 251
20.2 Operations on Complex Algebraic Numbers ..... 252
20.3 Export Constants in Haskell ..... 253

## 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 $\left.-\frac{1}{8}+\frac{1}{4} x+x^{3}\right)+\left(\right.$ root $\# 1$ of $\left.-\frac{31}{64}+\frac{9}{16} x^{2}-\frac{3}{2} x^{4}+x^{6}\right) \mathrm{i} "$, and "(root \#1 of $\left.-\frac{1}{8}+\frac{1}{4} x+x^{3}\right)+\left(\right.$ root $\# 2$ of $\left.-\frac{31}{64}+\frac{9}{16} x^{2}-\frac{3}{2} x^{4}+x^{6}\right) \mathrm{i}$ ".

## 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<br>HOL-Computational-Algebra.Fundamental-Theorem-Algebra<br>Polynomial-Interpolation.Newton-Interpolation<br>Polynomial-Factorization.Gauss-Lemma<br>Berlekamp-Zassenhaus.Unique-Factorization-Poly<br>Polynomial-Factorization.Square-Free-Factorization

## begin

lemma primitive-imp-unit-iff:
fixes $p::$ ' $a$ :: \{comm-semiring-1,semiring-no-zero-divisors $\}$ poly
assumes $p r$ : primitive $p$
shows $p$ dvd $1 \longleftrightarrow$ degree $p=0$
proof
assume degree $p=0$
from degree 0 -coeffs $[O F$ this $]$ obtain $p 0$ where $p: p=[: p 0:]$ by auto
then have $\forall c \in$ set (coeffs p). p0 dvd $c$ by (simp add: cCons-def)
with $p r$ have $p 0$ dvd 1 by (auto dest: primitiveD)
with $p$ show $p d v d 1$ by auto
next
assume $p d v d 1$
then show degree $p=0$ by (auto simp: poly-dvd-1)
qed
lemma dvd-all-coeffs-imp-dvd:
assumes $\forall a \in \operatorname{set}(c o e f f s p) . c$ dvd $a$ shows [:c:] dvd $p$
proof (insert assms, induct p)
case 0
then show? case by simp
next
case ( $p$ Cons a $p$ )
have $p$ Cons a $p=[: a:]+p$ Cons $0 p$ by simp
also have [:c:] dvd ...
proof (rule dvd-add)
from $p$ Cons show [:c:] dvd [:a:] by (auto simp: cCons-def)
from $p$ Cons have $[: c:] d v d p$ by auto
from Rings.dvd-mult[OF this]
show [:c:] dvd pCons $0 p$ by (subst pCons-0-as-mult)
qed
finally show ?case.
qed
lemma irreducible-content:
fixes $p::{ }^{\prime} a::\{$ comm-semiring-1,semiring-no-zero-divisors $\}$ poly
assumes irreducible $p$ shows degree $p=0 \vee$ primitive $p$
proof (rule ccontr)
assume not: $\neg$ ?thesis
then obtain $c$ where $c 1: \neg c$ dvd 1 and $\forall a \in \operatorname{set}$ (coeffs $p$ ). $c$ dvd $a$ by (auto elim: not-primitiveE)
from dvd-all-coeffs-imp-dvd[OF this(2)]
obtain $r$ where $p: p=r *[: c:]$ by (elim dvdE, auto)
from irreducible $D[O F$ assms this] have $r$ dvd $1 \vee[: c:] d v d 1$ by auto
with $c 1$ have $r d v d 1$ unfolding const-poly-dvd-1 by auto
then have degree $r=0$ unfolding poly-dvd-1 by auto
with $p$ have degree $p=0$ by auto
with not show False by auto

## qed

```
lemma linear-irreducible-field:
    fixes p :: ' }a\mathrm{ :: field poly
    assumes deg: degree p=1 shows irreducible p
proof (intro irreducibleI)
    from deg show p0: p\not=0 by auto
    from deg show }\negp\mathrm{ dvd 1 by (auto simp: poly-dvd-1)
    fix ab assume p: p=a*b
    with p0 have a0:a\not=0 and b0: b\not=0 by auto
    from degree-mult-eq[OF this, folded p] assms
    consider degree a=1 degree b=0 degree a=0 degree b=1 by force
    then show a dvd 1\veeb dvd 1
        by (cases; insert a0 b0, auto simp: primitive-imp-unit-iff)
qed
```

lemma linear-irreducible-int:
fixes $p$ :: int poly
assumes deg: degree $p=1$ and $c p$ : content $p$ dvd 1
shows irreducible $p$
proof (intro irreducibleI)
from deg show $p 0: p \neq 0$ by auto
from $\operatorname{deg}$ show $\neg p d v d 1$ by (auto simp: poly-dvd-1)
fix $a b$ assume $p: p=a * b$
note $*=c p[$ unfolded $p$ is-unit-content-iff, unfolded content-mult $]$
have a1: content a dvd 1 and b1: content $b$ dvd 1
using content-ge-0-int[of a] pos-zmult-eq-1-iff-lemma[OF *] * by (auto simp:
abs-mult)
with $p 0$ have $a 0: a \neq 0$ and $b 0: b \neq 0$ by auto
from degree-mult-eq[OF this, folded p] assms
consider degree $a=1$ degree $b=0 \mid$ degree $a=0$ degree $b=1$ by force
then show $a d v d 1 \vee b d v d 1$
by (cases; insert a1 b1, auto simp: primitive-imp-unit-iff)
qed
lemma irreducible-connect-rev:
fixes $p::$ ' $a::\{$ comm-semiring-1,semiring-no-zero-divisors $\}$ poly
assumes irr: irreducible $p$ and deg: degree $p>0$
shows irreducible $_{d} p$
proof (intro irreducible $_{d} I$ deg)
fix $q r$
assume degq: degree $q>0$ and diff: degree $q<$ degree $p$ and $p: p=q * r$
from $\operatorname{degq}$ have $n u$ : $\neg q d v d 1$ by (auto simp: poly-dvd-1)
from irreducible $D[O F$ irr $p] n u$ have $r d v d 1$ by auto
then have degree $r=0$ by (auto simp: poly-dvd-1)
with degq diff show False unfolding $p$ using degree-mult-le $[o f q r]$ by auto
qed

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

abbreviation ipoly where ipoly $f x \equiv$ poly (of-int-poly f) $x$
lemma poly-map-poly-code[code-unfold]: poly (map-poly $h$ p) $x=$ fold-coeffs $(\lambda a$
b. $h a+x * b) p 0$
by (induct $p$, auto)
abbreviation real-of-int-poly :: int poly $\Rightarrow$ real poly where real-of-int-poly $\equiv$ of-int-poly
abbreviation real-of-rat-poly :: rat poly $\Rightarrow$ real poly where real-of-rat-poly $\equiv$ map-poly of-rat
lemma of-rat-of-int[simp]: of-rat $\circ$ of-int $=o f$-int by auto
lemma ipoly-of-rat[simp]: ipoly $p(o f-r a t y)=o f-r a t(i p o l y p y)$
proof -
have $i d$ : of-int $=o f$-rat o of-int unfolding comp-def by auto
show ?thesis by (subst id, subst map-poly-map-poly[symmetric], auto)
qed
lemma ipoly-of-real[simp]:
ipoly $p$ (of-real $x::$ ' $a$ :: \{field,real-algebra-1 $\}$ ) $=$ of-real (ipoly $p x)$
proof -
have id: of-int $=$ of-real o of-int unfolding comp-def by auto
show ?thesis by (subst id, subst map-poly-map-poly[symmetric], auto)
qed
lemma finite-ipoly-roots: assumes $p \neq 0$
shows finite $\{x$ :: real. ipoly $p x=0\}$
proof -
let ?p $=$ real-of-int-poly $p$
from assms have ? $p \neq 0$ by auto
thus ?thesis by (rule poly-roots-finite)
qed

### 3.2 Algebraic Numbers - Definition, Inverse, and Roots

A number $x$ is algebraic iff it is the root of an integer polynomial. Whereas the Isabelle distribution this is defined via the embedding of integers in 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 $x \longleftrightarrow(\exists p$. ipoly $p x=0 \wedge p \neq 0)$
unfolding algebraic-def
proof (safe, goal-cases)
case (1 p)

```
    define the-int where the-int \(=\left(\lambda x::^{\prime} a\right.\). THE \(\left.r . x=o f-i n t r\right)\)
    define \(p^{\prime}\) where \(p^{\prime}=\) map-poly the-int \(p\)
    have of-int-the-int: of-int (the-int \(x\) ) \(=x\) if \(x \in \mathbb{Z}\) for \(x\)
    unfolding the-int-def by (rule sym, rule the \(I^{\prime}\) ) (insert that, auto simp: Ints-def)
    have the-int- 0 -iff: the-int \(x=0 \longleftrightarrow x=0\) if \(x \in \mathbb{Z}\)
    using of-int-the-int [OF that] by auto
    have map-poly of-int \(p^{\prime}=\) map-poly (of-int \(\circ\) the-int) \(p\)
        by (simp add: \(p^{\prime}\)-def map-poly-map-poly)
    also from 1 of-int-the-int have \(\ldots=p\)
    by (subst poly-eq-iff) (auto simp: coeff-map-poly)
    finally have \(p\) - \(p^{\prime}\) : map-poly of-int \(p^{\prime}=p\).
    show ?case
    proof (intro exI conjI notI)
    from 1 show ipoly \(p^{\prime} x=0\) by (simp add: \(p-p^{\prime}\) )
    next
    assume \(p^{\prime}=0\)
    hence \(p=0\) by (simp add: \(p-p^{\prime}[\) symmetric \(\left.]\right)\)
    with \(\langle p \neq 0\rangle\) show False by contradiction
    qed
next
    case (2 p)
    thus ?case by (intro exI[of-map-poly of-int p], auto)
qed
Definition of being algebraic with explicit witness polynomial.
definition represents :: int poly \(\Rightarrow\) ' \(a::\) field-char-0 \(\Rightarrow\) bool (infix represents 51) where \(p\) represents \(x=(\) ipoly \(p x=0 \wedge p \neq 0)\)
lemma represents \([\) intro]: ipoly \(p x=0 \Longrightarrow p \neq 0 \Longrightarrow p\) represents \(x\) unfolding represents-def by auto
lemma representsD:
assumes \(p\) represents \(x\) shows \(p \neq 0\) and ipoly \(p x=0\) using assms unfolding represents-def by auto
lemma representsE:
assumes \(p\) represents \(x\) and \(p \neq 0 \Longrightarrow\) ipoly \(p x=0 \Longrightarrow\) thesis
shows thesis using assms unfolding represents-def by auto
lemma represents-imp-degree:
fixes \(x\) :: ' \(a\) :: field-char-0
assumes \(p\) represents \(x\) shows degree \(p \neq 0\)
proof-
from assms have \(p \neq 0\) and \(p x\) : ipoly \(p x=0\) by (auto dest:representsD)
then have (of-int-poly \(p::{ }^{\prime} a\) poly) \(\neq 0\) by auto
then have degree (of-int-poly \(p::\) 'a poly) \(\neq 0\) by (fold poly-zero \([O F p x]\) )
then show ?thesis by auto
qed
```

```
lemma representsE-full[elim]:
    assumes rep: p represents x
        and main: p\not=0\Longrightarrow ipoly p x=0\Longrightarrow degree p}\not=0\Longrightarrow\mathrm{ thesis
    shows thesis
    by (rule main, insert represents-imp-degree[OF rep] rep, auto elim: representsE)
lemma represents-of-rat[simp]: p represents (of-rat x) = p represents x by (auto
elim!:representsE)
lemma represents-of-real[simp]: p represents (of-real x) = p represents x by (auto
elim!:representsE)
lemma algebraic-iff-represents: algebraic }x\longleftrightarrow(\exists\mathrm{ p.p represents x)
    unfolding algebraic-altdef-ipoly represents-def ..
lemma represents-irr-non-0:
    assumes irr: irreducible p and ap: p represents x and x0:x\not=0
    shows poly p 0}=
proof
    have nu:\neg [:0,1::int:] dvd 1 by (auto simp: poly-dvd-1)
    assume poly p 0=0
    hence dvd:[:0, 1 :] dvd p by (unfold dvd-iff-poly-eq-0, simp)
    then obtain q where pq: p=[:0,1:]*q by (elim dvdE)
    from irreducibleD[OF irr this] nu have q dvd 1 by auto
    from this obtain r where q=[:r:] r dvd 1 by (auto simp add: poly-dvd-1 dest:
degree0-coeffs)
    with pq have p=[:0,r:] by auto
    with ap have }x=0\mathrm{ by (auto simp: of-int-hom.map-poly-pCons-hom)
    with x0 show False by auto
qed
```

The polynomial encoding a rational number.
definition poly-rat :: rat $\Rightarrow$ int poly where
poly-rat $x=($ case quotient-of $x$ of $(n, d) \Rightarrow[:-n, d:])$
definition abs-int-poly:: int poly $\Rightarrow$ int poly where
abs-int-poly $p \equiv$ if lead-coeff $p<0$ then $-p$ else $p$
lemma pos-poly-abs-poly[simp]:
shows lead-coeff (abs-int-poly $p$ ) $>0 \longleftrightarrow p \neq 0$
proof-
have $p \neq 0 \longleftrightarrow$ lead-coeff $p * \operatorname{sgn}$ (lead-coeff $p$ ) > 0 by (fold abs-sgn, auto)
then show? thesis by (auto simp: abs-int-poly-def mult.commute)
qed
lemma abs-int-poly- 0 [simp]: abs-int-poly $0=0$
by (auto simp: abs-int-poly-def)
lemma abs-int-poly-eq- 0 -iff $[$ simp]: abs-int-poly $p=0 \longleftrightarrow p=0$
by (auto simp: abs-int-poly-def sgn-eq-0-iff)

```
lemma degree-abs-int-poly[simp]: degree (abs-int-poly p) = degree p
    by (auto simp: abs-int-poly-def sgn-eq-0-iff)
lemma abs-int-poly-dvd[simp]: abs-int-poly p dvd qup dvd q
    by (unfold abs-int-poly-def, auto)
lemma (in idom) irreducible-uminus[simp]: irreducible ( }-x)\longleftrightarrow\mathrm{ irreducible }
proof-
    have -x=-1*x by simp
    also have irreducible ... \longleftrightarrow irreducible x by (rule irreducible-mult-unit-left,
auto)
    finally show ?thesis.
qed
lemma irreducible-abs-int-poly[simp]:
    irreducible (abs-int-poly p) \longleftrightarrow irreducible p
    by (unfold abs-int-poly-def, auto)
lemma coeff-abs-int-poly[simp]:
    coeff (abs-int-poly p) n = (if lead-coeff p<0 then - coeff p n else coeff p n)
    by (simp add: abs-int-poly-def)
lemma lead-coeff-abs-int-poly[simp]:
    lead-coeff (abs-int-poly p) = abs (lead-coeff p)
    by auto
lemma ipoly-abs-int-poly-eq-zero-iff[simp]:
    ipoly (abs-int-poly p) (x :: 'a :: comm-ring-1)=0 \longleftrightarrow ipoly p x=0
    by (auto simp: abs-int-poly-def sgn-eq-0-iff of-int-poly-hom.hom-uminus)
lemma abs-int-poly-represents[simp]:
    abs-int-poly p represents }x\longleftrightarrowp\mathrm{ represents }x\mathrm{ by (auto elim!:representsE)
lemma content-pCons[simp]: content (pCons a p) = gcd a (content p)
    by (unfold content-def coeffs-pCons-eq-cCons cCons-def, auto)
lemma content-uminus[simp]:
    fixes p :: 'a :: ring-gcd poly shows content (-p)= content p
    by (induct p, auto)
lemma primitive-abs-int-poly[simp]:
    primitive (abs-int-poly p) \longleftrightarrow primitive p
    by (auto simp: abs-int-poly-def)
lemma abs-int-poly-inv[simp]: smult (sgn (lead-coeff p)) (abs-int-poly p) = p
```

by (cases lead-coeff $p>0$, auto simp: abs-int-poly-def)

```
definition cf-pos :: int poly \(\Rightarrow\) bool where
    cf-pos \(p=(\) content \(p=1 \wedge\) lead-coeff \(p>0)\)
definition \(c f\)-pos-poly :: int poly \(\Rightarrow\) int poly where
    cf-pos-poly \(f=(\) let
        \(c=\) content \(f\);
        \(d=(\operatorname{sgn}(\) lead-coeff \(f) * c)\)
        in sdiv-poly \(f d\) )
lemma sgn-is-unit[intro!]:
    fixes \(x\) :: ' \(a\) :: linordered-idom
    assumes \(x \neq 0\)
    shows sgn \(x\) dvd 1 using assms by(cases x 0::'a rule:linorder-cases, auto)
lemma \(c f\)-pos-poly- \(0[\) simp \(]:\) cf-pos-poly \(0=0\) by (unfold cf-pos-poly-def sdiv-poly-def,
auto)
lemma cf-pos-poly-eq- \(0[\) simp \(]\) : cf-pos-poly \(f=0 \longleftrightarrow f=0\)
proof (cases \(f=0\) )
    case True
    thus ?thesis unfolding cf-pos-poly-def Let-def by (simp add: sdiv-poly-def)
next
    case False
    then have lc 0 : lead-coeff \(f \neq 0\) by auto
    then have s0: sgn \((\) lead-coeff \(f) \neq 0(\) is \(? s \neq 0)\) and content \(f \neq 0\) (is \(? c \neq\)
0 ) by (auto simp: sgn-0-0)
    then have \(s c 0: ? s * ? c \neq 0\) by auto
    \{ fix \(i\)
        from content-dvd-coeff sgn-is-unit[OF lc0]
        have ?s * ?c dvd coeff \(f i\) by (auto simp: unit-dvd-iff)
    then have coefffidiv \((? s * ? c)=0 \longleftrightarrow\) coeff \(f i=0\) by (auto simp:dvd-div-eq-0-iff)
    \} note \(*=\) this
    show ?thesis unfolding cf-pos-poly-def Let-def sdiv-poly-def poly-eq-iff by (auto
simp: coeff-map-poly *)
qed
```


## lemma

```
shows \(c f\)-pos-poly-main: smult \((\operatorname{sgn}(l e a d-c o e f f f) *\) content \(f)(c f-p o s-p o l y f)=\) \(f\) (is ? \(g 1\) )
and content-cf-pos-poly[simp]: content (cf-pos-poly \(f)=(\) if \(f=0\) then 0 else 1) (is ? \(g 2\) )
and lead-coeff-cf-pos-poly[simp]: lead-coeff \((c f-p o s-p o l y f)>0 \longleftrightarrow f \neq 0\) (is ?g3)
and cf-pos-poly-dvd[simp]:cf-pos-poly \(f\) dvd \(f\) (is ?g4)
proof \((\) atomize \((f u l l),(\) cases \(f=0\); intro conjI))
```

```
    case True
    then show ?g1 ?g2 ?g3 ?g4 by simp-all
next
    case f0: False
    let ?s \(=\operatorname{sgn}(\) lead-coeff \(f)\)
    have \(s: ? s \in\{-1,1\}\) using \(f 0\) unfolding sgn-if by auto
    define \(g\) where \(g \equiv\) smult ?s \(f\)
    define \(d\) where \(d \equiv\) ? \(s *\) content \(f\)
    have content \(g=\) content \(([: ? s:] * f)\) unfolding \(g\)-def by simp
    also have \(\ldots=\) content \([: ? s:]\) * content \(f\) unfolding content-mult by simp
    also have content \([: ? s:]=1\) using \(s\) by (auto simp: content-def)
    finally have \(c g\) : content \(g=\) content \(f\) by simp
    from \(f 0\)
    have \(d\) : cf-pos-poly \(f=\) sdiv-poly \(f d\) by (auto simp: cf-pos-poly-def Let-def d-def)
    let \(? g=\) primitive-part \(g\)
    define \(n g\) where \(n g=\) primitive-part \(g\)
    note \(d\)
    also have sdiv-poly \(f d=\) sdiv-poly \(g\) (content \(g\) ) unfolding \(c g\) unfolding \(g\)-def
\(d\)-def
    by (rule poly-eqI, unfold coeff-sdiv-poly coeff-smult, insert s, auto simp: div-minus-right)
    finally have \(f g\) : cf-pos-poly \(f=\) primitive-part \(g\) unfolding primitive-part-alt-def
    have lead-coeff \(f \neq 0\) using \(f 0\) by auto
    hence \(l g\) : lead-coeff \(g>0\) unfolding \(g\)-def lead-coeff-smult
    by (meson linorder-neqE-linordered-idom sgn-greater sgn-less zero-less-mult-iff)
    hence \(g 0: g \neq 0\) by auto
    from \(f 0\) content-primitive-part[OF this]
    show ? \(g 2\) unfolding \(f g\) by auto
    from \(g 0\) have content \(g \neq 0\) by simp
    with arg-cong[OF content-times-primitive-part[of g], of lead-coeff, unfolded lead-coeff-smult]
    \(l g\) content-ge-0-int[of g] have \(l g^{\prime}\) : lead-coeff \(n g>0\) unfolding ng-def
    by (metis dual-order.antisym dual-order.strict-implies-order zero-less-mult-iff)
    with \(f 0\) show ? \(g 3\) unfolding \(f g n g\)-def by auto
    have \(d 0: d \neq 0\) using \(s f 0\) by (force simp add: \(d\)-def)
    have smult \(d\) (cf-pos-poly \(f)=\) smult ?s (smult (content \(f\) ) (sdiv-poly (smult ?s
f) (content f)) )
    unfolding fg primitive-part-alt-def \(c g\) by (simp add: \(g\)-def \(d\)-def)
    also have sdiv-poly (smult ?s \(f)(\) content \(f)=\) smult ?s (sdiv-poly \(f\) (content \(f)\) )
    using \(s\) by (metis cg g-def primitive-part-alt-def primitive-part-smult-int sgn-sgn)
    finally have smult \(d(c f\)-pos-poly \(f)=\) smult \((\) content \(f)(\) primitive-part \(f)\)
    unfolding primitive-part-alt-def using \(s\) by auto
    also have \(\ldots=f\) by (rule content-times-primitive-part)
    finally have \(d f\) : smult \(d(c f\)-pos-poly \(f)=f\).
    with \(d 0\) show ? g1 by (auto simp: d-def)
    from \(d f\) have \(*: f=c f\)-pos-poly \(f *[: d:]\) by simp
    from \(d v d I[O F\) this] show? ? 4 .
qed
```

```
lemma irreducible-connect-int:
    fixes p :: int poly
    assumes ir: irreducible e}p\mathrm{ p and c: content p=1
    shows irreducible p
    using c primitive-iff-content-eq-1 ir irreducible-primitive-connect by blast
lemma
    fixes x :: 'a :: {idom,ring-char-0}
    shows ipoly-cf-pos-poly-eq-0[simp]: ipoly (cf-pos-poly p) x=0 \longleftrightarrow 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}\not=0\Longrightarrowcf-pos (cf-pos-poly p
proof-
    show degree (cf-pos-poly p) = degree p
    by (subst(3) cf-pos-poly-main[symmetric], auto simp:sgn-eq-0-iff)
    {
    assume p: p\not=0
    show cf-pos (cf-pos-poly p) using cf-pos-poly-main p by (auto simp:cf-pos-def)
    have (ipoly (cf-pos-poly p)x=0)=(ipoly p x=0)
                apply (subst(3) cf-pos-poly-main[symmetric]) by (auto simp: sgn-eq-0-iff
hom-distribs)
    }
    then show (ipoly (cf-pos-poly p) x=0) = (ipoly p x = 0) by (cases p = 0,
auto)
qed
```

lemma cf-pos-poly-eq-1:cf-pos-poly $f=1 \longleftrightarrow$ degree $f=0 \wedge f \neq 0$ (is ?l $\longleftrightarrow$
? $r$ )
proof(intro iffI conjI)
assume ? $r$
then have $d f 0$ : degree $f=0$ and $f 0: f \neq 0$ by auto
from degree 0 -coeffs $[O F d f 0]$ obtain $f 0$ where $f: f=[: f 0:]$ by auto
show $c f$-pos-poly $f=1$ using $f 0$ unfolding $f c f$-pos-poly-def Let-def sdiv-poly-def
by (auto simp: content-def mult-sgn-abs)
next
assume $l:$ ?l
then have degree (cf-pos-poly $f$ ) $=0$ by auto
then show degree $f=0$ by simp
from $l$ have $c f$-pos-poly $f \neq 0$ by auto
then show $f \neq 0$ by simp
qed
lemma irr-cf-poly-rat[simp]: irreducible (poly-rat $x$ )
lead-coeff $($ poly-rat $x)>0$ primitive (poly-rat $x$ )
proof -
obtain $n d$ where $x$ : quotient-of $x=(n, d)$ by force

```
    hence id: poly-rat x = [:-n,d:] by (auto simp: poly-rat-def)
    from quotient-of-denom-pos[OF x] have d:d>0 by auto
    show lead-coeff (poly-rat x) > 0 primitive (poly-rat x)
        unfolding id cf-pos-def using d quotient-of-coprime[OF x] by (auto simp:
content-def)
    from this[unfolded cf-pos-def]
    show irr: irreducible (poly-rat x) unfolding id using d by (auto intro!: lin-
ear-irreducible-int)
qed
lemma poly-rat[simp]: ipoly (poly-rat x) (of-rat x :: 'a :: field-char-0) = 0 ipoly
(poly-rat x) x = 0
    poly-rat x\not=0 ipoly (poly-rat x) y=0 \longleftrightarrowy=(of-rat x :: 'a)
proof -
    from irr-cf-poly-rat(1)[of x] show poly-rat x}\not=
        unfolding Factorial-Ring.irreducible-def by auto
    obtain nd where x: quotient-of }x=(n,d)\mathrm{ by force
    hence id: poly-rat x = [:-n,d:] by (auto simp: poly-rat-def)
    from quotient-of-denom-pos[OF x] have d:d}\not=0\mathrm{ by auto
    have }y*\mathrm{ of-int d =of-int n #y=of-int n / of-int d using d
            by (simp add: eq-divide-imp)
    with d id show ipoly (poly-rat x) (of-rat x) = 0 ipoly (poly-rat x) x = 0
            ipoly (poly-rat x) y=0 \longleftrightarrowy=(of-rat x ::'a)
    by (auto simp: of-rat-minus of-rat-divide simp: quotient-of-div[OF x])
qed
lemma poly-rat-represents-of-rat: (poly-rat x) represents (of-rat x) by auto
lemma ipoly-smult-0-iff: assumes c:c\not=0
    shows (ipoly (smult c p) x = (0 :: real)) =(ipoly p x = 0)
    using c by (simp add: hom-distribs)
lemma not-irreducibleD:
    assumes \neg irreducible x and x}=0\mathrm{ and }\negx\mathrm{ dvd 1
    shows }\existsyz.x=y*z\wedge\negydvd 1\wedge\negzdvd 1 using assm
    apply (unfold Factorial-Ring.irreducible-def) by auto
lemma cf-pos-poly-represents[simp]: (cf-pos-poly p) represents }x\longleftrightarrowp\mathrm{ represents
x
    unfolding represents-def by auto
lemma coprime-prod:
    a\not=0\Longrightarrowc\not=0\Longrightarrow coprime (a*b) (c*d)\Longrightarrowcoprime b (d::'a::{semiring-gcd})
    by auto
lemma smult-prod:
```

```
        smult a b = monom a 0*b
        by (simp add: monom-0)
lemma degree-map-poly-2:
    assumes f (lead-coeff p)}\not=
    shows degree (map-poly f p) = degree p
proof (cases p=0)
    case False thus ?thesis
        unfolding degree-eq-length-coeffs Polynomial.coeffs-map-poly
        using assms by (simp add:coeffs-def)
qed auto
lemma irreducible-cf-pos-poly:
    assumes irr: irreducible p and deg: degree p}\not=
    shows irreducible (cf-pos-poly p) (is irreducible ?p)
proof (unfold irreducible-altdef, intro conjI allI impI)
    from irr show ?p}\not=0\mathrm{ by auto
    from deg have degree ?p }\not=0\mathrm{ by simp
    then show \neg?p dvd 1 unfolding poly-dvd-1 by auto
    fix b assume b dvd cf-pos-poly p
    also note cf-pos-poly-dvd
    finally have b dvd p.
    with irr[unfolded irreducible-altdef] have p dvd b\vee b dvd 1 by auto
    then show ?p dvd b\veeb dvd 1 by (auto dest:dvd-trans[OF cf-pos-poly-dvd])
qed
locale dvd-preserving-hom = comm-semiring-1-hom +
    assumes hom-eq-mult-hom-imp: hom x = hom y*hz\Longrightarrow\existsz. hz=hom z ^x
= y*z
begin
lemma hom-dvd-hom-iff[simp]: hom x dvd hom y 
proof
    assume hom x dvd hom y
    then obtain hz where hom y=hom x*hz by (elim dvdE)
    from hom-eq-mult-hom-imp[OF this] obtain z
    where hz=hom z and mult: }y=x*z\mathrm{ by auto
    then show }x\mathrm{ dvd y by auto
qed auto
sublocale unit-preserving-hom
proof unfold-locales
    fix x assume hom x dvd 1 then have hom x dvd hom 1 by simp
    then show x dvd 1 by (unfold hom-dvd-hom-iff)
qed
sublocale zero-hom-0
proof (unfold-locales)
    fix }a:::'
```

```
    assume hom a = 0
    then have hom 0 dvd hom a by auto
    then have 0 dvd a by (unfold hom-dvd-hom-iff)
    then show }a=0\mathrm{ by auto
qed
end
lemma smult-inverse-monom:p}\not=0\Longrightarrow\mathrm{ smult (inverse c) (p::rat poly)=1 }
p=[:c:]
    proof (cases c=0)
        case True thus p\not=0\Longrightarrow ?thesis by auto
    next
    case False thus ?thesis by (metis left-inverse right-inverse smult-1 smult-1-left
smult-smult)
    qed
lemma of-int-monom:of-int-poly p=[:rat-of-int c:]\longleftrightarrowp=[:c:] by (induct p,
auto)
lemma degree-0-content:
    fixes p :: int poly
    assumes deg: degree p = 0 shows content p =abs(coeff p 0)
proof-
    from deg obtain a where p:p=[:a:] by (auto dest: degree0-coeffs)
    show ?thesis by (auto simp: p)
qed
lemma prime-elem-imp-gcd-eq:
    fixes x::'a:: ring-gcd
    shows prime-elem x gcd x y = normalize }x\vee\operatorname{gcd}xy=
    using prime-elem-imp-coprime [of x y]
    by (auto simp add: gcd-proj1-iff intro: coprime-imp-gcd-eq-1)
lemma irreducible-pos-gcd:
    fixes p :: int poly
    assumes ir: irreducible p and pos: lead-coeff p>0 shows gcd p q\in{1,p}
proof-
    from pos have [:sgn (lead-coeff p):]=1 by auto
    with prime-elem-imp-gcd-eq[of p, unfolded prime-elem-iff-irreducible, OF ir, of
q]
    show ?thesis by (auto simp: normalize-poly-def)
qed
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 \(g c d p q=1 \vee p=q\)
```

```
proof (cases gcd p q=1)
    case False note pq= this
    have p=gcd pq using irreducible-pos-gcd [OF p,of q] pq
        by auto
    also have ... = q using irreducible-pos-gcd [OF q, of p] pq
        by (auto simp add: ac-simps)
    finally show ?thesis by auto
qed simp
interpretation of-rat-hom: field-hom-0' of-rat..
lemma poly-zero-imp-not-unit:
    assumes poly px=0 shows }\negpdvd 
proof (rule notI)
    assume pdvd 1
    from poly-hom.hom-dvd-1[OF this] have poly p x dvd 1 by auto
    with assms show False by auto
qed
lemma poly-prod-mset-zero-iff:
    fixes x :: 'a :: idom
    shows poly (prod-mset F) x=0 \longleftrightarrow(\existsf\in#F.poly f x=0)
    by (induct F, auto simp: poly-mult-zero-iff)
lemma algebraic-imp-represents-irreducible:
    fixes x :: ' }a\mathrm{ :: field-char-0
    assumes algebraic x
    shows \existsp.p represents x}\wedge\mathrm{ irreducible p
proof -
    from assms obtain p
    where px0: ipoly p x=0 and p0: p}\not=0\mathrm{ unfolding algebraic-altdef-ipoly by
auto
    from poly-zero-imp-not-unit[OF px0]
    have }\negp\mathrm{ dvd 1 by (auto dest: of-int-poly-hom.hom-dvd-1[where ' }a='='a]
    from mset-factors-exist[OF p0 this]
    obtain F where F: mset-factors F p by auto
    then have p=prod-mset F by auto
    also have (of-int-poly ... :: 'a poly) = prod-mset (image-mset of-int-poly F) by
simp
    finally have poly ... }x=0\mathrm{ using px0 by auto
    from this[unfolded poly-prod-mset-zero-iff]
    obtain f}\mathrm{ where f}\in#F\mathrm{ and fx0: ipoly f x=0 by auto
    with F have irreducible f by auto
    with fx0 show ?thesis by auto
qed
lemma algebraic-imp-represents-irreducible-cf-pos:
    assumes algebraic ( }x::'\a::{ield-char-0)
    shows \existsp.p represents }x\wedge\mathrm{ irreducible p}\wedge\mathrm{ lead-coeff p>0^ primitive p
```

```
proof -
    from algebraic-imp-represents-irreducible[OF assms(1)]
    obtain p where px: p represents x and irr: irreducible p by auto
    let ?p = cf-pos-poly p
    from px irr represents-imp-degree
    have 1:?p represents x and 2: irreducible?p and 3:cf-pos?p
        by (auto intro: irreducible-cf-pos-poly)
    then show ?thesis by (auto intro: exI[of - ?p] simp:cf-pos-def)
qed
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 -
    let ?ia = of-int-poly :: - = 'a poly
    let ?ir = of-int-poly :: - => rat poly
    let ?ra = map-poly of-rat :: - = 'a poly
    have id: ?ia x = ?ra (?ir x) for x by (subst map-poly-map-poly,auto)
    show ?thesis
        unfolding id
        unfolding of-rat-hom.map-poly-gcd[symmetric]
        unfolding gcd-rat-to-gcd-int by (auto simp: hom-distribs)
qed
lemma algebraic-imp-represents-unique:
    fixes }x\mathrm{ :: ' }a:::{\mathrm{ field-char-0,field-gcd}
    assumes algebraic x
    shows \exists! p.p represents x}\wedge\mathrm{ irreducible p \ lead-coeff p>0 (is Ex1 ?p)
proof -
    from assms obtain p
    where p:?p p and cfp:cf-pos p
        by (auto simp: cf-pos-def dest: algebraic-imp-represents-irreducible-cf-pos)
    show ?thesis
    proof (rule ex1I)
        show ?p p by fact
        fix q
        assume q: ?p q
        then have q represents x by auto
        from represents-imp-degree[OF this] q irreducible-content[of q]
        have cfq:cf-pos q by (auto simp:cf-pos-def)
        show }q=
        proof (rule ccontr)
            let ?ia = map-poly of-int :: int poly = 'a poly
            assume q}=
            with irreducible-pos-gcd-twice[of p q] p q cfp cfq have gcd: gcd p q=1 by
auto
            from pq have rt: ipoly p x = 0 ipoly qx=0 unfolding represents-def by
auto
            define c :: ' }a\mathrm{ where c= inverse (of-int (lead-coeff (gcd p q)))
```

```
        have rt: poly (?ia p) x=0 poly (?ia q) x=0 using rt by auto
        hence [:-x,1:] dvd ? ia p [:-x,1:] dvd ?ia q
            unfolding poly-eq-0-iff-dvd by auto
        hence [:-x,1:] dvd gcd (?ia p) (?ia q) by (rule gcd-greatest)
        also have ... = smult c (?ia (gcd p q)) unfolding gcd-of-int-poly c-def ..
        also have ? ia (gcd p q)=1 by (simp add: gcd)
        also have smult c 1 = [:c:] by simp
        finally show False using c-def gcd by (simp add: dvd-iff-poly-eq-0)
        qed
    qed
qed
lemma ipoly-poly-compose:
    fixes x :: 'a :: idom
    shows ipoly ( }p\mp@subsup{\circ}{p}{}q)x=\mathrm{ ipoly }p(\mathrm{ ipoly q x )
proof (induct p)
    case (pCons a p)
    have ipoly ((pCons a p) op q) x =of-int a + ipoly ( }q*p\mp@subsup{\circ}{p}{}q)x\mathrm{ by (simp add:
hom-distribs)
    also have ipoly ( }q*p\mp@subsup{\circ}{p}{}q)x=\mathrm{ ipoly q x * ipoly ( }p\mp@subsup{\circ}{p}{\prime}q)x\mathrm{ by (simp add:
hom-distribs)
    also have ipoly ( }p\mp@subsup{\circ}{p}{}q\mathrm{ ) x = ipoly p (ipoly q x) unfolding pCons(2) ..
    also have of-int a + ipoly qx*\ldots= ipoly (pCons a p) (ipoly q x)
        unfolding map-poly-pCons[OF pCons(1)] by simp
    finally show ?case .
qed simp
lemma algebraic-0[simp]: algebraic 0
    unfolding algebraic-altdef-ipoly
    by (intro exI[of - [:0,1:]], auto)
lemma algebraic-1[simp]: algebraic 1
    unfolding algebraic-altdef-ipoly
    by (intro exI[of - [:-1,1:]], auto)
        Polynomial for unary minus.
definition poly-uminus :: ' }a\mathrm{ :: ring-1 poly }=>\mp@subsup{|}{}{\prime}a\mathrm{ poly where [code del]:
    poly-uminus p \equiv\sumi\leqdegree p.monom ((-1)^i * coeff p i) i
lemma poly-uminus-pCons-pCons[simp]:
    poly-uminus (pCons a (pCons b p))=pCons a (pCons ( }-b\mathrm{ ) (poly-uminus p)) (is
?l = ?r)
proof(cases p=0)
    case False
    then have deg:degree (pCons a (pCons b p)) = Suc (Suc (degree p)) by simp
    show ?thesis
    by (unfold poly-uminus-def deg sum.atMost-Suc-shift monom-Suc monom-0 sum-pCons-0-commute,
simp)
next
```

```
    case True
    then show ?thesis by (auto simp add: poly-uminus-def monom-0 monom-Suc)
qed
fun poly-uminus-inner :: 'a :: ring-1 list }=>\mathrm{ ' '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 -
    have poly-uminus (Poly as) = poly-uminus-inner as for as :: 'a list
    proof (induct length as arbitrary:as rule: less-induct)
        case less
        show ?case
        proof(cases as)
            case Nil
            then show ?thesis by (simp add: poly-uminus-def)
        next
            case [simp]: (Cons a bs)
            show ?thesis
            proof (cases bs)
                    case Nil
                    then show ?thesis by (simp add: poly-uminus-def monom-0)
                next
                    case [simp]: (Cons b cs)
                    show ?thesis by (simp add: less)
                qed
        qed
    qed
    from this[of coeffs p]
    show ?thesis by simp
qed
lemma poly-uminus-inner-0[simp]: poly-uminus-inner as = 0 \longleftrightarrow Poly as=0
    by (induct as rule: poly-uminus-inner.induct, auto)
lemma degree-poly-uminus-inner[simp]: degree (poly-uminus-inner as) = degree
(Poly as)
    by (induct as rule: poly-uminus-inner.induct, auto)
lemma ipoly-uminus-inner[simp]:
    ipoly (poly-uminus-inner as) (x::'a::comm-ring-1) = ipoly (Poly as) ( }-x\mathrm{ ( 
    by (induct as rule: poly-uminus-inner.induct, auto simp: hom-distribs ring-distribs)
lemma represents-uminus: assumes alg: p represents x
    shows (poly-uminus p) represents (-x)
proof -
```

```
    from representsD[OF alg] have p\not=0 and rp: ipoly p x=0 by auto
    hence 0: poly-uminus p\not=0 by simp
    show ?thesis
    by (rule representsI[OF-0], insert rp, auto)
qed
```

lemma content-poly-uminus-inner [simp]:
fixes as :: ' $a$ :: ring-gcd list
shows content (poly-uminus-inner as) $=$ content $($ Poly as $)$
by (induct as rule: poly-uminus-inner.induct, auto)
Multiplicative inverse is represented by reflect-poly.
lemma inverse-pow-minus: assumes $x \neq(0::$ ' $a$ :: field $)$
and $i \leq n$
shows inverse $x^{\wedge} n * x^{\wedge} i=$ inverse $x^{\wedge}(n-i)$
using assms by (simp add: field-class.field-divide-inverse power-diff power-inverse)
lemma (in inj-idom-hom) reflect-poly-hom:
reflect-poly (map-poly hom $p$ ) $=$ map-poly hom (reflect-poly $p)$
proof -
obtain $x s$ where $x s$ : rev (coeffs $p$ ) $=x s$ by auto
show ?thesis unfolding reflect-poly-def coeffs-map-poly-hom rev-map
$x s$ by (induct xs, auto simp: hom-distribs)
qed
lemma ipoly-reflect-poly: assumes $x:(x::$ ' $a$ :: field-char-0) $\neq 0$
shows ipoly (reflect-poly $p$ ) $x=x^{\wedge}($ degree $p) *$ ipoly $p($ inverse $x)($ is $? l=? r)$
proof -
let ?or $=$ of-int $::$ int $\Rightarrow{ }^{\prime} a$
have hom: inj-idom-hom ?or ..
show ?thesis
using poly-reflect-poly-nz[OF x, of map-poly ?or p] by (simp add: inj-idom-hom.reflect-poly-hom[OF
hom])
qed
lemma represents-inverse: assumes $x: x \neq 0$
and alg: $p$ represents $x$
shows (reflect-poly $p$ ) represents (inverse $x$ )
proof (intro representsI)
from represents $D[$ OF alg $]$ have $p \neq 0$ and rp: ipoly $p x=0$ by auto
then show reflect-poly $p \neq 0$ by (metis reflect-poly-0 reflect-poly-at-0-eq-0-iff)
show ipoly (reflect-poly $p$ ) (inverse $x)=0$ by (subst ipoly-reflect-poly, insert $x$,
auto simp:rp)
qed
lemma inverse-roots: assumes $x:(x:: ' a$ :: field-char- 0$) \neq 0$
shows ipoly (reflect-poly $p$ ) $x=0 \longleftrightarrow$ ipoly $p($ inverse $x)=0$
using $x$ by (auto simp: ipoly-reflect-poly)

```
context
    fixes n :: nat
begin
    Polynomial for n-th root.
definition poly-nth-root :: 'a :: idom poly = 'a poly where
    poly-nth-root p = p op monom 1 n
lemma ipoly-nth-root:
    fixes }x :: 'a a:: idom
    shows ipoly (poly-nth-root p) x = ipoly p ( }\mp@subsup{x}{}{`}n
    unfolding poly-nth-root-def ipoly-poly-compose by (simp add: map-poly-monom
poly-monom)
context
    assumes n: n\not=0
begin
lemma poly-nth-root- 0[simp]: poly-nth-root p=0 < p=0
    unfolding poly-nth-root-def
    by (rule pcompose-eq-0, insert n, auto simp: degree-monom-eq)
lemma represents-nth-root:
    assumes y: y`n}=x\mathrm{ and alg: p represents }
    shows (poly-nth-root p) represents y
proof -
    from representsD[OF alg] have p\not=0 and rp: ipoly p x=0 by auto
    hence 0: poly-nth-root p\not=0 by simp
    show ?thesis
    by (rule representsI[OF-0], unfold ipoly-nth-root y rp, simp)
qed
lemma represents-nth-root-odd-real:
    assumes alg: p represents x and odd: odd n
    shows (poly-nth-root p) represents (root n x)
    by (rule represents-nth-root[OF odd-real-root-pow[OF odd] alg])
lemma represents-nth-root-pos-real:
    assumes alg: p represents x and pos: }x>
    shows (poly-nth-root p) represents (root n x)
proof -
    from n have id: Suc (n-1)=n by auto
    show ?thesis
    proof (rule represents-nth-root[OF - alg])
        show root n x^ n = x using id pos by auto
    qed
qed
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 -
    have rt: root n x = - root n ( }-x\mathrm{ ) unfolding real-root-minus by simp
    show ?thesis unfolding rt
    by (rule represents-uminus[OF represents-nth-root-pos-real[OF represents-uminus[OF
alg]]], insert neg, auto)
qed
end
end
lemma represents-csqrt:
    assumes alg: p represents x shows (poly-nth-root 2 p) represents (csqrt x)
    by (rule represents-nth-root[OF - alg], auto)
lemma represents-sqrt:
    assumes alg: p represents x and pos: x \geq0
    shows (poly-nth-root 2 p) represents (sqrt x)
    by (rule represents-nth-root[OF - alg], insert pos, auto)
lemma represents-degree:
    assumes p represents x shows degree p}\not=
proof
    assume degree p=0
    from degree0-coeffs[OF this] obtain c where p:p=[:c:] by auto
    from assms[unfolded represents-def p]
    show False by auto
qed
```

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 (\lambda (fi,i).fi*d^i* n^(k - Suc i)) (zip fs [0 ..<k])))
definition poly-mult-rat :: rat }=>\mathrm{ int poly }=>\mathrm{ int poly where
    poly-mult-rat r p \equivcase quotient-of r of ( }n,d)=>\mathrm{ poly-mult-rat-main n d p
lemma coeff-poly-mult-rat-main: coeff (poly-mult-rat-main n df) i=coeff fi*n
^(degree f-i)*d` i
proof -
    have id:coeff (poly-mult-rat-main n d f) i=(coeff fi* d^ i)* n^ (length
(coeffs f) - Suc i)
        unfolding poly-mult-rat-main-def Let-def poly-of-list-def coeff-Poly
        unfolding nth-default-coeffs-eq[symmetric]
        unfolding nth-default-def by auto
        show ?thesis unfolding id by (simp add: degree-eq-length-coeffs)
qed
lemma degree-poly-mult-rat-main: }n\not=0\Longrightarrow\mathrm{ degree (poly-mult-rat-main n d f)=
```

```
(if d=0 then 0 else degree f)
proof (cases d=0)
    case True
    thus ?thesis unfolding degree-def unfolding coeff-poly-mult-rat-main by simp
next
    case False
    hence id: (d=0) = False by simp
    show n\not=0\Longrightarrow?thesis unfolding degree-def coeff-poly-mult-rat-main id
    by (simp add: id)
qed
lemma ipoly-mult-rat-main:
    fixes }x:: ' a :: {field,ring-char-0}
    assumes d\not=0 and n\not=0
    shows ipoly (poly-mult-rat-main n d p) x = of-int n^ degree p * ipoly p (x*
of-int d / of-int n)
proof -
    from assms have d:(if d = 0 then t else f) = f for t f :: 'b by simp
    show ?thesis
    unfolding poly-altdef of-int-hom.coeff-map-poly-hom mult.assoc[symmetric] of-int-mult[symmetric]
        sum-distrib-left
    unfolding of-int-hom.degree-map-poly-hom degree-poly-mult-rat-main[OF assms(2)]
d
    proof (rule sum.cong[OF refl])
    fix }
    assume i\in{..degree p}
    hence i: i\leq degree p by auto
    hence id:of-int n^ (degree p-i)=(of-int n^degree p / of-int n^ i ::'a)
        by (simp add: assms(2) power-diff)
    thus of-int (coeff (poly-mult-rat-main n d p) i)*x^ i=of-int n^ degree p *
of-int (coeff pi)*(x* of-int d / of-int n) ^i
        unfolding coeff-poly-mult-rat-main
        by (simp add: field-simps)
    qed
qed
lemma degree-poly-mult-rat[simp]: assumes r\not=0 shows degree (poly-mult-rat r
p) = degree p
proof -
    obtain n d where quot: quotient-of r = ( n,d) by force
    from quotient-of-div[OF quot] have r: r =of-int n / of-int d by auto
    from quotient-of-denom-pos[OF quot] have d: d\not=0 by auto
    with assms r have n0: n\not=0 by simp
    from quot have id: poly-mult-rat r p = poly-mult-rat-main n d p unfolding
poly-mult-rat-def by simp
    show ?thesis unfolding id degree-poly-mult-rat-main[OF n0] using d by simp
qed
lemma ipoly-mult-rat:
```

```
    assumes r0: r\not=0
    shows ipoly (poly-mult-rat r p) x = of-int (fst (quotient-of r)) ^ degree p * ipoly
p(x* inverse (of-rat r))
proof -
    obtain nd where quot: quotient-of r = ( n,d) by force
    from quotient-of-div[OF quot] have r:r=of-int n / of-int d by auto
    from quotient-of-denom-pos[OF quot] have d: d\not=0 by auto
    from r r0 have n: n}=0\mathrm{ by simp
    from rd n have inv:of-int d / of-int n= inverse r by simp
    from quot have id: poly-mult-rat r p = poly-mult-rat-main n d p unfolding
poly-mult-rat-def by simp
    show ?thesis unfolding id ipoly-mult-rat-main[OF d n] quot fst-conv of-rat-inverse[symmetric]
inv[symmetric]
    by (simp add:of-rat-divide)
qed
lemma poly-mult-rat-main-0[simp]:
    assumes n\not=0d\not=0 shows poly-mult-rat-main nd p=0 < p=0
proof
    assume p=0 thus poly-mult-rat-main n d p=0
    by (simp add: poly-mult-rat-main-def)
next
    assume 0:poly-mult-rat-main n d p=0
    {
        fix }
        from 0 have coeff (poly-mult-rat-main n d p) i=0 by simp
        hence coeff p i=0 unfolding coeff-poly-mult-rat-main using assms by simp
        }
    thus p=0 by (intro poly-eqI, auto)
qed
```

lemma poly-mult-rat- $0[$ simp $]$ : assumes $r 0: r \neq 0$ shows poly-mult-rat $r p=0$
$\longleftrightarrow p=0$
proof -
obtain $n d$ where quot: quotient-of $r=(n, d)$ by force
from quotient-of-div[OF quot] have $r: r=o f-i n t n / o f-i n t d$ by auto
from quotient-of-denom-pos[OF quot] have $d: d \neq 0$ by auto
from r r0 have $n: n \neq 0$ by simp
from quot have id: poly-mult-rat $r$ p $=$ poly-mult-rat-main $n d p$ unfolding
poly-mult-rat-def by simp
show ?thesis unfolding id using $n d$ by simp
qed
lemma represents-mult-rat:
assumes $r: r \neq 0$ and $p$ represents $x$ shows (poly-mult-rat $r p$ ) represents (of-rat
$r * x)$
using assms
unfolding represents-def ipoly-mult-rat[OF r] by (simp add: field-simps)

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

```
definition poly-add-rat :: rat }=>\mathrm{ int poly }=>\mathrm{ int poly where
    poly-add-rat r p \equivcase quotient-of r of ( }n,d)
        (poly-mult-rat-main d 1 p op [:-n,d:])
lemma poly-add-rat-code[code]: poly-add-rat r p \equivcase quotient-of r of ( }n,d)
        let p}\mp@subsup{\mp@code{'}}{=}{=(let fs = coeffs p;k= length fs in poly-of-list (map (\lambda(fi,i).fi*d
(k - Suc i)) (zip fs [0..<k])));
            p}\mp@subsup{}{\prime\prime}{\prime\prime}=\mp@subsup{p}{}{\prime}\mp@subsup{o}{p}{}[:-n,d:
        in p"
    unfolding poly-add-rat-def poly-mult-rat-main-def Let-def by simp
lemma degree-poly-add-rat[simp]: degree (poly-add-rat r p) = degree p
proof -
    obtain nd where quot:quotient-of r = ( n,d) by force
    from quotient-of-div[OF quot] have r: r=of-int n / of-int d by auto
    from quotient-of-denom-pos[OF quot] have d: d\not=0 d>0 by auto
    show ?thesis unfolding poly-add-rat-def quot split
        by (simp add: degree-poly-mult-rat-main d)
qed
lemma ipoly-add-rat: ipoly (poly-add-rat r p)x = (of-int (snd (quotient-of r)) ^
degree p) * ipoly p(x-of-rat r)
proof -
    obtain nd where quot: quotient-of r=( n,d) by force
    from quotient-of-div[OF quot] have r: r=of-int n / of-int d by auto
    from quotient-of-denom-pos[OF quot] have d:d\not=0 d>0 by auto
    have id: ipoly [:- n, 1:] (x / of-int d :: 'a) = - of-int n + x / of-int d by simp
    show ?thesis unfolding poly-add-rat-def quot split
    by (simp add: ipoly-mult-rat-main ipoly-poly-compose d r degree-poly-mult-rat-main
field-simps id of-rat-divide)
qed
lemma poly-add-rat-O[simp]: poly-add-rat r p=0 < p=0
proof -
    obtain nd where quot: quotient-of r = ( n,d) by force
    from quotient-of-div[OF quot] have r: r =of-int n / of-int d by auto
    from quotient-of-denom-pos[OF quot] have d: d\not=0 d>0 by auto
    show ?thesis unfolding poly-add-rat-def quot split
    by (simp add: d pcompose-eq-0)
qed
lemma add-rat-roots: ipoly (poly-add-rat r p) x=0 \longleftrightarrow ipoly p (x-of-rat r)=
0
    unfolding ipoly-add-rat using quotient-of-nonzero by auto
lemma represents-add-rat:
    assumes p represents x shows (poly-add-rat r p) represents (of-rat r + x)
```

using assms unfolding represents-def ipoly-add-rat by simp
lemmas pos-mult $[$ simplified,simp $]=$ mult-less-cancel-left-pos $[$ of - 0] mult-less-cancel-left-pos $[$ of - - 0]
lemma ipoly-add-rat-pos-neg:
ipoly $($ poly-add-rat $r p)\left(x::{ }^{\prime} a:: l i n o r d e r e d-f i e l d\right)<0 \longleftrightarrow$ ipoly $p(x-$ of-rat $r)<$ 0 ipoly $($ poly-add-rat $r p)(x:: ' a:: l i n o r d e r e d-f i e l d)>0 \longleftrightarrow$ ipoly $p(x-$ of-rat $r)>$ 0 using quotient-of-nonzero unfolding ipoly-add-rat by auto
lemma sgn-ipoly-add-rat[simp]:
$\operatorname{sgn}\left(\right.$ ipoly $($ poly-add-rat r $\left.p)\left(x::^{\prime} a:: l i n o r d e r e d-f i e l d\right)\right)=\operatorname{sgn}($ ipoly $p(x-$ of-rat
$r))($ is $s g n ? l=s g n ? r)$
using ipoly-add-rat-pos-neg[of r p $x$ ]
by (cases ?r 0::'a rule: linorder-cases, auto simp: sgn-1-pos sgn-1-neg sgn-eq-0-iff)

## lemma deg-nonzero-represents:

assumes deg: degree $p \neq 0$ shows $\exists x$ :: complex. $p$ represents $x$
proof -
let $? p=o f-$ int-poly $p::$ complex poly
from fundamental-theorem-algebra-factorized $[$ of ?p]
obtain as $c$ where $i d:$ smult $c\left(\prod a \leftarrow a s\right.$. $\left.[:-a, 1:]\right)=? p$
and len: length as $=$ degree ? $p$ by blast
have degree ? $p=$ degree $p$ by simp
with deg len obtain $b$ bs where $a s: a s=b \# b s$ by (cases as, auto)
have $p$ represents $b$ unfolding represents-def id[symmetric] as using deg by auto
thus ?thesis by blast
qed
end

## 4 Resultants

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

### 4.1 Bivariate Polynomials

```
theory Bivariate-Polynomials
imports
    Polynomial-Interpolation.Ring-Hom-Poly
```


### 4.1.1 Evaluation of Bivariate Polynomials

definition poly2 :: ' $a$ ::comm-semiring-1 poly poly $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ where poly2 $p x y=$ poly $($ poly $p[: y:]) x$
lemma poly2-by-map: poly2 $p x=$ poly (map-poly $(\lambda c . p o l y c x) p)$ apply (rule ext) unfolding poly2-def by (induct $p$; simp)
lemma poly2-const[simp]: poly2 [:[:a:]:] $x y=a$ by (simp add: poly2-def)
lemma poly2-smult[simp,hom-distribs]: poly2 (smult a p) xy=poly a $x$ * poly2 $p$ $x y$ by (simp add: poly2-def)
interpretation poly2-hom: comm-semiring-hom $\lambda p$. poly2 $p x y$ by (unfold-locales; simp add: poly2-def)
interpretation poly2-hom: comm-ring-hom $\lambda$ p. poly2 p x $y$..
interpretation poly2-hom: idom-hom $\lambda$ p. poly2 p $x y$..
lemma poly2-pCons[simp,hom-distribs]: poly2 (pCons a p) xy=poly ax+y* poly2 $p x y$ by (simp add: poly2-def)
lemma poly2-monom: poly2 (monom a $n$ ) $x y=$ poly $a x * y へ n$ by (auto simp: poly-monom poly2-def)
lemma poly-poly-as-poly2: poly2 $p x($ poly $q x)=$ poly $($ poly $p q) x$ by (induct $p$; simp add:poly2-def)

The following lemma is an extension rule for bivariate polynomials.

```
lemma poly2-ext:
    fixes \(p q::{ }^{\prime} a\) :: \{ring-char-0, idom \(\}\) poly poly
    assumes \(\Lambda x y\). poly2 \(p x y=\) poly2 \(q x y\) shows \(p=q\)
proof (intro poly-ext)
    fix \(r x\)
    show poly (poly pr) \(x=\) poly \((\) poly \(q r) x\)
        unfolding poly-poly-as-poly2[symmetric] using assms by auto
qed
abbreviation (input) coeff-lift2 \(==\lambda a\). [:[: \(a::]:]\)
lemma coeff-lift2-lift: coeff-lift2 \(=\) coeff-lift \(\circ\) coeff-lift by auto
definition poly-lift \(=\) map-poly coeff-lift
definition poly-lift2 \(=\) map-poly coeff-lift2
lemma degree-poly-lift[simp]: degree (poly-lift \(p\) ) \(=\) degree \(p\)
    unfolding poly-lift-def by(rule degree-map-poly; auto)
```

```
lemma poly-lift- O[simp]: poly-lift 0 = 0 unfolding poly-lift-def by simp
lemma poly-lift-0-iff[simp]: poly-lift p=0 [ p cov
    unfolding poly-lift-def by(induct p;simp)
lemma poly-lift-pCons[simp]:
    poly-lift (pCons a p) = pCons [:a:] (poly-lift p)
    unfolding poly-lift-def map-poly-simps by simp
lemma coeff-poly-lift[simp]:
    fixes p:: 'a :: comm-monoid-add poly
    shows coeff (poly-lift p) i= coeff-lift (coeff p i)
    unfolding poly-lift-def by simp
lemma pcompose-conv-poly: pcompose p q = poly (poly-lift p) q
    by (induction p) auto
interpretation poly-lift-hom: inj-comm-monoid-add-hom poly-lift
proof-
    interpret map-poly-inj-comm-monoid-add-hom coeff-lift..
    show inj-comm-monoid-add-hom poly-lift by (unfold-locales, auto simp: poly-lift-def
hom-distribs)
qed
interpretation poly-lift-hom: inj-comm-semiring-hom poly-lift
proof
    interpret map-poly-inj-comm-semiring-hom coeff-lift..
    show inj-comm-semiring-hom poly-lift by (unfold-locales, auto simp add: poly-lift-def
hom-distribs)
qed
interpretation poly-lift-hom: inj-comm-ring-hom poly-lift..
interpretation poly-lift-hom: inj-idom-hom poly-lift..
lemma (in comm-monoid-add-hom) map-poly-hom-coeff-lift[simp, hom-distribs]:
    map-poly hom (coeff-lift a) = coeff-lift (hom a) by (cases a=0;simp)
lemma (in comm-ring-hom) map-poly-coeff-lift-hom:
    map-poly (coeff-lift \circ hom) p = map-poly (map-poly hom) (map-poly coeff-lift p)
proof (induct p)
    case (pCons a p) show ?case
        proof(cases a = 0)
            case True
                hence poly-lift p\not=0 using pCons(1) by simp
                thus ?thesis
                    unfolding map-poly-pCons[OF pCons(1)]
                    unfolding pCons(2) True by simp
            next case False
                hence coeff-lift a\not=0 by simp
                thus ?thesis
                unfolding map-poly-pCons[OF pCons(1)]
```

```
        unfolding pCons(2) by simp
    qed
qed auto
lemma poly-poly-lift[simp]:
    fixes p :: 'a :: comm-semiring-0 poly
    shows poly (poly-lift p) [:x:] = [: poly p x :]
proof (induct p)
    case 0 show ?case by simp
    next case (pCons a p) show ?case
        unfolding poly-lift-pCons
        unfolding poly-pCons
        unfolding pCons apply (subst mult.commute) by auto
qed
lemma degree-poly-lift2[simp]:
    degree (poly-lift2 p) = degree p unfolding poly-lift2-def by (induct p; auto)
lemma poly-lift2-0[simp]: poly-lift2 0 = 0 unfolding poly-lift2-def by simp
lemma poly-lift2-0-iff[simp]: poly-lift2 p=0 }\longleftrightarrowp=
    unfolding poly-lift2-def by(induct p;simp)
lemma poly-lift2-pCons[simp]:
    poly-lift2 (pCons a p) = pCons [:[:a:]:] (poly-lift2 p)
    unfolding poly-lift2-def map-poly-simps by simp
lemma poly-lift2-lift: poly-lift2 = poly-lift }\circ\mathrm{ poly-lift (is ?l = ?r)
proof
    fix p show ?l p=?r p
        unfolding poly-lift2-def coeff-lift2-lift poly-lift-def by (induct p; auto)
qed
lemma poly2-poly-lift[simp]: poly2 (poly-lift p) x y = poly p y by (induct p;simp)
lemma poly-lift2-nonzero:
    assumes p\not=0 shows poly-lift2 p\not=0
    unfolding poly-lift2-def
    apply (subst map-poly-zero)
    using assms by auto
```


### 4.1.2 Swapping the Order of Variables

## definition

poly-y-x $p \equiv \sum i \leq$ degree $p . \sum j \leq$ degree (coeff $p i$ ). monom (monom (coeff (coeff pi)j) i) $j$
lemma poly-y-x-fix-y-deg:
assumes ydeg: $\forall i \leq$ degree $p$. degree (coeff $p i) \leq d$

```
    shows poly-y-x p =(\sumi\leqdegree p. \sumj\leqd.monom (monom (coeff (coeff pi)j)
i) j)
    (is - = sum (\lambdai. sum (?f i) -) -)
    unfolding poly-y-x-def
    apply (rule sum.cong,simp)
    unfolding atMost-iff
proof -
    fix i assume i:i\leq degree p
    let ?d = degree (coeff p i)
    have {..d} = {..?d}\cup{Suc ?d .. d} using ydeg[rule-format, OF i] by auto
    also have sum (?f i) .. = sum (?f i) {..?d} + sum (?f i) {Suc ?d .. d}
        by(rule sum.union-disjoint,auto)
    also { fix j
        assume j: j\in{Suc ?d .. d}
        have coeff (coeff p i) j=0 apply (rule coeff-eq-0) using j by auto
        hence ?f i j = 0 by auto
    } hence sum (?f i) {Suc ?d .. d} = 0 by auto
    finally show sum (?f i) {..?d} = sum (?f i) {..d} by auto
qed
lemma poly-y-x-fixed-deg:
    fixes p :: ' }a\mathrm{ :: comm-monoid-add poly poly
    defines d \equiv Max { degree (coeff p i)| i. i\leq degree p }
    shows poly-y-x p = (\sumi\leqdegree p. \sumj\leqd.monom (monom (coeff (coeff pi)j)
i) j)
    apply (rule poly-y-x-fix-y-deg, intro allI impI)
    unfolding d-def
    by (subst Max-ge,auto)
lemma poly-y-x-swapped:
    fixes p :: 'a :: comm-monoid-add poly poly
    defines d \equiv Max { degree (coeff p i)| i. i\leq degree p }
    shows poly-y-x p =(\sumj\leqd. \sumi\leqdegree p.monom (monom (coeff (coeff p i)j)
i) j)
    using poly-y-x-fixed-deg[of p, folded d-def] sum.swap by auto
lemma poly2-poly-y-x[simp]: poly2 (poly-y-x p) x y = poly2 p y x
    using [[unfold-abs-def = false]]
    apply(subst(3) poly-as-sum-of-monoms[symmetric])
    apply(subst poly-as-sum-of-monoms[symmetric,of coeff p -])
    unfolding poly-y-x-def
    unfolding coeff-sum monom-sum
    unfolding poly2-hom.hom-sum
    apply(rule sum.cong, simp)
    apply(rule sum.cong, simp)
    unfolding poly2-monom poly-monom
    unfolding mult.assoc
    unfolding mult.commute..
```

```
context begin
private lemma poly-monom-mult:
    fixes p :: 'a :: comm-semiring-1
    shows poly (monom pi*q^j) y = poly (monom p j* [:y:]^ i) (poly q y)
    unfolding poly-hom.hom-mult
    unfolding poly-monom
    apply(subst mult.assoc)
    apply(subst(2) mult.commute)
    by (auto simp: mult.assoc)
lemma poly-poly-y-x:
    fixes p :: 'a :: comm-semiring-1 poly poly
    shows poly (poly (poly-y-x p) q) y = poly (poly p [:y:]) (poly q y)
    apply(subst(5) poly-as-sum-of-monoms[symmetric])
    apply(subst poly-as-sum-of-monoms[symmetric,of coeff p -])
    unfolding poly-y-x-def
    unfolding coeff-sum monom-sum
    unfolding poly-hom.hom-sum
    apply(rule sum.cong, simp)
    apply(rule sum.cong, simp)
    unfolding atMost-iff
    unfolding poly2-monom poly-monom
    apply(subst poly-monom-mult)..
```

end
interpretation poly-y-x-hom: zero-hom poly-y-x by (unfold-locales, auto simp:
poly-y-x-def)
interpretation poly-y-x-hom: one-hom poly-y-x by (unfold-locales, auto simp: poly-y-x-def
monom-0)
lemma map-poly-sum-commute:
assumes $h 0=0 \forall p q . h(p+q)=h p+h q$
shows sum ( $\lambda$ i. map-poly $h(f i)) S=$ map-poly $h(\operatorname{sum} f S)$
apply (induct $S$ rule: infinite-finite-induct)
using map-poly-add[OF assms] by auto
lemma poly-y-x-const: poly-y-x $[: p:]=$ poly-lift $p($ is $? l=? r)$
proof -
have $? l=\left(\sum j \leq\right.$ degree $p$. monom [:coeff $\left.\left.p j:\right] j\right)$
unfolding poly- $y$-x-def by (simp add: monom-0)
also have $\ldots=$ poly-lift $\left(\sum x \leq\right.$ degree $p$. monom (coeff $\left.p x\right) x$ )
unfolding poly-lift-hom.hom-sum unfolding poly-lift-def by simp
also have $\ldots=$ poly-lift $p$ unfolding poly-as-sum-of-monoms..
finally show ?thesis.
qed
lemma poly- $y-x-p$ Cons:

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

```
ij) for i
    proof(cases coeff pi=0)
    case True then show ?thesis by simp
    next
    case False
    show ?thesis
        apply (unfold degree-pCons-eq[OF False])
        apply (unfold sum.atMost-Suc-shift,simp)
        apply (fold monom-Suc)..
    qed
    show ?thesis
    apply (unfold poly-y-x-def)
    unfolding *..
qed
interpretation poly-y-x-hom: comm-monoid-add-hom poly-y-x :: ' a :: comm-monoid-add
poly poly }=>\mathrm{ -
proof (unfold-locales)
    fix p q :: 'a poly poly
    show poly-y-x ( p+q) = poly-y-x p + poly-y-x q
    proof (induct p arbitrary:q)
        case 0 show ?case by simp
    next
        case p:(pCons a p)
        show ?case
        proof (induct q)
            case q:(pCons b q)
            show ?case
            apply (unfold add-pCons)
            apply (unfold poly-y-x-pCons)
            apply (unfold p)
            by (simp add: poly-y-x-const ac-simps hom-distribs)
    qed auto
    qed
qed
            poly-y-x is bijective.
lemma poly-y-x-poly-lift:
    fixes p :: 'a :: comm-monoid-add poly
    shows poly-y-x (poly-lift p)=[:p:]
    apply(subst poly-y-x-fix-y-deg[of - 0],force)
    apply(subst(10) poly-as-sum-of-monoms[symmetric])
    by (auto simp add: monom-sum monom-0 hom-distribs)
lemma poly-y-x-id[simp]
    fixes p:: 'a :: comm-monoid-add poly poly
    shows poly-y-x (poly-y-x p)=p
proof (induct p)
```

    apply (unfold hom-distribs pCons-0-hom.degree-map-poly-hom pCons-0-hom.coeff-map-poly-hom)
    ```
    case 0
    then show ?case by simp
next
    case (pCons a p)
    interpret mm: map-poly-comm-monoid-add-hom \lambdax. monom x i for i..
    interpret mc: map-poly-comm-monoid-add-hom pCons 0 ..
    have pCons-as-add: pCons a p = [:a:] + pCons 0 p by simp
    from pCons show ?case
        apply (unfold pCons-as-add)
    by (simp add: poly-y-x-pCons poly-y-x-poly-lift poly-y-x-map-poly-pCons-0 hom-distribs)
qed
interpretation poly-y-x-hom:
    bijective poly-y-x :: 'a :: comm-monoid-add poly poly }=>\mathrm{ -
    by(unfold bijective-eq-bij, auto intro!:o-bij[of poly-y-x])
lemma inv-poly-y-x[simp]: Hilbert-Choice.inv poly-y-x = poly-y-x by auto
interpretation poly-y-x-hom: comm-monoid-add-isom poly-y-x
    by (unfold-locales, auto)
lemma pCons-as-add:
    fixes p :: 'a :: comm-semiring-1 poly
    shows pCons a p = [:a:]+ monom 11*p by (auto simp: monom-Suc)
lemma mult-pCons-0: (*)(pCons 0 1) = pCons 0 by auto
lemma pCons-0-as-mult:
    shows pCons (0 :: 'a :: comm-semiring-1) = (\lambdap.pCons 0 1 * p) by auto
lemma map-poly-pCons-0-as-mult:
    fixes p :: 'a :: comm-semiring-1 poly poly
    shows map-poly (pCons 0) p = [:pCons 0 1:] * p
    apply (subst(1) pCons-0-as-mult)
    apply (fold smult-as-map-poly) by simp
lemma poly-y-x-monom:
    fixes }a\mathrm{ :: ' }a\mathrm{ :: comm-semiring-1 poly
    shows poly-y-x (monom a n) = smult (monom 1 n) (poly-lift a)
proof (cases a=0)
    case True then show ?thesis by simp
next
    case False
    interpret map-poly-comm-monoid-add-hom \lambdax. c*x for c :: 'a poly..
    from False show ?thesis
        apply (unfold poly-y-x-def)
        apply (unfold degree-monom-eq)
        apply (subst(2) lessThan-Suc-atMost[symmetric])
        apply (unfold sum.lessThan-Suc)
```

```
    apply (subst sum.neutral,force)
    apply (subst(14) poly-as-sum-of-monoms[symmetric])
    apply (unfold smult-as-map-poly)
    by (auto simp: monom-altdef[unfolded x-as-monom x-pow-n,symmetric] hom-distribs)
qed
lemma poly-y-x-smult:
    fixes c :: 'a :: comm-semiring-1 poly
    shows poly-y-x (smult c p) = poly-lift c * poly-y-x p (is ?l = ?r)
proof-
    have smult c p = (\sumi\leqdegree p. monom (coeff (smult c p)i)i)
    by (metis (no-types, lifting) degree-smult-le poly-as-sum-of-monoms' sum.cong)
    also have ... =( \sumi\leqdegree p. monom (c* coeff pi) i)
        by auto
```



```
(coeff p i)))
    by (simp add: poly-y-x-monom hom-distribs)
    also have ... = poly-lift c* poly-y-x (\sumi\leqdegree p. monom (coeff pi) i)
    by (simp add: poly-y-x-monom hom-distribs)
    finally show ?thesis by (simp add: poly-as-sum-of-monoms)
qed
interpretation poly-y-x-hom:
    comm-semiring-isom poly-y-x :: ' a :: comm-semiring-1 poly poly }=>\mathrm{ -
proof
    fix p q :: 'a poly poly
    show poly-y-x (p*q) = poly-y-x p*poly-y-x q
    proof (induct p)
            case (pCons a p)
            show ?case
            apply (unfold mult-pCons-left)
            apply (unfold hom-distribs)
            apply (unfold poly-y-x-smult)
            apply (unfold poly-y-x-pCons-0)
            apply (unfold pCons)
            by (simp add: poly-y-x-pCons map-poly-pCons-0-as-mult field-simps)
    qed simp
qed
interpretation poly-y-x-hom: comm-ring-isom poly-y-x..
interpretation poly-y-x-hom: idom-isom poly-y-x..
lemma Max-degree-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 < degree p})
proof (cases p=0)
    case False show ?thesis
            unfolding degree-pCons-eq[OF False]
            unfolding image-Collect[symmetric]
```

```
    unfolding atMost-def[symmetric]
    apply(subst(1) atLeast0AtMost[symmetric])
    unfolding atLeastAtMost-insertL[OF le0,symmetric]
    unfolding image-insert
    apply(subst Max-insert,simp,simp)
    unfolding image-Suc-atLeastAtMost [symmetric]
    unfolding image-image
    unfolding atLeast0AtMost by simp
qed simp
lemma degree-poly-y-x:
    fixes p ::' 'a :: comm-ring-1 poly poly
    assumes p\not=0
    shows degree (poly-y-x p) = Max { degree (coeff pi)| i. i\leq degree p}
        (is - = ?d p)
    using assms
proof(induct p)
    interpret rhm: map-poly-comm-ring-hom coeff-lift ..
    let ?f = \lambdap i j. monom (monom (coeff (coeff p i)j) i) j
    case (pCons a p)
        show ?case
        proof(cases p=0)
            case True show ?thesis unfolding True unfolding poly-y-x-pCons by auto
            next case False
            note IH = pCons(2)[OF False]
            let ?a = poly-lift a
            let ?p = map-poly (pCons 0) (poly-y-x p)
            show ?thesis
                    proof(cases rule:linorder-cases[of degree ?a degree ?p])
                    case less
                            have dle: degree a\leqdegree (poly-y-x p)
                            apply(rule le-trans[OF less-imp-le[OF less[simplified]]])
                            using degree-map-poly-le by auto
                            show ?thesis
                                    unfolding poly-y-x-pCons
                                    unfolding degree-add-eq-right[OF less]
                                    unfolding Max-degree-coeff-pCons
                                    unfolding IH[symmetric]
                                    unfolding max-absorb2[OF dle]
                                    apply (rule degree-map-poly) by auto
                    next case equal
                                    have dega: degree ?a = degree a by auto
                                    have degp: degree (poly-y-x p)= degree a
                                    using equal[unfolded dega]
                                    using degree-map-poly[of pCons 0 poly-y-x p] by auto
                                    have *: degree (?a + ?p) = degree a
                                    proof(cases a=0)
                                    case True show ?thesis using equal unfolding True by auto
```

```
                    next case False show ?thesis
                    apply(rule antisym)
                    apply(rule degree-add-le, simp, fold equal, simp)
                    apply(rule le-degree)
                    unfolding coeff-add
                    using False
                    by auto
            qed
            show ?thesis unfolding poly-y-x-pCons
                unfolding *
                unfolding Max-degree-coeff-pCons
                unfolding IH[symmetric]
                    unfolding degp by auto
            next case greater
                have dge: degree a \geq degree (poly-y-x p)
                apply(rule le-trans[OF - less-imp-le[OF greater[simplified]]])
                by auto
                show ?thesis
                    unfolding poly-y-x-pCons
                unfolding degree-add-eq-left[OF greater]
            unfolding Max-degree-coeff-pCons
            unfolding IH[symmetric]
            unfolding max-absorb1[OF dge] by simp
        qed
    qed
qed auto
end
```


### 4.2 Resultant

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

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

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


### 4.2.1 Sylvester matrices and vector representation of polynomials

definition vec-of-poly-rev-shifted where
vec-of-poly-rev-shifted $p n j \equiv$
vec $n(\lambda i$. if $i \leq j \wedge j \leq$ 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 nj)=
n
    unfolding vec-of-poly-rev-shifted-def by auto
lemma col-sylvester:
    fixes pq
    defines m\equivdegree p and n\equivdegree q
    assumes j: j<m+n
    shows col (sylvester-mat p q) j=
        vec-of-poly-rev-shifted p nj @ @ vec-of-poly-rev-shifted qm j (is ?l =?r)
proof
    note [simp] = m-def[symmetric] n-def[symmetric]
    show dim-vec ?l = dim-vec ?r by simp
    fix i assume i<dim-vec ?r hence i:i<m+n by auto
    show ?l $ i=?r $ i
        unfolding vec-of-poly-rev-shifted-def
        apply (subst index-col) using i apply simp using j apply simp
        apply (subst sylvester-index-mat) using i apply simp using j apply simp
        apply (cases i<n) apply force using i by simp
qed
lemma inj-on-diff-nat2: inj-on (\lambdai. (n::nat) - i) {..n} by (rule inj-onI, auto)
lemma image-diff-atMost: (\lambdai.(n::nat) - i)'{..n}={..n} (is ?l=?r)
    unfolding set-eq-iff
proof (intro allI iffI)
    fix }x\mathrm{ assume }x:x\in
        thus x &?l unfolding image-def mem-Collect-eq
        by(intro bexI[of - n-x],auto)
qed auto
lemma sylvester-sum-mat-upper:
    fixes p q :: 'a a: comm-semiring-1 poly
    defines m\equivdegree p and n\equiv degree q
    assumes i:i<n
    shows (\sumj<m+n. monom (sylvester-mat p q $$ (i,j)) (m+n-Suc j))=
        monom 1 ( }n-\mathrm{ Suc i) * p (is sum ?f - = ?r)
proof -
    have n1: n\geq1 using i by auto
    define ni1 where ni1 = n-Suc i
    hence ni1: n-i=Suc ni1 using i by auto
    define l}\mathrm{ where l=m+n-1
    hence l: Suc l=m+n using n1 by auto
    let ?g = \lambdaj. monom (coeff (monom 1 ( }n-\mathrm{ Suc i) * p) j) j
    let ?p = \lambdaj. l-j
    have sum?f {..<m+n} = sum ?f {..l}
    unfolding l[symmetric] unfolding lessThan-Suc-atMost..
    also {
    fix j assume j: j\leql
```

```
    have ?f j = ((\lambdaj. monom (coeff (monom 1 (n-i)* p)(Suc j)) j)\circ ?p) j
        apply(subst sylvester-index-mat2)
        using ij unfolding l-def m-def[symmetric] n-def[symmetric]
        by (auto simp add: Suc-diff-Suc)
    also have ... = (?g\circ?p) j
        unfolding ni1
        unfolding coeff-monom-Suc
        unfolding ni1-def
        using i by auto
    finally have ?f j=(?g\circ?p) j.
}
hence ( }\sumj\leql\mathrm{ . ?f j) = ( }\sumj\leql. (?go?p) j) using l by aut
also have ... = (\sumj\leql. ?g j)
    unfolding l-def
    using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost].
    also have degree ?r }\leq
        using degree-mult-le[of monom 1 (n-Suc i) p]
        unfolding l-def m-def
        unfolding degree-monom-eq[OF one-neq-zero] using i by auto
    from poly-as-sum-of-monoms'[OF this]
    have ( }\sumj\leql. ?g j) = ?r
finally show?thesis.
qed
lemma sylvester-sum-mat-lower:
    fixes p q :: ' }a::\mathrm{ comm-semiring-1 poly
    defines m\equiv degree p}\mathrm{ and n}\equiv\mathrm{ degree }
    assumes ni:n\leqi and imn: i<m+n
    shows (\sumj<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 -
    define l where l=m+n-1
    hence l: Sucl=m+n using imn by auto
    define mni1 where mni1 =m+n-Suci
    hence mni1: m+n-i=Suc mni1 using imn by auto
    let ?g = \lambdaj. monom (coeff (monom 1 (m+n-Suci)*q)j)j
    let ? p = \lambdaj. l-j
    have sum ?f {..<m+n} = sum ?f {..l}
        unfolding l[symmetric] unfolding lessThan-Suc-atMost..
also {
    fix j assume j: j\leql
    have ?f j=((\lambdaj. monom (coeff (monom 1 (m+n-i)*q)(Suc j)) j)\circ?p) j
        apply(subst sylvester-index-mat2)
        using ni imn j unfolding l-def m-def[symmetric] n-def[symmetric]
        by (auto simp add: Suc-diff-Suc)
    also have ... = (?g\circ?p) j
        unfolding mni1
        unfolding coeff-monom-Suc
        unfolding mni1-def..
```

```
    finally have ?f j = ...
    }
    hence (\sumj\leql. ?f j) =( \sumj\leql. (?g\circ ?p) j) by auto
    also have ... = (\sumj\leql. ?g j)
    using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost].
    also have degree ?r }\leq
        using degree-mult-le[of monom 1 (m+n-1-i)q]
        unfolding l-def n-def[symmetric]
        unfolding degree-monom-eq[OF one-neq-zero] using ni imn by auto
    from poly-as-sum-of-monoms'[OF this]
    have ( }\sumj\leql. ?g j) = ?r
    finally show ?thesis.
qed
definition vec-of-poly p \equiv let m= degree p in vec (Suc m) (\lambdai. coeff p (m-i))
definition poly-of-vec v\equiv let d = dim-vec v in \sumi<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
    unfolding poly-of-vec-def vec-of-poly-def Let-def
    unfolding dim-vec
    unfolding lessThan-Suc-atMost
    using poly-as-sum-of-monoms[of p] by auto
lemma poly-of-vec-0[simp]: poly-of-vec ( }\mp@subsup{0}{v}{}n)=0\mathrm{ unfolding poly-of-vec-def Let-def
by auto
lemma poly-of-vec-0-iff[simp]:
    fixes v :: 'a :: comm-monoid-add vec
    shows poly-of-vec v=0\longleftrightarrowv= 0v (dim-vec v) (is ?v = - \longleftrightarrow-- ?z)
proof
    assume ?v = 0
    hence }\foralli\in{..<dim-vec v}.v $(dim-vec v-Suc i)=
        unfolding poly-of-vec-def Let-def
        by (subst sum-monom-0-iff[symmetric],auto)
    hence a:^i. i<dim-vec v\Longrightarrowv$(dim-vec v-Suc i)=0 by auto
    {fix i assume i<dim-vec v
        hence v$i=0 using a[of dim-vec v-Suci] by auto
    }
    thus v=?z by auto
    next assume r:v=?z
    show ?v = 0 apply (subst r) by auto
qed
lemma degree-sum-smaller:
```

assumes $n>0$ finite $A$
shows $(\bigwedge x . x \in A \Longrightarrow$ degree $(f x)<n) \Longrightarrow$ degree $\left(\sum x \in A . f x\right)<n$
using 〈finite $A$ 〉
by (induct rule: finite-induct)
(simp-all add: degree-add-less assms)

## 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 $v$
unfolding poly-of-vec-def Let-def
apply(rule degree-sum-smaller)
using dim apply force
apply force
unfolding lessThan-iff
by (metis degree-0 degree-monom-eq dim monom-eq-0-iff)
lemma coeff-poly-of-vec:
coeff (poly-of-vec $v$ ) $i=($ if $i<$ dim-vec $v$ then $v \$($ dim-vec $v-S u c i)$ else 0$)$ (is ?l $=$ ? $r$ )
proof -
have $? l=\left(\sum x<\operatorname{dim}-v e c v\right.$. if $x=i$ then $v \$($ dim-vec $v-$ Suc $x)$ else 0$)($ is = ? $m$ )
unfolding poly-of-vec-def Let-def coeff-sum coeff-monom ..
also have $\ldots=$ ? $r$
proof (cases $i<\operatorname{dim}$-vec $v$ )
case False
show ?thesis
by (subst sum.neutral, insert False, auto)
next
case True
show ?thesis
by (subst sum.remove $[$ of $-i]$, force, force simp: True, subst sum.neutral, insert
True, auto)
qed
finally show? ?thesis.
qed
lemma vec-of-poly-rev-shifted-scalar-prod:
fixes $p v$
defines $q \equiv$ 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 pn(n+m-Suc j) $v=\operatorname{coeff}(p * q) j$ (is ?l $=? r)$
proof -
have id1: $\wedge i . m+i-(n+m-S u c j)=i+S u c j-n$
using $j$ by auto
let ? $g=\lambda$. if $i \leq n+m-$ Suc $j \wedge n-$ Suc $j \leq i$ then coeff $p(i+$ Suc $j-$
$n) * v \$ i$ else 0

```
    have ?thesis \(=\left(\left(\sum i=0 . .<n\right.\right.\). ?g \(\left.i\right)=\)
    ( \(\sum i \leq j\). coeff \(p i *(\) if \(j-i<n\) then \(v \$(n-S u c(j-i))\) else 0\(\left.)\right)\) ) (is -
\(=(? l=? r))\)
    unfolding vec-of-poly-rev-shifted-def coeff-mult \(m\) scalar-prod-def \(n q\)-def
        coeff-poly-of-vec
    by (subst sum.cong, insert id1, auto)
also have ...
proof -
    have \(? r=\left(\sum i \leq j\right.\). \((\) if \(j-i<n\) then coeff \(p i * v \$(n-S u c(j-i))\) else 0\(\left.)\right)\)
(is - = sum ?f -)
        by (rule sum.cong, auto)
    also have sum? \(\{. . j\}=\operatorname{sum}\) ? \((\{i . i \leq j \wedge j-i<n\} \cup\{i . i \leq j \wedge \neg j-\)
\(i<n\}\) )
            \((\) is - \(=\) sum - \((? R 1 \cup ? R 2))\)
            by (rule sum.cong, auto)
    also have \(\ldots=\) sum ?f ? R1 + sum ?f ?R2
        by (subst sum.union-disjoint, auto)
    also have sum ?f ? R2 \(=0\)
        by (rule sum.neutral, auto)
    also have sum? ? ? \(11+0=\operatorname{sum}(\lambda i\). coeff \(p i * v \$(i+n-S u c j))\) ?R1
        (is - = sum ? F -)
        by (subst sum.cong, auto simp: ac-simps)
    also have \(\ldots=\operatorname{sum} ? F((? R 1 \cap\{. . m\}) \cup(? R 1-\{. . m\}))\)
        (is - = sum - \(\left.\left(? R \cup ? R^{\prime}\right)\right)\)
        by (rule sum.cong, auto)
    also have \(\ldots=\) sum ? F ? \(R+\) sum ? F ? \(R^{\prime}\)
        by (subst sum.union-disjoint, auto)
    also have sum ? \(F\) ? \(R^{\prime}=0\)
    proof -
        \{
        fix \(x\)
        assume \(x>m\)
        from coeff-eq- \(O[\) OF this[folded m]]
        have ? \(F x=0\) by simp
    \}
    thus ?thesis
        by (subst sum.neutral, auto)
    qed
    finally have \(r: ? r=s u m ? F\) ? \(R\) by simp
    have ?l \(=\operatorname{sum} ? g(\{i . i<n \wedge i \leq n+m-S u c j \wedge n-S u c j \leq i\}\)
        \(\cup\{i . i<n \wedge \neg(i \leq n+m-S u c j \wedge n-S u c j \leq i)\})\)
        (is - = sum - (?L1 \(\cup ? L 2))\)
        by (rule sum.cong, auto)
    also have \(\ldots=\) sum ? g ? L1 + sum ? g ? L2
        by (subst sum.union-disjoint, auto)
    also have sum ? \(g\) ? L2 \(=0\)
        by (rule sum.neutral, auto)
    also have sum ? \(g\) ? L1 \(+0=\operatorname{sum}(\lambda i\). coeff \(p(i+S u c j-n) * v \$ i) ? L 1\)
```

```
        (is - = sum ?G -)
        by (subst sum.cong, auto)
    also have ... = sum?G(?L1\cap{i.i +Suc j-n\leqm}\cup(?L1 - {i.i+
Suc j - n \leqm}))
    (is - = sum - (?L\cup?L'))
    by (subst sum.cong, auto)
    also have ... = sum??G?L + sum??G ?L'
    by (subst sum.union-disjoint, auto)
    also have sum?G? 'L'=0
    proof -
        {
            fix }
            assume }x+Sucj-n>
            from coeff-eq-O[OF this[folded m]]
            have ?G }x=0\mathrm{ by simp
        }
        thus ?thesis
            by (subst sum.neutral, auto)
    qed
    finally have l: ?l = sum?G ?L by simp
    let ?bij = \lambdai.i+n-Suc j
    {
        fix }
        assume x: j<m+nSuc (x+j)-n\leqmx<nn - Suc j\leqx
        define }y\mathrm{ where }y=x+Sucj-
        from }x\mathrm{ have }x+Sucj\geqn\mathrm{ by auto
        with }x\mathrm{ have xy: }x=\mathrm{ ?bij }y\mathrm{ unfolding y-def by auto
        from }x\mathrm{ have y:}y\in?R\mathrm{ unfolding }y\mathrm{ -def by auto
        have }x\in?bij'?R unfolding xy using y by blas
    } note tedious = this
    show ?thesis unfolding lr
        by (rule sum.reindex-cong[of ?bij], insert j, auto simp: inj-on-def tedious)
    qed
    finally show ?thesis by simp
qed
lemma sylvester-vec-poly:
    fixes p q :: 'a :: comm-semiring-0 poly
    defines m\equiv degree p
        and n\equiv degree q
    assumes v:v\incarrier-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 (rule poly-eqI)
    fix }
    note mn[simp] =m-def[symmetric] n-def[symmetric]
    let ?Tv = transpose-mat (sylvester-mat p q) *vv
    have dim: dim-vec (vec-first v n) = n dim-vec (vec-last v m) = m dim-vec ?Tv
```

```
=n+m
    using v by auto
    have if-distrib: \ x y z.(if x then y else (0 :: 'a))*z=(if x then y * z else 0)
    by auto
    show coeff ?l i}=\mathrm{ coeff ?r i
    proof (cases i<m+n)
    case False
        hence i-mn:i\geqm+n
            and i-n: \bigwedgex. x\leqi^x<n\longleftrightarrowx<n
            and i-m: \x. x\leqi^x<m\longleftrightarrow \longleftrightarrow < m by auto
        have coeff ?r i=
                    (\sumx<n. vec-first v n $ (n-Suc x)* coeff p (i-x)) +
                    (\sumx<m.vec-last v m $ (m-Suc x)* coeff q (i-x))
            (is - = sum ?f - + sum ?g -)
            unfolding coeff-add coeff-mult Let-def
            unfolding coeff-poly-of-vec dim if-distrib
            unfolding atMost-def
            apply(subst sum.inter-filter[symmetric],simp)
            apply(subst sum.inter-filter[symmetric],simp)
            unfolding mem-Collect-eq
            unfolding i-n i-m
            unfolding lessThan-def by simp
            also { fix }x\mathrm{ assume x: x<n
                have coeff p (i-x)=0
                    apply(rule coeff-eq-0) using i-mn x unfolding m-def by auto
            hence ?f }x=0\mathrm{ by auto
            } hence sum ?f {..<n}=0 by auto
            also { fix }x\mathrm{ assume x:x<m
                have coeff q}(i-x)=
                    apply(rule coeff-eq-0) using i-mn x unfolding n-def by auto
            hence ?g }x=0\mathrm{ by auto
            } hence sum?g {..<m}=0 by auto
            finally have coeff ?r i=0 by auto
            also from False have 0= coeff ?l i
                unfolding coeff-poly-of-vec dim sum.distrib[symmetric] by auto
            finally show ?thesis by auto
    next case True
            hence coeff ?l i = (transpose-mat (sylvester-mat p q) *vv)$(n+m-Suc
i)
            unfolding coeff-poly-of-vec dim sum.distrib[symmetric] by auto
        also have ... = coeff (p* poly-of-vec (vec-first v n) +q* poly-of-vec (vec-last
vm))}
            apply(subst index-mult-mat-vec) using True apply simp
            apply(subst row-transpose) using True apply simp
            apply(subst col-sylvester)
            unfolding mn using True apply simp
            apply(subst vec-first-last-append[of v n m,symmetric]) using v apply(simp
add: add.commute)
            apply(subst scalar-prod-append)
```

```
            apply (rule carrier-vecI,simp)+
            apply (subst vec-of-poly-rev-shifted-scalar-prod,simp,simp) using True apply
simp
            apply (subst add.commute[of n m])
            apply (subst vec-of-poly-rev-shifted-scalar-prod,simp,simp) using True apply
simp
            by simp
        also have ... =
            (\sumx\leqi. (if }x<n\mathrm{ then vec-first v n$ (n-Suc x) else 0) * coeff p (i-x))
+
            (\sumx\leqi. (if }x<m\mathrm{ then vec-last v m $ (m - Suc x) else 0) * coeff q (i-x))
            unfolding coeff-poly-of-vec[of vec-first v n,unfolded dim-vec-first,symmetric]
            unfolding coeff-poly-of-vec[of vec-last v m,unfolded dim-vec-last,symmetric]
            unfolding coeff-mult[symmetric] by (simp add: mult.commute)
            also have ... = coeff ?r i
            unfolding coeff-add coeff-mult Let-def
            unfolding coeff-poly-of-vec dim..
        finally show ?thesis.
    qed
qed
```


### 4.2.2 Homomorphism and Resultant

Here we prove Lemma 7.3.1 of the textbook.
lemma(in comm-ring-hom) resultant-sub-map-poly:
fixes $p q$ :: 'a poly
shows hom (resultant-sub $m n p q$ ) = resultant-sub $m$ (map-poly hom $p$ )
(map-poly hom q)
(is $? l=? r^{\prime}$ )
proof -
let $? m h=$ map-poly hom
have ?l $=$ det (sylvester-mat-sub $m n(? m h ~ p)(? m h ~ q))$
unfolding resultant-sub-def
apply (subst sylvester-mat-sub-map[symmetric]) by auto
thus ?thesis unfolding resultant-sub-def.
qed

### 4.2.3 Resultant as Polynomial Expression <br> context begin

This context provides notions for proving Lemma 7.2.1 of the textbook.
private fun $m k$-poly-sub where
$m k$-poly-sub A l $0=A$
$\mid$ mk-poly-sub A $l(S u c j)=$ mat-addcol $($ monom $1(S u c j)) l(l-S u c j)(m k$-poly-sub A $l j$ )
definition $m k$-poly $A=m k$-poly-sub (map-mat coeff-lift $A)(d i m-c o l A-1)$ (dim-col $A-1$ )

```
private lemma mk-poly-sub-dim[simp]:
    dim-row (mk-poly-sub A l j) = dim-row A
    dim-col (mk-poly-sub A l j) = dim-col A
    by (induct j,auto)
private lemma mk-poly-sub-carrier:
    assumes A\incarrier-mat nr nc shows mk-poly-sub A lj\incarrier-mat nr nc
    apply (rule carrier-matI) using assms by auto
private lemma mk-poly-dim[simp]:
    dim-col (mk-poly A)}=\operatorname{dim}-col 
    dim-row (mk-poly A) = dim-row A
    unfolding mk-poly-def by auto
private lemma mk-poly-sub-others[simp]:
    assumes l\not= j' and i<dim-row A and j'<dim-col A
    shows mk-poly-sub A l j$$ (i,j')=A $$ (i,j')
    using assms by (induct j; simp)
private 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):]
    unfolding mk-poly-def
    apply(subst mk-poly-sub-others)
    using ij by auto
private lemma mk-poly-delete[simp]:
    assumes i: i< dim-row }
    shows mat-delete (mk-poly A) i (dim-col A - 1) = map-mat coeff-lift (mat-delete
A i (dim-col A - 1))
    apply(rule eq-matI) unfolding mat-delete-def by auto
private lemma col-mk-poly-sub[simp]:
    assumes l\not=\mp@subsup{j}{}{\prime}}\mathrm{ and }\mp@subsup{j}{}{\prime}<dim-col 
    shows col (mk-poly-sub A lj) j' = col A j'
    by(rule eq-vecI; insert assms; simp)
private lemma det-mk-poly-sub:
    assumes A:(A:: ' }a::\mathrm{ comm-ring-1 poly mat) }\in\mathrm{ carrier-mat n n and i: i<n
    shows det (mk-poly-sub A (n-1) i) = det A
    using i
proof (induct i)
    case (Suc i)
        show ?case unfolding mk-poly-sub.simps
        apply(subst det-addcol[of - n])
            using Suc apply simp
            using Suc apply simp
            apply (rule mk-poly-sub-carrier[OF A])
```

using Suc by auto
qed $\operatorname{simp}$
private lemma det-mk-poly:
fixes $A$ :: ' $a$ :: comm-ring-1 mat
shows $\operatorname{det}(m k-p o l y A)=[: \operatorname{det} A:]$
proof (cases dim-row $A=\operatorname{dim}$-col $A$ )
case True
define $n$ where $n=\operatorname{dim}$-col $A$
have map-mat coeff-lift $A \in$ carrier-mat (dim-row $A$ ) (dim-col $A$ ) by simp
hence sq: map-mat coeff-lift $A \in$ carrier-mat (dim-col $A$ ) (dim-col $A$ ) unfolding True.
show ?thesis
proof (cases dim-col $A=0$ )
case True thus ?thesis unfolding det-def by simp
next case False thus ?thesis
unfolding $m k$-poly-def
by (subst det-mk-poly-sub[OF sq]; simp)
qed
next case False
hence f2: dim-row $A=$ dim-col $A \longleftrightarrow$ False by simp
hence f3: dim-row ( $m k$-poly $A$ ) $=$ dim-col $(m k$-poly $A) \longleftrightarrow$ False
unfolding mk-poly-dim by auto
show ?thesis unfolding det-def unfolding f2 f3 if-False by simp qed
private fun $m k$-poly2-row where
$m k$-poly2-row $A$ d j pv $0=p v$
$\mid m k$-poly2-row Adjpv (Suc n) $=$
$m k$-poly2-row $\left.A d j p v n\right|_{v} n \mapsto p v \$ n+\operatorname{monom}(A \$ \$(n, j)) d$
private fun $m k$-poly2-col where
$m k-p o l y 2-c o l A p v 0=p v$
| mk-poly2-col A pv (Suc m) =
mk-poly2-row A m(dim-col $A-S u c m)(m k-p o l y 2-c o l A p v m)(d i m-r o w A)$
private definition $m k$-poly2 $A \equiv m k$-poly2-col $A\left(0_{v}(\operatorname{dim}-r o w A)\right)(\operatorname{dim}-c o l A)$
private lemma mk-poly2-row-dim[simp]: dim-vec (mk-poly2-row A djpvi)= dim-vec pv
by (induct $i$ arbitrary: pv, auto)
private lemma $m k$-poly2-col-dim $[\operatorname{simp}]$ : dim-vec $(m k$-poly2-col A pvj) $=\operatorname{dim}$-vec $p v$
by (induct $j$ arbitrary: pv, auto)
private lemma $m k$-poly2-row:
assumes $n$ : $n \leq$ dim-vec $p v$
shows mk-poly2-row $A d j p v n \$ i=$
(if $i<n$ then $p v \$ i+\operatorname{monom}(A \$ \$(i, j)) d$ else $p v \$ i)$
using $n$
proof (induct $n$ arbitrary: pv)
case (Suc n) thus ?case
unfolding mk-poly2-row.simps by (cases rule: linorder-cases[of in],auto)
qed simp
private lemma $m k$-poly2-row-col:
assumes $\operatorname{dim}[\operatorname{simp}]$ : dim-vec $p v=n \operatorname{dim}$-row $A=n$ and $j: j<\operatorname{dim}$-col $A$
shows mk-poly2-row $A d j p v n=p v+\operatorname{map-vec}(\lambda a$. monom ad $)(\operatorname{col} A j)$
apply rule using $m k$-poly2-row $[o f-p v] j$ by auto
private lemma $m k$-poly2-col:
fixes $p v::$ ' $a$ :: comm-semiring-1 poly vec and $A$ :: ' $a$ mat
assumes $i$ : $i<$ dim-row $A$ and dim: dim-row $A=\operatorname{dim-vec~pv~}$
shows mk-poly2-col $A$ pv $j \$ i=p v \$ i+\left(\sum j^{\prime}<j\right.$. monom $(A \$ \$(i, d i m-c o l ~ A$

- Suc $\left.\left.\left.j^{\prime}\right)\right) j^{\prime}\right)$
using dim
proof (induct $j$ arbitrary: pv)
case (Suc j) show ?case
unfolding $m k$-poly2-col.simps
apply (subst mk-poly2-row)
using Suc apply simp
unfolding $\operatorname{Suc}(1)[O F \operatorname{Suc}(2)]$
using $i$ by (simp add: add.assoc)
qed $\operatorname{simp}$
private lemma $m k$-poly2-pre:
fixes $A$ :: 'a :: comm-semiring-1 mat
assumes $i$ : $i<$ dim-row $A$
shows mk-poly2 $A \$ i=\left(\sum j^{\prime}<d i m-c o l ~ A . m o n o m ~(A \$ \$(i, d i m-c o l A-S u c\right.$ $\left.j^{\prime}\right)$ ) $j^{\prime}$ )
unfolding $m k$-poly2-def
apply (subst $m k$-poly2-col) using $i$ by auto
private lemma $m k$-poly 2 :
fixes $A::{ }^{\prime} a::$ comm-semiring-1 mat
assumes $i: i<$ dim-row $A$
and $c: \operatorname{dim}-\operatorname{col} A>0$
shows mk-poly2 $A \$ i=\left(\sum j^{\prime}<d i m-c o l\right.$ A. monom $\left(A \$ \$\left(i, j^{\prime}\right)\right)($ dim-col $A-$ Suc $\left.j^{\prime}\right)$ )

$$
(\text { is } ? l=\text { sum ?f ?S })
$$

proof -
define $l$ where $l=\operatorname{dim}-\operatorname{col} A-1$
have dim: dim-col $A=$ Suc $l$ unfolding $l$-def using $i c$ by auto
let $? g=\lambda j . l-j$
have ?l $=\operatorname{sum}(? f \circ ? g)$ ?S unfolding $l$-def $m k$-poly2-pre $[O F i]$ by auto
also have $\ldots=$ sum ?f ? $S$
unfolding dim
unfolding lessThan-Suc-atMost
using sum.reindex[OF inj-on-diff-nat2,symmetric,unfolded image-diff-atMost]. finally show ?thesis.
qed
private lemma mk-poly2-sylvester-upper:
fixes $p q::{ }^{\prime} a$ :: comm-semiring-1 poly
assumes $i: i<$ degree $q$
shows mk-poly2 (sylvester-mat pq) \$i=monom 1 (degree $q-S u c i) * p$
apply (subst mk-poly2)
using $i$ apply simp using $i$ apply $\operatorname{simp}$
apply (subst sylvester-sum-mat-upper[OF i,symmetric])
apply (rule sum.cong)
unfolding sylvester-mat-dim lessThan-Suc-atMost apply simp
by auto
private lemma $m k$-poly2-sylvester-lower:
fixes $p q$ :: ' $a$ :: comm-semiring-1 poly
assumes $m i: i \geq$ degree $q$ and imn: $i<$ degree $p+$ degree $q$
shows mk-poly2 (sylvester-mat pq) \$i=monom 1 (degree $p+$ degree $q-$ Suc
i) $* q$
apply (subst mk-poly2)
using imn apply simp using mi imn apply simp
unfolding sylvester-mat-dim
using sylvester-sum-mat-lower[OF mi imn]
apply (subst sylvester-sum-mat-lower) using mi imn by auto

## private lemma foo:

fixes $v::{ }^{\prime} a::$ comm-semiring-1 vec
shows monom $1 d \cdot{ }_{v}$ map-vec coeff-lift $v=\operatorname{map-vec}(\lambda a$. monom a d) $v$
apply (rule eq-vecI)
unfolding index-map-vec index-col
by (auto simp add: Polynomial.smult-monom)
private lemma $m k$-poly-sub-corresp:
assumes $\operatorname{dim} A[\operatorname{simp}]: \operatorname{dim}-\operatorname{col} A=S u c l$ and $\operatorname{dimpv}[\operatorname{simp}]: \operatorname{dim}-v e c p v=\operatorname{dim}$-row A
and $j: j<\operatorname{dim}-\operatorname{col} A$
shows $p v+\operatorname{col}(m k$-poly-sub (map-mat coeff-lift $A) l j) l=$
$m k-p o l y 2-c o l A p v(S u c j)$
proof $($ insert $j$, induct $j$ )
have le: dim-row $A \leq$ dim-vec $p v$ using dimpv by simp
have $l: l<\operatorname{dim}-c o l A$ using $\operatorname{dim} A$ by $\operatorname{simp}$
\{ case 0 show ?case
apply (rule eq-vecI)
using $m k$-poly2-row[OF le]
by (auto simp add: monom-0)
\}
\{ case (Suc j)

```
    hence j: j<dim-col A by simp
    show ?case
        unfolding mk-poly-sub.simps
    apply(subst col-addcol)
        apply simp
        apply simp
        apply(subst(2) comm-add-vec)
        apply(rule carrier-vecI, simp)
        apply(rule carrier-vecI, simp)
        apply(subst assoc-add-vec[symmetric])
            apply(rule carrier-vecI, rule refl)
            apply(rule carrier-vecI, simp)
    apply(rule carrier-vecI, simp)
    unfolding Suc(1)[OF j]
    apply(subst(2) mk-poly2-col.simps)
    apply(subst mk-poly2-row-col)
        apply simp
        apply simp
        using Suc apply simp
        apply(subst col-mk-poly-sub)
        using Suc apply simp
        using Suc apply simp
        apply(subst col-map-mat)
        using }\operatorname{dim}A\mathrm{ apply simp
        unfolding foo dimA by simp
    }
qed
private lemma col-mk-poly-mk-poly2:
    fixes }A:: ' 'a :: comm-semiring-1 mat
    assumes dim: dim-col A>0
    shows col (mk-poly A) (dim-col A - 1) = mk-poly2 A
proof -
    define l where l= dim-col A-1
    have dim: dim-col A = Suc l unfolding l-def using dim by auto
    show ?thesis
        unfolding mk-poly-def mk-poly2-def dim
        apply(subst mk-poly-sub-corresp[symmetric])
            apply(rule dim)
            apply simp
            using dim apply simp
        apply(subst left-zero-vec)
            apply(rule carrier-vecI) using dim apply simp
        apply simp
        done
qed
private lemma mk-poly-mk-poly2:
    fixes }A :: ' a :: comm-semiring-1 mat
```

```
    assumes dim: dim-col A>0 and i: i< dim-row A
    shows mk-poly A $$ (i,dim-col A - 1)=mk-poly2 A $ i
proof -
    have mk-poly A $$ (i,dim-col A - 1) = col (mk-poly A) (dim-col A - 1)$ i
    apply (subst index-col(1)) using dim i by auto
    also note col-mk-poly-mk-poly2[OF dim]
    finally show ?thesis.
qed
lemma mk-poly-sylvester-upper:
    fixes p q ::' 'a :: comm-ring-1 poly
    defines }m\equiv\mathrm{ degree }p\mathrm{ and }n\equiv\mathrm{ degree }
    assumes i: i<n
    shows mk-poly (sylvester-mat p q) $$ (i,m + n-1) = monom 1 ( n-Suc i)
* p(is ?l = ?r)
proof -
    let ?S = sylvester-mat pq
    have c: m+n= dim-col ?S and r:m+n=dim-row?S unfolding m-def n-def
by auto
    hence dim-col ?S > 0 i< dim-row ?S using i by auto
    from mk-poly-mk-poly2[OF this]
    have ?l = mk-poly2 (sylvester-mat p q) $ i unfolding m-def n-def by auto
    also have ... = ?r
        apply(subst mk-poly2-sylvester-upper)
            using i unfolding n-def m-def by auto
    finally show ?thesis.
qed
lemma mk-poly-sylvester-lower:
    fixes p q :: ' a :: comm-ring-1 poly
    defines }m\equiv\mathrm{ degree }p\mathrm{ and }n\equiv\mathrm{ degree }
    assumes ni: n\leqi and imn: i<m+n
    shows mk-poly (sylvester-mat p q) $$ (i,m+n-1) = monom 1 (m+n-
Suc i)*q(is ?l = ?r)
proof -
    let ?S = sylvester-mat p q
    have c:m+n=dim-col ?S and r: m+n= dim-row ?S unfolding m-def n-def
by auto
    hence dim-col ?S>0 i< dim-row ?S using imn by auto
    from mk-poly-mk-poly2[OF this]
    have ?l = mk-poly2 (sylvester-mat p q) $ i unfolding m-def n-def by auto
    also have ... = ?r
        apply(subst mk-poly2-sylvester-lower)
            using ni imn unfolding n-def m-def by auto
    finally show ?thesis.
qed
```

The next lemma corresponds to Lemma 7.2.1.
lemma resultant-as-poly:
fixes $p$ :: ' $a$ :: comm-ring-1 poly
assumes degp: degree $p>0$ and degq: degree $q>0$
shows $\exists p^{\prime} q^{\prime}$. degree $p^{\prime}<$ degree $q \wedge$ degree $q^{\prime}<$ degree $p \wedge$
$[:$ resultant $p q:]=p^{\prime} * p+q^{\prime} * q$
proof (intro exI conjI)
define $m$ where $m=$ degree $p$
define $n$ where $n=$ degree $q$
define $d$ where $d=$ dim-row ( $m k$-poly (sylvester-mat $p q$ ))
define $c$ where $c=(\lambda i$. coeff-lift (cofactor (sylvester-mat p q) $i(m+n-1))$ )
define $p^{\prime}$ where $p^{\prime}=\left(\sum i<n\right.$. monom $\left.1(n-S u c i) * c i\right)$
define $q^{\prime}$ where $q^{\prime}=\left(\sum i<m\right.$. monom $\left.1(m-S u c i) * c(n+i)\right)$
have degc: $\bigwedge i$. degree ( $c i$ ) $=0$ unfolding $c$-def by auto
have $d m n: d=m+n$ and $m n d: m+n=d$ unfolding $d$-def $m$-def $n$-def by auto
have $[:$ resultant $p$ : $:]=$
$\left(\sum i<d . m k\right.$-poly $($ sylvester-mat p q) $\$ \$(i, m+n-1) *$
cofactor (mk-poly (sylvester-mat p q)) $i(m+n-1))$
unfolding resultant-def
unfolding det-mk-poly[symmetric]
unfolding $m$-def $n$-def $d$-def
apply(rule laplace-expansion-column $[$ of - degree $p+$ degree $q-1]$ )
apply (rule carrier-matI) using degp by auto
also \{ fix $i$ assume $i: i<d$
have d2: $d=$ dim-row (sylvester-mat $p$ ) unfolding $d$-def by auto
have cofactor $(m k$-poly (sylvester-mat $p q)) i(m+n-1)=$
$(-1) \wedge(i+(m+n-1)) * \operatorname{det}($ mat-delete $(m k-p o l y(s y l v e s t e r-m a t ~ p q)) i$ ( $m+n-1$ )
using cofactor-def.
also have ... $=$
$(-1) \wedge(i+m+n-1) *$ coeff-lift (det (mat-delete (sylvester-mat p q) $i$
$(m+n-1)))$
using $m k$-poly-delete[OF i[unfolded d2]] degp degq
unfolding $m$-def $n$-def by (auto simp add: add.assoc)
also have $i+m+n-1=i+(m+n-1)$ using $i[$ folded $m n d]$ by auto
finally have cofactor ( $m k$-poly (sylvester-mat pq)) $i(m+n-1)=c i$ unfolding $c$-def cofactor-def hom-distribs by simp
\}
hence $\ldots=\left(\sum i<d\right.$. $m k$-poly $($ sylvester-mat $\left.p q) \$ \$(i, m+n-1) * c i\right)$
(is - = sum ? f -) by auto
also have $\ldots=\operatorname{sum}$ ?f $(\{. .<n\} \cup\{n \ldots<d\})$ unfolding dmn apply(subst ivl-disj-un(8)) by auto
also have $\ldots=$ sum ?f $\{. .<n\}+$ sum ?f $\{n . .<d\}$ apply(subst sum.union-disjoint) by auto
also $\{$ fix $i$ assume $i: i<n$
have ?f $i=$ monom $1(n-$ Suc $i) * c i * p$
unfolding $m$-def $n$-def
apply (subst mk-poly-sylvester-upper)
using $i$ unfolding $n$-def by auto \}
hence sum ?f $\{. .<n\}=p^{\prime} * p$ unfolding $p^{\prime}$-def sum-distrib-right by auto
also \{fix $i$ assume $i: i \in\{n . .<d\}$
have ?f $i=$ monom $1(m+n-S u c i) * c i * q$
unfolding $m$-def $n$-def
apply (subst $m k$-poly-sylvester-lower)
using $i$ unfolding dmn $n$-def $m$-def by auto
\}
hence sum ?f $\{n . .<d\}=\left(\sum i=n . .<d\right.$. monom $\left.1(m+n-S u c i) * c i\right) * q$
(is $-=$ sum ? $h-*-$ ) unfolding sum-distrib-right by auto
also have $\{n . .<d\}=(\lambda i . i+n) '\{0 . .<m\}$
by (simp add: dmn)
also have sum ? $h \ldots=\operatorname{sum}(? h \circ(\lambda i . i+n))\{0 . .<m\}$
apply(subst sum.reindex[symmetric])
apply (rule inj-onI) by auto
also have $\ldots=q^{\prime}$ unfolding $q^{\prime}$-def apply(rule sum.cong) by (auto simp add: add.commute)
finally show main: [:resultant $p q:]=p^{\prime} * p+q^{\prime} * q$.
show degree $p^{\prime}<n$
unfolding $p^{\prime}$-def
apply (rule degree-sum-smaller)
using degq[folded $n$-def] apply force +
proof -
fix $i$ assume $i: i \in\{. .<n\}$
show degree (monom $1(n-S u c i) * c i)<n$
apply (rule order.strict-trans1)
apply (rule degree-mult-le)
unfolding add.right-neutral degc
apply (rule order.strict-trans1)
apply (rule degree-monom-le) using $i$ by auto
qed
show degree $q^{\prime}<m$
unfolding $q^{\prime}$-def
apply (rule degree-sum-smaller)
using degp [folded m-def] apply force+
proof -
fix $i$ assume $i: i \in\{. .<m\}$
show degree (monom $1(m-S u c i) * c(n+i))<m$
apply (rule order.strict-trans1)
apply (rule degree-mult-le)
unfolding add.right-neutral degc
apply (rule order.strict-trans1)
apply (rule degree-monom-le) using $i$ by auto
qed
qed
end

### 4.2.4 Resultant as Nonzero Polynomial Expression

```
lemma resultant-zero:
    fixes p q :: 'a :: comm-ring-1 poly
    assumes deg: degree p>0\vee degree q>0
        and xp: poly p x=0 and xq: poly q x=0
    shows resultant pq=0
proof -
    { assume degp: degree p>0 and degq: degree q>0
    obtain }\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}\mathrm{ where [: resultant pq:] = p'*p+ q}**
        using resultant-as-poly[OF degp degq] by force
    hence resultant pq= poly ( }\mp@subsup{p}{}{\prime}*p+\mp@subsup{q}{}{\prime}*q)
            using mpoly-base-conv(2)[of resultant pq] by auto
    also have ... = poly px* poly p'x+ poly q x * poly q' 
            unfolding poly2-def by simp
    finally have ?thesis using xp xq by simp
    } moreover
    { assume degp: degree p=0
    have p:p=[:0:] using xp degree-0-id[OF degp,symmetric] by (metis mpoly-base-conv(2))
    have ?thesis unfolding p using degp deg by simp
    } moreover
    { assume degq: degree q=0
    have q:q=[:0:] using xq degree-0-id[OF degq,symmetric] by (metis mpoly-base-conv(2))
        have ?thesis unfolding q using degq deg by simp
    }
    ultimately show ?thesis by auto
qed
lemma poly-resultant-zero:
    fixes pq::' 'a :: comm-ring-1 poly poly
    assumes deg: degree p>0\vee degree q>0
    assumes p0: poly2 p x y = 0 and q0: poly2 q x y =0
    shows poly (resultant p q) }x=
proof -
    { assume degree p>0 degree q>0
        from resultant-as-poly[OF this]
        obtain p}\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}\mathrm{ where [: resultant pq:] = p'* p+ q}\mp@subsup{q}{}{\prime}*q\mathrm{ by force
        hence resultant p q= poly ( }\mp@subsup{p}{}{\prime}*p+\mp@subsup{q}{}{\prime}*q) [:y:
            using mpoly-base-conv(2)[of resultant pq] by auto
        also have poly ... x= poly2 pxy* poly2 p'xy+poly2 q x y * poly2 q' x y
            unfolding poly2-def by simp
    finally have ?thesis unfolding p0 q0 by simp
    } moreover {
    assume degp: degree p=0
    hence p:p=[:coeff p 0:] by(subst degree-0-id[OF degp,symmetric],simp)
    hence resultant pq= coeff p 0^ degree qusing resultant-const(1) by metis
    also have poly ... x voly(coeff p 0) x^ degree q by auto
    also have ... = poly2 pxy^degree q unfolding poly2-def by(subst p,auto)
    finally have ?thesis unfolding p0 using deg degp zero-power by auto
    } moreover {
```

```
    assume degq: degree q=0
    hence q: q=[:coeff q 0:] by(subst degree-0-id[OF degq,symmetric],simp)
    hence resultant pq= coeff q0^ degree pusing resultant-const(2) by metis
    also have poly ... x= poly (coeff q 0) x^ degree p by auto
    also have ... = poly2 q x y ^ degree p unfolding poly2-def by(subst q, auto)
    finally have ?thesis unfolding q0 using deg degq zero-power by auto
}
    ultimately show ?thesis by auto
qed
lemma resultant-as-nonzero-poly-weak:
    fixes p q :: ' }a\mathrm{ :: idom poly
    assumes degp: degree p>0 and degq: degree q>0
        and r0: resultant p q}=
    shows \exists\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime}\mathrm{ . degree }\mp@subsup{p}{}{\prime}<\mathrm{ degree q}\wedge\mathrm{ degree }\mp@subsup{q}{}{\prime}<\mathrm{ degree }p\wedge
        [: resultant p q:] = p'* * + \mp@subsup{q}{}{\prime}*q\wedge p
proof -
    obtain p}\mp@subsup{p}{}{\prime}\mp@subsup{q}{}{\prime
    where deg: degree }\mp@subsup{p}{}{\prime}<\mathrm{ degree q degree q}\mp@subsup{q}{}{\prime}<\mathrm{ degree p
            and main: [: resultant p q:] = \mp@subsup{p}{}{\prime}*p+\mp@subsup{q}{}{\prime}*q
            using resultant-as-poly[OF degp degq] by auto
    have p0:p\not=0 using degp by auto
    have q0:q\not=0 using degq by auto
    show ?thesis
    proof (intro exI conjI notI)
        assume p
        hence [: resultant p q:] = q'*q using main by auto
        also hence d0:0 = degree ( }\mp@subsup{q}{}{\prime}*q)\mathrm{ by (metis degree-pCons-0)
            { assume q' = 0
                hence degree ( }\mp@subsup{q}{}{\prime}*q)=\mathrm{ degree }\mp@subsup{q}{}{\prime}+\mathrm{ degree }
                        apply(rule degree-mult-eq) using q0 by auto
                hence False using d0 degq by auto
            } hence q' = 0 by auto
        finally show False using r0 by auto
    next
        assume q}\mp@subsup{q}{}{\prime}=
        hence [: resultant p q:] = p'* p using main by auto
        also
            hence d0:0 = degree ( }\mp@subsup{p}{}{\prime}*p)\mathrm{ by (metis degree-pCons-0)
            { assume p}\mp@subsup{p}{}{\prime}\not=
                hence degree ( }\mp@subsup{p}{}{\prime}*p)=\mathrm{ degree }\mp@subsup{p}{}{\prime}+\mathrm{ degree }
                        apply(rule degree-mult-eq) using p0 by auto
            hence False using d0 degp by auto
            } hence p}\mp@subsup{p}{}{\prime}=0\mathrm{ by auto
    finally show False using r0 by auto
    qed fact+
qed
```

Next lemma corresponds to Lemma 7.2.2 of the textbook

```
lemma resultant-as-nonzero-poly:
    fixes \(p\) :: ' \(a\) :: idom poly
    defines \(m \equiv\) degree \(p\) and \(n \equiv\) degree \(q\)
    assumes degp: \(m>0\) and degq: \(n>0\)
    shows \(\exists p^{\prime} q^{\prime}\). degree \(p^{\prime}<n \wedge\) degree \(q^{\prime}<m \wedge\)
        \([:\) resultant \(p q:]=p^{\prime} * p+q^{\prime} * q \wedge p^{\prime} \neq 0 \wedge q^{\prime} \neq 0\)
proof (cases resultant p \(q=0\) )
    case False
    thus ?thesis
        using resultant-as-nonzero-poly-weak \(\operatorname{deg} p \operatorname{degq}\)
        unfolding \(m\)-def \(n\)-def by auto
next case True
    define \(S\) where \(S=\) transpose-mat (sylvester-mat \(p q\) )
    have \(S: S \in\) carrier-mat \((m+n)(m+n)\) unfolding \(S\)-def \(m\)-def \(n\)-def by auto
    have det \(S=0\) using True
        unfolding resultant-def \(S\)-def apply (subst det-transpose) by auto
    then obtain \(v\)
        where \(v: v \in\) carrier-vec \((m+n)\) and \(v 0: v \neq O_{v}(m+n)\) and \(S *_{v} v=O_{v}\)
( \(m+n\) )
        using det-0-iff-vec-prod-zero \([O F S]\) by auto
    hence poly-of-vec \(\left(S *_{v} v\right)=0\) by auto
    hence main: poly-of-vec (vec-first v \(n\) ) * \(p+\) poly-of-vec (vec-last v \(m\) ) \(* q=0\)
        (is ? \(p *-+? q *-=-\) )
        using sylvester-vec-poly[OF v[unfolded m-def \(n\)-def], folded m-def \(n\)-def \(S\)-def]
        by auto
    have split: vec-first vn @ \({ }_{v}\) vec-last v \(m=v\)
        using vec-first-last-append[simplified add.commute] \(v\) by auto
    show ?thesis
    proof(intro exI conjI)
        show [: resultant \(p q:]=? p * p+? q * q\) unfolding True using main by
auto
        show \(? p \neq 0\)
        proof
            assume \(p^{\prime} 0: ? p=0\)
            hence ? \(q * q=0\) using main by auto
            hence \(? q=0\) using \(\operatorname{degq} n\)-def by auto
            hence vec-last \(v m=0_{v} m\) unfolding poly-of-vec- 0 -iff by auto
            also have vec-first vn @ \({ }_{v} \ldots=0_{v}(m+n)\) using \(p^{\prime} 0\) unfolding poly-of-vec- 0 -iff
by auto
            finally have \(v=O_{v}(m+n)\) using split by auto
            thus False using v0 by auto
                    qed
                    show \(? q \neq 0\)
                    proof
                        assume \(q^{\prime} 0: ? q=0\)
                        hence ? \(p * p=0\) using main by auto
            hence \(? p=0\) using degp \(m\)-def by auto
            hence vec-first \(v n=0_{v} n\) unfolding poly-of-vec- 0 -iff by auto
            also have \(\ldots @_{v}\) vec-last \(v m=0_{v}(m+n)\) using \(q^{\prime} 0\) unfolding poly-of-vec- 0 -iff
```

by auto
finally have $v=O_{v}(m+n)$ using split by auto
thus False using v0 by auto
qed
show degree ? $p<n$ using degree-poly-of-vec-less[of vec-first $v n]$ using $\operatorname{degq}$
by auto
show degree ? $q<m$ using degree-poly-of-vec-less[of vec-last $v m$ ] using degp by auto
qed
qed
Corresponds to Lemma 7.2.3 of the textbook
lemma resultant-zero-imp-common-factor:
fixes $p q::{ }^{\prime} a$ :: ufd poly
assumes deg: degree $p>0 \vee$ degree $q>0$ and $r 0$ : resultant $p q=0$
shows $\neg$ coprime $p q$
unfolding neq0-conv[symmetric]
proof -
\{ assume degp: degree $p>0$ and degq: degree $q>0$
assume cop: coprime $p q$
obtain $p^{\prime} q^{\prime}$ where $p^{\prime} * p+q^{\prime} * q=0$
and $p^{\prime}$ : degree $p^{\prime}<$ degree $q$ and $q^{\prime}$ : degree $q^{\prime}<$ degree $p$
and $p^{\prime} 0: p^{\prime} \neq 0$ and $q^{\prime} 0: q^{\prime} \neq 0$
using resultant-as-nonzero-poly[OF degp degq] r0 by auto
hence $p^{\prime} * p=-q^{\prime} * q$ by (simp add: eq-neg-iff-add-eq-0)
from some-gcd.coprime-mult-cross-dvd[OF cop this]
have $p d v d q^{\prime}$ by auto
from dvd-imp-degree-le[OF this $\left.q^{\prime} 0\right]$
have degree $p \leq$ degree $q^{\prime}$ by auto
hence False using $q^{\prime}$ by auto
\}
moreover
\{ assume degp: degree $p=0$
then obtain $x$ where $p=[: x:]$ by (elim degree-eq-zeroE)
moreover hence resultant $p q=x^{\wedge}$ degree $q$ using resultant-const by auto
hence $x=0$ using $r 0$ by auto
ultimately have $p=0$ by auto
hence ?thesis unfolding not-coprime-iff-common-factor
by (metis deg degp dvd-0-right dvd-refl less-numeral-extra(3) poly-dvd-1)
\}
moreover
\{ assume degq: degree $q=0$
then obtain $x$ where $q=[: x:]$ by (elim degree-eq-zero $E$ )
moreover hence resultant $p q=x^{\wedge}$ degree $p$ using resultant-const by auto
hence $x=0$ using $r 0$ by auto
ultimately have $q=0$ by auto
hence ?thesis unfolding not-coprime-iff-common-factor
by (metis deg degq dvd-0-right dvd-refl less-numeral-extra(3) poly-dvd-1)

```
}
ultimately show ?thesis by auto
qed
lemma resultant-non-zero-imp-coprime:
    assumes nz: resultant (f :: 'a :: field poly) g}\not=
    and nz':}f\not=0\veeg\not=
shows coprime f g
proof (cases degree f=0\vee degree g=0)
    case False
    define }r\mathrm{ where }r=[:resultant f g:]
    from nz have r: r}\not=0\mathrm{ unfolding r-def by auto
    from False have degree f>0 degree g>0 by auto
    from resultant-as-nonzero-poly-weak[OF this nz]
    obtain pq where degree p<degree g degree q< degree f
    and id:r=p*f+q*g
    and p\not=0q\not=0 unfolding r-def by auto
    define h}\mathrm{ where }h=\mathrm{ some-gcd fg
    have h dvd f h dvd g}\mathrm{ unfolding h-def by auto
    then obtain jk where f:f=h*j and g:g=h*k unfolding dvd-def by
auto
    from id[unfolded fg] have id: }h*(p*j+q*k)=r by (auto simp: field-simps
    from arg-cong[OF id, of degree] have degree (h* (p*j+q*k))=0
        unfolding r-def by auto
    also have degree (h*(p*j+q*k)) = degree h + degree ( }p*j+q*k
    by (subst degree-mult-eq, insert id r, auto)
    finally have h: degree h=0 h\not=0 using rid by auto
    thus ?thesis unfolding h-def using is-unit-iff-degree some-gcd.gcd-dvd-1 by
blast
next
    case True
    thus ?thesis
    proof
    assume deg-g: degree g=0
    show ?thesis
    proof (cases g=0)
        case False
        then show ?thesis using divides-degree[of - g, unfolded deg-g]
                by (simp add: is-unit-right-imp-coprime)
    next
            case g: True
            then have g=[:0:] by auto
            from nz[unfolded this resultant-const] have degree f = 0 by auto
            with nz' show ?thesis unfolding g}\mathrm{ by auto
        qed
    next
        assume deg-f: degree f}=
        show ?thesis
    proof (cases f=0)
```

```
        case False
        then show ?thesis using divides-degree[of - f, unfolded deg-f]
        by (simp add: is-unit-left-imp-coprime)
        next
            case f:True
            then have f=[:0:] by auto
            from nz[unfolded this resultant-const] have degree g=0 by auto
            with nz' show ?thesis unfolding f by auto
        qed
    qed
qed
end
```


## 5 Algebraic Numbers: Addition and Multiplication

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

```
theory Algebraic-Numbers
    imports
    Algebraic-Numbers-Prelim
    Resultant
    Polynomial-Factorization.Polynomial-Divisibility
begin
interpretation coeff-hom: monoid-add-hom \lambdap. coeff p i by (unfold-locales, auto)
interpretation coeff-hom: comm-monoid-add-hom \lambdap. coeff p i..
interpretation coeff-hom: group-add-hom \lambdap. coeff p i..
interpretation coeff-hom: ab-group-add-hom \lambdap. coeff p i..
interpretation coeff-0-hom: monoid-mult-hom \lambdap. coeff p 0 by (unfold-locales,
auto simp: coeff-mult)
interpretation coeff-0-hom: semiring-hom \lambdap.coeff p 0..
interpretation coeff-0-hom:comm-monoid-mult-hom \lambdap. coeff p 0..
interpretation coeff-0-hom: comm-semiring-hom \lambdap. coeff p 0..
```


### 5.1 Addition of Algebraic Numbers

definition $x-y \equiv[:[: 0,1:],-1:]$
definition poly-x-minus-y $p=$ poly-lift $p \circ_{p} x-y$
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 $\left.(n-k)) *(-1)^{\wedge} k\right)(n-k)$
proof -
define $X$ :: 'a poly poly where $X=$ monom (monom 1 1) 0
define $Y::$ 'a poly poly where $Y=$ monom $(-1) 1$
have [simp]: monom $1 b *(-1)^{\wedge} k=\operatorname{monom}((-1) \wedge k:: ' a) b$ for $b k$
by (auto simp: monom-altdef minus-one-power-iff)
have $(X+Y){ }^{\wedge} n=\left(\sum i \leq n\right.$. of-nat $(n$ choose $\left.i) * X^{\wedge} i * Y^{\wedge}(n-i)\right)$
by (subst binomial-ring) auto
also have $\ldots=\left(\sum i \leq n\right.$. of-nat ( $n$ choose $\left.i\right) * \operatorname{monom}(\operatorname{monom}((-1) \wedge(n-$ i)) i) $(n-i))$
by (simp add: X-def $Y$-def monom-power mult-monom mult.assoc)
also have $\ldots=\left(\sum i \leq n\right.$. monom (monom (of-nat ( $n$ choose $\left.i\right) *(-1)^{\wedge}(n-$
i)) i) $(n-i))$
by (simp add: of-nat-poly smult-monom)
also have coeff ... $k=$
( $\sum i \leq n$. if $n-i=k$ then monom (of-nat ( $n$ choose $\left.\left.i\right) *(-1)^{\wedge}(n-i)\right) i$ else 0)
by (simp add: of-nat-poly coeff-sum)
also have $\ldots=\left(\sum i \in\{n-k\}\right.$. monom (of-nat (n choose $\left.\left.i\right) *(-1)^{\wedge}(n-i)\right)$
i)
using $\langle k \leq n\rangle$ by (intro sum.mono-neutral-cong-right) auto
also have $X+Y=x-y$
by (simp add: X-def $Y$-def $x$ - $y$-def monom-altdef)
finally show ?thesis
using $\langle k \leq n\rangle$ by $\operatorname{simp}$
qed
The following polynomial represents the sum of two algebraic numbers.
definition poly-add $::$ ' $a$ :: comm-ring-1 poly $\Rightarrow$ 'a poly $\Rightarrow$ ' $a$ poly where poly-add p $q=$ resultant $($ poly-x-minus-y $p)($ poly-lift $q)$

### 5.1.1 poly-add has desired root

interpretation poly-x-minus-y-hom:
comm-ring-hom poly-x-minus-y by (unfold-locales; simp add: poly-x-minus-y-def hom-distribs)
lemma poly2-x-y[simp]:
fixes $x$ :: ' $a$ :: comm-ring-1
shows poly2 $x-y x y=x-y$ unfolding poly2-def by (simp add: $x-y$-def)
lemma degree-poly-x-minus-y[simp]:
fixes $p::$ ' $a:$ :idom poly
shows degree (poly-x-minus-y $p$ ) $=$ degree $p$ unfolding poly-x-minus- $y$-def $x$ - $y$-def by auto
lemma poly-x-minus-y-pCons[simp]:
poly-x-minus-y (pCons a $p$ ) $=[:[: a:]:]+$ poly-x-minus-y $p * x-y$
unfolding poly-x-minus- $y$-def $x$ - $y$-def by simp
lemma poly-poly-poly-x-minus-y[simp]:
fixes $p$ :: ' $a$ :: comm-ring-1 poly
shows poly (poly (poly-x-minus-y $p) q) x=$ poly $p(x-\operatorname{poly} q x)$
by (induct $p$; simp add: ring-distribs $x$ - $y$-def)
lemma poly2-poly-x-minus-y[simp]:
fixes $p::$ ' $a::$ comm-ring-1 poly
shows poly2 (poly-x-minus-y $p$ ) $x y=$ poly $p(x-y)$ unfolding poly2-def by simp
interpretation $x$ - $y$-mult-hom: zero-hom-0 $\lambda p$ :: ' $a$ :: comm-ring-1 poly poly. $x-y$ *
p
proof (unfold-locales)
fix $p$ :: 'a poly poly
assume $x-y * p=0$
then show $p=0$ apply (simp add: $x$ - $y$-def)
by (metis eq-neg-iff-add-eq-0 minus-equation-iff minus-pCons synthetic-div-unique-lemma)
qed
lemma $x$ - $y$-nonzero $[$ simp $]: x-y \neq 0$ by (simp add: $x-y$-def)
lemma degree- $x-y[$ simp $]$ : degree $x-y=1$ by (simp add: $x$ - $y$-def)
interpretation $x$-y-mult-hom: inj-comm-monoid-add-hom $\lambda p$ :: 'a :: idom poly
poly. $x-y * p$
proof (unfold-locales)
show $x-y * p=x-y * q \Longrightarrow p=q$ for $p q::$ 'a poly poly
proof (induct $p$ arbitrary:q)
case 0
then show ?case by simp
next
case $p:(p$ Cons a $p)$
from $p(3)$ [unfolded mult-pCons-right]
have $x-y *($ monom a $0+p$ Cons $01 * p)=x-y * q$
apply (subst(asm) pCons-0-as-mult)
apply (subst(asm) smult-prod) by (simp only: field-simps distrib-left)
then have monom a $0+p$ Cons $01 * p=q$ by simp then show $p$ Cons a $p=q$ using $p$ Cons-as-add by (simp add: monom-0 monom-Suc)
qed
qed
interpretation poly-x-minus-y-hom: inj-idom-hom poly-x-minus-y
proof
fix $p$ :: 'a poly
assume 0: poly-x-minus-y $p=0$
then have poly-lift $p \circ_{p} x-y=0$ by (simp add: poly-x-minus-y-def)
then show $p=0$
proof (induct p)
case 0
then show ?case by simp

```
    next
        case (pCons a p)
        note p = this[unfolded poly-lift-pCons pcompose-pCons]
    show ?case
    proof (cases a=0)
        case a0: True
        with p have }x-y*\mathrm{ poly-lift p }\mp@subsup{\circ}{p}{}x-y=0\mathrm{ by simp
        then have poly-lift p op
        then show ?thesis using p by simp
    next
        case a0: False
        with p have p0:p\not=0 by auto
    from p have [:[:a:]:] = - x-y* poly-lift p op x-y by (simp add: eq-neg-iff-add-eq-0)
        then have degree [:[:a:]:] = degree (x-y* poly-lift p op
        also have ... = degree (x-y::'a poly poly) + degree (poly-lift p op x-y)
            apply (subst degree-mult-eq)
                apply simp
                apply (subst pcompose-eq-0)
                apply (simp add: x-y-def)
                apply (simp add: p0)
                apply simp
        done
        finally have False by simp
        then show ?thesis..
    qed
    qed
qed
lemma poly-add:
    fixes p q :: 'a ::comm-ring-1 poly
    assumes q0: q}\not=0\mathrm{ and x: poly p x=0 and y: poly q y=0
    shows poly (poly-add p q) (x+y) = 0
proof (unfold poly-add-def, rule poly-resultant-zero[OF disjI2])
    have degree q>0 using poly-zero q0 y by auto
    thus degq: degree (poly-lift q)>0 by auto
qed (insert x y, simp-all)
```


### 5.1.2 poly-add is nonzero

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


## interpretation poly-lift-hom:

```
unit-preserving-hom poly-lift :: 'a :: \{comm-semiring-1,semiring-no-zero-divisors \(\}\)
poly \(\Rightarrow\) -
proof
fix \(x\) :: 'a poly
assume poly-lift \(x\) dvd 1
then have poly-y-x (poly-lift \(x\) ) dvd poly- \(y\)-x 1
by \(\operatorname{simp}\)
```

```
    then show \(x\) dvd 1
    by (auto simp add: poly-y-x-poly-lift)
qed
interpretation poly-lift-hom:
    factor-preserving-hom poly-lift::'a::idom poly \(\Rightarrow\) 'a poly poly
proof unfold-locales
    fix \(p\) :: 'a poly
    assume \(p\) : irreducible \(p\)
    show irreducible (poly-lift p)
    proof (rule ccontr)
        from \(p\) have \(p 0: p \neq 0\) and \(\neg p\) dvd 1 by (auto dest: irreducible-not-unit)
        with poly-lift-hom.hom-dvd[of p1] have p1: ᄀpoly-lift pdvd 1 by auto
        assume \(\neg\) irreducible (poly-lift p)
        from this[unfolded irreducible-altdef,simplified] p0 p1
        obtain \(q\) where \(q\) dvd poly-lift \(p\) and \(p q: \neg\) poly-lift \(p d v d q\) and \(q: \neg q\) dvd 1
by auto
    then obtain \(r\) where \(q * r=\) poly-lift \(p\) by (elim dvdE, auto)
    then have poly-y-x \((q * r)=\) poly- \(y-x\) (poly-lift \(p\) ) by auto
    also have \(\ldots=[: p:]\) by (auto simp: poly- \(y\)-x-poly-lift monom-0)
    also have poly-y-x \((q * r)=\) poly- \(y-x q * p o l y-y-x r\) by (auto simp: hom-distribs)
    finally have \(\ldots=[: p:]\) by auto
    then have \(q p: p o l y-y-x ~ q ~ d v d ~[: p:]\) by (metis dvdI)
    from dvd-const[OF this] p0 have degree (poly-y-x q) \(=0\) by auto
    from degree-0-id[OF this,symmetric] obtain \(s\)
            where \(q s\) : poly- \(y-x q=[: s:]\) by auto
    have poly-lift \(s=\) poly- \(y-x\) (poly- \(y-x(\) poly-lift s) \()\) by auto
            also have \(\ldots=\) poly- \(y\)-x [:s:] by (auto simp: poly-y-x-poly-lift monom-0)
            also have \(\ldots=q\) by (auto simp: \(q s[\) symmetric \(]\) )
    finally have sq: poly-lift \(s=q\) by auto
    from \(q p\) [unfolded \(q s]\) have \(s p\) : \(s\) dvd \(p\) by (auto simp: const-poly-dvd)
    from irreducible \(D^{\prime}[O F\) p this] sq q pq show False by auto
    qed
qed
```

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 = ( \lambdap.pop x-y)\circ poly-lift
    by (intro ext, unfold poly-x-minus-y-def, auto)
context idom-isom begin
    sublocale comm-semiring-isom..
end
interpretation poly-x-minus-y-hom:
    factor-preserving-hom poly-x-minus-y :: 'a :: idom poly }=>\mp@subsup{}{}{\prime}'a poly poly
proof -
    have }\langlep\mp@subsup{\circ}{p}{}x-y\mp@subsup{\circ}{p}{}x-y=p\rangle\mathrm{ for p :: <'a poly poly>
    proof (induction p)
```

```
    case 0
    show ?case
        by simp
    next
    case (pCons a p)
    then show ?case
        by (unfold x-y-def hom-distribs pcompose-pCons) simp
    qed
    then interpret x-y-hom: bijective \lambdap :: 'a poly poly. p op x-y
    by (unfold bijective-eq-bij) (rule involuntory-imp-bij)
    interpret x-y-hom: idom-isom \lambdap :: 'a poly poly.p op x-y
    by standard simp-all
have〈factor-preserving-hom ( }\lambdap::: 'a poly poly. p op x-y)
    and <factor-preserving-hom (poly-lift :: 'a poly = 'a poly poly)>
    ..
then show factor-preserving-hom (poly-x-minus-y :: 'a poly => -)
    by (unfold poly-x-minus-y-as-comp) (rule factor-preserving-hom-comp)
qed
```

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:]:] $=[:[: a:]:]$ by (simp add: poly-y-x-def monom-0)

## context begin

private abbreviation $y-x==[:[: 0,-1:], 1:]$
lemma poly-y-x-x-y[simp]: poly-y-x x-y=y-x by (simp add: x-y-def poly-y-x-def monom-Suc monom-0)
private lemma $y$-x[simp]: fixes $x::{ }^{\prime} a$ :: comm-ring-1 shows poly2 $y$-x $x y=y$ - $x$
unfolding poly2-def by simp
private definition poly-y-minus-x $p \equiv$ poly-lift $p \circ_{p} y-x$
private lemma poly-y-minus- $x-0[$ simp $]$ : poly- $y$-minus-x $0=0$ by (simp add: poly-y-minus-x-def)
private lemma poly-y-minus-x-pCons[simp]:
poly-y-minus-x (pCons a $p$ ) $=[:[: a:]:]+$ poly-y-minus-x $p * y$-x by (simp add:
poly-y-minus-x-def)
private lemma poly-y-x-poly-x-minus-y:
fixes $p::$ ' $a$ :: idom poly
shows poly-y-x (poly-x-minus-y $p$ ) $=$ poly- $y$-minus-x $p$
apply (induct $p$, simp)
apply (unfold poly-x-minus-y-pCons hom-distribs) by simp

```
lemma degree-poly-y-minus-x [simp]:
    fixes p :: 'a :: idom poly
    shows degree (poly-y-x (poly-x-minus-y p)) = degree p
    by (simp add: poly-y-minus-x-def poly-y-x-poly-x-minus-y)
end
lemma dvd-all-coeffs-iff:
    fixes x :: 'a :: comm-semiring-1
    shows (\forallpi\in set (coeffs p). x dvd pi)\longleftrightarrow(\foralli.x dvd coeff p i)(is ?l = ?r)
proof-
    have ?r = (\forall i\in{..degree p}\cup{Suc (degree p)..}. x dvd coeff p i) by auto
    also have ... = ( }\foralli\leq\mathrm{ degree p. x dvd coeff p i) by (auto simp add: ball-Un co-
eff-eq-0)
    also have ... = ?l by (auto simp: coeffs-def)
    finally show ?thesis..
qed
lemma primitive-imp-no-constant-factor:
    fixes p :: 'a :: {comm-semiring-1, semiring-no-zero-divisors} poly
    assumes pr: primitive p and F:mset-factors F p and fF:f\in#F
    shows degree f}\not=
proof
    from FfF have irr: irreducible f and fp: fdvd p by (auto dest: mset-factors-imp-dvd)
    assume deg: degree f=0
    then obtain f0 where f0:f=[:f0:] by (auto dest:degree0-coeffs)
    with fp have [:f0:] dvd p by simp
    then have f0 dvd coeff pi for i by (simp add: const-poly-dvd-iff)
    with primitiveD[OF pr] dvd-all-coeffs-iff have f0 dvd 1 by (auto simp: coeffs-def)
    with f0 irr show False by auto
qed
lemma coprime-poly-x-minus-y-poly-lift:
    fixes p q :: ' }a\mathrm{ :: 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(rule ccontr)
    from degp have p:\negp dvd 1 by (auto simp:dvd-const)
    from degp have p0: p\not=0 by auto
    from mset-factors-exist[of p, OF p0 p]
    obtain F where F: mset-factors F p by auto
    with poly-x-minus-y-hom.hom-mset-factors
    have pF: mset-factors (image-mset poly-x-minus-y F) (poly-x-minus-y p) by auto
    from degq have q:\negq dvd 1 by (auto simp:dvd-const)
    from }\operatorname{leg}q\mathrm{ have q0:q}q=0\mathrm{ by auto
    from mset-factors-exist[OF q0 q]
    obtain G where G: mset-factors G q by auto
```

with poly-lift-hom.hom-mset-factors
have $p G$ : mset-factors (image-mset poly-lift $G$ ) (poly-lift q) by auto

```
assume \neg coprime (poly-x-minus-y p)(poly-lift q)
from this[unfolded not-coprime-iff-common-factor]
obtain r
where rp: r dvd (poly-x-minus-y p)
    and rq: r dvd (poly-lift q)
    and rU:\negr dvd 1 by auto note poly-lift-hom.hom-dvd
from rp p0 have r0:r\not=0 by auto
from mset-factors-exist[OF r0 rU]
obtain H where H: mset-factors H r by auto
then have H}={#}\mathrm{ by auto
then obtain h}\mathrm{ where hH:h 斩
with H mset-factors-imp-dvd have hr: hdvdr and h: irreducible h by auto
from irreducible-not-unit[OF h] have hU: \neg h dvd 1 by auto
from hr rp have h dvd (poly-x-minus-y p) by (rule dvd-trans)
from irreducible-dvd-imp-factor[OF this h pF] p0
obtain f}\mathrm{ where f:f}\in#F\mathrm{ and fh: poly-x-minus-y f ddvd h by auto
from hr rq have h dvd (poly-lift q) by (rule dvd-trans)
from irreducible-dvd-imp-factor[OF this h pG] q0
obtain g}\mathrm{ where g:g}\in#G\mathrm{ and gh: poly-lift g ddvd h by auto
from fh gh have poly-x-minus-y f ddvd poly-lift g using ddvd-trans by auto
then have poly-y-x (poly-x-minus-y f) ddvd poly-y-x (poly-lift g) by simp
also have poly-y-x (poly-lift g) =[:g:] unfolding poly-y-x-poly-lift monom-0 by
auto
    finally have ddvd: poly-y-x (poly-x-minus-y f) ddvd [:g:] by auto
    then have degree (poly-y-x (poly-x-minus-y f)) = 0 by (metis degree-pCons-0
dvd-0-left-iff dvd-const)
    then have degree f=0 by simp
    with primitive-imp-no-constant-factor[OF pr F f] show False by auto
qed
lemma poly-add-nonzero:
    fixes p q :: ' }a\mathrm{ :: ufd poly
    assumes p0:p\not=0 and q0:q\not=0 and x: poly px=0 and y: poly q y=0
        and pr: primitive p
    shows poly-add p q\not=0
proof
    have degp: degree p>0 using le-0-eq order-degree order-root p0 x by (metis
grOI)
    have degq: degree q>0 using le-0-eq order-degree order-root q0 y by (metis
gr0I)
    assume 0: poly-add p q=0
    from resultant-zero-imp-common-factor[OF - this[unfolded poly-add-def]] degp
        and coprime-poly-x-minus-y-poly-lift[OF degp degq pr]
    show False by auto
qed
```


### 5.1.3 Summary for addition

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

```
lemma (in comm-ring-hom) map-poly-x-minus-y:
    map-poly (map-poly hom) (poly-x-minus-y \(p\) ) \(=\) poly-x-minus-y (map-poly hom \(p\) )
proof -
    interpret mp: map-poly-comm-ring-hom hom..
    interpret mmp: map-poly-comm-ring-hom map-poly hom..
    show ?thesis
        apply (induct \(p\), simp)
        apply (unfold \(x\) - \(y\)-def hom-distribs poly-x-minus-y-pCons, simp) done
qed
lemma (in comm-ring-hom) hom-poly-lift[simp]:
    map-poly (map-poly hom) (poly-lift \(q)=\) poly-lift \((\) map-poly hom \(q)\)
proof -
    show ?thesis
        unfolding poly-lift-def
        unfolding map-poly-map-poly[of coeff-lift,OF coeff-lift-hom.hom-zero]
        unfolding map-poly-coeff-lift-hom by simp
qed
```

lemma lead-coeff-poly-x-minus-y:
fixes $p::$ ' $a::$ idom poly
shows lead-coeff (poly-x-minus-y $p)=[$ :lead-coeff $p *((-1)$ ^degree $p)$ :] (is ?l
$=? r)$
proof-
have $? l=$ Polynomial.smult (lead-coeff $p)((-1)$ ^ degree $p)$
by (unfold poly-x-minus- $y$-def, subst lead-coeff-comp; simp add: $x$ - $y$-def)
also have $\ldots=? r$ by (unfold hom-distribs, simp add: smult-as-map-poly[symmetric])
finally show ?thesis.
qed
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 -
consider $i=$ degree $p \mid i>$ degree $p \mid i<$ degree $p$
by force
thus ?thesis
proof cases
assume $i>$ degree $p$
thus ?thesis by (subst coeff-eq-0) auto
next
assume $i=$ degree $p$
thus ?thesis using lead-coeff-poly-x-minus-y[of p]
by (simp add: lead-coeff-poly-x-minus-y)

```
    next
        assume i< degree p
    define }n\mathrm{ where }n=\mathrm{ degree }
    have degree (coeff (poly-x-minus-y p) i) =
            degree (\sumj\leqn. [:coeff p j:] * coeff (x-y^ j) i) (is - = degree (sum ?f -))
        by (simp add: poly-x-minus-y-def pcompose-conv-poly poly-altdef coeff-sum
n-def)
    also have {..n}= insert n{..<n}
        by auto
    also have sum ?f ... = ?f n + sum ?f {..<n}
        by (subst sum.insert) auto
    also have degree ... = n-i
    proof -
        have degree (?f n) = n-i
            using <i < degree p> by (simp add: n-def coeff-xy-power degree-monom-eq)
        moreover have degree (sum ?f {..<n})<n-i
        proof (intro degree-sum-smaller)
            fix j assume j { {..<n}
            have degree ([:coeff p j:]* coeff (x-y^ j) i)\leqj-i
            proof (cases i\leqj)
                case True
                thus ?thesis
                    by (auto simp: n-def coeff-xy-power degree-monom-eq)
            next
                case False
                hence coeff (x-y ^ j :: 'a poly poly) i=0
                    by (subst coeff-eq-0) (auto simp: degree-power-eq)
                    thus ?thesis by simp
            qed
            also have ... < n-i
                using <j \in{..<n}><i< degree p> by (auto simp: n-def)
            finally show degree ([:coeff p j:] * coeff (x-y ^ j) i)<n-i .
            qed (use <i< degree p> in <auto simp: n-def>)
            ultimately show ?thesis
            by (subst degree-add-eq-left) auto
    qed
    finally show ?thesis
        by (simp add: n-def)
    qed
qed
lemma coeff-0-poly-x-minus-y [simp]: coeff (poly-x-minus-y p) 0 = p
    by (induction p) (auto simp: poly-x-minus-y-def x-y-def)
lemma (in idom-hom) poly-add-hom:
    assumes p0: hom (lead-coeff p)}\not=0\mathrm{ and q0: hom (lead-coeff q)}\not=
    shows map-poly hom (poly-add p q) = poly-add (map-poly hom p) (map-poly hom
q)
proof -
```

```
    interpret mh: map-poly-idom-hom..
    show ?thesis unfolding poly-add-def
    apply (subst mh.resultant-map-poly(1)[symmetric])
        apply (subst degree-map-poly-2)
        apply (unfold lead-coeff-poly-x-minus-y, unfold hom-distribs, simp add: p0)
        apply simp
    apply (subst degree-map-poly-2)
        apply (simp-all add: q0 map-poly-x-minus-y)
    done
qed
lemma(in zero-hom) hom-lead-coeff-nonzero-imp-map-poly-hom:
    assumes hom (lead-coeff p)}\not=
    shows map-poly hom p}\not=
proof
    assume map-poly hom p=0
    then have coeff (map-poly hom p) (degree p)=0 by simp
    with assms show False by simp
qed
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)
\not=0
            and x: ipoly p x = 0 and y: ipoly q y = 0
    shows ipoly (poly-add p q) (x+y) =0
    using assms of-int-hom.hom-lead-coeff-nonzero-imp-map-poly-hom[OF q0]
    by (auto intro: poly-add simp: of-int-hom.poly-add-hom[OF p0 q0])
lemma (in comm-monoid-gcd) gcd-list-eq-O-iff[simp]: listgcd xs = 0 \longleftrightarrow(\forallx\in
set xs. }x=0\mathrm{ )
    by (induct xs,auto)
lemma primitive-field-poly[simp]: primitive ( }p::='a :: field poly) \longleftrightarrowp\not=
    by (unfold primitive-iff-some-content-dvd-1,auto simp: dvd-field-iff coeffs-def)
lemma ipoly-poly-add-nonzero:
    fixes x y :: 'a :: field
    assumes }p\not=0\mathrm{ and q}\not=0\mathrm{ and ipoly p x=0 and ipoly q y=0
        and (of-int (lead-coeff p):: 'a)}\not=0\mathrm{ and (of-int (lead-coeff q) :: ' }a)\not=
    shows poly-add p q\not=0
proof-
    from assms have (of-int-poly (poly-add p q) :: 'a poly) }=
    apply (subst of-int-hom.poly-add-hom,simp,simp)
    by (rule poly-add-nonzero, auto dest:of-int-hom.hom-lead-coeff-nonzero-imp-map-poly-hom)
    then show ?thesis by auto
qed
lemma represents-add:
```

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

### 5.2 Division of Algebraic Numbers

```
definition poly-x-mult-y where
    [code del]: poly-x-mult-y \(p \equiv\left(\sum i \leq\right.\) degree \(p\). monom (monom (coeff pi) 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 (cases degree \(p<i\) )
    case \(i\) : False
    have ?l \(=\operatorname{sum}(\lambda j\). if \(j=i\) then (monom (coeff \(p j) j\) ) else 0) \(\{\).. degree \(p\}\)
    (is - = sum ?f ?A) by (simp add: poly-x-mult- \(y\)-def coeff-sum)
    also have \(\ldots=\) sum ?f \(\{i\}\) using \(i\) by (intro sum.mono-neutral-right, auto)
    also have \(\ldots=\) ?f \(i\) by simp
    also have \(\ldots=\) ? \(r\) by auto
    finally show ?thesis.
next
    case True then show ?thesis by (auto simp: poly-x-mult-y-def coeff-eq-0 co-
eff-sum)
qed
lemma poly-x-mult-y-code[code]: poly-x-mult-y \(p=(\) let cs \(=\) coeffs \(p\)
    in poly-of-list (map ( \(\lambda(i\), ai). monom ai \(i)(z i p[0 \quad . .<\) length \(c s] c s)))\)
    unfolding Let-def poly-of-list-def
proof (rule poly-eqI, unfold coeff-poly-x-mult-y)
    fix \(n\)
    let ?xs \(=z i p[0 . .<\) length \((\) coeffs \(p)](\) coeffs \(p)\)
    let ?f \(=(\lambda(i, a i)\). monom ai \(i)\)
    show monom (coeff \(p n\) ) \(n=\operatorname{coeff}(\) Poly (map ?f ? \(x s)\) ) \(n\)
    proof (cases \(n<\) length (coeffs \(p\) ))
        case True
        hence \(n: n<\) length (map ?f ?xs) and \(n n: n<\) length ?xs
            unfolding degree-eq-length-coeffs by auto
        show ?thesis unfolding coeff-Poly nth-default-nth[OF n] nth-map[OF nn]
            using True by (simp add: nth-coeffs-coeff)
    next
        case False
        hence id: coeff (Poly (map ?f ?xs)) \(n=0\) unfolding coeff-Poly
            by (subst nth-default-beyond, auto)
        from False have \(n>\) degree \(p \vee p=0\) unfolding degree-eq-length-coeffs by
(cases n, auto)
    hence monom (coeff \(p n\) ) \(n=0\) using coeff-eq- \(0[\) of \(p n]\) by auto
    thus ?thesis unfolding id by simp
    qed
qed
```

definition poly-div $::$ ' $a$ :: comm-ring-1 poly $\Rightarrow$ ' $a$ poly $\Rightarrow$ 'a poly where
poly-div p $q=$ resultant $($ poly-x-mult-y $p)($ poly-lift $q)$
poly-div has desired roots.
lemma poly2-poly-x-mult-y:
fixes $p::$ ' $a$ :: comm-ring-1 poly
shows poly2 (poly-x-mult-y p) x $y=$ poly $p(x * y)$
apply (subst(3) poly-as-sum-of-monoms[symmetric])
apply (unfold poly-x-mult-y-def hom-distribs)
by (auto simp: poly2-monom poly-monom power-mult-distrib ac-simps)
lemma poly-div:
fixes $p q::$ ' $a$ ::field poly
assumes $q 0: q \neq 0$ and $x:$ poly $p x=0$ and $y:$ poly $q y=0$ and $y 0: y \neq 0$
shows poly (poly-div $p$ q) $(x / y)=0$
proof (unfold poly-div-def, rule poly-resultant-zero[OF disjI2])
have degree $q>0$ using poly-zero $q 0 y$ by auto
thus degq: degree (poly-lift $q$ ) $>0$ by auto
qed (insert $x$ y y0, simp-all add: poly2-poly-x-mult-y)
poly-div is nonzero.
interpretation poly-x-mult-y-hom: ring-hom poly-x-mult-y :: ' $a::\{$ idom,ring-char- 0$\}$
poly $\Rightarrow$ -
by (unfold-locales, auto intro: poly2-ext simp: poly2-poly-x-mult-y hom-distribs)
interpretation poly-x-mult-y-hom: inj-ring-hom poly-x-mult-y :: ' $a::\{$ idom,ring-char-0 $\}$
poly $\Rightarrow$ -
proof
let $? h=$ poly- $x-m u l t-y$
fix $f$ :: 'a poly
assume ?h $f=0$
then have poly $2(? h f) x 1=0$ for $x$ by simp
from this[unfolded poly2-poly-x-mult-y]
show $f=0$ by auto
qed
lemma degree-poly-x-mult-y[simp]:
fixes $p::$ ' $a::\{$ idom, ring-char-0 $\}$ poly
shows degree (poly-x-mult-y $p$ ) $=$ degree $p(i s ? l=? r)$
proof (rule antisym)
show ?r $\leq$ ?l by (cases $p=0$, auto intro: le-degree simp: coeff-poly-x-mult-y)
show ?l $\leq$ ?r unfolding poly-x-mult-y-def
by (auto intro: degree-sum-le le-trans[OF degree-monom-le])
qed
interpretation poly-x-mult-y-hom: unit-preserving-hom poly-x-mult-y :: 'a :: field-char-0
poly $\Rightarrow$ -
proof (unfold-locales)
let $? h=$ poly-x-mult-y :: 'a poly $\Rightarrow$ -
fix $f$ :: 'a poly
assume unit: ?h $f$ dvd 1
then have degree $(? h f)=0$ and coeff $(? h f) 0$ dvd 1 unfolding poly-dvd- 1 by auto
then have deg: degree $f=0$ by (auto simp add: degree-monom-eq)
with unit show $f$ dvd 1 by (cases $f=0$, auto)
qed
lemmas poly-y-x-o-poly-lift $=o$-def[of poly-y-x poly-lift, unfolded poly-y-x-poly-lift]
lemma irreducible-dvd-degree: assumes ( $f:::^{\prime a}::$ field poly) dvd $g$
irreducible $g$
degree $f>0$
shows degree $f=$ degree $g$
using assms
by (metis irreducible-altdef degree-0 dvd-refl is-unit-field-poly linorder-neqE-nat
poly-divides-conv0)
lemma coprime-poly-x-mult-y-poly-lift:
fixes $p q$ :: ' $a$ :: field-char-0 poly
assumes degp: degree $p>0$ and degq: degree $q>0$
and nz: poly p $0 \neq 0 \vee$ poly q $0 \neq 0$
shows coprime (poly-x-mult-y p) (poly-lift q)
proof (rule ccontr)
from degp have $p: \neg p d v d 1$ by (auto simp: dvd-const)
from degp have $p 0: p \neq 0$ by auto
from mset-factors-exist $[o f ~ p, O F p 0 p]$
obtain $F$ where $F$ : mset-factors $F p$ by auto
then have $p F$ : prod-mset (image-mset poly-x-mult-y $F$ ) $=$ poly-x-mult-y $p$ by (auto simp: hom-distribs)
from $\operatorname{deg} q$ have $q$ : $\neg$ is-unit $q$ by (auto simp: dvd-const)
from $\operatorname{deg} q$ have $q 0: q \neq 0$ by auto
from mset-factors-exist[OF q0 q]
obtain $G$ where $G$ : mset-factors $G q$ by auto
with poly-lift-hom.hom-mset-factors
have $p G$ : mset-factors (image-mset poly-lift $G$ ) (poly-lift q) by auto
from poly-y-x-hom.hom-mset-factors[OF this]
have $p G$ : mset-factors (image-mset coeff-lift $G$ ) [:q:]
by (auto simp: poly-y-x-poly-lift monom-0 image-mset.compositionality poly-y-x-o-poly-lift)
assume $\neg$ coprime $($ poly-x-mult-y $p)($ poly-lift $q)$
then have $\neg$ coprime $($ poly- $y-x($ poly-x-mult-y $p))($ poly- $y-x($ poly-lift $q))$
by (simp del: coprime-iff-coprime)
from this[unfolded not-coprime-iff-common-factor]
obtain $r$
where rp: $r$ dvd poly- $y$-x (poly-x-mult-y $p$ )
and rq: $r$ dvd poly- $y-x$ (poly-lift $q$ )
and $r U$ : $\neg r$ dvd 1 by auto
from $r p p 0$ have $r 0: r \neq 0$ by auto
from mset-factors-exist[OF r0 rU]
obtain $H$ where $H$ : mset-factors $H r$ by auto
then have $H \neq\{\#\}$ by auto
then obtain $h$ where $h H: h \in \# H$ by fastforce
with $H$ mset-factors-imp-dvd have $h r: h d v d r$ and $h$ : irreducible $h$ by auto
from irreducible-not-unit $[O F h]$ have $h U: \neg h$ dvd 1 by auto
from $h r r p$ have $h$ dvd poly- $y$-x (poly-x-mult-y $p$ ) by (rule dvd-trans)
note this[folded $p F$, unfolded poly-y-x-hom.hom-prod-mset image-mset.compositionality]
from prime-elem-dvd-prod-mset[OF h[folded prime-elem-iff-irreducible] this]
obtain $f$ where $f: f \in \# F$ and $h f: h$ dvd poly- $y-x$ (poly-x-mult-y $f$ ) by auto
have irr $F$ : irreducible $f$ using $f F$ by blast
from dvd-trans[OF hr rq] have $h$ dvd [:q:] by (simp add: poly-y-x-poly-lift monom-0)
from irreducible-dvd-imp-factor $[O F$ this $h p G] q 0$
obtain $g$ where $g: g \in \# G$ and $g h:[: g:] d v d h$ by auto
from dvd-trans[OF gh hf] have $*:[: g:]$ dvd poly-y-x (poly-x-mult-y $f$ ) using dvd-trans by auto
show False
proof (cases poly f $0=0$ )
case $f$ - 0 : False
from poly-hom.hom-dvd[OF *]
have $g$ dvd poly (poly-y-x (poly-x-mult-y f)) [:0:] by simp
also have $\ldots=[$ :poly $f 0:]$ by (intro poly-ext, fold poly2-def, simp add: poly2-poly-x-mult-y)
also have ... dvd 1 using $f-0$ by auto
finally have $g d v d 1$.
with $g$ G show False by (auto elim!: mset-factorsE dest!: irreducible-not-unit)
next
case True
hence $[: 0,1:]$ dvd $f$ by (unfold dvd-iff-poly-eq- 0 , simp)
from irreducible-dvd-degree[OF this irrF]
have degree $f=1$ by auto
from degree1-coeffs[OF this] True obtain $c$ where $c: c \neq 0$ and $f: f=[: 0, c:]$
by auto
from $g G$ have irr $G$ : irreducible $g$ by auto
from poly-hom.hom-dvd[OF *]
have $g$ dvd poly (poly-y-x (poly-x-mult-y f)) 1 by simp
also have $\ldots=f$ by (auto simp: $f$ poly-x-mult- $y$-code Let-def c poly- $y-x-p$ Cons map-poly-monom poly-monom poly-lift-def)
also have $\ldots$ dvd [:0, $1:]$ unfolding $f d v d$-def using $c$
by (intro exI[of - [: inverse c:]], auto)
finally have $g 01: g$ dvd [:0,1:].
from divides-degree[OF this] $\operatorname{irr} G$ have degree $g=1$ by auto
from degree1-coeffs [OF this] obtain $a b$ where $g: g=[: b, a:]$ and $a: a \neq 0$ by auto
from $g 01$ [unfolded dvd-def] $g$ obtain $k$ where $i d:[: 0,1:]=g * k$ by auto
from id have $0: g \neq 0 k \neq 0$ by auto
from arg-cong[OF id, of degree $]$ have degree $k=0$ unfolding degree-mult-eq $[O F$

```
0]
            unfolding g using a by auto
    from degree0-coeffs[OF this] obtain kk where k: k= [:kk:] by auto
    from id[unfolded g k] a have b=0 by auto
    hence poly g 0 = 0 by (auto simp: g)
    from True this nz\langlef\in#F\rangle\langleg\in#G\rangleFG
    show False by (auto dest!:mset-factors-imp-dvd elim:dvdE)
    qed
qed
lemma poly-div-nonzero:
    fixes p q :: 'a :: field-char-0 poly
    assumes p0:p\not=0 and q0:q\not=0 and x: poly px=0 and y: poly q y=0
            and p-0: poly p 0}=0\vee\mathrm{ poly q 0}=
    shows poly-div p q}\not=
proof
    have degp: degree p>0 using le-0-eq order-degree order-root p0 x by (metis
grOI)
    have degq: degree q > 0 using le-0-eq order-degree order-root q0 y by (metis
gr0I)
    assume 0: poly-div p q=0
    from resultant-zero-imp-common-factor[OF - this[unfolded poly-div-def]] degp
    and coprime-poly-x-mult-y-poly-lift[OF degp degq] p-0
    show False by auto
qed
```


### 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 -
interpret mh: map-poly-inj-comm-ring-hom..
interpret mmh: map-poly-inj-comm-ring-hom map-poly hom..
show ?thesis unfolding poly-x-mult-y-def by (simp add: hom-distribs)
qed
lemma (in inj-comm-ring-hom) poly-div-hom:
map-poly hom (poly-div $p$ q) $=$ poly-div (map-poly hom $p)($ map-poly hom $q)$
proof -
have zero: $\forall x$. hom $x=0 \longrightarrow x=0$ by simp
interpret mh: map-poly-inj-comm-ring-hom..
show ?thesis unfolding poly-div-def mh.resultant-hom[symmetric]
by (simp add: poly-x-mult-y-hom)
qed
lemma ipoly-poly-div:
fixes $x$ y :: ' $a$ :: field-char-0

```
    assumes q\not=0 and ipoly px=0 and ipoly q y=0 and y\not=0
    shows ipoly (poly-div p q) (x/y)=0
    by (unfold of-int-hom.poly-div-hom, rule poly-div, insert assms, auto)
lemma ipoly-poly-div-nonzero:
    fixes x y :: 'a :: field-char-0
    assumes p\not=0 and q\not=0 and ipoly p x=0 and ipoly q y=0 and poly p 0
# 0\vee poly q 0}\not=
    shows poly-div p q\not=0
proof -
    from assms have (of-int-poly (poly-div p q) :: 'a poly) = 0 using of-int-hom.poly-map-poly[of
p]
    by (subst of-int-hom.poly-div-hom, subst poly-div-nonzero, auto)
    then show ?thesis by auto
qed
lemma represents-div:
    fixes x y :: 'a :: field-char-0
    assumes p represents x and q represents y and poly q 0}=
    shows (poly-div p q) represents ( }x/y\mathrm{ )
    using assms by (intro representsI ipoly-poly-div ipoly-poly-div-nonzero, auto)
```


### 5.3 Multiplication of Algebraic Numbers

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

### 5.4 Summary: Closure Properties of Algebraic Numbers

lemma algebraic-representsI: $p$ represents $x \Longrightarrow$ algebraic $x$ unfolding represents-def algebraic-altdef-ipoly by auto
lemma algebraic-of-rat: algebraic (of-rat x)
by (rule algebraic-representsI[OF poly-rat-represents-of-rat])
lemma algebraic-uminus: algebraic $x \Longrightarrow$ algebraic $(-x)$
by (auto dest: algebraic-imp-represents-irreducible intro: algebraic-representsI rep-
resents-uminus)
lemma algebraic-inverse: algebraic $x \Longrightarrow$ algebraic (inverse $x$ )

```
    using algebraic-of-rat[of 0]
    by (cases x = 0, auto dest: algebraic-imp-represents-irreducible intro: algebraic-representsI
represents-inverse)
lemma algebraic-plus: algebraic }x\Longrightarrow\mathrm{ algebraic }y\Longrightarrow\mathrm{ algebraic ( }x+y\mathrm{ )
    by (auto dest!: algebraic-imp-represents-irreducible-cf-pos intro!: algebraic-representsI[OF
represents-add])
lemma algebraic-div:
    assumes x: algebraic x and y:algebraic y shows algebraic (x/y)
proof(cases y=0\vee (x=0)
    case True
    then show ?thesis using algebraic-of-rat[of 0] by auto
next
    case False
    then have x0:x\not=0 and y0:y\not=0 by auto
    from x y obtain pq
    where px: p represents x and irr: irreducible q and qy: q represents y
    by (auto dest!: algebraic-imp-represents-irreducible)
    show ?thesis
    using False px represents-irr-non-0[OF irr qy]
    by (auto intro!: algebraic-representsI[OF represents-div] qy)
qed
lemma algebraic-times: algebraic x \Longrightarrow algebraic y \Longrightarrow algebraic (x*y)
    using algebraic-div[OF - algebraic-inverse, of x y] by (simp add: field-simps)
lemma algebraic-root: algebraic x \Longrightarrow algebraic (root n x)
proof -
    assume algebraic x
    then obtain p}\mathrm{ where p: p represents x by (auto dest: algebraic-imp-represents-irreducible-cf-pos)
    from
        algebraic-representsI[OF represents-nth-root-neg-real[OF - this, of n]]
        algebraic-representsI[OF represents-nth-root-pos-real[OF - this, of n].
        algebraic-of-rat[of 0]
    show ?thesis by (cases n=0,force, cases n>0,force, cases n<0,auto)
qed
lemma algebraic-nth-root: n}\not=0\Longrightarrow\mathrm{ algebraic }x\Longrightarrow\\widehat{ n}=x\Longrightarrow\mathrm{ algebraic y
    by (auto dest: algebraic-imp-represents-irreducible-cf-pos intro: algebraic-representsI
represents-nth-root)
```


### 5.5 More on algebraic integers

definition poly-add-sign :: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a::$ comm-ring-1 where
poly-add-sign $m n=\operatorname{signof}(\lambda i$. if $i<n$ then $m+i$ else if $i<m+n$ then $i-$
$n$ else $i$ )
lemma lead-coeff-poly-add:
fixes $p$ : : ' $a::\{$ idom, semiring-char-0 $\}$ poly
defines $m \equiv$ degree $p$ and $n \equiv$ degree $q$
assumes lead-coeff $p=1$ lead-coeff $q=1 m>0 n>0$
shows lead-coeff (poly-add p $q$ :: 'a poly) $=$ poly-add-sign $m n$
proof -
from assms have $[$ simp $]: p \neq 0 q \neq 0$
by auto
define $M$ where $M=$ sylvester-mat (poly-x-minus-y $p$ ) (poly-lift $q$ )
define $\pi$ :: nat $\Rightarrow$ nat where

$$
\pi=(\lambda i \text {. if } i<n \text { then } m+i \text { else if } i<m+n \text { then } i-n \text { else } i)
$$

have $\pi: \pi$ permutes $\{0 . .<m+n\}$
by (rule inj-on-nat-permutes) (auto simp: $\pi$-def inj-on-def)
have $n z: M \$ \$(i, \pi i) \neq 0$ if $i<m+n$ for $i$
using that by (auto simp: M-def $\pi$-def sylvester-index-mat m-def $n$-def)
have indices-eq: $\{0 . .<m+n\}=\{. .<n\} \cup(+) n^{\prime}\{. .<m\}$
by (auto simp flip: atLeast0LessThan)
define $f$ where $f=\left(\lambda \sigma\right.$. signof $\left.\sigma *\left(\prod i=0 . .<m+n . M \$ \$(i, \sigma i)\right)\right)$
have degree $(f \pi)=$ degree $\left(\prod i=0 . .<m+n . M \$ \$(i, \pi i)\right)$
using $n z$ by (auto simp: f-def degree-mult-eq sign-def)
also have $\ldots=\left(\sum i=0 . .<m+n\right.$. degree $\left.(M \$ \$(i, \pi i))\right)$
using $n z$ by (subst degree-prod-eq-sum-degree) auto
also have $\ldots=\left(\sum i<n\right.$. degree $\left.(M \$ \$(i, \pi i))\right)+\left(\sum i<m\right.$. degree $(M \$ \$(n$ $+i, \pi(n+i))))$
by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
also have $\left(\sum i<n\right.$. degree $\left.(M \$ \$(i, \pi i))\right)=\left(\sum i<n . m\right)$
by (intro sum.cong) (auto simp: $M$-def sylvester-index-mat $\pi$-def m-def $n$-def)
also have $\left(\sum i<m\right.$. degree $\left.(M \$ \$(n+i, \pi(n+i)))\right)=\left(\sum i<m .0\right)$
by (intro sum.cong) (auto simp: M-def sylvester-index-mat $\pi$-def m-def $n$-def)
finally have deg-f1: degree $(f \pi)=m * n$
by $\operatorname{simp}$
have deg-f2: degree $(f \sigma)<m * n$ if $\sigma$ permutes $\{0 . .<m+n\} \sigma \neq \pi$ for $\sigma$
proof (cases $\exists i \in\{0 . .<m+n\} . M \$ \$(i, \sigma i)=0)$
case True
hence $*:\left(\prod i=0 . .<m+n . M \$ \$(i, \sigma i)\right)=0$
by auto
show ?thesis using $\langle m>0\rangle\langle n>0\rangle$
by (simp add: $f$-def *)
next
case False
note $n z=$ this
from that have $\sigma$-less: $\sigma i<m+n$ if $i<m+n$ for $i$ using permutes-in-image $[O F$ < $\sigma$ permutes -$\rangle]$ that by auto
have degree $(f \sigma)=$ degree $\left(\prod i=0 . .<m+n . M \$ \$(i, \sigma i)\right)$ using $n z$ by (auto simp: f-def degree-mult-eq sign-def)

```
    also have \(\ldots=\left(\sum i=0 . .<m+n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)\)
        using \(n z\) by (subst degree-prod-eq-sum-degree) auto
    also have \(\ldots=\left(\sum i<n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)+\left(\sum i<m\right.\). degree \((M \$ \$(n\)
\(+i, \sigma(n+i))))\)
        by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
    also have \(\left(\sum i<m\right.\). degree \(\left.(M \$ \$(n+i, \sigma(n+i)))\right)=\left(\sum i<m\right.\). 0\()\)
        using \(\sigma\)-less by (intro sum.cong) (auto simp: M-def sylvester-index-mat \(\pi\)-def
\(m\)-def \(n\)-def)
    also have \(\left(\sum i<n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)<\left(\sum i<n . m\right)\)
    proof (rule sum-strict-mono-ex1)
        show \(\forall x \in\{. .<n\}\). degree \((M \$ \$(x, \sigma x)) \leq m\) using \(\sigma\)-less
        by (auto simp: \(M\)-def sylvester-index-mat \(\pi\)-def \(m\)-def \(n\)-def degree-coeff-poly-x-minus-y)
    next
    have \(\exists i<n . \sigma i \neq \pi i\)
    proof (rule ccontr)
        assume nex: \(\sim(\exists i<n . \sigma i \neq \pi i)\)
        have \(\forall i \geq m+n-k . \sigma i=\pi i\) if \(k \leq m\) for \(k\)
        using that
        proof (induction \(k\) )
        case 0
        thus ? case using « \(\pi\) permutes \(\rightarrow\langle\sigma\) permutes \(\rightarrow\)
            by (fastforce simp: permutes-def)
        next
        case (Suc \(k\) )
        have \(I H: \sigma i=\pi i\) if \(i \geq m+n-k\) for \(i\)
            using Suc.prems Suc.IH that by auto
        from \(n z\) have \(M \$ \$(m+n-\) Suc \(k, \sigma(m+n-\) Suc \(k)) \neq 0\)
                using Suc.prems by auto
        moreover have \(m+n-\) Suc \(k \geq n\)
                using Suc.prems by auto
            ultimately have \(\sigma(m+n-\) Suc \(k) \geq m-\) Suc \(k\)
                using assms \(\sigma\)-less[of \(m+n-\) Suc \(k\) ] Suc.prems
                by (auto simp: \(M\)-def sylvester-index-mat \(m\)-def \(n\)-def split: \(i f\)-splits)
        have \(\neg(\sigma(m+n-\) Suc \(k)>m-\) Suc \(k)\)
        proof
            assume *: \(\sigma(m+n-\) Suc \(k)>m-\) Suc \(k\)
            have less: \(\sigma(m+n-\) Suc \(k)<m\)
            proof (rule ccontr)
                    assume \(*: \neg \sigma(m+n-\) Suc \(k)<m\)
                    define \(j\) where \(j=\sigma(m+n-\) Suc \(k)-m\)
                    have \(\sigma(m+n-\) Suc \(k)=m+j\)
                        using * by (simp add: \(j\)-def)
                    moreover \{
                        have \(j<n\)
                        using \(\sigma\)-less[of \(m+n-\) Suc \(k]\langle m>0\rangle\langle n>0\rangle\) by (simp add: \(j\)-def)
                    hence \(\sigma j=\pi j\)
                        using nex by auto
                    with \(\langle j<n\rangle\) have \(\sigma j=m+j\)
```

```
        by (auto simp: \(\pi\)-def)
    \}
    ultimately have \(\sigma(m+n-S u c k)=\sigma j\)
        by \(\operatorname{simp}\)
    hence \(m+n-\) Suc \(k=j\)
        using permutes-inj[OF < \(\sigma\) permutes -\(\rangle\) ] unfolding inj-def by blast
    thus False using \(\langle n \leq m+n-\) Suc \(k\rangle \sigma\)-less \([\) of \(m+n-\) Suc \(k]\langle n>\)
by linarith
show ?case
proof safe
fix \(i::\) nat
assume \(i: i \geq m+n-S u c k\)
show \(\sigma i=\pi i\)
using eq Suc.prems \(\langle m>0\rangle\) IH \(i\)
proof (cases \(i=m+n-S u c k\) )
case True
thus ?thesis using eq Suc.prems \(\langle m>0\) 〉
by (auto simp: \(\pi\)-def)
qed (use IH \(i\) in auto)
qed
qed
from this [of \(m\) ] and nex have \(\sigma i=\pi i\) for \(i\)
by (cases \(i \geq n\) ) auto
```

```
        hence }\sigma=\pi\mathrm{ by force
        thus False using < }\sigma\not=\pi\rangle\mathrm{ by contradiction
    qed
    then obtain i where i:i<n\sigmai\not=\pii
        by auto
    have \sigmai<m+n
        using i by (intro \sigma-less) auto
    moreover have \pi}i=m+
    using i by (auto simp: \pi-def)
    ultimately have degree ( M $$ (i,\sigma i)) < m using i<m>0\rangle
    by (auto simp: M-def m-def n-def sylvester-index-mat degree-coeff-poly-x-minus-y)
    thus }\existsi\in{..<n}.degree (M$$(i,\sigmai))<
        using i by blast
    qed auto
    finally show degree (f \sigma)<m*n
    by (simp add: mult-ac)
qed
have lead-coeff (f \pi) = poly-add-sign m n
proof -
    have lead-coeff (f \pi)= signof }\pi*(\prodi=0..<m+n.lead-coeff (M $$ (i,
i)))
    by (simp add: f-def sign-def lead-coeff-prod)
    also have (\prodi=0..<m+n. lead-coeff (M$$ (i,\pii)))=
        (\prodi<n.lead-coeff (M $$ (i,\pii)))*(\prodi<m.lead-coeff (M $$ (n+
i,\pi(n+i))))
            by (subst indices-eq, subst prod.union-disjoint) (auto simp: prod.reindex)
    also have (\prodi<n. lead-coeff (M $$ (i,\pii))) = (\prodi<n.lead-coeff p)
        by (intro prod.cong) (auto simp: M-def m-def n-def \pi-def sylvester-index-mat)
    also have (\prodi<m. lead-coeff (M$$ (n+i,\pi(n+i))))=(\prodi<m. lead-coeff
q)
    by (intro prod.cong) (auto simp: M-def m-def n-def \pi-def sylvester-index-mat)
    also have signof }\pi=\mathrm{ poly-add-sign m n
        by (simp add: \pi-def poly-add-sign-def m-def n-def cong: if-cong)
    finally show ?thesis
        using assms by simp
qed
have lead-coeff (poly-add p q) =
            lead-coeff (det (sylvester-mat (poly-x-minus-y p) (poly-lift q)))
    by (simp add: poly-add-def resultant-def)
also have det (sylvester-mat (poly-x-minus-y p)(poly-lift q)) =
                        (\sum\pi|\pi permutes {0..<m+n}.f\pi)
    by (simp add: det-def m-def n-def M-def f-def)
also have {\pi.\pi permutes {0..<m+n}}= insert \pi({\pi.\pi permutes {0..<m+n}}
- {\pi})
    using }\pi\mathrm{ by auto
also have (\sum\sigma\in\ldots.f\sigma)=(\sum\sigma\in{\sigma.\sigma permutes {0..<m+n}}-{\pi}.f\sigma)+f
```

```
\pi
    by (subst sum.insert) (auto simp: finite-permutations)
    also have lead-coeff ... = lead-coeff ( f \pi
    proof -
        have degree (\sum\sigma\in{\sigma.\sigma permutes {0..<m+n}}-{\pi}.f\sigma)<m*n using
assms
    by (intro degree-sum-smaller deg-f2) (auto simp: m-def n-def finite-permutations)
    with deg-f1 show ?thesis
        by (subst lead-coeff-add-le) auto
    qed
    finally show ?thesis
    using <lead-coeff (f \pi})=->\mathrm{ by simp
qed
lemma lead-coeff-poly-mult:
    fixes p q :: 'a :: {idom, ring-char-0} poly
    defines }m\equiv\mathrm{ degree }p\mathrm{ and }n\equiv\mathrm{ degree }
    assumes lead-coeff p=1 lead-coeff q=1 m>0n>0
    assumes coeff q 0}=
    shows lead-coeff (poly-mult p q :: 'a poly) = 1
proof -
    from assms have [simp]: p\not=0 q\not=0
        by auto
    have [simp]: degree (reflect-poly q) = n
    using assms by (subst degree-reflect-poly-eq) (auto simp: n-def)
    define M where M = sylvester-mat (poly-x-mult-y p) (poly-lift (reflect-poly q))
    have nz:M$$ (i,i)\not=0 if i<m+n for i
    using that by (auto simp: M-def sylvester-index-mat m-def n-def coeff-poly-x-mult-y)
    have indices-eq: {0..<m+n}={..<n}\cup(+) n'{..<m}
    by (auto simp flip: atLeast0LessThan)
    define f}\mathrm{ where f=( }\lambda\sigma.\mathrm{ signof }\sigma*(\prodi=0..<m+n.M$$ (i,\sigmai))
    have degree (fid)= degree (\prodi=0..<m+n.M$$(i,i))
    using nz by (auto simp: f-def degree-mult-eq sign-def)
    also have ... = (\sumi=0..<m+n. degree (M $$ (i,i)))
        using nz by (subst degree-prod-eq-sum-degree) auto
    also have ... = (\sumi<n. degree (M$$ (i,i))) +(\sumi<m. degree (M $$ (n+
i,n+i)))
    by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
    also have (\sumi<n. degree (M$$(i,i)))=(\sumi<n.m)
            by (intro sum.cong)
            (auto simp: M-def sylvester-index-mat m-def n-def coeff-poly-x-mult-y de-
gree-monom-eq)
    also have (\sumi<m. degree (M$$(n+i,n+i)))=(\sumi<m.0)
            by (intro sum.cong) (auto simp: M-def sylvester-index-mat m-def n-def)
    finally have deg-f1: degree (f id) =m*n
            by (simp add: mult-ac id-def)
```

```
have deg-f2: degree \((f \sigma)<m * n\) if \(\sigma\) permutes \(\{0 . .<m+n\} \sigma \neq i d\) for \(\sigma\)
proof (cases \(\exists i \in\{0 . .<m+n\} . M \$ \$(i, \sigma i)=0)\)
    case True
    hence \(*:\left(\prod i=0 . .<m+n . M \$ \$(i, \sigma i)\right)=0\)
        by auto
    show ?thesis using \(\langle m>0\rangle\langle n>0\rangle\)
        by (simp add: \(f\)-def *)
    next
    case False
    note \(n z=\) this
    from that have \(\sigma\)-less: \(\sigma i<m+n\) if \(i<m+n\) for \(i\)
        using permutes-in-image \([O F\langle\sigma\) permutes -\(\rangle]\) that by auto
    have degree \((f \sigma)=\) degree \(\left(\prod i=0 . .<m+n . M \$ \$(i, \sigma i)\right)\)
        using \(n z\) by (auto simp: f-def degree-mult-eq sign-def)
    also have \(\ldots=\left(\sum i=0 . .<m+n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)\)
        using \(n z\) by (subst degree-prod-eq-sum-degree) auto
    also have \(\ldots=\left(\sum i<n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)+\left(\sum i<m\right.\). degree \((M \$ \$(n\)
\(+i, \sigma(n+i))))\)
            by (subst indices-eq, subst sum.union-disjoint) (auto simp: sum.reindex)
    also have \(\left(\sum i<m\right.\). degree \(\left.(M \$ \$(n+i, \sigma(n+i)))\right)=\left(\sum i<m .0\right)\)
        using \(\sigma\)-less by (intro sum.cong) (auto simp: M-def sylvester-index-mat m-def
\(n\)-def)
    also have \(\left(\sum i<n\right.\). degree \(\left.(M \$ \$(i, \sigma i))\right)<\left(\sum i<n . m\right)\)
    proof (rule sum-strict-mono-ex1)
        show \(\forall x \in\{. .<n\}\). degree \((M \$ \$(x, \sigma x)) \leq m\) using \(\sigma\)-less
        by (auto simp: M-def sylvester-index-mat m-def \(n\)-def degree-coeff-poly-x-minus-y
coeff-poly-x-mult-y
                intro: order.trans[OF degree-monom-le])
    next
        have \(\exists i<n . \sigma i \neq i\)
        proof (rule ccontr)
            assume nex: \(\neg(\exists i<n . \sigma i \neq i)\)
            have \(\sigma i=i\) for \(i\)
                using that
            proof (induction i rule: less-induct)
                case (less \(i\) )
                consider \(i<n|i \in\{n . .<m+n\}| i \geq m+n\)
                    by force
                thus ?case
                proof cases
                    assume \(i<n\)
                    thus ?thesis using nex by auto
                next
                    assume \(i \geq m+n\)
                    thus ?thesis using 〈 \(\sigma\) permutes -\(\rangle\)
                        by (auto simp: permutes-def)
                next
                        assume \(i: i \in\{n . .<m+n\}\)
```

```
            have IH: \sigma j=j if j<i for j
                    using that less.prems by (intro less.IH) auto
            from nz have M $$ (i,\sigma i)\not=0
                    using i by auto
            hence \sigmai\leqi
                using i \sigma-less[of i] by (auto simp: M-def sylvester-index-mat m-def
n-def)
            moreover have \sigmai\geqi
            proof (rule ccontr)
                    assume *: \neg\sigma i\geqi
                    from * have \sigma(\sigma i)=\sigmai
                        by (subst IH) auto
                    hence \sigmai=i
                            using permutes-inj[OF <\sigma permutes ->] unfolding inj-def by blast
                    with * show False by simp
                    qed
                    ultimately show ?case by simp
            qed
        qed
        hence }\sigma=i
            by force
        with }\langle\sigma\not=id\rangle\mathrm{ show False
            by contradiction
        qed
        then obtain i where i: i<n \sigmai\not=i
        by auto
        have \sigmai<m+n
            using i by (intro \sigma-less) auto
    hence degree (M $$ (i,\sigma i))<m using i<m> 0\rangle
    by (auto simp:M-def m-def n-def sylvester-index-mat degree-coeff-poly-x-minus-y
                coeff-poly-x-mult-y intro:le-less-trans[OF degree-monom-le])
    thus \existsi\in{..<n}. degree (M $$ (i,\sigmai))<m
        using i by blast
    qed auto
    finally show degree (f \sigma)<m*n
        by (simp add: mult-ac)
    qed
    have lead-coeff (f id) = 1
    proof -
    have lead-coeff (f id) = (\prodi=0..<m + n.lead-coeff (M $$ (i, i)))
        by (simp add: f-def lead-coeff-prod)
    also have (\prodi=0..<m+n.lead-coeff (M$$ (i,i)))=
        (\prodi<n.lead-coeff (M $$ (i,i)))*(\prodi<m. lead-coeff (M $$ (n+i,
n+i)))
        by (subst indices-eq, subst prod.union-disjoint) (auto simp: prod.reindex)
        also have (\prodi<n.lead-coeff (M $$ (i,i)))=(\prodi<n.lead-coeff p) using
```

```
assms
            by (intro prod.cong) (auto simp: M-def m-def n-def sylvester-index-mat
                                    coeff-poly-x-mult-y degree-monom-eq)
    also have (\prodi<m. lead-coeff (M $$ (n+i,n+i)))=(\prodi<m.lead-coeff q)
        by (intro prod.cong) (auto simp: M-def m-def n-def sylvester-index-mat)
    finally show ?thesis
        using assms by (simp add:id-def)
    qed
    have lead-coeff (poly-mult p q) = lead-coeff (det M)
    by (simp add: poly-mult-def resultant-def M-def poly-div-def)
    also have det M = (\sum\pi|\pi permutes {0..<m+n}.f\pi)
    by (simp add: det-def m-def n-def M-def f-def)
    also have {\pi. \pi permutes {0..<m+n}} = insert id ({\pi.\pi permutes {0..<m+n}}
- {id})
    by (auto simp: permutes-id)
    also have (\sum\sigma\in\ldots..f\sigma)=(\sum\sigma\in{\sigma.\sigma permutes {0..<m+n}}-{id}.f\sigma)+
fid
    by (subst sum.insert) (auto simp: finite-permutations)
    also have lead-coeff ... = lead-coeff (f id)
    proof -
        have degree (\sum\sigma\in{\sigma.\sigma permutes {0..<m+n}}-{id}.f\sigma)<m*n using
assms
    by (intro degree-sum-smaller deg-f2) (auto simp: m-def n-def finite-permutations)
    with deg-f1 show ?thesis
        by (subst lead-coeff-add-le) auto
    qed
    finally show ?thesis
        using〈lead-coeff (f id) = 1` by simp
qed
lemma algebraic-int-plus [intro]:
    fixes x y :: 'a :: field-char-0
    assumes algebraic-int x algebraic-int y
    shows algebraic-int (x+y)
proof -
    from assms(1) obtain p where p: lead-coeff p=1 ipoly p x=0
        by (auto simp: algebraic-int-altdef-ipoly)
    from assms(2) obtain q}\mathrm{ where q: lead-coeff q=1 ipoly q y =0
        by (auto simp: algebraic-int-altdef-ipoly)
    have deg-pos: degree p>0 degree q>0
        using p q by (auto intro!: Nat.grOI elim!: degree-eq-zeroE)
    define }r\mathrm{ where }r=\mathrm{ poly-add-sign (degree p) (degree q) * poly-add p q
    have lead-coeff r=1 using p q deg-pos
    by (simp add: r-def lead-coeff-mult poly-add-sign-def sign-def lead-coeff-poly-add)
    moreover have ipoly r (x+y)=0
    using p q by (simp add: ipoly-poly-add r-def of-int-poly-hom.hom-mult)
    ultimately show ?thesis
```

```
    by (auto simp: algebraic-int-altdef-ipoly)
qed
lemma algebraic-int-times [intro]:
    fixes x y :: 'a :: field-char-0
    assumes algebraic-int x algebraic-int y
    shows algebraic-int (x*y)
proof (cases y=0)
    case [simp]: False
    from assms(1) obtain p where p: lead-coeff p=1 ipoly p x=0
        by (auto simp: algebraic-int-altdef-ipoly)
    from assms(2) obtain q}\mathrm{ where q: lead-coeff q=1 ipoly q y =0
        by (auto simp: algebraic-int-altdef-ipoly)
    have deg-pos: degree p>0 degree q>0
    using p q by (auto intro!: Nat.grOI elim!: degree-eq-zeroE)
    have [simp]: q}\not=
        using q by auto
    define n where n= Polynomial.order 0q
    have monom 1 n dvd q
        by (simp add: n-def monom-1-dvd-iff)
    then obtain q' where q-split: q= q'* monom 1 n
        by auto
    have Polynomial.order 0 q = Polynomial.order 0 q
        using }\langleq\not=0\rangle\mathrm{ unfolding q-split by (subst order-mult) auto
    hence poly q' 0}=
        unfolding n-def using «q\not=0` by (simp add: q-split order-root)
    have q': ipoly q}\mp@subsup{q}{}{\prime}y=0\mathrm{ lead-coeff q}\mp@subsup{q}{}{\prime}=1\mathrm{ using q-split q
    by (auto simp: of-int-poly-hom.hom-mult poly-monom lead-coeff-mult degree-monom-eq)
    from this have deg-pos': degree q'>0
        by (intro Nat.grOI) (auto elim!: degree-eq-zeroE)
    from 〈poly q' 0 = 0 > have [simp]: coeff q' 0 = 0
        by (auto simp: monom-1-dvd-iff' poly-0-coeff-0)
    have p represents x q' represents y
    using p q' by (auto simp: represents-def)
    hence poly-mult p q' represents x*y
        by (rule represents-mult) (simp add: poly-0-coeff-0)
    moreover have lead-coeff (poly-mult p q')=1 using p deg-pos q' deg-pos'
    by (simp add: lead-coeff-mult lead-coeff-poly-mult)
    ultimately show ?thesis
    by (auto simp: algebraic-int-altdef-ipoly represents-def)
qed auto
lemma algebraic-int-power [intro]:
    algebraic-int (x :: 'a :: field-char-0) \Longrightarrow algebraic-int ( }\mp@subsup{x}{}{\wedge}n
    by (induction n) auto
```

```
lemma algebraic-int-diff [intro]:
    fixes x y :: 'a :: field-char-0
    assumes algebraic-int x algebraic-int y
    shows algebraic-int (x-y)
    using algebraic-int-plus[OF assms(1) algebraic-int-minus[OF assms(2)]] by simp
lemma algebraic-int-sum [intro]:
    (\bigwedgex. x 位\Longrightarrow algebraic-int (f x :: 'a :: field-char-0))
        \Longrightarrow \text { algebraic-int (sum f A)}
    by (induction A rule: infinite-finite-induct) auto
lemma algebraic-int-prod [intro]:
    (\bigwedgex. x 位\Longrightarrow algebraic-int (f x :: 'a :: field-char-0))
        \Longrightarrow \text { algebraic-int (prod f A)}
    by (induction A rule: infinite-finite-induct) auto
lemma algebraic-int-nth-root-real-iff:
    algebraic-int (root n x) \longleftrightarrown=0\vee algebraic-int x
proof -
    have algebraic-int x if algebraic-int (root n x) n}\not=
    proof -
        from that(1) have algebraic-int (root n x` n)
            by auto
        also have root n x` n = (if even n then |x| else x)
            using sgn-power-root[of n x] that(2) by (auto simp: sgn-if split: if-splits)
        finally show ?thesis
            by (auto split: if-splits)
    qed
    thus?thesis by auto
qed
lemma algebraic-int-power-iff:
    algebraic-int (x ^ n :: 'a :: field-char-0) \longleftrightarrown=0\vee algebraic-int x
proof -
    have algebraic-int x if algebraic-int ( ( ^^ n) n>0
    proof (rule algebraic-int-root)
        show poly (monom 1 n) x= x`n
            by (auto simp: poly-monom)
    qed (use that in <auto simp: degree-monom-eq`)
    thus?thesis by auto
qed
lemma algebraic-int-power-iff' [simp]:
    n>0\Longrightarrow algebraic-int ( }x\mathrm{ ^ n ::' 'a :: field-char-0) 山 algebraic-int x
    by (subst algebraic-int-power-iff) auto
lemma algebraic-int-sqrt-iff [simp]: algebraic-int (sqrt x) \longleftrightarrow algebraic-int x
    by (simp add: sqrt-def algebraic-int-nth-root-real-iff)
```

```
lemma algebraic-int-csqrt-iff [simp]: algebraic-int (csqrt x) \longleftrightarrow algebraic-int x
proof
    assume algebraic-int (csqrt x)
    hence algebraic-int (csqrt x ^ 2)
        by (rule algebraic-int-power)
    thus algebraic-int x
        by simp
qed auto
lemma algebraic-int-norm-complex [intro]:
    assumes algebraic-int (z :: complex)
    shows algebraic-int (norm z)
proof -
    from assms have algebraic-int (z*cnj z)
        by auto
    also have z*cnj z=of-real (norm z` 2)
        by (rule complex-norm-square [symmetric])
    finally show ?thesis
        by simp
qed
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-gcd, semiring-no-zero-divisors} poly
    shows poly (primitive-part p) x=0 \longleftrightarrow poly p x=0
```

```
proof(cases p=0)
    case True
    then show ?thesis by auto
next
    case False
    have poly p x = content p * poly (primitive-part p) x
        by (metis content-times-primitive-part poly-smult)
    also have ... = 0 \longleftrightarrow poly (primitive-part p) x=0 by (simp add: False)
    finally show ?thesis by auto
qed
```

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

### 6.1 Interface for Separating Roots

For a given rational polynomial, we need to know how many real roots are in a given closed interval, and how many real roots are in an interval $(-\infty, r]$. datatype root-info $=$ Root-Info $(l-r: r a t \Rightarrow r a t \Rightarrow n a t)($ number-root: rat $\Rightarrow$ nat $)$ hide-const (open) $l-r$ hide-const (open) number-root

```
definition count-roots-interval-sf :: real poly \(\Rightarrow(\) real \(\Rightarrow\) real \(\Rightarrow\) nat \() \times(\) real \(\Rightarrow\)
nat) where
    count-roots-interval-sf \(p=(\) let \(p s=\) sturm-squarefree \(p\)
    in \(((\lambda a b\). sign-changes ps \(a-\) sign-changes ps \(b+(\) if poly \(p a=0\) then 1 else
\(0)\) ),
    ( \(\lambda\) a. sign-changes-neg-inf \(p s-\) sign-changes \(p s a))\) )
definition count-roots-interval \(::\) real poly \(\Rightarrow(\) real \(\Rightarrow\) real \(\Rightarrow\) nat \() \times(\) real \(\Rightarrow\) nat \()\)
where
    count-roots-interval \(p=(\) let \(p s=\) sturm \(p\)
    in \(((\lambda a b\). sign-changes ps \(a-\) sign-changes ps \(b+(\) if poly \(p a=0\) then 1 else
\(0)\) ),
            ( \(\lambda\) a. sign-changes-neg-inf \(p s-\) sign-changes \(p s a))\) )
lemma count-roots-interval-iff: square-free \(p \Longrightarrow\) count-roots-interval \(p=\) count-roots-interval-sf
\(p\)
    unfolding count-roots-interval-def count-roots-interval-sf-def sturm-squarefree-def
        square-free-iff-separable separable-def by (cases \(p=0\), auto)
lemma count-roots-interval-sf: assumes \(p: p \neq 0\)
    and \(c r\) : count-roots-interval-sf \(p=(c r, n r)\)
    shows \(a \leq b \Longrightarrow c r a b=(\operatorname{card}\{x . a \leq x \wedge x \leq b \wedge\) poly \(p x=0\})\)
        nr \(a=\operatorname{card}\{x . x \leq a \wedge\) poly \(p x=0\}\)
```

```
proof -
    have id: a\leqb\Longrightarrow{x.a\leqx^x\leqb^ poly px=0}=
        {x.a<x\wedgex\leqb^ poly px=0}\cup(if poly pa=0 then {a} else {})
        (is - \Longrightarrow-= ?R \cup?S) using not-less by force
    have RS: finite ?R finite ?S ?R \cap?S = {} using p by (auto simp: poly-roots-finite)
```

    show \(a \leq b \Longrightarrow c r a b=(\operatorname{card}\{x . a \leq x \wedge x \leq b \wedge\) poly \(p x=0\})\)
        nr \(a=\) card \(\{x . x \leq a \wedge\) poly \(p x=0\}\) using cr unfolding arg-cong[OF id,
    of card] card-Un-disjoint[OF RS]
count-roots-interval-sf-def count-roots-between-correct[symmetric]
count-roots-below-correct[symmetric] count-roots-below-def
count-roots-between-def Let-def using $p$ by auto
qed
lemma count-roots-interval: assumes cr: count-roots-interval $p=(c r, n r)$
and sf: square-free $p$
shows $a \leq b \Longrightarrow c r a b=(\operatorname{card}\{x . a \leq x \wedge x \leq b \wedge$ poly $p x=0\})$
nr $a=\operatorname{card}\{x . x \leq a \wedge$ poly $p x=0\}$
using count-roots-interval-sf[OF - cr[unfolded count-roots-interval-iff [OF sf]]]
sf[unfolded square-free-def] by blast+
definition root-cond $::$ int poly $\times$ rat $\times$ rat $\Rightarrow$ real $\Rightarrow$ bool where
root-cond plr $x=($ case plr of $(p, l, r) \Rightarrow$ of-rat $l \leq x \wedge x \leq$ of-rat $r \wedge$ ipoly $p x$ $=0$ )
definition root-info-cond $::$ root-info $\Rightarrow$ int poly $\Rightarrow$ bool where
root-info-cond ri $p \equiv(\forall a b . a \leq b \longrightarrow$ root-info.l-r ri $a b=$ card $\{x$. root-cond $(p, a, b) x\})$
$\wedge(\forall$ a. root-info.number-root ri $a=$ card $\{x . x \leq$ real-of-rat $a \wedge$ ipoly $p x=$ 0\})
lemma root-info-condD: root-info-cond ri $p \Longrightarrow a \leq b \Longrightarrow$ root-info.l-r ri a $b=$ card $\{x$. root-cond $(p, a, b) x\}$
root-info-cond ri $p \Longrightarrow$ root-info.number-root ri $a=$ card $\{x . x \leq$ real-of-rat $a \wedge$ ipoly $p x=0\}$
unfolding root-info-cond-def by auto
definition count-roots-interval-sf-rat :: int poly $\Rightarrow$ root-info where
count-roots-interval-sf-rat $p=$ (let pp $=$ real-of-int-poly $p$; $(c r, n r)=$ count-roots-interval-sf pp
in Root-Info ( $\lambda$ a b. cr (of-rat a) $(o f-r a t b))(\lambda a . n r(o f-r a t a)))$
definition count-roots-interval-rat $::$ int poly $\Rightarrow$ root-info where
[code del]: count-roots-interval-rat $p=$ (let pp = real-of-int-poly $p ;$ $(c r, n r)=$ count-roots-interval $p p$
in Root-Info ( $\lambda$ a b. cr (of-rat a) (of-rat b)) ( $\lambda a . n r(o f-r a t a)))$
definition count-roots-rat :: int poly $\Rightarrow$ nat where

```
    [code del]: count-roots-rat p = (count-roots (real-of-int-poly p))
lemma count-roots-interval-sf-rat: assumes p: p\not=0
    shows root-info-cond (count-roots-interval-sf-rat p) p
proof -
    let ?p = real-of-int-poly p
    let ?r = real-of-rat
    let ?ri = count-roots-interval-sf-rat p
    from p have p:?p}\not=0\mathrm{ by auto
    obtain cr nr where cr: count-roots-interval-sf ?p = (cr,nr) by force
    have ?ri = Root-Info (\lambdaa b.cr (?r a) (?r b)) (\lambdaa.nr (?r a))
        unfolding count-roots-interval-sf-rat-def Let-def cr by auto
    hence id: root-info.l-r ?ri = (\lambdaa b.cr (?r a) (?r b)) root-info.number-root ?ri =
(\lambdaa.nr (?r a))
    by auto
    note cr = count-roots-interval-sf[OF p cr]
    show ?thesis unfolding root-info-cond-def id
    proof (intro conjI impI allI)
        fix }
        show nr (?r a) = card {x. x\leq (?r a) ^ ipoly p x = 0}
            using cr(2)[of ?r a] by simp
    next
        fix a b :: rat
        assume ab:a\leqb
        from ab have ab: ?r a\leq?r b by (simp add: of-rat-less-eq)
        from cr(1)[OF this] show cr (?r a) (?r b) = card (Collect (root-cond (p,a,
b)))
        unfolding root-cond-def[abs-def] split by simp
    qed
qed
lemma of-rat-of-int-poly: map-poly of-rat (of-int-poly p) = of-int-poly p
    by (subst map-poly-map-poly, auto simp: o-def)
lemma square-free-of-int-poly: assumes square-free p
    shows square-free (of-int-poly p :: 'a :: {field-gcd, field-char-0} poly)
proof -
    have square-free (map-poly of-rat (of-int-poly p) :: 'a poly)
        unfolding of-rat-hom.square-free-map-poly by (rule square-free-int-rat[OF assms])
    thus ?thesis unfolding of-rat-of-int-poly .
qed
lemma count-roots-interval-rat: assumes sf: square-free p
    shows root-info-cond (count-roots-interval-rat p) p
proof -
    from sf have sf: square-free (real-of-int-poly p) by (rule square-free-of-int-poly)
    from sf have p:p\not=0 unfolding square-free-def by auto
    show ?thesis
    using count-roots-interval-sf-rat[OF p]
```

unfolding count-roots-interval-rat-def count-roots-interval-sf-rat-def Let-def count-roots-interval-iff $[O F s f]$. qed
lemma count-roots-rat: count-roots-rat $p=$ card $\{x$. ipoly $p x=(0::$ real $)\}$ unfolding count-roots-rat-def count-roots-correct ..

### 6.2 Implementing Sturm on Rational Polynomials

```
function sturm-aux-rat where
sturm-aux-rat ( }p::\mathrm{ rat poly) q=
    (if degree q = 0 then [p,q] else p # sturm-aux-rat q (-( }p\operatorname{mod}q))\mathrm{ )
    by (pat-completeness, simp-all)
termination by (relation measure (degree ○ snd),
    simp-all add: o-def degree-mod-less')
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 (induct p q rule: sturm-aux-rat.induct)
    case (1 p q)
    interpret map-poly-inj-idom-hom of-rat..
    note deg = of-int-hom.degree-map-poly-hom
    show ?case
        unfolding sturm-aux.simps[of real-of-rat-poly p] sturm-aux-rat.simps[of p]
        using 1 by (cases degree q=0; simp add: hom-distribs)
qed
definition sturm-rat where sturm-rat p = sturm-aux-rat p (pderiv p)
lemma sturm-rat: sturm (real-of-rat-poly p) = map real-of-rat-poly (sturm-rat p)
    unfolding sturm-rat-def sturm-def
    apply (fold of-rat-hom.map-poly-pderiv)
    unfolding sturm-aux-rat..
definition poly-number-rootat :: rat poly }=>\mathrm{ rat where
    poly-number-rootat p \equivsgn (coeff p (degree p))
definition poly-neg-number-rootat :: rat poly }=>\mathrm{ rat where
    poly-neg-number-rootat p\equiv if even (degree p) then sgn (coeff p (degree p))
        else - sgn (coeff p (degree p))
```

lemma poly-number-rootat: poly-inf (real-of-rat-poly $p$ ) = real-of-rat (poly-number-rootat
p)
unfolding poly-inf-def poly-number-rootat-def of-int-hom.degree-map-poly-hom
of-rat-hom.coeff-map-poly-hom
real-of-rat-sgn by simp
lemma poly-neg-number-rootat: poly-neg-inf (real-of-rat-poly $p$ ) = real-of-rat (poly-neg-number-rootat
definition sign-changes-rat where
sign-changes-rat ps $(x:: r a t)=$
length $($ remdups-adj $($ filter $(\lambda x . x \neq 0)(\operatorname{map}(\lambda p . \operatorname{sgn}(p o l y p x)) p s)))-1$
definition sign-changes-number-rootat where
sign-changes-number-rootat $p s=$
length $($ remdups-adj $($ filter $(\lambda x . x \neq 0)($ map poly-number-rootat $p s)))-1$
definition sign-changes-neg-number-rootat where
sign-changes-neg-number-rootat ps $=$
length $($ remdups-adj $($ filter $(\lambda x . x \neq 0)($ map poly-neg-number-rootat $p s)))-$
1
lemma real-of-rat-list-neq: list-neq (map real-of-rat xs) 0
$=$ map real-of-rat (list-neq xs 0)
by (induct xs, auto)
lemma real-of-rat-remdups-adj: remdups-adj (map real-of-rat xs) $=$ map real-of-rat (remdups-adj xs)
by (induct xs rule: remdups-adj.induct, auto)
lemma sign-changes-rat: sign-changes (map real-of-rat-poly ps) (real-of-rat $x$ )
$=$ sign-changes-rat ps $x($ is $? l=? r)$
proof -
define $x s$ where $x s=$ list-neq $(\operatorname{map}(\lambda p . \operatorname{sgn}($ poly $p x)) p s) 0$
have ?l = length (remdups-adj (list-neq (map real-of-rat (map ( $\lambda$ xa. ( $\operatorname{sgn}$ (poly
xa $x$ )) ) ps )) 0)) - 1
by (simp add: sign-changes-def real-of-rat-sgn o-def)
also have $\ldots=$ ? $r$ unfolding sign-changes-rat-def real-of-rat-list-neq
unfolding real-of-rat-remdups-adj by simp
finally show ?thesis .
qed
lemma sign-changes-neg-number-rootat: sign-changes-neg-inf (map real-of-rat-poly ps)
$=$ sign-changes-neg-number-rootat ps $(\mathbf{i s} ? l=? r)$
proof -
have ?l = length (remdups-adj (list-neq (map real-of-rat (map poly-neg-number-rootat ps)) 0)) - 1
by (simp add: sign-changes-neg-inf-def o-def real-of-rat-sgn poly-neg-number-rootat) also have $\ldots=? r$ unfolding sign-changes-neg-number-rootat-def real-of-rat-list-neq
unfolding real-of-rat-remdups-adj by simp
finally show? ?thesis .

## qed

lemma sign-changes-number-rootat: sign-changes-inf (map real-of-rat-poly ps)
$=$ sign-changes-number-rootat ps (is ?l $=? r$ )
proof -
have ?l = length (remdups-adj (list-neq (map real-of-rat (map poly-number-rootat ps)) 0)) - 1
unfolding sign-changes-inf-def
unfolding map-map o-def real-of-rat-sgn poly-number-rootat ..
also have $\ldots=$ ? $r$ unfolding sign-changes-number-rootat-def real-of-rat-list-neq
unfolding real-of-rat-remdups-adj by simp finally show ?thesis .
qed
lemma count-roots-interval-rat-code[code]:
count-roots-interval-rat $p=($ let $r p=$ map-poly rat-of-int $p ; p s=$ sturm-rat $r p$
in Root-Info
( $\lambda$ a b. sign-changes-rat ps $a-$ sign-changes-rat $p s b+($ if poly rp $a=0$ then 1 else 0))
( $\lambda$ a.sign-changes-neg-number-rootat ps - sign-changes-rat ps a))
unfolding count-roots-interval-rat-def Let-def count-roots-interval-def split of-rat-of-int-poly[symmetric, where ' $a=$ real]
sturm-rat sign-changes-rat
by (simp add: sign-changes-neg-number-rootat)
lemma count-roots-rat-code[code]:
count-roots-rat $p=$ (let rp $=$ map-poly rat-of-int $p$ in if $p=0$ then 0 else let $p s$
= sturm-rat rp
in sign-changes-neg-number-rootat ps - sign-changes-number-rootat ps)
unfolding count-roots-rat-def Let-def sturm-rat count-roots-code of-rat-of-int-poly[symmetric, where ' $a=$ real]
sign-changes-neg-number-rootat sign-changes-number-rootat
by $\operatorname{simp}$
hide-const (open) count-roots-interval-sf-rat
Finally we provide an even more efficient implementation which avoids the "poly $\mathrm{p} \mathrm{x}=0$ " test, but it is restricted to irreducible polynomials.

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

lemma root-info:

```
    assumes irr: irreducible p and deg: degree p>0
    shows root-info-cond (root-info p) p
proof (cases degree p=1)
    case deg: True
    from degree1-coeffs[OF this] obtain a b where p: p=[:b,a:] and a\not=0 by auto
    from deg have degree (real-of-int-poly p)=1 by simp
    from roots1[OF this, unfolded roots1-def] p
    have id:(ipoly px=0) = ((x :: real)=-b/a) for }x\mathrm{ by auto
    have idd: {x. real-of-rat aa \leqx^
                        x <real-of-rat ba ^x= real-of-int (-b) / real-of-int a}
    =(if real-of-rat aa \leqreal-of-int (-b) / real-of-int a ^
        real-of-int (-b) / real-of-int a s real-of-rat ba then {real-of-int (-
b) / real-of-int a} else {})
        for aa ba by auto
    have iddd: {x. x\leq real-of-rat aa }\wedgex=real-of-int (-b) / real-of-int a
        =(if real-of-int (-b)/ real-of-int a \leqreal-of-rat aa then {real-of-int (-b) /
real-of-int a} else {}) for aa
    by auto
    have id4: real-of-int x = real-of-rat (rat-of-int x) for }x\mathrm{ by simp
    show ?thesis unfolding root-info-def deg unfolding root-info-cond-def id root-cond-def
split
    unfolding p Fract-of-int-quotient Let-def idd iddd
    unfolding id4 of-rat-divide[symmetric] of-rat-less-eq by auto
next
    case False
    have irr-d: irreducible d p by (simp add: deg irr irreducible-connect-rev)
    from irreducible d-int-rat[OF this]
    have irreducible (of-int-poly p :: rat poly) by auto
    from irreducible-root-free[OF this]
    have idd: (poly (of-int-poly p) a=0)= False for a :: rat
        unfolding root-free-def using False by auto
    have id: root-info p = count-roots-interval-rat p
        unfolding root-info-def if-False count-roots-interval-rat-code Let-def idd using
False by auto
    show ?thesis unfolding id
        by (rule count-roots-interval-rat[OF irreducible d-square-free[OF irr-d]])
qed
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 \longleftrightarrow
    degree (gcd (of-int-poly q :: 'a :: {field-char-0, field-gcd} poly) (of-int-poly r))}\not=
proof -
    let ?r =of-rat :: rat # 'a
    interpret rpoly: field-hom' ?r
        by (unfold-locales, auto simp: of-rat-add of-rat-mult)
    {
        fix p
        have of-int-poly p = map-poly (?r o of-int) p unfolding o-def
            by auto
        also have ... = map-poly ?r (map-poly of-int p)
            by (subst map-poly-map-poly, auto)
        finally have of-int-poly p = map-poly ?r (map-poly of-int p).
    } note id = this
    show ?thesis unfolding id by (fold hom-distribs, simp add: gcd-rat-to-gcd-int)
qed
definition factors-of-int-poly :: int poly }=>\mathrm{ int poly list where
    factors-of-int-poly p = map (abs-int-poly o fst) (snd (factorize-int-poly p))
lemma factors-of-int-poly-const: assumes degree p=0
    shows factors-of-int-poly p = []
proof -
    from degree0-coeffs[OF assms] obtain a where p:p=[:a:] by auto
    show ?thesis unfolding p factors-of-int-poly-def
            factorize-int-poly-generic-def x-split-def
            by (cases a = 0, auto simp add: Let-def factorize-int-last-nz-poly-def)
qed
lemma factors-of-int-poly:
```



```
    assumes factors-of-int-poly p = qs
    shows }\q.q\in\mathrm{ set qs # irreducible q}\\\mathrm{ lead-coeff }q>0\wedge\mathrm{ degree }q\leq\mathrm{ degree
p\wedge degree q}\not=
    p\not=0\Longrightarrowrp px=0\longleftrightarrow(\existsq\in set qs.rp q x = 0)
    p\not=0\Longrightarrowrp px=0\Longrightarrow\exists!q\in set qs.rp q x = 0
    distinct qs
proof -
    obtain c qis where factt: factorize-int-poly p = (c,qis) by force
    from assms[unfolded factors-of-int-poly-def factt]
    have qs:qs = map (abs-int-poly \circfst) (snd (c, qis)) by auto
    note fact = factorize-int-poly(1)[OF factt]
    note fact-mem = factorize-int-poly(2,3)[OF factt]
```

```
    have sqf: square-free-factorization p (c, qis) by (rule fact(1))
    note sff = square-free-factorizationD[OF sqf]
    have sff': p = Polynomial.smult c (\prod (a,i)\leftarrowqis.a^ Suc i)
    unfolding sff(1) prod.distinct-set-conv-list[OF sff(5)] ..
{
    fix q
    assume q:q\in set qs
    then obtain ri where qi:(r,i)\in set qis and qr:q=abs-int-polyr unfolding
qs by auto
    from split-list[OF qi] obtain qis1 qis2 where qis: qis = qis1 @ (r,i) # qis2
by auto
    have dvd: r dvd p unfolding sff' qis dvd-def
        by (intro exI[of-smult c (r^i*(\prod(a,i)\leftarrowqis1@ qis2. a ^ Suc i))],auto)
    from fact-mem[OF qi] have r0:r\not=0 by auto
    from qi factt have p: p\not=0 by (cases p, auto)
    with dvd have deg: degree r\leq degree p by (metis dvd-imp-degree-le)
    with fact-mem[OF qi] r0
    show irreducible q\wedge lead-coeff q>0^ degree q\leq degree p}\wedge\mathrm{ degree q}\not=
        unfolding qr lead-coeff-abs-int-poly by auto
    } note * = this
    show distinct qs unfolding distinct-conv-nth
    proof (intro allI impI)
    fix ij
    assume i< length qs j< length qs and diff: i\not=j
    hence ij:i< length qis j< length qis
        and id:qs!i=abs-int-poly (fst (qis!i)) qs !j=abs-int-poly (fst (qis!j))
unfolding qs by auto
    obtain qi I where qi: qis ! i = (qi,I) by force
    obtain qj J where qj: qis ! j = (qj, J) by force
    from sff(5)[unfolded distinct-conv-nth, rule-format, OF ij diff] qi qj
    have diff: (qi,I)}\not=(qj,J) by aut
    from ij qi qj have (qi,I) \in set qis (qj, J) \in set qis unfolding set-conv-nth
by force+
    from sff(3)[OF this diff] sff(2) this
    have cop: coprime qi qj degree qi}\not=0\mathrm{ degree qj}\not=0\mathrm{ by auto
    note i = cf-pos-poly-main[of qi, unfolded smult-prod monom-0]
    note j = cf-pos-poly-main[of qj, unfolded smult-prod monom-0]
    from cop(2) i have deg: degree (qs ! i)\not=0 by (auto simp: id qi)
    have cop: coprime (qs !i) (qs! j)
        unfolding id qi qj fst-conv
        apply (rule coprime-prod[of [:sgn (lead-coeff qi):] [:sgn (lead-coeff qj):]])
        using cop
        unfolding ij by (auto simp: sgn-eq-0-iff)
show qs!i\not=qs!j
proof
        assume id:qs !i=qs!j
        have degree (gcd (qs !i) (qs!j)) = degree (qs!i) unfolding id by simp
        also have ... =0 using deg by simp
        finally show False using cop by simp
```

qed
qed
assume $p: p \neq 0$
from fact(1) $p$ have $c: c \neq 0$ using sff(1) by auto
let ? $r=o f$-int $::$ int $\Rightarrow{ }^{\prime} a$
let ${ }^{2} r p=$ map-poly $? r$
have $r p: \wedge x p . r p p x=0 \longleftrightarrow$ poly $(? r p p) x=0$ unfolding rp-def ..
have $r p p x=0 \longleftrightarrow r p\left(\prod(x, y) \leftarrow q i s\right.$. $x$ ^Suc $\left.y\right) x=0$ unfolding $s f f^{\prime}(1)$
unfolding rp hom-distribs using $c$ by simp
also have $\ldots=\left(\exists(q, i) \in\right.$ set qis. poly $\left(\right.$ ? $r p\left(q^{\wedge}\right.$ Suc $\left.\left.\left.i\right)\right) x=0\right)$
unfolding qs rp of-int-poly-hom.hom-prod-list poly-prod-list-zero-iff set-map by
fastforce
also have $\ldots=(\exists(q, i) \in$ set qis. poly $($ ?rp $q) x=0)$
unfolding of-int-poly-hom.hom-power poly-power-zero-iff by auto
also have $\ldots=(\exists q \in f s t$ ' set qis. poly $(? r p q) x=0)$ by force
also have $\ldots=(\exists q \in$ set $q$ s. rp $q x=0)$ unfolding rp qs snd-conv o-def
bex-simps set-map
by $\operatorname{simp}$
finally show iff: $r p p x=0 \longleftrightarrow(\exists q \in$ set $q s . r p q x=0)$ by auto
assume $r p p x=0$
with iff obtain $q$ where $q: q \in$ set $q s$ and $r t q: r p q x=0$ by auto
then obtain $i q^{\prime}$ where $q i:\left(q^{\prime}, i\right) \in$ set qis and $q q^{\prime}: q=a b s$-int-poly $q^{\prime}$ unfolding $q s$ by auto
show $\exists!q \in$ set $q s . r p q x=0$
proof (intro ex1I, intro conjI, rule $q$, rule rtq, clarify)
fix $r$
assume $r \in$ set $q s$ and rtr: rp $r x=0$
then obtain $j r^{\prime}$ where $r j:\left(r^{\prime}, j\right) \in$ set qis and $r r^{\prime}: r=$ abs-int-poly $r^{\prime}$
unfolding $q s$ by auto
from $r t r r t q$ have $r t r: r p r^{\prime} x=0$ and $r t q: r p q^{\prime} x=0$
unfolding $r p r r^{\prime} q q^{\prime}$ by auto
from $r$ tr $r t q$ have $[:-x, 1:] d v d$ ? $r p q^{\prime}[:-x, 1:] d v d$ ? $r p r^{\prime}$ unfolding $r p$
by (auto simp: poly-eq-0-iff-dvd)
hence $[:-x, 1:]$ dvd gcd (?rp $\left.q^{\prime}\right)\left(? r p r^{\prime}\right)$ by simp
hence $g c d\left(? r p q^{\prime}\right)\left(? r p r^{\prime}\right)=0 \vee$ degree $\left(g c d\left(? r p q^{\prime}\right)\left(? r p r^{\prime}\right)\right) \neq 0$
by (metis is-unit-gcd-iff is-unit-iff-degree is-unit-pCons-iff one-poly-eq-simps(1))
hence $g c d q^{\prime} r^{\prime}=0 \vee$ degree $\left(g c d q^{\prime} r^{\prime}\right) \neq 0$
unfolding gcd-eq-0-iff degree-of-gcd[of $q^{\prime} r^{\prime}$, symmetric $]$ by auto
hence $\neg$ coprime $q^{\prime} r^{\prime}$ by auto
with $\operatorname{sff}$ (3) $[O F q i r j]$ have $q^{\prime}=r^{\prime}$ by auto
thus $r=q$ unfolding $r r^{\prime} q q^{\prime}$ by simp
qed
qed
lemma factors-int-poly-represents:
fixes $x::{ }^{\prime} a::\{$ field-char- 0, field-gcd $\}$
assumes $p: p$ represents $x$
shows $\exists q \in$ set (factors-of-int-poly $p$ ).
$q$ represents $x \wedge$ irreducible $q \wedge$ lead-coeff $q>0 \wedge$ degree $q \leq$ degree $p$

```
proof -
    from representsD[OF p] have p: p\not=0 and rt: ipoly p x = 0 by auto
    note fact = factors-of-int-poly[OF refl]
    from fact(2)[OF p,of x] rt obtain q where q: q\in set (factors-of-int-poly p)
and
    rt: ipoly q x = 0 by auto
    from fact(1)[OF q] rt show ?thesis
        by (intro bexI[OF - q], auto simp: represents-def irreducible-def)
qed
corollary irreducible-represents-imp-degree:
    fixes x :: 'a :: {field-char-0,field-gcd}
    assumes irreducible f}\mathrm{ and frepresents x and g represents x
    shows degree f}\leq\mathrm{ degree g
proof -
    from factors-of-int-poly(1)[OF refl, of - g] factors-of-int-poly(3)[OF refl, of g x]
        assms(3) obtain h}\mathrm{ where *: h represents x degree h}\leq\mathrm{ degree g irreducible h
        by blast
    let ?af =abs-int-poly f
    let ?ah = abs-int-poly h
    from assms have af: irreducible ?af ?af represents x lead-coeff ?af > 0 by
fastforce+
    from * have ah: irreducible ?ah ?ah represents x lead-coeff ?ah > 0 by fastforce+
    from algebraic-imp-represents-unique[of x] af ah have id: ?af = ?ah
        unfolding algebraic-iff-represents by blast
    show ?thesis using arg-cong[OF id, of degree] <degree h}\leq\mathrm{ degree g> by simp
qed
lemma irreducible-preservation:
    fixes x :: 'a :: {field-char-0,field-gcd}
    assumes irr: irreducible p
    and x: p represents x
    and y:q represents y
    and deg: degree p\geq degree q
    and f:^ \.q represents y\Longrightarrow(fq) represents }x\wedge\mathrm{ degree (f q) \ degree q
    and pr: primitive q
    shows irreducible q
proof (rule ccontr)
    define pp where pp=abs-int-poly p
    have dp: degree p}=0\mathrm{ using }x\mathrm{ by (rule represents-degree)
    have dq: degree q}=0\mathrm{ using y by (rule represents-degree)
    from dp have p0: p\not=0 by auto
    from x deg irr p0
    have irr: irreducible pp and x: pp represents x and
        deg: degree pp \geqdegree q and cf-pos: lead-coeff pp>0
        unfolding pp-def lead-coeff-abs-int-poly by (auto intro!: representsI)
    from }x\mathrm{ have ax: algebraic x unfolding algebraic-altdef-ipoly represents-def by
blast
    assume \neg ?thesis
```

from this irreducible-connect-int $[o f q] p r$ have $\neg$ irreducible $_{d} q$ by auto
from this $d q$ obtain $r$ where
$r$ : degree $r \neq 0$ degree $r<$ degree $q$ and $r d v d q$ by auto
then obtain $r r$ where $q: q=r * r r$ unfolding $d v d-d e f$ by auto
have degree $q=$ degree $r+$ degree $r r$ using $d q$ unfolding $q$
by (subst degree-mult-eq, auto)
with $r$ have $r r$ : degree $r r \neq 0$ degree $r r<$ degree $q$ by auto
from represents $D(2)[O F y$, unfolded $q$ hom-distribs $]$
have ipoly $r y=0 \vee$ ipoly $r r y=0$ by auto
with $r$ rr have $r$ represents $y \vee r r$ represents $y$ unfolding represents-def by auto
with $r r r$ obtain $r$ where $r$ : represents y degree $r<$ degree $q$ by blast
from $f[O F r(1)]$ deg $r(2)$ obtain $r$ where $r$ : represents $x$ degree $r<$ degree $p p$ by auto
from factors-int-poly-represents[OF $r(1)] r(2)$ obtain $r$ where
$r: r$ represents $x$ irreducible $r$ lead-coeff $r>0$ and deg: degree $r<$ degree $p p$ by force
from algebraic-imp-represents-unique[OF ax] rirr cf-pos $x$ have $r=p p$ by auto with deg show False by auto
qed
declare irreducible-const-poly-iff [simp]
lemma poly-uminus-irreducible:
assumes $p$ : irreducible ( $p::$ int poly) and deg: degree $p \neq 0$
shows irreducible (poly-uminus $p$ )
proof-
from deg-nonzero-represents[OF deg] obtain $x$ :: complex where $x: p$ represents
$x$ by auto
from represents-uminus $[O F x]$
have $y$ : poly-uminus $p$ represents $(-x)$.
show ?thesis
proof (rule irreducible-preservation[OF $p x y$ ], force)
from deg irreducible-imp-primitive $[O F p]$ have primitive $p$ by auto
then show primitive (poly-uminus $p$ ) by simp
fix $q$
assume $q$ represents $(-x)$
from represents-uminus[OF this] have (poly-uminus q) represents $x$ by simp
thus (poly-uminus $q$ ) represents $x \wedge$ degree (poly-uminus $q$ ) $\leq$ degree $q$ by auto
qed
qed
lemma reflect-poly-irreducible:
fixes $x::{ }^{\prime} a::\{$ field-char- 0, field-gcd $\}$
assumes $p$ : irreducible $p$ and $x: p$ represents $x$ and $x 0: x \neq 0$
shows irreducible (reflect-poly p)
proof -
from represents-inverse[OF x0 x]
have $y$ : (reflect-poly $p$ ) represents (inverse $x$ ) by simp

```
    from x0 have ix0: inverse }x\not=0\mathrm{ by auto
    show ?thesis
    proof (rule irreducible-preservation[OF prey])
        from x irreducible-imp-primitive[OF p]
        show primitive (reflect-poly p) by (auto simp: content-reflect-poly)
    fix q
    assume q represents (inverse x)
    from represents-inverse[OF ix0 this] have (reflect-poly q) represents x by simp
    with degree-reflect-poly-le
    show (reflect-poly q) represents x ^ degree (reflect-poly q) \leq degree q by auto
    qed (insert p, auto simp: degree-reflect-poly-le)
qed
lemma poly-add-rat-irreducible:
    assumes p: irreducible p and deg: degree p}\not=
    shows irreducible (cf-pos-poly (poly-add-rat r p))
proof -
    from deg-nonzero-represents[OF deg] obtain x :: complex where x: p represents
x by auto
    from represents-add-rat[OF x]
    have y:cf-pos-poly (poly-add-rat r p) represents (of-rat r + x) by simp
    show ?thesis
    proof (rule irreducible-preservation[OF p x y], force)
        fix q
        assume q represents (of-rat r + x)
        from represents-add-rat[OF this, of - r] have (poly-add-rat (-r) q) represents
x by (simp add: of-rat-minus)
            thus (poly-add-rat (-r) q) represents x ^ degree (poly-add-rat (-r)q)\leq
degree q by auto
    qed (insert p,auto)
qed
lemma poly-mult-rat-irreducible:
    assumes p: irreducible p and deg: degree p\not=0 and r:r\not=0
    shows irreducible (cf-pos-poly (poly-mult-rat r p))
proof -
    from deg-nonzero-represents[OF deg] obtain x :: complex where x: p represents
x by auto
    from represents-mult-rat[OF r x]
    have y:cf-pos-poly (poly-mult-rat r p) represents (of-rat r * x) by simp
    show ?thesis
    proof (rule irreducible-preservation[OF p x y], force simp:r)
        fix q
        from r have r': inverse r}\not=0\mathrm{ by simp
    assume q represents (of-rat r*x)
    from represents-mult-rat[OF r' this] have (poly-mult-rat (inverse r) q) repre-
sents x using r
            by (simp add: of-rat-divide field-simps)
            thus (poly-mult-rat (inverse r) q) represents x ^ degree (poly-mult-rat (inverse
```

```
r) q)}\leq\mathrm{ degree }
        using r by auto
    qed (insert p r, auto)
qed
interpretation coeff-lift-hom:
    factor-preserving-hom coeff-lift :: ' a :: {comm-semiring-1,semiring-no-zero-divisors}
=> -
    by (unfold-locales, auto)
```

end

## 8 The minimal polynomial of an algebraic number

theory Min-Int-Poly imports<br>Algebraic-Numbers-Prelim<br>begin

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

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

```
definition min-int-poly :: ' \(a\) :: field-char-0 \(\Rightarrow\) int poly where
    min-int-poly \(x=\)
        (if algebraic \(x\) then THE p. p represents \(x \wedge\) irreducible \(p \wedge\) lead-coeff \(p>0\)
        else [:0, 1:])
lemma
    fixes \(x::\) ' \(a\) :: \(\{\) field-char-0, field-gcd \(\}\)
    shows min-int-poly-represents [intro]: algebraic \(x \Longrightarrow\) 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 -
    note \(*=\) theI \({ }^{\prime}[\) OF algebraic-imp-represents-unique, of \(x]\)
    show min-int-poly \(x\) represents \(x\) if algebraic \(x\)
        using \(*[\) OF that \(]\) by (simp add: that min-int-poly-def)
    have irreducible [:0, \(1::\) int:]
        by (rule irreducible-linear-poly) auto
    thus irreducible (min-int-poly \(x\) )
            using \(*\) by (auto simp: min-int-poly-def)
    show lead-coeff (min-int-poly \(x\) ) \(>0\)
        using * by (auto simp: min-int-poly-def)
qed
```

```
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)}\not=
proof -
    show degree (min-int-poly x)>0
    proof (cases algebraic x)
        case True
        hence min-int-poly x represents x
            by auto
        thus ?thesis by blast
    qed (auto simp: min-int-poly-def)
    thus degree (min-int-poly x)}\not=
        by blast
qed
lemma min-int-poly-primitive [intro]:
    fixes }x:: 'a :: {field-char-0, field-gcd
    shows primitive (min-int-poly x)
    by (rule irreducible-imp-primitive) auto
lemma min-int-poly-content [simp]:
    fixes }x\mathrm{ :: 'a :: {field-char-0, field-gcd}
    shows content (min-int-poly x) = 1
    using min-int-poly-primitive[of x] by (simp add: primitive-def)
lemma ipoly-min-int-poly [simp]:
    algebraic x\Longrightarrow ipoly (min-int-poly x) (x :: ' a :: {field-gcd, field-char-0}) = 0
    using min-int-poly-represents[of x] by (auto simp: represents-def)
lemma min-int-poly-nonzero [simp]:
    fixes }x:: ' a :: { field-char-0, field-gcd
    shows min-int-poly x\not=0
    using lead-coeff-min-int-poly-pos[of x] by auto
lemma min-int-poly-normalize [simp]:
    fixes }x\mathrm{ :: 'a :: {field-char-0, field-gcd}
    shows normalize (min-int-poly x) = min-int-poly x
    unfolding normalize-poly-def using lead-coeff-min-int-poly-pos[of x] by simp
lemma min-int-poly-prime-elem [intro]:
    fixes }x::' 'a :: {field-char-0, field-gcd
    shows prime-elem (min-int-poly x)
    using min-int-poly-irreducible[of x] by blast
lemma min-int-poly-prime [intro]:
    fixes }x::' 'a :: {field-char-0, field-gcd
    shows prime (min-int-poly x)
    using min-int-poly-prime-elem[of x]
```

```
    by (simp only: prime-normalize-iff [symmetric] min-int-poly-normalize)
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 -
    from assms(1) have x: algebraic x
        using algebraic-iff-represents by blast
    thus ?thesis
        using the1-equality[OF algebraic-imp-represents-unique[OF x], of p] assms
        unfolding min-int-poly-def by auto
qed
lemma min-int-poly-of-int [simp]:
    min-int-poly (of-int n :: 'a :: {field-char-0, field-gcd}) = [:-of-int n, 1:]
    by (intro min-int-poly-unique irreducible-linear-poly) auto
lemma min-int-poly-of-nat [simp]:
    min-int-poly (of-nat n :: 'a :: {field-char-0, field-gcd}) = [:-of-nat n, 1:]
    using min-int-poly-of-int[of int n] by (simp del: min-int-poly-of-int)
lemma min-int-poly-0 [simp]: min-int-poly (0 :: 'a :: { field-char-0, field-gcd}) =
[:0, 1:]
    using min-int-poly-of-int[of 0] unfolding of-int-0 by simp
lemma min-int-poly-1 [simp]: min-int-poly (1 :: 'a :: {field-char-0, field-gcd})=
[:-1, 1:]
    using min-int-poly-of-int[of 1] unfolding of-int-1 by simp
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 \longleftrightarrowx=0
proof
    assume *: poly (min-int-poly x) 0=0
    show }x=
    proof (rule ccontr)
        assume x\not=0
        hence poly (min-int-poly x) 0}\not=
            using assms by (intro represents-irr-non-0) auto
        with * show False by contradiction
    qed
qed auto
lemma min-int-poly-eqI:
    fixes }x:: 'a :: {field-char-0, field-gcd
    assumes p represents x irreducible p lead-coeff p\geq0
    shows min-int-poly x = p
```

```
proof -
    from assms have [simp]: p\not=0
        by auto
    have lead-coeff p}\not=
        by auto
    with assms(3) have lead-coeff p>0
        by linarith
    moreover have algebraic x
        using <p represents x> by (meson algebraic-iff-represents)
    ultimately show ?thesis
        unfolding min-int-poly-def
    using the1-equality[OF algebraic-imp-represents-unique[OF <algebraic x〉], of p]
assms by auto
qed
Implementation for real and rational numbers
lemma min-int-poly-of-rat: min-int-poly (of-rat r :: 'a :: {field-char-0, field-gcd})
= poly-rat r
    by (intro min-int-poly-unique, auto)
definition min-int-poly-real :: real }=>\mathrm{ 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
    by simp
lemma min-int-poly-real-basic-impl[code]: min-int-poly-real (real-of-rat x)=poly-rat
x
    unfolding min-int-poly-real-def by (rule min-int-poly-of-rat)
lemma min-int-poly-rat-code-unfold [code-unfold]: min-int-poly = poly-rat
    by (intro ext, insert min-int-poly-of-rat[where ?' a = rat], auto)
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 $g c d$-int-poly is not preferable to the default implementation of $g c d$, 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 (intro iffI conjI, unfold conj-imp-eq-imp-imp)
assume Ex1 P P $x$ then show Collect $P=\{x\}$ by blast
next
assume Px: Collect $P=\{x\}$
then have $P y \longleftrightarrow x=y$ for $y$ by auto
then show Ex1 P by auto
from $P x$ show $P x$ by auto
qed
lemma ex1-Collect-singleton[consumes 2]:
assumes $\exists!x . P x$ and $P x$ and Collect $P=\{x\} \Longrightarrow$ thesis shows thesis
by (rule assms(3), subst ex1-imp-Collect-singleton[symmetric], insert assms(1,2), auto)
lemma ex1-iff-Collect-singleton: $P x \Longrightarrow(\exists!x . P x) \longleftrightarrow$ Collect $P=\{x\}$
by (subst ex1-imp-Collect-singleton[symmetric], auto)
context
fixes $f$
assumes bij: bijf
begin
lemma bij-imp-ex1-iff: $(\exists!x . P(f x)) \longleftrightarrow(\exists!y . P y)$ (is ?l $=? r)$
proof (intro iffI)
assume $l$ : ?l
then obtain $x$ where $P(f x)$ by auto
with $l$ have $*:\{x\}=$ Collect $(P o f)$ by auto
also have $f$ ' $\ldots=\{y . P(f($ Hilbert-Choice.inv $f y))\}$ using bij-image-Collect-eq[OF
bij] by auto
also have $\ldots=\{y . P y\}$
proof-
have $f$ (Hilbert-Choice.inv $f y)=y$ for $y$ by (meson bij bij-inv-eq-iff)
then show ?thesis by simp
qed
finally have Collect $P=\{f x\}$ by auto
then show ?r by (fold ex1-imp-Collect-singleton, auto)

```
next
    assume r: ?r
    then obtain y where P y by auto
    with }r\mathrm{ have {y}= Collect P by auto
    also have Hilbert-Choice.inv f' ... = Collect ( }P\circf\mathrm{ )
    using bij-image-Collect-eq[OF bij-imp-bij-inv[OF bij]] bij by (auto simp: inv-inv-eq)
    finally have Collect (Pof)={Hilbert-Choice.inv fy} by (simp add:o-def)
    then show ?l by (fold ex1-imp-Collect-singleton, auto)
qed
lemma bij-ex1-imp-the-shift:
    assumes ex1: \exists!y.P y shows (THE x. P (f x)) = Hilbert-Choice.inv f (THE
y.Py)(is ?l = ?r)
proof-
    from ex1 have P(THE y. P y) by (rule the1I2)
    moreover from ex1[folded bij-imp-ex1-iff] have P(f(THE x.P (fx))) by (rule
the1I2)
    ultimately have (THE y.P y) =f(THE x. P (fx)) using ex1 by auto
    also have Hilbert-Choice.inv f \ldots.=(THE x.P (fx)) using bij by (simp add:
bij-is-inj)
    finally show ?l = ?r by auto
qed
lemma bij-imp-Collect-image: {x.P (fx)}=Hilbert-Choice.inv f'{y.P y} (is ?l
=?g'-)
proof-
    have ?l = ?g'f' ?l by (simp add: image-comp inv-o-cancel[OF bij-is-inj[OF
bij]])
    also have f'?l ={fx|x.P(fx)} by auto
    also have \ldots. ={y.Py} by (metis bij bij-iff)
    finally show ?thesis.
qed
lemma bij-imp-card-image: card ( f` X) = card X
    by (metis bij bij-iff card.infinite finite-imageD inj-onI inj-on-iff-eq-card)
end
definition poly-cond :: int poly }=>\mathrm{ 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 using assms by
(auto simp: poly-cond-def)
lemma poly-condD:
    assumes poly-cond p
    shows irreducible p and lead-coeff p>0 and root-free p and square-free p and
p\not=0
```

using assms unfolding poly-cond-def using irreducible-root-free irreducible-imp-square-free cf-pos-def by auto

```
lemma poly-condE[elim]:
    assumes poly-cond p
    and irreducible p\Longrightarrow lead-coeff p>0 root-free p\Longrightarrow square-free p 
p\not=0\Longrightarrow thesis
    shows thesis
    using assms by (auto dest:poly-condD)
lemma poly-cond-abs-int-poly[simp]: irreducible p \Longrightarrow poly-cond (abs-int-poly p)
    unfolding poly-cond-def by (cases p = 0, auto)
```

definition poly-uminus-abs :: int poly $\Rightarrow$ int poly where
poly-uminus-abs $p=a b s$-int-poly (poly-uminus $p$ )
lemma irreducible-poly-uminus[simp]: irreducible $p \Longrightarrow$ irreducible (poly-uminus
( $p$ :: int poly))
proof (cases degree $p=0$ )
case True
from degree 0-coeffs[OF this]
obtain $a$ where $p: p=[: a:]$ by auto
have poly-uminus $p=p$ unfolding $p$ by (cases $a=0$, auto)
thus irreducible $p \Longrightarrow$ irreducible (poly-uminus $p$ ) by auto
next
case False
from poly-uminus-irreducible $[O F-t h i s]$
show irreducible $p \Longrightarrow$ irreducible (poly-uminus $p$ ).
qed
lemma irreducible-poly-uminus-abs[simp]: irreducible $p \Longrightarrow$ irreducible (poly-uminus-abs
p)
unfolding poly-uminus-abs-def using irreducible-poly-uminus $[o f ~ p]$ by auto
lemma poly-cond-poly-uminus-abs[simp]: poly-cond $p \Longrightarrow$ poly-cond (poly-uminus-abs
p)
by (auto simp: poly-cond-def, unfold poly-uminus-abs-def, subst pos-poly-abs-poly,
auto)
lemma ipoly-poly-uminus-abs-zero[simp]: ipoly (poly-uminus-abs p) ( $x$ :: 'a :: idom)
$=0 \longleftrightarrow$ ipoly $p(-x)=0$
unfolding poly-uminus-abs-def by simp
lemma degree-poly-uminus-abs[simp]: degree (poly-uminus-abs $p$ ) $=$ degree $p$
unfolding poly-uminus-abs-def by auto
definition poly-inverse :: int poly $\Rightarrow$ int poly where
poly-inverse $p=a b s-i n t-p o l y(r e f l e c t-p o l y ~ p)$
lemma irreducible-poly-inverse[simp]: coeff $p 0 \neq 0 \Longrightarrow$ irreducible $p \Longrightarrow$ irreducible (poly-inverse p)
unfolding poly-inverse-def by (auto simp: irreducible-reflect-poly)
lemma degree-poly-inverse[simp]: coeff p $0 \neq 0 \Longrightarrow$ degree (poly-inverse $p$ ) $=$ degree $p$
unfolding poly-inverse-def by auto
lemma ipoly-poly-inverse $[$ simp $]$ : assumes coeff $p 0 \neq 0$
shows ipoly (poly-inverse $p$ ) ( $x::$ ' $a$ :: field-char- 0 ) $=0 \longleftrightarrow$ ipoly $p$ (inverse $x$ ) $=0$
unfolding poly-inverse-def ipoly-abs-int-poly-eq-zero-iff
proof (cases $x=0$ )
case False
thus $($ ipoly $($ reflect-poly $p) x=0)=($ ipoly $p($ inverse $x)=0)$
by (subst ipoly-reflect-poly, auto)
next
case True
show (ipoly (reflect-poly $p$ ) $x=0)=($ ipoly $p($ inverse $x)=0)$ unfolding True using assms by (auto simp: poly-0-coeff-0)
qed
lemma ipoly-roots-finite: $p \neq 0 \Longrightarrow$ finite $\left\{x::{ }^{\prime} a::\{\right.$ idom, ring-char-0 $\}$. ipoly $p$ $x=0$ \}
by (rule poly-roots-finite, simp)
lemma root-sign-change: assumes
p0: poly ( $p:$ :real poly) $x=0$ and
pd-ne0: poly (pderiv $p$ ) $x \neq 0$
obtains $d$ where
$0<d$
$\operatorname{sgn}($ poly $p(x-d)) \neq \operatorname{sgn}($ poly $p(x+d))$
$\operatorname{sgn}($ poly $p(x-d)) \neq 0$
$0 \neq \operatorname{sgn}($ poly $p(x+d))$
$\forall d^{\prime}>0 . d^{\prime} \leq d \longrightarrow \operatorname{sgn}\left(\right.$ poly $\left.p\left(x+d^{\prime}\right)\right)=\operatorname{sgn}($ poly $p(x+d)) \wedge \operatorname{sgn}($ poly
$\left.p\left(x-d^{\prime}\right)\right)=\operatorname{sgn}($ poly $p(x-d))$
proof -
assume $a:(\bigwedge d .0<d \Longrightarrow$
$\operatorname{sgn}(\operatorname{poly} p(x-d)) \neq \operatorname{sgn}($ poly $p(x+d)) \Longrightarrow$
$\operatorname{sgn}(\operatorname{poly} p(x-d)) \neq 0 \Longrightarrow$
$0 \neq \operatorname{sgn}(\operatorname{poly} p(x+d)) \Longrightarrow$
$\forall d^{\prime}>0 . d^{\prime} \leq d \longrightarrow$
$\operatorname{sgn}\left(\right.$ poly $\left.p\left(x+d^{\prime}\right)\right)=\operatorname{sgn}($ poly $p(x+d)) \wedge \operatorname{sgn}\left(\right.$ poly $\left.p\left(x-d^{\prime}\right)\right)$
$=\operatorname{sgn}($ poly $p(x-d)) \Longrightarrow$
thesis)
from pd-ne0 consider poly (pderiv p) $x>0 \mid \operatorname{poly}($ pderiv $p) x<0$ by linarith
thus ?thesis proof (cases)
case 1
obtain $d 1$ where $d 1: \wedge h .0<h \Longrightarrow h<d 1 \Longrightarrow$ poly $p(x-h)<0 d 1>0$ using DERIV-pos-inc-left[OF poly-DERIV 1] p0 by auto
obtain d2 where $d 2: \wedge h .0<h \Longrightarrow h<d 2 \Longrightarrow$ poly $p(x+h)>0 d 2>0$ using DERIV-pos-inc-right[OF poly-DERIV 1] p0 by auto
have $g 0: 0<(\min d 1 d 2) / 2$ using $d 1 d 2$ by auto
hence m1:min d1 d2 / $2<d 1$ and m2:min d1 d2 / $2<d 2$ by auto $\{$ fix $d$
assume $a 1: 0<d$ and $a 2: d<\min d 1 d 2$
have $\operatorname{sgn}($ poly $p(x-d))=-1 \operatorname{sgn}($ poly $p(x+d))=1$ using $d 1$ (1) [OF a1] $d 2(1)\left[\begin{array}{ll}\text { F } & \text { a1] }] \text { a2 by auto }\end{array}\right.$
\} note $d=$ this
show ?thesis by (rule a[OF g0];insert d g0 m1 m2, simp)

## next

case 2
obtain $d 1$ where $d 1: \backslash h .0<h \Longrightarrow h<d 1 \Longrightarrow$ poly $p(x-h)>0 d 1>0$ using DERIV-neg-dec-left[OF poly-DERIV 2] p0 by auto
obtain d2 where d2: $\backslash h .0<h \Longrightarrow h<d 2 \Longrightarrow$ poly $p(x+h)<0 d 2>0$ using DERIV-neg-dec-right[OF poly-DERIV 2] p0 by auto
have $g 0: 0<(\min d 1 d 2) / 2$ using $d 1 d 2$ by auto
hence m1:min d1 d2 / 2 2 d1 and m2: min $d 1$ d2 / $2<d 2$ by auto
\{ fix $d$
assume a1:0<d and a2: $d<\min d 1 d 2$
have $\operatorname{sgn}($ poly $p(x-d))=1 \operatorname{sgn}($ poly $p(x+d))=-1$
using $d 1$ (1) [OF a1] d2(1)[OF a1] a2 by auto
\} note $d=$ this
show ?thesis by(rule a[OF g0];insert d g0 m1 m2, simp)
qed
qed
lemma gt-rat-sign-change-square-free:
assumes ur: $\exists$ ! x. root-cond plr $x$
and $p l r[s i m p]: p l r=(p, l, r)$
and sf: square-free $p$ and in-interval: $l \leq y y \leq r$
and py0: ipoly $p y \neq 0$ and pr0: ipoly $p r \neq 0$
shows $(\operatorname{sgn}($ ipoly $p y)=\operatorname{sgn}($ ipoly $p r))=($ of-rat $y>($ THE x. root-cond plr
x)) $($ is $? g t=-)$
proof (rule ccontr)
define $u r$ where $u r=(T H E x$. root-cond plr $x)$
assume $\neg$ ?thesis
hence ? gt $\neq($ real-of-rat $y>u r)$ unfolding ur-def by auto
note $a=$ this[unfolded plr]
from py0 have $p \neq 0$ unfolding irreducible-def by auto
hence $p 0$-real: real-of-int-poly $p \neq(0::$ real poly $)$ by auto
let $? p=$ real-of-int-poly $p$
let $? r=$ real-of-rat
from in-interval have in':?r $l \leq$ ?r $y$ ?r $y \leq$ ?r $r$ unfolding of-rat-less-eq by auto
from sf square-free-of-int-poly[of p] square-free-rsquarefree

```
    have rsf:rsquarefree ?p by auto
    from ur have root-cond plr ur by (metis ur-def theI')
    note urD = this[unfolded root-cond-def plr split] this[unfolded plr]
    have ur3:poly ?p ur = 0 using urD by auto
    from urD have ur \leqof-rat r by auto
    moreover
    from pr0 have ipoly p (real-of-rat r)}\not=0\mathrm{ by auto
    with ur3 have real-of-rat r\not=ur by force
    ultimately have ur < ?r r by auto
    hence ur2: 0 < ?r r - ur by linarith
    from rsquarefree-roots rsf ur3
    have pd-nonz:poly (pderiv ?p) ur }\not=0\mathrm{ by auto
    obtain d}\mathrm{ where d}\mp@subsup{d}{}{\prime}:\bigwedge\mp@subsup{d}{}{\prime}..\mp@subsup{d}{}{\prime}>0\Longrightarrow\mp@subsup{d}{}{\prime}\leqd
                        sgn }(\mathrm{ poly ?p (ur + d'}))=\operatorname{sgn}(\mathrm{ poly ?p }(ur+d))
                            sgn (poly ?p (ur - d')) = sgn (poly ?p (ur - d))
    sgn (poly ?p (ur - d)) = sgn (poly ?p (ur + d))
    sgn (poly ?p (ur + d))}\not=
    and d-ge-0:d>0
    by (metis root-sign-change[OF ur3 pd-nonz])
    have sr:sgn (poly ?p (ur +d)) = sgn (poly ?p (?r r))
    proof (cases ?r r - ur \leqd)
    case True show ?thesis using d'(1)[OF ur2 True] by auto
next
    case False hence less:ur + d<? ? r by auto
    show ?thesis
    proof(rule no-roots-inbetween-imp-same-sign[OF less,rule-format],goal-cases)
        case (1 x)
        from ur 1 d-ge-0 have ran: real-of-rat l \leq x x seal-of-rat r using urD by
auto
            from 1 d-ge-0 have ur }\not=x\mathrm{ by auto
            with ur urD have }\neg\mathrm{ root-cond ( }p,l,r)x\mathrm{ by (auto simp: root-cond-def)
            with ran show ?case by (auto simp: root-cond-def)
        qed
    qed
    consider ?r l<ur - d ?r l<ur | 0<ur - ?r l ur - ?r l\leqd |ur=?r l
    using urD by argo
    hence sl:sgn (poly ?p (ur - d)) = sgn (poly ?p (?r l))\vee 0 = sgn (poly ?p (?r
l))
    proof (cases)
    case 1
    have sgn (poly ?p (?r l)) = sgn (poly ?p (ur - d))
    proof(rule no-roots-inbetween-imp-same-sign[OF 1(1),rule-format],goal-cases)
        case (1 x)
        from ur 1 d-ge-0 urD have ran: real-of-rat l \leqx x \leq real-of-rat r by auto
        from 1 d-ge-0 have ur }\not=x\mathrm{ by auto
        with ur urD have }\neg\mathrm{ root-cond ( p,l,r) x by (auto simp: root-cond-def)
        with ran show ?case by (auto simp: root-cond-def)
    qed
    thus ?thesis by auto
```

```
    next
    case 2 show ?thesis using d'(1)[OF 2] by simp
    qed (insert ur3,simp)
    have diff-sign: sgn (ipoly p l)}\not=\operatorname{sgn}(\mathrm{ ipoly p r)
    using d'(2-) sr sl real-of-rat-sgn by auto
    have ur':\x. real-of-rat l\leqx^x\leq real-of-rat y m ipoly p x=0\Longrightarrow\neg(?r y
\lequr)
    proof(standard +,goal-cases)
    case (1 x)
    {
        assume id: ur = ?r y
        with urD ur py0 have False by auto
    } note neq = this
    have x:root-cond (p,l,r)x unfolding root-cond-def
        using 1 a ur urD by auto
    from ur urD x have ur-eqI:ur =x
        by auto
    with 1 have ur=of-rat y by auto
    with urD(1) py0 show False by auto
qed
    hence ur'':}\forallx\mathrm{ . real-of-rat }y\leqx\wedgex\leqreal-of-rat r \longrightarrow poly (real-of-int-poly p
x\not=0\Longrightarrow\neg (?r y \lequr)
    using urD by auto
    have (sgn (ipoly p y)=sgn (ipoly pr))=(?r y > ur)
    proof(cases sgn (ipoly pr)=sgn (ipoly p y))
    case True
    have sgn:sgn (poly ?p (real-of-rat l))}\not=\operatorname{sgn}(\mathrm{ poly ?p (real-of-rat y)) using True
diff-sign
            by (simp add: real-of-rat-sgn)
    have ly:of-rat l < (of-rat y::real) using in-interval True diff-sign less-eq-rat-def
of-rat-less by auto
    with no-roots-inbetween-imp-same-sign[OF ly,of ?p] sgn ur' True
    show ?thesis by force
    next
    case False
    hence ne:sgn (ipoly p (real-of-rat y)) = sgn (ipoly p (real-of-rat r)) by (simp
add: real-of-rat-sgn)
    have ry:of-rat y<(of-rat r::real) using in-interval False diff-sign less-eq-rat-def
of-rat-less by auto
    obtain x where x:real-of-rat y \leq x x seal-of-rat r ipoly p x = 0
            using no-roots-inbetween-imp-same-sign[OF ry,of ?p] ne by auto
    hence lx:real-of-rat l \leqx using in-interval
        using False a urD by auto
    with }x\mathrm{ have root-cond (p,l,r) x by (auto simp: root-cond-def)
    with urD ur
    have ur = x by auto
    then show ?thesis using False x by auto
qed
thus False using diff-sign(1) a py0 by(cases ipoly p r = 0;auto simp:sgn-0-0)
```


## qed

definition algebraic-real :: real $\Rightarrow$ bool where
[simp]: algebraic-real = algebraic
lemma algebraic-real-iff[code-unfold]: algebraic = algebraic-real by simp
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 \(\Rightarrow\) int \(\Rightarrow\) int where
    div-ceiling \(x y=(\) let \(q=x\) div \(y\) in if \(q * y=x\) then \(q\) else \(q+1)\)
definition root-bound \(::\) int poly \(\Rightarrow\) rat where
    root-bound \(p \equiv\) let
        \(n=\) degree \(p\);
        \(m=1+\) div-ceiling \((\) max-list-non-empty \((\operatorname{map}(\lambda i . a b s(\operatorname{coeff} p i))[0 . .<n]))\)
            (abs (lead-coeff p))
        - round to the next higher number \(2 \wedge n\), so that bisection will
        - stay on integers for as long as possible
    in of-int (2 \({ }^{\wedge}\) (log-ceiling 2 m) )
```

lemma root-imp-deg-nonzero: assumes $p \neq 0$ poly $p x=0$
shows degree $p \neq 0$
proof
assume degree $p=0$
from degree 0 -coeffs[OF this] assms show False by auto
qed
lemma cauchy-root-bound: fixes $x::$ ' $a$ :: real-normed-field
assumes $x$ : poly $p x=0$ and $p: p \neq 0$
shows norm $x \leq 1+$ max-list-non-empty $(\operatorname{map}(\lambda i$. norm $(\operatorname{coeff} p i))[0 . .<$
degree $p]$ )
/ norm (lead-coeff p) (is - $\leq-+$ ?max / ?nlc)

```
proof -
    let \(? n=\) degree \(p\)
    let \(? p=\) coeff \(p\)
    let ?lc \(=\) lead-coeff \(p\)
    define \(m l\) where \(m l=\) ? max / ?nlc
    from \(p\) have \(l c: ? l c \neq 0\) by auto
    hence nlc: norm ?lc \(>0\) by auto
    from root-imp-deg-nonzero \([O F \quad p x]\) have \(*: 0 \in \operatorname{set}[0 . .<\) degree \(p]\) by auto
    have \(0 \leq \operatorname{norm}(? p\) 0) by \(\operatorname{simp}\)
    also have ... \(\leq\) ? max
        by (rule max-list-non-empty, insert *, auto)
    finally have \(\max 0\) : ? \(\max \geq 0\).
    with \(n l c\) have \(m l 0: m l \geq 0\) unfolding \(m l-d e f\) by auto
    hence easy: norm \(x \leq 1 \Longrightarrow\) ?thesis unfolding ml-def[symmetric] by auto
    show ?thesis
    proof (cases norm \(x \leq 1\) )
        case True
        thus ?thesis using easy by auto
    next
        case False
        hence nx: norm \(x>1\) by simp
    hence \(x 0: x \neq 0\) by auto
    hence xn0: \(0<\) norm \(x\) ? \(n\) by auto
    from \(x\left[\right.\) unfolded poly-altdef] have \(x^{\wedge} ? n * ? l c=x^{\wedge} ? n * ? l c-\left(\sum i \leq ? n . x\right.\)
^ \(i * ? p i)\)
        unfolding poly-altdef by (simp add: ac-simps)
    also have \(\left(\sum i \leq ? n . x^{\wedge} i * ? p i\right)=x^{\wedge} ? n * ? l c+\left(\sum i<? n . x^{\wedge} i * ? p i\right)\)
        by (subst sum.remove [of - ? \(n\) ], auto intro: sum.cong)
    finally have \(x^{\wedge}\) ? \(n * ? l c=-\left(\sum i<? n . x^{\wedge} i * ? p i\right)\) by simp
        with \(l c\) have \(x^{\wedge} ? n=-\left(\sum i<? n . x^{\wedge} i * ? p i\right) / ? l c\) by (simp add:
field-simps)
        from arg-cong[OF this, of norm]
        have norm \(x^{\wedge}\) ? \(n=\operatorname{norm}\left(\left(\sum i<? n . x^{\wedge} i * ? p i\right) / ? l c\right)\) unfolding
norm-power by simp
    also have \(\left(\sum i<\right.\) ?n. \(\left.x^{\wedge} i * ? p i\right) / ? l c=\left(\sum i<? n . x^{\wedge} i * ? p i / ? l c\right)\)
        by (rule sum-divide-distrib)
    also have norm \(\ldots \leq\left(\sum i<? n\right.\). norm \(\left.\left(x^{\wedge} i *(? p i / ? l c)\right)\right)\)
        by (simp add: field-simps, rule norm-sum)
    also have \(\ldots=\left(\sum i<\right.\) ?n. norm \(\left.x{ }^{\wedge} i * \operatorname{norm}(? p i / ? l c)\right)\)
        unfolding norm-mult norm-power ..
    also have \(\ldots \leq\left(\sum i<\right.\) ? \(n\). norm \(\left.x{ }^{\wedge} i * m l\right)\)
    proof (rule sum-mono)
        fix \(i\)
        assume \(i \in\{. .<? n\}\)
        hence \(i: i<? n\) by simp
        show norm \(x^{\wedge} i *\) norm (?p \(\left.i / ? l c\right) \leq\) norm \(x{ }^{\wedge} i * m l\)
        proof (rule mult-left-mono)
            show \(0 \leq\) norm \(x{ }^{\wedge} i\) using \(n x\) by auto
            show norm (?p \(i / ? l c) \leq m l\) unfolding norm-divide ml-def
```

```
                by (rule divide-right-mono[OF max-list-non-empty], insert nlc i, auto)
        qed
    qed
    also have ... = ml* (\sumi< ?n. norm x^ i)
        unfolding sum-distrib-right[symmetric] by simp
    also have (\sumi< ?n. norm x^ i)=(norm x^ ?n - 1) / (norm x - 1)
        by (rule geometric-sum, insert nx, auto)
    finally have norm x^`?n\leqml*(norm x^ ?n - 1)/(norm x - 1) by simp
    from mult-left-mono[OF this, of norm x - 1]
    have (norm x - 1)*(norm x^? ?n) \leqml*(norm x^? n n - 1) using nx by
auto
    also have \ldots=(ml*(1-1/(norm x^? n)))* norm x^ ? n
        using nx False x0 by (simp add: field-simps)
    finally have (norm x - 1)*(norm x^? n) \leq(ml*(1-1/(norm x^?n))}
* norm x^ ?n.
    from mult-right-le-imp-le[OF this xn0]
    have norm x - 1\leqml*(1-1/(norm x^ ?n)) by simp
    hence norm x\leq1+ml-ml/(norm x^? ? ) by (simp add: field-simps)
    also have ... \leq1+ml using ml0 xn0 by auto
    finally show ?thesis unfolding ml-def .
    qed
qed
lemma div-le-div-ceiling: x div y \leqdiv-ceiling x y
    unfolding div-ceiling-def Let-def by auto
lemma div-ceiling: assumes q: q\not=0
    shows (of-int x :: 'a :: floor-ceiling) / of-int q \leq of-int (div-ceiling x q)
proof (cases q dvd x)
    case True
    then obtain k where xqk:x=q*k unfolding dvd-def by auto
    hence id: div-ceiling x q=k unfolding div-ceiling-def Let-def using q by auto
    show ?thesis unfolding id unfolding xqk using q by simp
next
    case False
    {
        assume x div q*q=x
        hence }x=q*(x\mathrm{ div q) by (simp add: ac-simps)
        hence qdvd x unfolding dvd-def by auto
        with False have False by simp
    }
    hence id: div-ceiling x q = x div q+1
        unfolding div-ceiling-def Let-def using q by auto
    show ?thesis unfolding id
    by (metis floor-divide-of-int-eq le-less add1-zle-eq floor-less-iff)
qed
lemma max-list-non-empty-map: assumes hom: \ x y. max (fx) (fy)=f(max
```

```
x y)
    shows xs \not=[]\Longrightarrow max-list-non-empty (map f xs) = f (max-list-non-empty xs)
    by (induct xs rule: max-list-non-empty.induct, auto simp: hom)
lemma root-bound: assumes root-bound p = B and deg: degree p>0
    shows ipoly p (x :: 'a :: real-normed-field ) = 0 \Longrightarrow norm }x\leq\mathrm{ of-rat B B }\geq
proof -
    let ?r = of-rat :: - = 'a
    let ?i =of-int :: - = ' }
    let ?p = map-poly ?i p
    define }n\mathrm{ where }n=\mathrm{ degree }
    let ?lc = coeff p n
    let ?list = map (\lambdai. abs (coeff p i)) [0..<n]
    let ?list' = (map (\lambdai.real-of-int (abs ((coeff p i)))) [0..<n])
    define m}\mathrm{ where m= max-list-non-empty ?list
    define m-up where m-up=1 + div-ceiling m (abs ?lc)
    define C where C=rat-of-int (2`(log-ceiling 2 m-up))
    from deg have p0:p\not=0 by auto
    from p0 have alc0:abs ?lc \not=0 unfolding n-def by auto
    from deg have mem: abs (coeff p 0) \in set ?list unfolding n-def by auto
    from max-list-non-empty[OF this, folded m-def]
    have m0: m\geq0 by auto
    have div-ceiling m (abs ?lc)\geq0
    by (rule order-trans[OF - div-le-div-ceiling[of m abs ?lc]], subst
    pos-imp-zdiv-nonneg-iff, insert p0 m0, auto simp: n-def)
    hence mup: m-up \geq1 unfolding m-up-def by auto
    have m-up \leq2 ^(log-ceiling 2 m-up) using mup log-ceiling-sound(1) by auto
    hence Cmup:C\geqof-int m-up unfolding C-def by linarith
    with mup have C:C\geq1 by auto
    from assms(1)[unfolded root-bound-def Let-def]
    have B:C=B unfolding C-def m-up-def n-def m-def by auto
    note dc = div-le-div-ceiling[of m abs ?lc]
    with C show }B\geq0\mathrm{ unfolding B by auto
    assume ipoly p x=0
    hence rt: poly ?p x=0 by simp
    from root-imp-deg-nonzero[OF - this] p0 have n0: n}\not=0\mathrm{ unfolding n-def by
auto
    from cauchy-root-bound[OF rt] p0
    have norm x \leq 1 + max-list-non-empty ?list' / real-of-int (abs ?lc)
        by (simp add: n-def)
    also have ?list' = map real-of-int ?list by simp
    also have max-list-non-empty ... = real-of-int m unfolding m-def
        by (rule max-list-non-empty-map, insert mem, auto)
    also have 1 +m/real-of-int (abs ?lc)\leq real-of-int m-up
        unfolding m-up-def using div-ceiling[OF alc0, of m] by auto
    also have ... \leqreal-of-rat C using Cmup using of-rat-less-eq by force
    finally have norm x \leq real-of-rat C .
    thus norm x < real-of-rat B unfolding B by simp
qed
```

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 (intro iffI conjI, unfold conj-imp-eq-imp-imp)
assume Ex1 P P x then show Collect $P=\{x\}$ by blast
next
assume Px: Collect $P=\{x\}$
then have $P y \longleftrightarrow x=y$ for $y$ by auto
then show Ex1 P by auto
from $P x$ show $P x$ by auto
qed
lemma ex1-Collect-singleton[consumes 2]:
assumes $\exists!x . P x$ and $P x$ and Collect $P=\{x\} \Longrightarrow$ thesis shows thesis
by (rule assms(3), subst ex1-imp-Collect-singleton[symmetric], insert assms (1, 2),
auto)
lemma ex1-iff-Collect-singleton: $P x \Longrightarrow(\exists!x . P x) \longleftrightarrow$ Collect $P=\{x\}$
by (subst ex1-imp-Collect-singleton[symmetric], auto)
lemma bij-imp-card: assumes bij: bij $f$ shows card $\{x . P(f x)\}=\operatorname{card}\{x . P x\}$
unfolding bij-imp-Collect-image $[O F$ bij] bij-imp-card-image $[O F$ bij-imp-bij-inv[OF
bij]]..

```
lemma bij-add: bij ( \(\lambda x . x+y\) :: 'a :: group-add) (is ?g1)
    and bij-minus: bij ( \(\lambda x . x-y::{ }^{\prime} a\) ) (is ? \(g 2\) )
    and inv-add[simp]: Hilbert-Choice.inv \((\lambda x . x+y)=(\lambda x . x-y)(\) is ?g3)
```

```
    and inv-minus[simp]: Hilbert-Choice.inv ( }\lambdax.x-y)=(\lambdax.x+y)(is?g4
proof-
    have 1:(\lambdax.x-y)\circ}(\lambdax.x+y)=id and 2: (\lambdax.x+y)\circ(\lambdax.x-y)=i
by auto
    from o-bij[OF 1 2] show ?g1.
    from o-bij[OF 2 1] show ?g2.
    from inv-unique-comp[OF 2 1] show ?g3.
    from inv-unique-comp[OF 1 2] show ?g4.
qed
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\mathrm{ :: 'a :: group-add. P y
    shows (THE x.P (x+d)) = (THE y.P y) -d
    and (THE x.P (x-d)) =(THE y.P y) +d
    unfolding bij-ex1-imp-the-shift[OF bij-add ex1] bij-ex1-imp-the-shift[OF bij-minus
ex1] by auto
lemma card-shift-image[simp]:
    shows card ((\lambdax :: 'a :: group-add. x + d)' X) = card X
        and card ((\lambdax. x-d)'X) = card X
    by (auto simp: bij-imp-card-image[OF bij-add] bij-imp-card-image[OF bij-minus])
lemma irreducible-root-free:
    fixes p :: ' }a\mathrm{ :: {idom,comm-ring-1} poly
    assumes irr: irreducible p shows root-free p
proof (cases degree p 1::nat rule: linorder-cases)
    case greater
    {
        fix }
        assume poly p x=0
        hence [:-x,1:] dvd p using poly-eq-0-iff-dvd by blast
        then obtain r where p:p=r*[:-x,1:] by (elim dvdE, auto)
    have deg: degree [:-x,1:]=1 by simp
    have dvd:\neg[:-x,1:] dvd 1 by (auto simp: poly-dvd-1)
        from greater have degree }r\not=0\mathrm{ using degree-mult-le[of r [:-x,1:], unfolded
deg, folded p] by auto
    then have \negr dvd 1 by (auto simp: poly-dvd-1)
    with p irr irreducibleD[OF irr p] dvd have False by auto
    }
    thus ?thesis unfolding root-free-def by auto
next
    case less then have deg: degree p=0 by auto
    from deg obtain p0 where p: p=[:p0:] using degree0-coeffs by auto
    with irr have }p\not=0\mathrm{ by auto
    with p}\mathrm{ have poly px}=0\mathrm{ for }x\mathrm{ by auto
    thus ?thesis by (auto simp: root-free-def)
qed (auto simp: root-free-def)
```


### 11.1 Real Algebraic Numbers - Innermost Layer

We represent a real algebraic number $\alpha$ by a tuple ( $\mathrm{p}, \mathrm{l}, \mathrm{r}$ ): $\alpha$ is the unique root in the interval $[1, r]$ 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. ( $\ldots, \mathrm{x}-3,3,3$ ) is equivalent to ( $\ldots, \mathrm{x}-3,2,10$ ).

### 11.1.1 Basic Definitions

type-synonym real-alg-1 $=$ int poly $\times r a t \times r a t$
fun poly-real-alg-1 :: real-alg-1 $\Rightarrow$ int poly where poly-real-alg-1 $(p,-,-)=p$
fun rai-ub :: real-alg-1 $\Rightarrow$ rat where rai-ub $(-,-, r)=r$
fun rai-lb :: real-alg-1 $\Rightarrow$ rat where rai-lb $(-, l,-)=l$
abbreviation roots-below $p x \equiv\{y::$ real. $y \leq x \wedge$ ipoly $p y=0\}$
abbreviation(input) unique-root :: real-alg-1 $\Rightarrow$ bool where
unique-root plr $\equiv(\exists!x$. root-cond plr $x)$
abbreviation the-unique-root :: real-alg-1 $\Rightarrow$ real where
the-unique-root plr $\equiv($ THE $x$. root-cond plr $x)$
abbreviation real-of- 1 where real-of- $1 \equiv$ the-unique-root
lemma root-condI[intro]:
assumes of-rat (rai-lb plr) $\leq x$ and $x \leq o f-r a t(r a i-u b p l r)$ and ipoly (poly-real-alg-1 $p l r) x=0$
shows root-cond plr $x$
using assms by (auto simp: root-cond-def)
lemma root-condE[elim]:
assumes root-cond plr $x$ and of-rat (rai-lb plr) $\leq x \Longrightarrow x \leq$ of-rat (rai-ub plr) $\Longrightarrow$ ipoly (poly-real-alg-1 plr) $x=0 \Longrightarrow$ thesis
shows thesis
using assms by (auto simp: root-cond-def)

## lemma

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 $r \equiv r a i-u b p l r$
shows unique-rootD: of-rat $l \leq x x \leq o f$-rat $r$ ipoly $p x=0$ root-cond plr $x$ $x=y \longleftrightarrow$ root-cond plr $y y=x \longleftrightarrow$ root-cond plr $y$
and the-unique-root-eqI: root-cond plr $y \Longrightarrow y=x$ root-cond plr $y \Longrightarrow x=y$ proof -
from ur show $x$ : root-cond plr $x$ unfolding $x$-def by (rule theI')
have $p l r=(p, l, r)$ by (cases $p l r$, auto simp: $p$-def $l$-def $r$-def)
from $x[$ unfolded this] show of-rat $l \leq x x \leq o f$-rat $r$ ipoly $p x=0$ by auto
from $x$ ur
show root-cond plr $y \Longrightarrow y=x$ and root-cond plr $y \Longrightarrow x=y$
and $x=y \longleftrightarrow$ root-cond plr $y$ and $y=x \longleftrightarrow$ root-cond plr $y$ by auto
qed
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 $r \equiv r a i-u b p l r$
assumes main: of-rat $l \leq x \Longrightarrow x \leq$ of-rat $r \Longrightarrow$ ipoly $p x=0 \Longrightarrow$ root-cond plr $x \Longrightarrow$

$$
(\bigwedge y \cdot x=y \longleftrightarrow \text { root-cond plr } y) \Longrightarrow(\bigwedge y \cdot y=x \longleftrightarrow \text { root-cond plr } y) \Longrightarrow
$$ thesis

shows thesis by (rule main, unfold $x$-def $p$-def l-def r-def; rule unique-root $D[O F$ $u r])$
lemma unique-rootI:
assumes $\bigwedge y$. root-cond plr $y \Longrightarrow x=y$ root-cond plr $x$
shows unique-root plr using assms by blast
definition invariant-1 :: real-alg-1 $\Rightarrow$ bool where
invariant-1 tup $\equiv$ case tup of $(p, l, r) \Rightarrow$
unique-root $(p, l, r) \wedge \operatorname{sgn} l=\operatorname{sgn} r \wedge$ poly-cond $p$
lemma invariant-1I:
assumes unique-root plr and sgn (rai-lb plr) $=\operatorname{sgn}$ (rai-ub plr) and poly-cond (poly-real-alg-1 plr)
shows invariant-1 plr
using assms by (auto simp: invariant-1-def)

## lemma

assumes invariant-1 plr
defines $x \equiv$ 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 $x$
$\operatorname{sgn} l=\operatorname{sgn} r \operatorname{sgn} x=o f$-rat (sgn $r$ ) unique-root plr poly-cond $p$ degree $p>0$ primitive $p$
and invariant-1-root-cond: $\wedge y$. root-cond plr $y \longleftrightarrow y=x$
proof -
let ?l $=$ of-rat $l::$ real
let ? $r=o f-r a t r::$ real
have $p l r: p l r=(p, l, r)$ by (cases plr, auto simp: $p$-def l-def $r$-def)
from assms
show ur: unique-root plr and sgn: sgn $l=\operatorname{sgn} r$ and $p c$ : poly-cond $p$ by (auto simp: invariant-1-def)
from ur show rc: root-cond plr $x$ by (auto simp add: $x$-def plr intro: theI')

```
    from this[unfolded plr] have \(x\) : ipoly \(p x=0\) and bnd: ?l \(\leq x x \leq ? r\) by auto
    show \(\operatorname{sgn} x=o f\)-rat (sgn \(r\) )
    proof (cases \(0::\) real \(x\) rule:linorder-cases)
        case less
        with bnd(2) have \(0<\) ? \(r\) by arith
        thus ?thesis using less by simp
    next
    case equal
    with bnd have ?l \(\leq 0\) ? \(r \geq 0\) by auto
    hence \(l \leq 0 r \geq 0\) by auto
    with \(\langle s g n ~ l=s g n ~ r\rangle\) have \(l=0 r=0\) unfolding sgn-rat-def by (auto split:
if-splits)
    with rc[unfolded plr]
    show ?thesis by auto
    next
    case greater
    with \(b n d(1)\) have ?l \(<0\) by arith
    thus ?thesis unfolding «sgn \(l=\operatorname{sgn} r\rangle[\) symmetric \(]\) using greater by simp
    qed
    from the-unique-root-eqI[OF ur] rc
    show \(\wedge y\). root-cond plr \(y \longleftrightarrow y=x\) by metis
    \{
    assume degree \(p=0\)
    with poly-zero [OF \(x\), simplified \(]\) sgn bnd have \(p=0\) by auto
    with \(p c\) have False by auto
    \}
    then show degree \(p>0\) by auto
    with \(p c\) show primitive \(p\) by (intro irreducible-imp-primitive, auto)
qed
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 \(x \Longrightarrow\)
        \(\operatorname{sgn} l=\operatorname{sgn} r \Longrightarrow \operatorname{sgn} x=\) of-rat \((\operatorname{sgn} r) \Longrightarrow\) unique-root \(p l r \Longrightarrow\) poly-cond \(p\)
\(\Longrightarrow\) degree \(p>0 \Longrightarrow\)
        primitive \(p \Longrightarrow\) thesis
    shows thesis apply (rule main)
    using assms(1) unfolding \(x\)-def \(p\)-def l-def r-def by (auto dest: invariant-1D)
lemma invariant-1-realI:
    fixes plr :: real-alg-1
    defines \(p \equiv\) poly-real-alg-1 plr and \(l \equiv\) rai-lb plr and \(r \equiv\) rai-ub plr
    assumes \(x\) : root-cond plr \(x\) and \(\operatorname{sgn} l=\operatorname{sgn} r\)
        and ur: unique-root plr
        and poly-cond \(p\)
    shows invariant-1 plr \(\wedge\) real-of- 1 plr \(=x\)
    using the-unique-root-eqI[OF ur x] assms by (cases plr, auto intro: invariant-1I)
```

```
lemma real-of-1-0:
    assumes invariant-1 ( \(p, l, r\) )
    shows [simp]: the-unique-root \((p, l, r)=0 \longleftrightarrow r=0\)
    and [dest]: \(l=0 \Longrightarrow r=0\)
    and [intro]: \(r=0 \Longrightarrow l=0\)
    using assms by (auto simp: sgn-0-0)
```

lemma invariant-1-pos: assumes $r c:$ 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(atomize(full),goal-cases)
case 1
let $? r=$ real-of-rat
from assms[unfolded invariant-1-def]
have ur: unique-root ( $p, l, r$ ) and sgn: sgn $l=\operatorname{sgn} r$ by auto
from unique-rootD(1-2) $[$ OF ur $]$ have $l e$ : ?r $l \leq$ ?x ? $x \leq$ ?r $r$ by auto
from rc show ?case
proof (cases r 0:: rat rule:linorder-cases)
case greater
with $\operatorname{sgn}$ have $\operatorname{sgn} l=1$ by $\operatorname{simp}$
hence $l 0: l>0$ by (auto simp: sgn-1-pos)
hence ? $r l>0$ by auto
hence ? $x>0$ using $l e(1)$ by arith
with greater $l 0$ show ?thesis by auto
next
case equal
with real-of-1-0 [OF rc] show ?thesis by auto
next
case less
hence ? $r$ r $<0$ by auto
with le(2) have ? $x<0$ by arith
with less sgn show ?thesis by (auto simp: sgn-1-neg)
qed
qed
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 $p$ by

```
(simp add: poly-cond2-def)
lemma poly-cond2D:
    assumes poly-cond2 p
    shows poly-cond p and degree p>1 using assms by (auto simp: poly-cond2-def)
lemma poly-cond2E[elim!]:
    assumes poly-cond2 p and poly-cond p\Longrightarrow degree p>1\Longrightarrow thesis shows thesis
    using assms by (auto simp: poly-cond2-def)
lemma invariant-1-2-poly-cond2: invariant-1-2 rai \Longrightarrow poly-cond2 (poly-real-alg-1
rai)
    unfolding invariant-1-def invariant-1-2-def poly-cond2-def by auto
lemma invariant-1-2I[intro!]:
    assumes invariant-1 rai and degree (poly-real-alg-1 rai) > 1 shows invariant-1-2
rai
    using assms by (auto simp: invariant-1-2-def)
lemma invariant-1-2E[elim!]:
    assumes invariant-1-2 rai
                            and invariant-1 rai \Longrightarrow degree (poly-real-alg-1 rai) > 1 \Longrightarrow thesis
    shows thesis using assms[unfolded invariant-1-2-def] by auto
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 ^ real-of-1 plr =x
    using invariant-1-realI[OF x] p sgn ur unfolding p-def l-def r-def by auto
```


### 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 $\mid$ Irrational nat real-alg-1

```
fun invariant-2 :: real-alg-2 }=>\mathrm{ bool where
    invariant-2 (Irrational n rai)}=(\mathrm{ invariant-1-2 rai
    \wedge n = \operatorname { c a r d ( r o o t s - b e l o w ~ ( p o l y - r e a l - a l g - 1 ~ r a i ) ~ ( r e a l - o f - 1 ~ r a i ) ) ) }
| invariant-2 (Rational r) = True
```

fun real-of-2 :: real-alg-2 $\Rightarrow$ real where
real-of-2 (Rational $r$ ) $=$ of-rat $r$
$\mid$ 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 $x$ )
by (auto simp: of-rat-2-def)
typedef real-alg-3 $=$ Collect invariant-2
morphisms rep-real-alg-3 Real-Alg-Invariant
by (rule exI[of - Rational 0], auto)
setup-lifting type-definition-real-alg-3
lift-definition real-of-3 :: real-alg-3 $\Rightarrow$ real is real-of-2 .

### 11.2.2 Definitions and Algorithms on Quotient Type

quotient-type real-alg $=$ real-alg-3 $/ \lambda x y$. real-of-3 $x=$ real-of-3 $y$ morphisms rep-real-alg Real-Alg-Quotient
by (auto simp: equivp-def) metis
lift-definition real-of :: real-alg $\Rightarrow$ real is real-of-3 .
lemma real-of-inj: $($ real-of $x=$ real-of $y)=(x=y)$
by (transfer, simp)

### 11.2.3 Sign

definition sgn-1 :: real-alg-1 $\Rightarrow$ rat where $\operatorname{sgn}-1 x=\operatorname{sgn}(r a i-u b x)$
lemma sgn-1: invariant-1 $x \Longrightarrow$ real-of-rat $(\operatorname{sgn-1} x)=\operatorname{sgn}($ real-of- $1 x)$ unfolding sgn-1-def by auto
lemma sgn-1-inj: invariant-1 $x \Longrightarrow$ invariant-1 $y \Longrightarrow$ real-of-1 $x=$ real-of- $1 y \Longrightarrow$ sgn-1 $x=\operatorname{sgn}-1 y$
by (auto simp: sgn-1-def elim!: invariant-1E)

### 11.2.4 Normalization: Bounds Close Together

```
lemma unique-root-lr: assumes ur: unique-root plr shows rai-lb plr \(\leq\) rai-ub plr
(is ?l \(\leq ? r\) )
proof -
    let \(? p=\) poly-real-alg-1 plr
    from ur[unfolded root-cond-def]
    have ex 1: \(\exists\) ! \(x\) :: real. of-rat ?l \(\leq x \wedge x \leq o f\)-rat ? \(r \wedge\) ipoly ? \(p x=0\) by (cases
plr, simp)
    then obtain \(x::\) real where bnd: of-rat ?l \(\leq x x \leq\) of-rat ?r and rt: ipoly ?p \(x\)
\(=0\) by auto
    from bnd have real-of-rat ?l \(\leq\) of-rat ?r by linarith
    thus ?l \(\leq\) ?r by (simp add: of-rat-less-eq)
qed
locale map-poly-zero-hom-0 = base: zero-hom-0
begin
    sublocale zero-hom-0 map-poly hom by (unfold-locales,auto)
end
interpretation of-int-poly-hom:
    map-poly-zero-hom-0 of-int :: int \(\Rightarrow^{\prime} a::\{\) ring-1, ring-char-0 \(\} ..\)
```

lemma roots-below-the-unique-root:
assumes ur: unique-root ( $p, l, r$ )
shows roots-below $p$ (the-unique-root $(p, l, r))=$ roots-below $p$ (of-rat $r$ ) (is roots-below
$p ? x=-$ )
proof-
from ur have rc: root-cond ( $p, l, r$ ) ? $x$ by (auto dest!: unique-rootD)
with $u r$ have $x:\{x$. root-cond $(p, l, r) x\}=\{? x\}$ by (auto intro: the-unique-root-eqI)
from $r c$ have $? x \in\{y . ? x \leq y \wedge y \leq o f$-rat $r \wedge$ ipoly $p y=0\}$ by auto
with $r c$ have $l 1 x: \ldots=\{? x\}$ by (intro equalityI, fold $x(1)$, force, simp add: $x$ )
have rb:roots-below $p$ (of-rat $r$ ) $=$ roots-below $p ? x \cup\{y . ? x<y \wedge y \leq o f$-rat $r$
$\wedge$ ipoly $p$ y $=0$ \}
using $r c$ by auto
have emp: $\bigwedge x$. the-unique-root $(p, l, r)<x \Longrightarrow$
$x \notin\{r a . ? x \leq r a \wedge r a \leq$ real-of-rat $r \wedge$ ipoly $p r a=0\}$
using $l 1 x$ by auto
with $r b$ show ?thesis by auto
qed
lemma unique-root-sub-interval:
assumes ur: unique-root ( $p, l, r$ )
and $r c$ : root-cond $\left(p, l^{\prime}, r^{\prime}\right)$ (the-unique-root $(p, l, r)$ )
and between: $l \leq l^{\prime} r^{\prime} \leq r$
shows unique-root ( $p, l^{\prime}, r^{\prime}$ )
and the-unique-root $\left(p, l^{\prime}, r^{\prime}\right)=$ the-unique-root $(p, l, r)$
proof -
from between have ord: real-of-rat $l \leq$ of-rat $l^{\prime}$ real-of-rat $r^{\prime} \leq o f-r a t r$ by (auto

```
simp:of-rat-less-eq)
    from rc have lr':real-of-rat l' }\leq\mathrm{ of-rat r' by auto
    with ord have lr: real-of-rat l}\leq\mathrm{ real-of-rat r by auto
    show }\exists\mathrm{ ! x. root-cond ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})
    proof (rule, rule rc)
    fix y
    assume root-cond ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})
    with ord have root-cond (p,l,r) y by (auto intro!:root-condI)
    from the-unique-root-eqI[OF ur this] show }y=\mathrm{ the-unique-root ( }p,l,r)\mathrm{ by simp
    qed
    from the-unique-root-eqI[OF this rc]
    show the-unique-root ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=\mathrm{ the-unique-root ( }p,l,r)\mathrm{ by simp
qed
lemma invariant-1-sub-interval:
    assumes rc: invariant-1 ( p,l,r)
        and sub: root-cond ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})(\mathrm{ the-unique-root ( }p,l,r)\mathrm{ )
        and between: l\leql' r}\mp@subsup{r}{}{\prime}\leq
    shows invariant-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ and real-of-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=\mathrm{ real-of-1 ( }p,l,r
proof -
    let ?r = real-of-rat
    note rcD = invariant-1D[OF rc]
    from rc
    have ur: unique-root ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}
    and id: the-unique-root ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=\mathrm{ the-unique-root ( }p,l,r\mathrm{ )
    by (atomize(full), intro conjI unique-root-sub-interval[OF - sub between], auto)
    show real-of-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=\mathrm{ real-of-1 ( }p,l,r
    using id by simp
    from rcD(1)[unfolded split] have ?r l \leq ? r r by auto
    hence lr:l}\leqr\mathrm{ by (auto simp: of-rat-less-eq)
    from unique-rootD[OF ur] have ?r l' 
    hence lr': l'}\leq\mp@subsup{r}{}{\prime}\mathrm{ by (auto simp: of-rat-less-eq)
    have sgn l' = sgn r'
    proof (cases r 0::rat rule: linorder-cases)
    case less
    with lr lr' between have l < 0 l'<0 r'<0 r<0 by auto
    thus ?thesis unfolding sgn-rat-def by auto
next
    case equal with rcD(2) have l=0 using sgn-0-0 by auto
    with equal between lr' have l'=0 r'=0 by auto then show ?thesis by auto
    next
    case greater
    with rcD(4) have sgn r=1 unfolding sgn-rat-def by (cases r = 0, auto)
    with }rcD(2) have sgn l=1 by sim
    hence l:l>0 unfolding sgn-rat-def by (cases l=0; cases l<0;auto)
    with lr lr'}\mathrm{ between have l>0 l'>0 r'>0 r>0 by auto
    thus ?thesis unfolding sgn-rat-def by auto
qed
with between ur rc show invariant-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by (auto simp add: invariant-1-def
```

id)
qed
lemma rational-root-free-degree-iff: assumes rf: root-free (map-poly rat-of-int p) and $r$ : ipoly $p x=0$
shows $(x \in \mathbb{Q})=($ degree $p=1)$
proof
assume $x \in \mathbb{Q}$
then obtain $y$ where $x: x=$ of-rat $y$ (is $-=? x$ ) unfolding Rats-def by blast
from $r t[$ unfolded $x]$ have poly (map-poly rat-of-int $p$ ) $y=0$ by simp
with $r f$ show degree $p=1$ unfolding root-free-def by auto
next
assume degree $p=1$
from degree1-coeffs[OF this]
obtain $a b$ where $p: p=[: a, b:]$ and $b: b \neq 0$ by auto
from $r t[$ unfolded $p$ hom-distribs] have of-int $a+x *$ of-int $b=0$ by auto
from arg-cong $[O F$ this, of $\lambda x$. $(x-$ of-int $a) /$ of-int $b]$
have $x=-$ of-rat (of-int $a$ ) / of-rat (of-int b) using $b$ by auto
also have $\ldots=$ of-rat ( - of-int $a /$ of-int b) unfolding of-rat-minus of-rat-divide
finally show $x \in \mathbb{Q}$ by auto
qed
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 (rule rational-root-free-degree-iff[OF - assms(2)])
from poly-condD[OF assms(1)] irreducible-connect-rev[of p] assms(3)
have $p$ : irreducible $_{d} p$ by auto
from irreducible $_{d}$-int-rat[OF this]
have irreducible (map-poly rat-of-int p) by simp
thus root-free (map-poly rat-of-int p) by (rule irreducible-root-free)
qed
lemma poly-cond-degree-gt-1: assumes poly-cond $p$ degree $p>1$ ipoly $p x=0$ shows $x \notin \mathbb{Q}$
using rational-poly-cond-iff[OF assms(1,3)] assms(2) by simp
lemma poly-cond2-no-rat-root: assumes poly-cond2 $p$
shows ipoly $p$ (real-of-rat $x) \neq 0$
using poly-cond-degree-gt-1 [of p real-of-rat $x]$ assms by auto
context
fixes $p::$ int poly
and $x::$ rat
begin
lemma gt-rat-sign-change:
assumes ur: unique-root plr

```
    defines \(p \equiv\) poly-real-alg-1 plr and \(l \equiv\) rai-lb plr and \(r \equiv\) rai-ub plr
    assumes \(p\) : poly-cond2 \(p\) and in-interval: \(l \leq y y \leq r\)
    shows \((\operatorname{sgn}(\) ipoly \(p y)=\operatorname{sgn}(\) ipoly \(p r))=(\) of-rat \(y>\) the-unique-root plr \()\)
proof -
    have plr: plr \(=(p, l, r)\) by (cases plr, auto simp: \(p\)-def l-def r-def)
    show ?thesis
    proof (rule gt-rat-sign-change-square-free[OF ur plr - in-interval])
        note \(n z=\) poly-cond2-no-rat-root [OF p]
        from \(n z[o f y]\) show ipoly \(p y \neq 0\) by auto
    from \(n z[o f r]\) show ipoly \(p r \neq 0\) by auto
    from \(p\) have irreducible \(p\) by auto
    thus square-free \(p\) by (rule irreducible-imp-square-free)
    qed
qed
definition tighten-poly-bounds :: rat \(\Rightarrow\) rat \(\Rightarrow\) rat \(\Rightarrow\) rat \(\times\) rat \(\times\) rat where
    tighten-poly-bounds \(l r\) sr \(=(\) let \(m=(l+r) / 2 ; s m=\operatorname{sgn}(\) ipoly \(p m)\) in
    if \(s m=s r\)
        then \((l, m, s m)\) else \((m, r, s r))\)
lemma tighten-poly-bounds: assumes res: tighten-poly-bounds l r sr \(=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)\)
    and ur: unique-root ( \(p, l, r\) )
    and \(p\) : poly-cond2 \(p\)
    and \(s r: s r=\operatorname{sgn}(\) ipoly \(p r)\)
    shows root-cond \(\left(p, l^{\prime}, r^{\prime}\right)\) (the-unique-root \(\left.(p, l, r)\right) l \leq l^{\prime} l^{\prime} \leq r^{\prime} r^{\prime} \leq r\)
    \(\left(r^{\prime}-l^{\prime}\right)=(r-l) / 2 s r^{\prime}=\operatorname{sgn}\left(\right.\) ipoly \(\left.p r^{\prime}\right)\)
proof -
    let \(? x=\) the-unique-root \((p, l, r)\)
    let \(? x^{\prime}=\) the-unique-root \(\left(p, l^{\prime}, r^{\prime}\right)\)
    let \(? m=(l+r) / 2\)
    note \(d=\) tighten-poly-bounds-def Let-def
    from unique-root-lr[OF ur] have \(l r: l \leq r\) by auto
    thus \(l \leq l^{\prime} l^{\prime} \leq r^{\prime} r^{\prime} \leq r\left(r^{\prime}-l^{\prime}\right)=(r-l) / 2 s r^{\prime}=\operatorname{sgn}\left(\right.\) ipoly \(\left.p r^{\prime}\right)\)
    using res sr unfolding \(d\) by (auto split: if-splits)
    hence \(l \leq ? m\) ? \(m \leq r\) by auto
    note \(l e=g t\)-rat-sign-change[OF ur,simplified,OF pthis]
    note \(u r D=\) unique-root \(D[O F u r]\)
    show root-cond \(\left(p, l^{\prime}, r^{\prime}\right) ? x\)
    proof (cases sgn (ipoly \(p\) ? \(m\) ) \(=\operatorname{sgn}(\) ipoly \(p r)\) )
        case *: False
        with res sr have \(i d: l^{\prime}=? m r^{\prime}=r\) unfolding \(d\) by auto
        from \(*[\) unfolded le] urD show ?thesis unfolding id by auto
    next
        case *: True
        with res sr have \(i d: l^{\prime}=l r^{\prime}=? m\) unfolding \(d\) by auto
        from \(*[u n f o l d e d ~ l e] ~ u r D ~ s h o w ~ ? t h e s i s ~ u n f o l d i n g ~ i d ~ b y ~ a u t o ~\)
    qed
qed
```

partial-function (tailrec) tighten-poly-bounds-epsilon :: rat $\Rightarrow$ rat $\Rightarrow$ rat $\Rightarrow$ rat $\times$ rat $\times$ rat where
[code]: tighten-poly-bounds-epsilon lr sr $=($ if $r-l \leq x$ then $(l, r, s r)$ else (case tighten-poly-bounds lr sr of $\left(l^{\prime}, r^{\prime}, s r^{\prime}\right) \Rightarrow$ tighten-poly-bounds-epsilon $l^{\prime} r^{\prime}$ $\left.s r^{\prime}\right)$ )
partial-function (tailrec) tighten-poly-bounds-for-x :: rat $\Rightarrow$ rat $\Rightarrow$ rat $\Rightarrow$ rat $\times$ rat $\times$ rat where
[code]: tighten-poly-bounds-for-x lr sr $=($ if $x<l \vee r<x$ then $(l, r, s r)$ else
(case tighten-poly-bounds $l r \operatorname{sr}$ of $\left(l^{\prime}, r^{\prime}, s r^{\prime}\right) \Rightarrow$ tighten-poly-bounds-for-x $l^{\prime} r^{\prime}$ $\left.s r^{\prime}\right)$ )
lemma tighten-poly-bounds-epsilon:
assumes ur: unique-root ( $p, l, r$ )
defines $u: u \equiv$ the-unique-root $(p, l, r)$
assumes $p$ : poly-cond2 $p$
and res: tighten-poly-bounds-epsilon lr sr $=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
and $s r: s r=\operatorname{sgn}($ ipoly $p r)$
and $x: x>0$
shows $l \leq l^{\prime} r^{\prime} \leq r$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u r^{\prime}-l^{\prime} \leq x s r^{\prime}=\operatorname{sgn}\left(\right.$ ipoly $\left.p r^{\prime}\right)$
proof -
let ?u $=$ the-unique-root $(p, l, r)$
define delta where delta $=x / 2$
have delta: delta $>0$ unfolding delta-def using $x$ by auto
let ? dist $=\lambda(l, r, s r) . r-l$
let ?rel $=$ inv-image $\{(x, y)$. $0 \leq y \wedge$ delta-gt delta $x y\}$ ?dist
note $S N=S N$-inv-image[OF delta-gt-SN[OF delta], of ?dist]
note simps $=$ res[unfolded tighten-poly-bounds-for-x.simps[of l r]]
let ? $P=\lambda(l, r, s r)$. unique-root $(p, l, r) \longrightarrow u=$ the-unique-root $(p, l, r)$
$\longrightarrow$ tighten-poly-bounds-epsilon l r sr $=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
$\longrightarrow s r=\operatorname{sgn}($ ipoly $p r)$
$\longrightarrow l \leq l^{\prime} \wedge r^{\prime} \leq r \wedge r^{\prime}-l^{\prime} \leq x \wedge$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u \wedge s r^{\prime}=\operatorname{sgn}($ ipoly $p$ $\left.r^{\prime}\right)$
have ? $P(l, r, s r)$
proof (induct rule: $S N$-induct [OF $S N]$ )
case (1 lr)
obtain $l r$ sr where $l r$ : $l r=(l, r, s r)$ by (cases $l r$, auto)
show ?case unfolding lr split
proof (intro impI)
assume ur: unique-root $(p, l, r)$
and $u: u=$ the-unique-root $(p, l, r)$
and res: tighten-poly-bounds-epsilon l $r$ sr $=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
and $s r: s r=\operatorname{sgn}($ ipoly $p r$ )
note tur $=$ unique-root $D[O F u r]$
note simps $=$ tighten-poly-bounds-epsilon.simps[of l r sr]
show $l \leq l^{\prime} \wedge r^{\prime} \leq r \wedge r^{\prime}-l^{\prime} \leq x \wedge$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u \wedge s r^{\prime}=\operatorname{sgn}($ ipoly $\left.p r^{\prime}\right)$
proof (cases $r-l \leq x$ )
case True
with res[unfolded simps] ur tur(4) u sr
show ?thesis by auto
next
case False
hence $x: r-l>x$ by auto
let ?tight $=$ tighten-poly-bounds l r sr
obtain $L R S R$ where tight: ?tight $=(L, R, S R)$ by (cases ?tight, auto)
note tighten $=$ tighten-poly-bounds $[$ OF tight $[$ unfolded sr] ur $p]$
from unique-root-sub-interval $[$ OF ur tighten(1-2,4)] $p$
have $u r^{\prime}$ : unique-root $(p, L, R) u=$ the-unique-root $(p, L, R)$ unfolding $u$ by auto
from res[unfolded simps tight] False sr have tighten-poly-bounds-epsilon $L$ $R S R=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$ by auto
note $I H=1\left[\right.$ of $(L, R, S R)$, unfolded tight split lr, rule-format, $O F-u r^{\prime}$ this]
have $L \leq l^{\prime} \wedge r^{\prime} \leq R \wedge r^{\prime}-l^{\prime} \leq x \wedge$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u \wedge s r^{\prime}=s g n$ (ipoly $p r^{\prime}$ )
by (rule IH, insert tighten False, auto simp: delta-gt-def delta-def)
thus ?thesis using tighten by auto
qed
qed
qed
from this[unfolded split $u$, rule-format, OF ur refl res sr]
show $l \leq l^{\prime} r^{\prime} \leq r$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u r^{\prime}-l^{\prime} \leq x s r^{\prime}=\operatorname{sgn}\left(\right.$ ipoly $\left.p r^{\prime}\right)$ using $u$ by auto
qed
lemma tighten-poly-bounds-for-x:
assumes ur: unique-root ( $p, l, r$ )
defines $u: u \equiv$ the-unique-root ( $p, l, r$ )
assumes $p$ : poly-cond2 $p$
and res: tighten-poly-bounds-for-x l r sr $=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
and $s r: s r=s g n($ ipoly $p r)$
shows $l \leq l^{\prime} l^{\prime} \leq r^{\prime} r^{\prime} \leq r$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u \neg\left(l^{\prime} \leq x \wedge x \leq r^{\prime}\right) s r^{\prime}=s g n$
(ipoly $p r^{\prime}$ ) unique-root ( $p, l^{\prime}, r^{\prime}$ )
proof -
let $? u=$ the-unique-root $(p, l, r)$
let $? x=$ real-of-rat $x$
define delta where delta $=a b s((u-? x) / 2)$
let $? p=$ real-of-int-poly $p$
note $r u=$ unique-root $D[$ OF ur]
\{
assume $u=$ ? $x$
note $u=$ this[unfolded $u$ ]
from poly-cond2-no-rat-root[OF p] ur have False by (elim unique-rootE, auto simp: u)
\}
hence delta: delta $>0$ unfolding delta-def by auto
let ?dist $=\lambda(l, r, s r)$. real-of-rat $(r-l)$
let ?rel $=$ inv-image $\{(x, y) .0 \leq y \wedge$ delta-gt delta $x y\}$ ?dist
note $S N=S N$-inv-image $[$ OF delta-gt-SN[OF delta], of ?dist $]$
note simps $=$ res[unfolded tighten-poly-bounds-for-x.simps[of l r]]
let ? $P=\lambda(l, r, s r)$. unique-root $(p, l, r) \longrightarrow u=$ the-unique-root $(p, l, r)$
$\longrightarrow$ tighten-poly-bounds-for-x lr sr $=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
$\longrightarrow s r=\operatorname{sgn}($ ipoly $p r)$
$\longrightarrow l \leq l^{\prime} \wedge r^{\prime} \leq r \wedge \neg\left(l^{\prime} \leq x \wedge x \leq r^{\prime}\right) \wedge$ root-cond $\left(p, l^{\prime}, r^{\prime}\right) u \wedge s r^{\prime}=s g n$
(ipoly $p r^{\prime}$ )
have ? $P(l, r, s r)$
proof (induct rule: $S N$-induct [OF $S N]$ )
case (1 lr)
obtain $l r s r$ where $l r: l r=(l, r, s r)$ by (cases $l r$, auto)
let $? l=$ real-of-rat $l$
let $? r=$ real-of-rat $r$
show ?case unfolding lr split
proof (intro impI)
assume ur: unique-root ( $p, l, r$ )
and $u: u=$ the-unique-root $(p, l, r)$
and res: tighten-poly-bounds-for-x l r $s r=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$
and $s r: s r=\operatorname{sgn}($ ipoly $p r)$
note tur $=$ unique-root $D[O F u r]$
note simps $=$ tighten-poly-bounds-for-x.simps[of l r]
show $l \leq l^{\prime} \wedge r^{\prime} \leq r \wedge \neg\left(l^{\prime} \leq x \wedge x \leq r^{\prime}\right) \wedge \operatorname{root-cond}\left(p, l^{\prime}, r^{\prime}\right) u \wedge s r^{\prime}=$ sgn (ipoly $p r^{\prime}$ )
proof (cases $x<l \vee r<x$ )
case True
with res[unfolded simps] ur tur(4) u sr
show ?thesis by auto
next
case False
hence $x: ? l \leq ? x ? x \leq ? r$ by (auto simp: of-rat-less-eq)
let ?tight $=$ tighten-poly-bounds l $r$ sr
obtain $L R S R$ where tight: ?tight $=(L, R, S R)$ by (cases ?tight, auto)
note tighten $=$ tighten-poly-bounds $[$ OF tight ur $p$ sr]
from unique-root-sub-interval $[$ OF ur tighten(1-2,4)] $p$
have $u r^{\prime}$ : unique-root $(p, L, R) u=$ the-unique-root $(p, L, R)$ unfolding $u$ by
auto
from res[unfolded simps tight] False have tighten-poly-bounds-for-x L R SR
$=\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)$ by auto
note $I H=1\left[\right.$ of ?tight, unfolded tight split lr, rule-format, $O F-u r^{\prime}$ this]
let ?DIFF $=$ real-of-rat $(R-L)$ let ?diff $=$ real-of-rat $(r-l)$
have diff0: $0 \leq$ ? DIFF using tighten(3)
by (metis cancel-comm-monoid-add-class.diff-cancel diff-right-mono of-rat-less-eq of-rat-hom.hom-zero)
have $*: r-l-(r-l) / 2=(r-l) / 2$ by (auto simp: field-simps)
have delta-gt delta ?diff ?DIFF $=($ abs $(u-$ of-rat $x) \leq$ real-of-rat $(r-l)$ * 1)
unfolding delta-gt-def tighten(5) delta-def of-rat-diff[symmetric] * by (simp add: hom-distribs)
also have real-of-rat $(r-l) * 1=? r-? l$

```
            unfolding of-rat-divide of-rat-mult of-rat-diff by auto
            also have abs (u - of-rat x) \leq?r - ?l using x ur by (elim unique-rootE,
auto simp: u)
    finally have delta: delta-gt delta ?diff ?DIFF .
    have L\leq l'^ r'\leqR\wedge\neg(l'\leqx\wedgex\leqr')^root-cond (p, l', r')u\wedges\mp@subsup{r}{}{\prime}
= sgn (ipoly p r')
            by (rule IH, insert delta diffo tighten(6), auto)
            with }\langlel\leqL\rangle\langleR\leqr\rangle\mathrm{ show ?thesis by auto
        qed
    qed
    qed
    from this[unfolded split u, rule-format, OF ur refl res sr]
    show *:l\leql' }\mp@subsup{r}{}{\prime}\leqr\mathrm{ root-cond (p, l',r')u}\neg(\mp@subsup{l}{}{\prime}\leqx\wedgex\leqr')s\mp@subsup{r}{}{\prime}=\operatorname{sgn}(\mathrm{ ipoly 
p r') unfolding u
    by auto
    from *(3)[unfolded split] have real-of-rat l'}\leq\mathrm{ of-rat r' by auto
    thus \mp@subsup{l}{}{\prime}}\leq\mp@subsup{r}{}{\prime}\mathrm{ unfolding of-rat-less-eq.
    show unique-root ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}\mathrm{ ) using ur*(1-3) p poly-condD(5) u unique-root-sub-interval(1)
by blast
qed
end
definition real-alg-precision :: rat where
    real-alg-precision \equiv Rat.Fract 1 2
lemma real-alg-precision: real-alg-precision > 0
    by eval
definition normalize-bounds-1-main :: rat }=>\mathrm{ real-alg-1 }=>\mathrm{ real-alg-1 where
    normalize-bounds-1-main eps rai = (case rai of (p,l,r) =>
        let (l', r',s\mp@subsup{r}{}{\prime})= tighten-poly-bounds-epsilon p eps lr (sgn (ipoly p r));
            fr = rat-of-int (floor r}\mp@subsup{r}{}{\prime})
            (l',},\mp@subsup{r}{}{\prime\prime},-)= tighten-poly-bounds-for-x p fr l' r'sr'*
        in (p,\mp@subsup{l}{}{\prime\prime},\mp@subsup{r}{}{\prime\prime}))
definition normalize-bounds-1 :: real-alg-1 }=>\mathrm{ real-alg-1 where
    normalize-bounds-1 = (normalize-bounds-1-main real-alg-precision)
context
    fixes pq and lr :: rat
    assumes cong: \ x. real-of-rat l\leqx\Longrightarrow < < of-rat r \Longrightarrow (ipoly p x = (0 ::
real))}=(\mathrm{ ipoly q }x=0
begin
lemma root-cond-cong: root-cond ( }p,l,r)=root-cond ( q,l,r
    by (intro ext, insert cong, auto simp: root-cond-def)
lemma the-unique-root-cong:
    the-unique-root ( }p,l,r\mathrm{ ) = the-unique-root ( }q,l,r\mathrm{ )
    unfolding root-cond-cong ..
```

```
lemma unique-root-cong:
    unique-root ( }p,l,r)=\mathrm{ unique-root ( }q,l,r
    unfolding root-cond-cong ..
end
lemma normalize-bounds-1-main: assumes eps:eps > 0 and rc: invariant-1-2 x
    defines y: }y\equiv\mathrm{ normalize-bounds-1-main eps }
    shows invariant-1-2 y ^(real-of-1 y = real-of-1 x)
proof -
    obtain plr where x: x = (p,l,r) by (cases x) auto
    note rc = rc[unfolded x]
    obtain l' r' sr' where tb: tighten-poly-bounds-epsilon p eps lr (sgn (ipoly pr))
=(l',r',s\mp@subsup{r}{}{\prime})
    by (cases rule: prod-cases3, auto)
    let ?fr = rat-of-int (floor r')
    obtain l" }\mp@subsup{r}{}{\prime\prime}s\mp@subsup{r}{}{\prime\prime}\mathrm{ where tbx: tighten-poly-bounds-for-x p ?fr l' r' sr'}=(\mp@subsup{l}{}{\prime\prime},\mp@subsup{r}{}{\prime\prime},s\mp@subsup{r}{}{\prime\prime}
        by (cases rule: prod-cases3, auto)
    from y[unfolded normalize-bounds-1-main-def x] tb tbx
    have y:}y=(p,\mp@subsup{l}{}{\prime\prime},\mp@subsup{r}{}{\prime\prime}
        by (auto simp: Let-def)
    from rc have unique-root ( }p,l,r\mathrm{ ) and p2: poly-cond2 p by auto
    from tighten-poly-bounds-epsilon[OF this tb refl eps]
    have bnd:l\leq l' r}\mp@subsup{r}{}{\prime}\leqr\mathrm{ and }r\mp@subsup{c}{}{\prime}: root-cond (p, l', r') (the-unique-root ( p,l,r)
        and eps: r' - l'\leqeps
        and }s\mp@subsup{r}{}{\prime}:s\mp@subsup{r}{}{\prime}=\operatorname{sgn}(\mathrm{ ipoly p r ') by auto
    from invariant-1-sub-interval[OF - rc' bnd] rc
    have inv': invariant-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ and eq: real-of-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=real-of-1 (p,l,r
by auto
    have bnd: l' }\leq\mp@subsup{l}{}{\prime\prime}\mp@subsup{r}{}{\prime\prime}\leq\mp@subsup{r}{}{\prime}\mathrm{ and rc': root-cond ( p, l', , r') (the-unique-root ( }p,\mp@subsup{l}{}{\prime}\mathrm{ ,
r'))
    by (rule tighten-poly-bounds-for-x[OF - p2 tbx sr ], fact invariant-1D[OF inv ])+
    from invariant-1-sub-interval[OF inv' rc' bnd] p2 eq
    show ?thesis unfolding y x by auto
qed
lemma normalize-bounds-1: assumes x: invariant-1-2 x
    shows invariant-1-2 (normalize-bounds-1 x) ^(real-of-1 (normalize-bounds-1 x)
= real-of-1 x)
proof(cases x)
    case xx:(fields pl r)
    let ?res = ( p,l,r)
    have norm: normalize-bounds-1 x = (normalize-bounds-1-main real-alg-precision
    ?res)
        unfolding normalize-bounds-1-def by (simp add: xx)
    from x have x: invariant-1-2 ?res real-of-1 ?res = real-of-1 }x\mathrm{ unfolding }xx\mathrm{ by
auto
    from normalize-bounds-1-main[OF real-alg-precision x(1)] x(2-)
    show ?thesis unfolding normalize-bounds-1-def xx by auto
```


## qed

lemma normalize-bound-1-poly: poly-real-alg-1 (normalize-bounds-1 rai) $=$ poly-real-alg-1 rai
unfolding normalize-bounds-1-def normalize-bounds-1-main-def Let-def
by (auto split: prod.splits)
definition real-alg-2-main :: root-info $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
real-alg-2-main ri rai $\equiv$ let $p=$ poly-real-alg-1 rai
in (if degree $p=1$ then Rational (Rat.Fract $(-\operatorname{coeff} p 0)(\operatorname{coeff} p 1))$ else (case normalize-bounds-1 rai of $\left(p^{\prime}, l, r\right) \Rightarrow$ Irrational (root-info.number-root ri r) $\left.\left(p^{\prime}, l, r\right)\right)$ )
definition real-alg-2 :: real-alg-1 $\Rightarrow$ real-alg-2 where
real-alg-2 rai $\equiv$ let $p=$ poly-real-alg-1 rai in (if degree $p=1$ then Rational (Rat.Fract $(-\operatorname{coeff} p 0)(\operatorname{coeff} p 1))$ else (case normalize-bounds-1 rai of ( $\left.p^{\prime}, l, r\right) \Rightarrow$ Irrational (root-info.number-root (root-info p)r)( $\left.\left.p^{\prime}, l, r\right)\right)$ )
lemma degree-1-ipoly: assumes degree $p=$ Suc 0
shows ipoly $p x=0 \longleftrightarrow(x=$ real-of-rat $($ Rat.Fract $(-\operatorname{coeff} p 0)(\operatorname{coeff} p 1)))$
proof -
from roots1 [of map-poly real-of-int p] assms
have ipoly $p x=0 \longleftrightarrow x \in\{$ roots1 (real-of-int-poly $p$ ) $\}$ by auto
also have $\ldots=(x=$ real-of-rat $($ Rat.Fract $(-\operatorname{coeff} p 0)(\operatorname{coeff} p 1)))$
unfolding Fract-of-int-quotient roots1-def hom-distribs
by auto
finally show ?thesis.
qed
lemma invariant-1-degree-0:
assumes inv: invariant-1 rai
shows degree (poly-real-alg-1 rai) $\neq 0$ (is degree $? p \neq 0$ )
proof (rule notI)
assume deg: degree ? $p=0$
from inv have ipoly?p (real-of-1 rai) $=0$ by auto
with deg have ? $p=0$ by (meson less-Suc0 representsI represents-degree)
with inv show False by auto
qed
lemma real-alg-2-main:
assumes inv: invariant-1 rai
defines $[$ simp $]: p \equiv$ poly-real-alg-1 rai
assumes ric: irreducible (poly-real-alg-1 rai) $\Longrightarrow$ root-info-cond ri (poly-real-alg-1 rai)
shows invariant-2 (real-alg-2-main ri rai) real-of-2 (real-alg-2-main ri rai) $=$ real-of-1 rai
proof (atomize(full))
define $l r$ where $[s i m p]: l \equiv$ rai-lb rai and $[\operatorname{simp}]: r \equiv$ rai-ub rai

```
    show invariant-2 (real-alg-2-main ri rai) ^ real-of-2 (real-alg-2-main ri rai) =
real-of-1 rai
    unfolding id using invariant-1D
    proof (cases degree p Suc 0 rule: linorder-cases)
    case deg: equal
        hence id: real-alg-2-main ri rai = Rational (Rat.Fract (- coeff p 0) (coeff p
1))
            unfolding real-alg-2-main-def Let-def by auto
    note rc = invariant-1D[OF inv]
    from degree-1-ipoly[OF deg, of the-unique-root rai] rc(1)
    show ?thesis unfolding id by auto
    next
    case deg: greater
    with inv have inv: invariant-1-2 rai unfolding p-def by auto
    define rai' where rai' = normalize-bounds-1 rai
    have rai': real-of-1 rai = real-of-1 rai' and inv': invariant-1-2 rai'
        unfolding rai'-def using normalize-bounds-1[OF inv] by auto
    obtain p}\mp@subsup{p}{}{\prime}\mp@subsup{l}{}{\prime}\mp@subsup{r}{}{\prime}\mathrm{ where rai' = ( }\mp@subsup{p}{}{\prime},\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by (cases rai')
    with arg-cong[OF rai'-def, of poly-real-alg-1, unfolded normalize-bound-1-poly]
split
    have split:rai' = (p, l',r') by auto
    from inv'[unfolded split]
    have poly-cond p by auto
    from poly-condD[OF this] have irr: irreducible p by simp
    from ric irr have ric: root-info-cond ri p by auto
    have id: real-alg-2-main ri rai = (Irrational (root-info.number-root ri r') rai')
        unfolding real-alg-2-main-def Let-def using deg split rai'-def
        by (auto simp: rai'-def rai')
    show ?thesis unfolding id using rai' root-info-condD(2)[OF ric]
                inv'[unfolded split]
        apply (elim invariant-1-2E invariant-1E) using inv'
        by(auto simp: split roots-below-the-unique-root)
    next
    case deg: less then have degree p=0 by auto
    from this invariant-1-degree-0[OF inv] have p=0 by simp
    with inv show ?thesis by auto
    qed
qed
lemma real-alg-2: assumes invariant-1 rai
    shows invariant-2 (real-alg-2 rai) real-of-2 (real-alg-2 rai) = real-of-1 rai
proof -
    have deg: 0 < degree (poly-real-alg-1 rai) using assms by auto
    have real-alg-2 rai = real-alg-2-main (root-info (poly-real-alg-1 rai)) rai
        unfolding real-alg-2-def real-alg-2-main-def Let-def by auto
    from real-alg-2-main[OF assms root-info, folded this, simplified] deg
    show invariant-2 (real-alg-2 rai) real-of-2 (real-alg-2 rai) = real-of-1 rai by auto
qed
```


## lemma invariant-2-realI:

fixes $p l r::$ 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 $l=\operatorname{sgn} r$
and ur: unique-root plr and $p$ : poly-cond $p$
shows invariant-2 (real-alg-2 plr) $\wedge$ real-of-2 (real-alg-2 plr) $=x$
using invariant-1-realI[OF x,folded $p$-def l-def r-def] sgn ur $p$
real-alg-2[of plr] by auto

### 11.2.5 Comparisons

```
fun compare-rat-1 :: rat \(\Rightarrow\) real-alg-1 \(\Rightarrow\) order where
    compare-rat- \(1 x(p, l, r)=(\) if \(x<l\) then Lt else if \(x>r\) then \(G t\) else
        if \(\operatorname{sgn}(\) ipoly \(p x)=\operatorname{sgn}(\) ipoly \(p r)\) then Gt else Lt)
lemma compare-rat-1: assumes rai: invariant-1-2 y
    shows compare-rat-1 \(x y=\) compare \((\) of-rat \(x)(\) real-of-1 \(y)\)
proof-
    define \(p l r\) where \(p \equiv\) poly-real-alg-1 y \(l \equiv\) rai-lb y \(r \equiv\) rai-ub y
    then have \(y[\) simp \(]: y=(p, l, r)\) by (cases \(y\), auto)
    from rai have ur: unique-root \(y\) by auto
    show ?thesis
    proof (cases \(x<l \vee x>r\) )
        case True
        \{
        assume \(x l\) : \(x<l\)
        hence real-of-rat \(x<\) of-rat \(l\) unfolding of-rat-less by auto
        with rai have of-rat \(x<\) the-unique-root \(y\) by (auto elim!: invariant-1E)
        with \(x l\) rai have ?thesis by (cases \(y\), auto simp: compare-real-def compara-
tor-of-def)
        \}
        moreover
        \{
            assume \(x r: \neg x<l x>r\)
            hence real-of-rat \(x>\) of-rat \(r\) unfolding of-rat-less by auto
            with rai have of-rat \(x>\) the-unique-root \(y\) by (auto elim!: invariant-1E)
            with xr rai have ?thesis by (cases y, auto simp: compare-real-def compara-
tor-of-def)
            \}
            ultimately show ?thesis using True by auto
    next
            case False
            have 0: ipoly \(p\) (real-of-rat \(x) \neq 0\) by (rule poly-cond2-no-rat-root, insert rai,
auto)
            with rai have diff: real-of-1 \(y \neq\) of-rat \(x\) by (auto elim!: invariant-1E)
            have \(\wedge P .(1<\) degree \((\) poly-real-alg-1 \(y) \Longrightarrow \exists!x\). root-cond \(y x \Longrightarrow\) poly-cond
\(p \Longrightarrow P) \Longrightarrow P\)
            using poly-real-alg-1.simps y rai invariant-1-2E invariant-1E by metis
```

```
    from this[OF gt-rat-sign-change] False
    have left: compare-rat-1 x y = (if real-of-rat x < the-unique-root y then Lt else
Gt)
    by (auto simp:poly-cond2-def)
    also have ... = compare (real-of-rat x) (real-of-1 y) using diff
        by (auto simp: compare-real-def comparator-of-def)
    finally show ?thesis.
    qed
qed
lemma cf-pos-0[simp]: \neg cf-pos 0
    unfolding cf-pos-def by auto
```


### 11.2.6 Negation

fun uminus-1 :: real-alg-1 $\Rightarrow$ real-alg- 1 where uminus-1 $(p, l, r)=($ abs-int-poly $($ poly-uminus $p),-r,-l)$
lemma uminus-1: assumes $x$ : invariant-1 $x$
defines $y: y \equiv$ uminus- $1 x$
shows invariant-1 $y \wedge$ (real-of-1 $y=-$ real-of-1 $x)$
proof (cases $x$ )
case plr: (fields plr)
from $x$ plr have inv: invariant-1 $(p, l, r)$ by auto
note $*=$ invariant $-1 D[$ OF this $]$
from $p l r$ have $x: x=(p, l, r)$ by simp
let $? p=$ poly-uminus $p$
let $? m p=a b s$-int-poly ? $p$
have $y: y=(? m p,-r,-l)$
unfolding $y$ plr by (simp add: Let-def)
\{
fix $y$
assume root-cond $(? m p,-r,-l) y$
hence mpy: ipoly ? $m p y=0$ and bnd: - of-rat $r \leq y y \leq-o f$-rat $l$
unfolding root-cond-def by (auto simp: of-rat-minus)
from mpy have id: ipoly $p(-y)=0$ by auto
from bnd have bnd: of-rat $l \leq-y-y \leq$ of-rat $r$ by auto
from id bnd have root-cond ( $p, l, r$ ) ( $-y$ ) unfolding root-cond-def by auto
with inv $x$ have real-of-1 $x=-y$ by (auto intro!: the-unique-root-eqI)
then have - real-of-1 $x=y$ by auto
\} note $i n j=$ this
have $r c$ : root-cond $(? m p,-r,-l)(-$ real-of-1 $x)$
using * unfolding root-cond-def y $x$ by (auto simp: of-rat-minus sgn-minus-rat)
from inj rc have $u r^{\prime}$ : unique-root (? $m p,-r,-l$ ) by (auto intro: unique-rootI)
with $r c$ have the: - real-of-1 $x=$ the-unique-root $(? m p,-r,-l)$ by (auto intro: the-unique-root-eqI)
have xp: $p$ represents (real-of-1 $x$ ) using * unfolding root-cond-def split repre-sents-def $x$ by auto
from $*$ have mon: lead-coeff ? mp $>0$ by (unfold pos-poly-abs-poly, auto)
from poly-uminus-irreducible * have mi : irreducible ?mp by auto
from mi mon have $c^{\prime}$ : poly-cond ? $m p$ by (auto simp: cf-pos-def)
from poly-condD[OF pc $]$ have irr: irreducible ? $m p$ by auto
show ?thesis unfolding $y$ apply (intro invariant-1-realI ur'rc) using $p c^{\prime}$ inv
by auto
qed
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 -
from $x$ have invariant- $1 x$ by auto
from uminus- 1 [OF this] have $*$ : real-of- $1 y=-$ real-of- $1 x$
invariant-1 $y$ unfolding $y$ by auto
obtain $p l r$ where $i d: x=(p, l, r)$ by (cases $x)$
from $x[$ unfolded $i d]$ have degree $p>1$ by auto
moreover have poly-real-alg-1 $y=a b s$-int-poly (poly-uminus $p$ )
unfolding $y$ id uminus-1.simps split Let-def by auto
ultimately have degree (poly-real-alg-1 y) >1 by simp
with $*$ show ?thesis by auto
qed
fun uminus-2 :: real-alg-2 $\Rightarrow$ real-alg-2 where
uminus-2 (Rational $r$ ) $=$ Rational $(-r)$
|uminus-2 (Irrational $n x)=$ real-alg-2 $(u m i n u s-1 x)$
lemma uminus-2: assumes invariant-2 $x$
shows real-of-2 (uminus-2 $x$ ) $=$ uminus $($ real-of- $2 x)$
invariant-2 (uminus-2 $x$ )
using assms real-alg-2 uminus-1 by (atomize(full), cases x, auto simp: hom-distribs)
declare uminus-1.simps[simp del]
lift-definition uminus-3 :: real-alg-3 $\Rightarrow$ real-alg-3 is uminus-2
by (auto simp: uminus-2)
lemma uminus-3: real-of-3 (uminus-3 $x$ ) $=-$ real-of-3 $x$
by (transfer, auto simp: uminus-2)
instantiation real-alg :: uminus
begin
lift-definition uminus-real-alg :: real-alg $\Rightarrow$ real-alg is uminus-3
by (simp add: uminus-3)
instance ..
end
lemma uminus-real-alg: $-($ real-of $x)=$ real-of $(-x)$

```
by (transfer, rule uminus-3[symmetric])
```


### 11.2.7 Inverse

```
fun inverse-1 :: real-alg-1 => real-alg-2 where
    inverse-1 ( }p,l,r)=\mathrm{ 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 }=\mathrm{ of-rat }
proof -
    obtain plr where rai: rai = (p,l,r) by (cases rai,auto)
    from rc[unfolded rai]
    have poly-cond2 p ipoly p (the-unique-root ( }p,l,r)\mathrm{ ) = 0 by (auto elim!: invari-
ant-1E)
    from poly-cond2-no-rat-root[OF this(1), of x] this(2) show ?thesis unfolding
rai by auto
qed
lemma inverse-1:
    assumes rcx: invariant-1-2 x
    defines }y\mathrm{ : }y\equiv\mathrm{ inverse-1 }
    shows invariant-2 y ^(real-of-2 y = inverse (real-of-1 x))
proof (cases x)
    case x:(fields p l r)
    from x rcx have rcx: invariant-1-2 (p,l,r) by auto
    from invariant-1-2-poly-cond2[OF rcx] have pc2: poly-cond2 p by simp
    have x0: real-of-1 ( }p,l,r)\not=0\mathrm{ using invariant-1-2-of-rat[OF rcx, of 0] x by auto
    let ?x = real-of-1 ( }p,l,r
    let ?mp =abs-int-poly (reflect-poly p)
    from x0 rcx have lr0:l\not=0 and r\not=0 by auto
    from x0 rcx have y:y= real-alg-2 (?mp, inverse r, inverse l)
        unfolding y x Let-def inverse-1.simps by auto
    from rcx have mon: lead-coeff ?mp>0 by (unfold lead-coeff-abs-int-poly,auto)
    {
        fix y
        assume root-cond (?mp, inverse r, inverse l) y
        hence mpy: ipoly ?mp y = 0 and bnd: inverse (of-rat r)\leqy y \leqinverse
(of-rat l)
            unfolding root-cond-def by (auto simp: of-rat-inverse)
    from sgn-real-mono[OF bnd(1)] sgn-real-mono[OF bnd(2)]
    have sgn (of-rat r) \leq sgn y sgn y \leq sgn (of-rat l)
            by (simp-all add: algebra-simps)
    with rcx have sgn: sgn (inverse (of-rat r)) = sgn y sgn y = sgn (inverse (of-rat
l))
            unfolding sgn-inverse inverse-sgn
            by (auto simp add: real-of-rat-sgn intro: order-antisym)
    from sgn[simplified, unfolded real-of-rat-sgn] lr0 have y}=0\mathrm{ by (auto simp:sgn-0-0)
    with mpy have id: ipoly p (inverse y)=0 by (auto simp: ipoly-reflect-poly)
    from inverse-le-sgn[OF sgn(1) bnd(1)] inverse-le-sgn[OF sgn(2) bnd(2)]
```

```
    have bnd: of-rat l \leq inverse y inverse y s of-rat r by auto
    from id bnd have root-cond ( }p,l,r\mathrm{ ) (inverse y) unfolding root-cond-def by
auto
    from rcx this x0 have ? }x=\mathrm{ inverse }y\mathrm{ by auto
    then have inverse ? }x=y\mathrm{ by auto
    } note inj = this
    have rc: root-cond (?mp, inverse r, inverse l) (inverse ?x)
        using rcx x0 apply (elim invariant-1-2E invariant-1E)
    by (simp add: root-cond-def of-rat-inverse real-of-rat-sgn inverse-le-iff-sgn ipoly-reflect-poly)
    from inj rc have ur: unique-root (?mp, inverse r, inverse l) by (auto intro:
unique-rootI)
    with rc have the: the-unique-root (?mp, inverse r, inverse l) = inverse ?x by
(auto intro: the-unique-root-eqI)
    have xp: p represents ?x unfolding split represents-def using rcx by (auto elim!:
invariant-1E)
    from reflect-poly-irreducible[OF - xp x0] poly-condD rcx
    have mi: irreducible ?mp by auto
    from mi mon have un: poly-cond ?mp by (auto simp: poly-cond-def)
    show ?thesis using rcx rc ur unfolding y
    by (intro invariant-2-realI, auto simp: x y un)
qed
fun inverse-2 :: real-alg-2 }=>\mathrm{ real-alg-2 where
    inverse-2 (Rational r)= Rational (inverse r)
| inverse-2 (Irrational n x) = inverse-1 }
lemma inverse-2: assumes invariant-2 x
    shows real-of-2 (inverse-2 x) = inverse (real-of-2 x)
    invariant-2 (inverse-2 x)
        using assms
    by (atomize(full), cases x, auto simp: real-alg-2 inverse-1 hom-distribs)
lift-definition inverse-3 :: real-alg-3 => real-alg-3 is inverse-2
    by (auto simp: inverse-2)
lemma inverse-3: real-of-3 (inverse-3 x) = inverse (real-of-3 x)
    by (transfer, auto simp: inverse-2)
```


### 11.2.8 Floor

```
fun floor-1 :: real-alg-1 \(\Rightarrow\) int where
floor-1 \((p, l, r)=(\) let
\(\left(l^{\prime}, r^{\prime}, s r^{\prime}\right)=\) tighten-poly-bounds-epsilon \(p(1 / 2) l r(\operatorname{sgn}(\) ipoly \(p r))\);
fr \(=\) floor \(r^{\prime}\);
\(f l=\) floor \(l^{\prime}\);
\(f r^{\prime}=r a t-o f-i n t ~ f r\)
in (if fr \(=\) fl then fr else
let \(\left(l^{\prime \prime}, r^{\prime \prime}, s r^{\prime \prime}\right)=\) tighten-poly-bounds-for-x \(p f r^{\prime} l^{\prime} r^{\prime} s r^{\prime}\)
in if \(f r^{\prime}<l^{\prime \prime}\) then fr else fl))
```

```
lemma floor-1: assumes invariant-1-2 x
    shows floor (real-of-1 x) = floor-1 }
proof (cases x)
    case (fields p l r)
    obtain l' r'sr' where tbe: tighten-poly-bounds-epsilon p (1 / 2) lr (sgn (ipoly
pr)) = ( l', r',sr')
    by (cases rule: prod-cases3, auto)
    let ?fr = floor r'
    let ?fl = floor l}\mp@subsup{l}{}{\prime
    let ?fr' = rat-of-int ?fr
    obtain l' }\mp@subsup{l}{}{\prime\prime}\mp@subsup{r}{}{\prime\prime}s\mp@subsup{r}{}{\prime\prime}\mathrm{ where tbx: tighten-poly-bounds-for-x p?fr'}\mp@subsup{l}{}{\prime}\mp@subsup{r}{}{\prime}s\mp@subsup{r}{}{\prime}=(\mp@subsup{l}{}{\prime\prime},\mp@subsup{r}{}{\prime\prime},s\mp@subsup{r}{}{\prime\prime}
    by (cases rule: prod-cases3, auto)
    note rc = assms[unfolded fields]
    hence rc1: invariant-1 ( p,l,r) by auto
    have id: floor-1 x= ((if ?fr = ?fl then ?fr
        else if ?fr' < l' then ?fr else ?fl))
        unfolding fields floor-1.simps tbe Let-def split tbx by simp
    let ?}x=\mathrm{ real-of-1 }
    have x: ? }x=\mathrm{ the-unique-root ( p,l,r) unfolding fields by simp
    have bnd:l\leql' r ' 
        and rc': root-cond (p, l', r') (the-unique-root ( }p,l,r)
        and }s\mp@subsup{r}{}{\prime}:s\mp@subsup{r}{}{\prime}=sgn(ipoly p r '
        by (atomize(full), intro conjI tighten-poly-bounds-epsilon[OF - tbe refl],insert
rc,auto elim!: invariant-1E)
    let ?r = real-of-rat
    from rc'[folded x, unfolded split]
    have ineq: ?r l' }\mp@subsup{l}{}{\prime}\leq??\mathrm{ ? ? }x\leq\mathrm{ ?r r r' ?r l l' }\leq\mathrm{ ?r r r' by auto
    hence lr': l' }\leq\mp@subsup{r}{}{\prime}\mathrm{ unfolding of-rat-less-eq by simp
    have flr:?fl \leq?fr
        by (rule floor-mono[OF lr ])
    from invariant-1-sub-interval[OF rc1 rc' bnd(1,2)]
    have rc': invariant-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}
        and id': the-unique-root ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})=\mathrm{ the-unique-root ( }p,l,r)\mathrm{ by auto
    with rc have rc\mp@subsup{Q}{}{\prime}: invariant-1-2 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by auto
    have }x:
    unfolding fields using id' by simp
    {
    assume ?fr }\not=\mathrm{ ?fl
    with flr have flr:?fl \leq?fr - 1 by simp
    have ?fr'}\leq\mp@subsup{r}{}{\prime}\mp@subsup{l}{}{\prime}\leq?f\mp@subsup{r}{}{\prime}\mathrm{ using flr bnd by linarith+
    } note fl-diff = this
    show ?thesis
    proof (cases ?fr = ?fl)
    case True
    hence id1: floor-1 x = ?fr unfolding id by auto
    from True have id: floor (?r l') = floor (?r r')
        by simp
```

```
    have floor ? }x\leq\mathrm{ floor (?r r r)
    by (rule floor-mono[OF ineq(2)])
    moreover have floor (?r l') \leq floor ?x
    by (rule floor-mono[OF ineq(1)])
    ultimately have floor ?}x=\mathrm{ floor (?r r )
    unfolding id by (simp add: id)
    then show ?thesis by (simp add: id1)
next
    case False
    with id have id: floor-1 }x=(\mathrm{ if ?fr' < l" then ?fr else ?f) by simp
    from rcQ' have unique-root ( p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}) poly-cond2 p by auto
    from tighten-poly-bounds-for-x[OF this tbx sr']
    have ineq':}\mp@subsup{l}{}{\prime}\leq\mp@subsup{l}{}{\prime\prime}\mp@subsup{r}{}{\prime\prime}\leq\mp@subsup{r}{}{\prime}\mathrm{ and lr':}\mp@subsup{r}{}{\prime\prime}\leq\mp@subsup{r}{}{\prime\prime}\mathrm{ and rc': root-cond (p, l",},\mp@subsup{r}{}{\prime\prime})?
        and fr':}\neg(\mp@subsup{l}{}{\prime\prime}\leq?f\mp@subsup{r}{}{\prime}\wedge?f\mp@subsup{r}{}{\prime}\leq\mp@subsup{r}{}{\prime\prime})\mathrm{ unfolding }x\mathrm{ by auto
    from rc'"[unfolded split]
```



```
    from False have ?fr \not= ?f by auto
    note fr =fl-diff[OF this]
    show ?thesis
    proof (cases ?.fr' < l')
        case True
        with id have id: floor-1 x = ?fr by simp
        have floor?s}\leq\mathrm{ ?fr using floor-mono[OF ineq(2)] by simp
        moreover
        from True have ?r ?fr' < ?r l" unfolding of-rat-less .
        with ineq"(1) have ?r ?fr' }\leq?x\mathrm{ by simp
        from floor-mono[OF this]
        have?fr \leq floor ?x by simp
        ultimately show ?thesis unfolding id by auto
    next
        case False
        with id have id: floor-1 x=?f by simp
        from False have l'\ ?fr' by auto
        from floor-mono[OF ineq(1)] have ?f }\leq\mathrm{ floor ?x by simp
        moreover have floor ?x }\leq\mathrm{ ?fl
        proof -
            from False fr' have fr': r' < ?fr' by auto
            hence floor r" < ?fr by linarith
            with floor-mono[OF ineq'(2)]
            have floor? ? }\leq\mathrm{ ?fr - 1 by auto
            also have ?fr - 1 = floor (r' - 1) by simp
            also have ... \leq?fl
                    by (rule floor-mono, insert bnd, auto)
            finally show ?thesis.
        qed
        ultimately show ?thesis unfolding id by auto
        qed
    qed
qed
```


### 11.2.9 Generic Factorization and Bisection Framework

```
lemma card-1-Collect-ex1: assumes card (Collect P) = 1
    shows }\exists\mathrm{ ! x. P x
proof -
    from assms[unfolded card-eq-1-iff] obtain x where Collect P ={x} by auto
    thus ?thesis
        by (intro ex1I[of - x], auto)
qed
fun sub-interval :: rat }\times\mathrm{ rat }=>\mathrm{ rat }\times\mathrm{ rat }=>\mathrm{ bool where
    sub-interval (l,r) ( l',r') =( l'\leql^r\leqr')
fun in-interval :: rat }\times\mathrm{ rat }=>\mathrm{ real }=>\mathrm{ bool where
    in-interval (l,r) x = (of-rat l \leqx^x\leqof-rat r)
definition converges-to :: (nat }=>\mathrm{ rat }\times\mathrm{ rat ) }=>\mathrm{ real }=>\mathrm{ bool where
    converges-to fx\equiv(\forall n. in-interval (f n) x ^ sub-interval (f (Suc n)) (f n))
    \wedge(\forall(eps :: real)>0.\exists nlr.fn=(l,r)\wedgeof-ratr-of-rat l\leqeps)
context
    fixes bnd-update :: ' }a>>'
    and bnd-get :: 'a = rat }\times\mathrm{ rat
begin
definition at-step :: (nat }=>\mathrm{ rat }\times\mathrm{ rat ) }=>\mathrm{ nat }=>\mp@subsup{}{}{\prime}a=>\mathrm{ bool where
    at-step f n a \equiv\forall i. bnd-get ((bnd-update ~~ i) a)=f(n+i)
partial-function (tailrec) select-correct-factor-main
    :: 'a }=>\mathrm{ (int poly }\times\mathrm{ root-info)list }=>\mathrm{ (int poly }\times\mathrm{ root-info)list
        rat }=>\mathrm{ rat }=>\mathrm{ nat }=>\mathrm{ (int poly }\times\mathrm{ root-info })\times\mathrm{ rat }\times\mathrm{ 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 }=>\mathrm{ let m= root-info.l-r ri l r in
        if m}=0\mathrm{ then select-correct-factor-main bnd todo old l r n
        else select-correct-factor-main bnd todo ((p,ri) # old) lr (n+m))
definition select-correct-factor :: ' }a=>\mathrm{ (int poly }\times\mathrm{ root-info)list }
        (int poly }\times\mathrm{ root-info) }\times\mathrm{ rat }\times\mathrm{ 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 lr n = ((q,ri-fin),(l-fin,r-fin))
    and bnd: bnd-get a = (l,r)
    and ri: \bigwedge q ri. (q,ri) \in set todo \cup set old \Longrightarrow root-info-cond ri q
    and q0:\bigwedge q ri. (q,ri) \in set todo \cup set old \Longrightarrowq}\Longrightarrow二
```

and ex: $\exists q . q \in f s t$ 'set todo $\cup f$ ft' set old $\wedge$ ipoly $q x=0$
and dist: distinct (map fst (todo @ old))
and old: $\bigwedge q$ ri. $(q, r i) \in$ set old $\Longrightarrow$ root-info.l-r ri l $r \neq 0$
and un: $\bigwedge x::$ real. $(\exists q . q \in$ fst'set todo $\cup$ fst'set old $\wedge$ ipoly $q x=0) \Longrightarrow$
$\exists!q . q \in f s t$ ' set todo $\cup$ fst'set old $\wedge$ ipoly $q x=0$
and $n$ : $n=$ sum-list (map ( $\lambda(q$, ri $)$. root-info.l-r ri l r) old)
shows unique-root $(q, l$-fin,r-fin $) \wedge(q, r i$-fin $) \in$ set todo $\cup$ set old $\wedge x=$ the-unique-root
( $q, l$-fin, $r$-fin)
proof -
define orig where orig $=$ set todo $\cup$ set old
have orig: set todo $\cup$ set old $\subseteq$ orig unfolding orig-def by auto
let ? $\mathrm{rts}=\{x::$ real. $\exists$ qri. $(q$, ri $) \in$ orig $\wedge$ ipoly $q x=0\}$
define rts where rts $=$ ? rts
let $? h=\lambda(x, y)$. abs $(x-y)$
let $? r=$ real-of-rat
have rts: ? $r$ ts $=(\bigcup((\lambda(q, r i) .\{x$. ipoly $q x=0\})$ 'set (todo @ old $)))$ unfolding
orig-def by auto
have finite rts unfolding rts rts-def
using finite-ipoly-roots[OF q0] finite-set[of todo @ old] by auto
hence fin: finite (rts $\times$ rts $-I d$ ) by auto
define diffs where diffs $=$ insert $1\{a b s(x-y) \mid x y . x \in r t s \wedge y \in r t s \wedge x \neq$ $y\}$
have finite $\{a b s(x-y) \mid x y . x \in r t s \wedge y \in r t s \wedge x \neq y\}$
by (rule subst[of - finite, OF - finite-imageI[OF fin, of ?h]], auto)
hence diffs: finite diffs diffs $\neq\{ \}$ unfolding diffs-def by auto
define eps where eps = Min diffs / 2
have $\bigwedge x . x \in$ diffs $\Longrightarrow x>0$ unfolding diffs-def by auto
with Min-gr-iff [OF diffs] have eps: eps $>0$ unfolding eps-def by auto
note conv $=$ conv[unfolded converges-to-def]
from conv eps obtain $N L R$ where
$N: f N=(L, R)$ ?r $R-$ ?r $L \leq e p s$ by auto
obtain pair where pair: pair $=($ todo,$i)$ by auto
define rel where rel $=$ measures $[\lambda(t, i) . N-i, \lambda(t::($ int poly $\times$ root-info $)$
list, $i$ ). length $t$ ]
have wf: wf rel unfolding rel-def by simp
show ?thesis
using at res bnd ri q0 ex dist old un n pair orig
proof (induct pair arbitrary: todo $i$ old a l r n rule: wf-induct[OF wf])
case (1 pair todo $i$ old a l r n)
note $I H=1(1)[$ rule-format $]$
note $a t=1$ (2)
note res $=1(3)[$ unfolded select-correct-factor-main.simps[of - todo] $]$
note $b n d=1$ (4)
note $r i=1(5)$
note $q 0=1(6)$
note $e x=1(7)$
note dist $=1(8)$
note old $=1(9)$
note $u n=1(10)$
note $n=1$ (11)
note pair $=1$ (12)
note orig $=1$ (13)
from $a t[$ unfolded at-step-def, rule-format, of 0 ] bnd have $f$ : $f i=(l, r)$ by auto
with conv have inx: in-interval ( $f i$ ) $x$ by blast
hence lxr: ? $r l \leq x x \leq$ ? $r r$ unfolding $f i$ by auto
from order.trans[OF this] have $l r: l \leq r$ unfolding of-rat-less-eq.
show ?case
proof (cases todo)
case (Cons rri tod)
obtain $s$ ri where rri: rri $=(s, r i)$ by force
with Cons have todo: todo $=(s$, ri $) \#$ tod by simp
note res $=$ res $[$ unfolded todo list.simps split Let-def $]$
from root-info-condD(1)[OF ri[of s ri, unfolded todo] $l r$ ]
have ri' $^{\prime}$ : root-info.l-r ri l r $=$ card $\{x$. root-cond $(s, l, r) x\}$ by auto
from $q 0$ have $s 0: s \neq 0$ unfolding todo by auto
from finite-ipoly-roots[OF s0] have fins: finite $\{x$. root-cond $(s, l, r) x\}$
unfolding root-cond-def by auto
have rel: $(($ tod,$i)$, pair $) \in$ rel unfolding rel-def pair todo by simp
show ?thesis
proof (cases root-info.l-r ri l r $\quad$ )
case True
with res have res: select-correct-factor-main a tod old lr $n=((q$, ri-fin $)$,
$l$-fin, $r$-fin) by auto
from $r i^{\prime}[$ symmetric, unfolded True $]$ fins have empty: $\{x$. root-cond $(s, l, r)$
$x\}=\{ \}$ by simp
from ex lxr empty have ex': $(\exists q . q \in f s t$ 'set tod $\cup f s t$ 'set old $\wedge$ ipoly $q$
$x=0$ )
unfolding todo root-cond-def split by auto
have unique-root $(q, l$-fin, $r$-fin $) \wedge(q$, ri-fin $) \in$ set tod $\cup$ set old $\wedge$
$x=$ the-unique-root ( $q, l$-fin, $r$-fin)
proof (rule IH[OF rel at res bnd ri - ex' - - n refl], goal-cases)
case (5y) thus ?case using un [of $y$ ] unfolding todo by auto
next
case 2 thus ?case using $q 0$ unfolding todo by auto
qed (insert dist old orig, auto simp: todo)
thus ?thesis unfolding todo by auto
next
case False
with res have res: select-correct-factor-main a tod ((s, ri) \# old) lr
$(n+$ root-info.l-r ri l r $)=((q$, ri-fin $), l$-fin, $r$-fin $)$ by auto
from $e x$ have $e x^{\prime}: \exists q . q \in f s t$ 'set tod $\cup f s t$ 'set $((s, r i) \#$ old $) \wedge$ ipoly $q$ $x=0$
unfolding todo by auto
from dist have dist: distinct (map fst (tod @ (s, ri) \# old)) unfolding todo by auto
have $i d$ : set todo $\cup$ set old $=$ set tod $\cup$ set $((s$, ri $) \#$ old $)$ unfolding todo by $\operatorname{simp}$
show ?thesis unfolding id

```
            proof (rule IH[OF rel at res bnd ri - ex' dist], goal-cases)
                    case 4 thus ?case using un unfolding todo by auto
            qed (insert old False orig, auto simp: q0 todo n)
        qed
    next
        case Nil
        note res = res[unfolded Nil list.simps Let-def]
        from ex[unfolded Nil] lxr obtain s where s\infst'set old }\wedge root-cond (s,l,r
```

    then obtain \(q 1\) ri1 old' where old': old \(=(q 1\), ri1 \() \#\) old' using id by (cases
    old, auto)
let ?ri $=$ root-info.l-r ri1 lr
from old[unfolded old ] have 0 : ? ri $\neq 0$ by auto
from $n$ [unfolded old $\left.{ }^{\prime}\right] 0$ have $n 0: n \neq 0$ by auto
from ri[unfolded old ] have ri': root-info-cond ri1 q1 by auto
show ?thesis
proof (cases $n=1$ )
case False
with $n 0$ have $n 1: n>1$ by auto
obtain $l^{\prime} r^{\prime}$ where $b n d^{\prime}$ : bnd-get (bnd-update $\left.a\right)=\left(l^{\prime}, r^{\prime}\right)$ by force
with res False have res: select-correct-factor-main (bnd-update a) old [] $l^{\prime}$
$r^{\prime} 0=$
((q, ri-fin), l-fin, r-fin) by auto
have at': at-step $f$ (Suc i) (bnd-update a) unfolding at-step-def
proof (intro allI, goal-cases)
case (1 n)
have id: (bnd-update ~Suc n) $a=($ bnd-update $\sim n)$ (bnd-update a)
by (induct $n$, auto)
from at[unfolded at-step-def, rule-format, of Suc n]
show ?case unfolding id by simp
qed
from $0\left[\right.$ unfolded root-info-condD $\left.(1)\left[O F r i^{\prime} l r\right]\right]$ obtain $y 1$ where $y 1$ :
root-cond ( $q 1, l, r$ ) y1
by $($ cases Collect $($ root-cond $(q 1, l, r))=\{ \}$, auto)
from $n 1$ [unfolded $n$ old]
have ?ri $>1 \vee$ sum-list ( $\operatorname{map}(\lambda(q, r i)$. root-info.l-r ri l r) old $) \neq 0$
by (cases sum-list (map ( $\lambda$ ( $q$, ri). root-info.l-r ri l r) old'), auto)
hence $\exists$ q2 ri2 y2. $\left(q_{2}\right.$, ri2 $) \in$ set old $\wedge$ root-cond $(q 2, l, r) y 2 \wedge y 1 \neq y 2$
proof
assume ? $r i>1$
with root-info-condD(1)[OF ri' lr] have card $\{x$. root-cond $(q 1, l, r) x\}$
$>1$ by $\operatorname{simp}$
from card-gt-1D[OF this] y1 obtain y2 where root-cond ( $q 1, l, r$ ) y2 and
$y 1 \neq y 2$ by auto
thus ?thesis unfolding old' by auto
next
assume sum-list (map $\left(\lambda(q\right.$, ri $)$. root-info.l-r ri l r) old $\left.{ }^{\prime}\right) \neq 0$
then obtain $q 2$ ri2 where mem: $(q 2$, ri2 $) \in$ set old' and ri2: root-info.l-r
ri2 l $r \neq 0$ by auto
with $q 0$ ri have root-info-cond ri2 q2 unfolding old' by auto from ri2[unfolded root-info-condD(1)[OF this lr $]]$ obtain $y^{2} 2$ where $y 2$ :
root-cond (q2,l,r) y2 by $($ cases Collect $($ root-cond $(q 2, l, r))=\{ \}$, auto)
from dist[unfolded old] split-list[OF mem] have diff: q1 $\neq q 2$ by auto from $y 1$ have $q 1: q 1 \in f s t '$ set todo $\cup f s t '$ set old $\wedge$ ipoly $q 1$ y1 $=0$ unfolding old' root-cond-def by auto from $y^{2}$ have $q^{2}: q^{2} \in f s t$ ' set todo $\cup f s t$ 'set old $\wedge$ ipoly $q 2$ y2 $=0$ unfolding old' root-cond-def using mem by force have $y 1 \neq y^{2}$ proof
assume $i d$ : $y 1=y 2$ from $q 1$ have $\exists q 1 . q 1 \in f s t$ 'set todo $\cup f s t$ 'set old $\wedge$ ipoly q1 y1 $=0$ by blast
from $u n[O F$ this $] q 1 q 2[$ folded $i d]$ have $q 1=q 2$ by auto
with diff show False by simp
qed
with mem y2 show ?thesis unfolding old' by auto
qed
then obtain q2 ri2 y2 where
mem2: $\left(q_{2}\right.$, ri2 $) \in$ set old and y2: root-cond $(q 2, l, r) y_{2}$ and diff: $y 1 \neq$ y2 by auto
from mem2 orig have $(q 1$, ri1 $) \in$ orig $(q 2$, ri2 $) \in$ orig unfolding old' by auto
with y1 y2 diff have abs $(y 1-y 2) \in$ diffs unfolding diffs-def rts-def root-cond-def by auto
from Min-le[OF diffs(1) this] have abs $(y 1-y \mathcal{L}) \geq 2 *$ eps unfolding eps-def by auto
with eps have eps: abs $(y 1-y 2)>e p s$ by auto
from $y 1$ y2 have $l$ : of-rat $l \leq \min y 1$ y2 unfolding root-cond-def by auto
from y1 y2 have $r$ : of-rat $r \geq$ max y1 y2 unfolding root-cond-def by auto
from $l r$ eps have eps: of-rat $r-$ of-rat $l>e p s$ by auto
have $i<N$
proof (rule ccontr)
assume $\neg i<N$
hence $\exists k . i=N+k$ by presburger
then obtain $k$ where $i: i=N+k$ by auto
\{
fix $k l r$
assume $f(N+k)=(l, r)$
hence of-rat $r-o f$-rat $l \leq e p s$
proof (induct $k$ arbitrary: $l r$ )
case 0
with $N$ show ?case by auto
next
case (Suc klr)
obtain $l^{\prime} r^{\prime}$ where $f: f(N+k)=\left(l^{\prime}, r^{\prime}\right)$ by force
from $\operatorname{Suc}(1)[O F$ this $]$ have $I H$ : ?r $r^{\prime}-$ ?r $l^{\prime} \leq e p s$ by auto

```
            from f Suc(2) conv[THEN conjunct1, rule-format, of N+k]
            have ?r l \geq ?r l' ?r r \leq ? r r'
            by (auto simp: of-rat-less-eq)
            thus ?case using IH by auto
                qed
            } note * = this
            from at[unfolded at-step-def i, rule-format, of 0] bnd have f}(N+k)
(l,r) by auto
            from *[OF this] eps
            show False by auto
            qed
            hence rel: ((old, Suc i), pair) \in rel unfolding pair rel-def by auto
            from dist have dist: distinct (map fst (old @ [])) unfolding Nil by auto
            have id: set todo \cup set old = set old \cup set [] unfolding Nil by auto
            show ?thesis unfolding id
            proof (rule IH[OF rel at' res bnd' ri - dist - - refl], goal-cases)
            case 2 thus ?case using q0 by auto
            qed (insert ex un orig Nil, auto)
next
    case True
    with res old' have id: q = q1 ri-fin = ri1 l-fin =l r-fin =r by auto
    from n[unfolded True old] 0 have 1: ?ri = 1
        by (cases ?ri; cases ?ri - 1, auto)
    from root-info-condD(1)[OF ri' lr] 1 have card {x. root-cond (q1,l,r) x}
=1 by auto
    from card-1-Collect-ex1[OF this]
    have unique: unique-root (q1,l,r).
    from ex[unfolded Nil old'] consider (A) ipoly q1 x = 0
        | (B) q where q\in fst ' set old' ipoly q x = 0 by auto
    hence }x=\mathrm{ the-unique-root (q1,l,r)
    proof (cases)
        case }
        with lxr have root-cond (q1,l,r) x unfolding root-cond-def by auto
        from the-unique-root-eqI[OF unique this] show ?thesis by simp
    next
        case (Bq)
        with lxr have root-cond ( }q,l,r)x\mathrm{ unfolding root-cond-def by auto
        hence empty: {x. root-cond ( }q,l,r)x}\not={}\mathrm{ by auto
        from B(1) obtain ri' where mem: (q,ri') \in set old' by force
        from q0[unfolded old'] mem have q0: q\not=0 by auto
        from finite-ipoly-roots[OF this] have finite {x.root-cond (q,l,r)x}
            unfolding root-cond-def by auto
        with empty have card: card {x. root-cond (q,l,r) x} \not=0 by simp
        from ri[unfolded old] mem have root-info-cond ri' q by auto
    from root-info-condD(1)[OF this lr] card have root-info.l-r ri' l r\not=0 by
auto
        with n[unfolded True old'] 1 split-list[OF mem] have False by auto
        thus ?thesis by simp
    qed
```

```
            thus ?thesis unfolding id using unique ri' unfolding old' by auto
        qed
    qed
    qed
qed
lemma select-correct-factor: assumes
    conv: converges-to (\lambda i. bnd-get ((bnd-update ~ i) init)) x
    and res: select-correct-factor init polys = ((q,ri),(l,r))
    and ri: \bigwedge q ri. (q,ri) \in set polys \Longrightarrow root-info-cond ri q
    and q0: \bigwedge q ri. (q,ri) \in set polys \Longrightarrowq\not=0
    and ex:\existsq. q\in fst' set polys ^ ipoly q x = 0
    and dist: distinct (map fst polys)
    and un: \bigwedgex :: real. ( }\existsq.q\infst'set polys ^ ipoly q x = 0)
    \exists!q. q\in fst'set polys }\wedge\mathrm{ ipoly q x = 0
    shows unique-root ( }q,l,r)\wedge(q,ri)\in\mathrm{ set polys }\wedgex=\mathrm{ the-unique-root ( }q,l,r
proof -
    obtain l' r' where init: bnd-get init = ( l', r') by force
    from res[unfolded select-correct-factor-def init split]
    have res: select-correct-factor-main init polys [] l' r}\mp@subsup{r}{}{\prime}0=((q,ri),l,r) by aut
    have at: at-step ( }\lambda\mathrm{ i. bnd-get ((bnd-update ~ i ) init)) 0 init unfolding at-step-def
by auto
    have unique-root (q,l,r)^(q,ri)\in set polys \cup set [] ^x= the-unique-root ( }q,l,r
    by (rule select-correct-factor-main[OF conv at res init ri], insert dist un ex q0,
auto)
    thus ?thesis by auto
qed
definition real-alg-2' :: root-info }=>\mathrm{ int poly }=>\mathrm{ rat }=>\mathrm{ rat }=>\mathrm{ real-alg-2 where
    [code del]: real-alg-2' ri p l r = (
        if degree p = 1 then Rational (Rat.Fract ( }-\operatorname{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',}\mp@subsup{r}{}{\prime},s\mp@subsup{r}{}{\prime})=>(p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}))
lemma real-alg-2'-code[code]: real-alg-2' ri p l r =
    (if degree p = 1 then Rational (Rat.Fract ( }-\operatorname{coeff p 0) (coeff p 1))
        else case normalize-bounds-1
            (case tighten-poly-bounds-for-x p 0 l r (sgn (ipoly pr)) of (l', r', sr') => (p,
l',}\mp@subsup{r}{}{\prime})
        of ( }\mp@subsup{p}{}{\prime},l,r)=>\mathrm{ Irrational (root-info.number-root ri r) ( }\mp@subsup{p}{}{\prime},l,r)
    unfolding real-alg-2'-def real-alg-2-main-def
    by (cases tighten-poly-bounds-for-x p 0 l r (sgn (ipoly p r)), simp add: Let-def)
definition real-alg-2'\prime :: root-info }=>\mathrm{ int poly }=>\mathrm{ rat }=>\mathrm{ rat }=>\mathrm{ real-alg-2 where
    real-alg-2" ri p l r = (case normalize-bounds-1
            (case tighten-poly-bounds-for-x p Olr (sgn (ipoly pr)) of (l', r', sr') =>(p,
l',}\mp@subsup{r}{}{\prime})
        of ( }\mp@subsup{p}{}{\prime},l,r)=>\mathrm{ Irrational (root-info.number-root ri r) ( }\mp@subsup{p}{}{\prime},l,r)
```

```
lemma real-alg-2': degree p\not=1\Longrightarrow real-alg-2''ri p l r = real-alg-2' ri p l r
    unfolding real-alg-2'-code real-alg-2''-def by auto
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\not=0
proof
    from pc have p\not=0 by auto
    moreover assume ipoly p x=0
        note poly-zero[OF this]
    ultimately show False using deg by auto
qed
lemma real-alg-2':
    assumes ur: unique-root (q,l,r) and pc: poly-cond q and ri: root-info-cond ri q
        shows invariant-2 (real-alg-2' ri q l r) ^ real-of-2 (real-alg-2' ri q l r)=
the-unique-root (q,l,r) (is - ^ - = ?x)
proof (cases degree q Suc 0 rule: linorder-cases)
    case deg: less
    then have degree q=0 by auto
    from poly-cond-degree-0-imp-no-root[OF pc this] ur have False by force
    then show ?thesis by auto
next
    case deg: equal
    hence id: real-alg-2' ri q l r = Rational (Rat.Fract (- coeff q 0) (coeff q 1))
        unfolding real-alg-2'-def by auto
    show ?thesis unfolding id using degree-1-ipoly[OF deg]
        using unique-rootD(4)[OF ur] by auto
next
    case deg: greater
    with pc have pc2: poly-cond2 q by auto
    let ?rai = real-alg-2' ri q l r
    let ?r = real-of-rat
    obtain l' r'sr' where tight: tighten-poly-bounds-for-x q 0 lr (sgn (ipoly q r)) =
( l', r',sr')
    by (cases rule: prod-cases3, auto)
    let ?rai' = ( q, l', r')
    have rai': ?rai = real-alg-2-main ri ?rai'
        unfolding real-alg-2'-def using deg tight by auto
    hence rai: real-of-1 ?rai' = the-unique-root ( }q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by auto
    note tight = tighten-poly-bounds-for-x[OF ur pc2 tight refl]
    let ?x = the-unique-root (q,l,r)
    from tight have tight: root-cond (q, l', r') ?x l\leq l' l' \leq r'r r'}\leqr\mp@subsup{l}{}{\prime}>0\vee\mp@subsup{r}{}{\prime}
0 \text { by auto}
    from unique-root-sub-interval[OF ur tight(1) tight(2,4)] poly-condD[OF pc]
    have ur': unique-root ( }q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}\mathrm{ ) and x: ?x = the-unique-root ( }q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by auto
    from tight(2-) have sgn: sgn l}\mp@subsup{l}{}{\prime}=\operatorname{sgn}\mp@subsup{r}{}{\prime}\mathrm{ by auto
    show ?thesis unfolding rai' using real-alg-2-main[of ?rai' ri] invariant-1-realI[of
    ?rai' ?x]
```

```
    by (auto simp: tight(1) sgn pc ri ur')
qed
definition select-correct-factor-int-poly \(::\) ' \(a \Rightarrow\) int poly \(\Rightarrow\) real-alg-2 where
    select-correct-factor-int-poly init \(p \equiv\)
        let qs = factors-of-int-poly \(p\);
        polys \(=\operatorname{map}(\lambda q .(q\), root-info \(q)) q s ;\)
        \(((q, r i),(l, r))=\) select-correct-factor init polys
        in real-alg-2' ri q l r
lemma select-correct-factor-int-poly: assumes
    conv: converges-to ( \(\lambda\) 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 \(p: p \neq 0\)
    shows invariant-2 rai \(\wedge\) real-of-2 rai \(=x\)
proof -
    obtain \(q s\) where fact: factors-of-int-poly \(p=q s\) by auto
    define polys where polys \(=\operatorname{map}(\lambda q .(q\), root-info \(q)) q s\)
    obtain \(q\) ri \(l r\) where res: select-correct-factor init polys \(=((q, r i),(l, r))\)
    by (cases select-correct-factor init polys, auto)
    have fst: map fst polys \(=q s f s t\) ' set polys \(=\) set \(q s\) unfolding polys-def map-map
\(o-d e f\)
    by force+
    note fact \(^{\prime}=\) factors-of-int-poly \([\) OF fact \(]\)
    note rai \(=\) rai[unfolded select-correct-factor-int-poly-def Let-def fact,
    folded polys-def, unfolded res split]
    from fact' fst have dist: distinct (map fst polys) by auto
    from fact \(^{\prime}(2)\) [OF \(p\), of \(\left.x\right] x\) fst
    have ex: \(\exists q . q \in\) fst'set polys \(\wedge\) ipoly \(q x=0\) by auto
    \{
        fix \(q r i\)
        assume \((q, r i) \in\) set polys
        hence \(r i: r i=\) root-info \(q\) and \(q: q \in\) set \(q s\) unfolding polys-def by auto
        from fact \(^{\prime}(1)[O F q]\) have \(*\) : lead-coeff \(q>0\) irreducible \(q\) degree \(q>0\) by auto
        from \(*\) have \(q 0: q \neq 0\) by auto
        from root-info[OF \(*(2-3)]\) ri have ri: root-info-cond ri \(q\) by auto
        note ri q0 *
    \} note polys \(=\) this
    have unique-root \((q, l, r) \wedge(q, r i) \in\) set polys \(\wedge x=\) the-unique-root \((q, l, r)\)
    by (rule select-correct-factor[OF conv res polys(1) - ex dist, unfolded fst, OF -
- fact' (3) [OF p]],
    insert fact'(2)[OF p] polys(2), auto)
    hence ur: unique-root \((q, l, r)\) and mem: \((q, r i) \in\) set polys and \(x: x=\) the-unique-root
( \(q, l, r\) ) by auto
    note polys \(=\) polys [OF mem]
    from polys \((3-4)\) have ty: poly-cond \(q\) by (simp add: poly-cond-def)
    show ?thesis unfolding \(x\) rai[symmetric] by (intro real-alg-2' ur ty polys(1))
qed
```

end

### 11.2.10 Addition

lemma ipoly- $0-0[$ simp $]$ : ipoly $f\left(0::^{\prime} a::\{\right.$ comm-ring- 1 , ring-char- 0$\left.\}\right)=0 \longleftrightarrow$ poly $f 0=0$
unfolding poly- 0 -coeff- 0 by simp
lemma add-rat-roots-below[simp]: roots-below (poly-add-rat rp) $x=(\lambda y . y+o f-r a t$ $r$ )' roots-below $p$ ( $x-$ of-rat $r$ )
proof (unfold add-rat-roots image-def, intro Collect-eqI, goal-cases)
case (1 $y$ ) then show ?case by (auto intro: exI[of-y-real-of-rat r])
qed
lemma add-rat-root-cond:
shows root-cond (cf-pos-poly (poly-add-rat m $p$ ),l,r) $x=$ root-cond $(p, l-m, r$ - m) (x - of-rat m)
by (unfold root-cond-def, auto simp add: add-rat-roots hom-distribs)
lemma add-rat-unique-root: unique-root (cf-pos-poly (poly-add-rat m p), l,r)= unique-root ( $p, l-m, r-m$ )
by (auto simp: add-rat-root-cond)

```
fun add-rat-1 :: rat }=>\mathrm{ real-alg-1 }=>\mathrm{ 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 $)=c f$-pos-poly $($ poly-add-rat $r($ poly-real-alg-1 $y))$ by (cases y, auto simp: Let-def split: prod.split)
lemma sgn-cf-pos:
assumes lead-coeff $p>0$ shows sgn (ipoly (cf-pos-poly $p$ ) ( $x::{ }^{\prime} a::$ linordered-field) $)$
$=\operatorname{sgn}($ ipoly $p x)$
proof (cases $p=0$ )
case True with assms show ?thesis by auto
next
case False
from $c f$-pos-poly-main False obtain $d$ where $p^{\prime}$ : Polynomial.smult $d$ ( $c f$-pos-poly $p)=p$ by auto
have $d>0$
proof (rule zero-less-mult-pos2)
from False assms have $0<$ lead-coeff $p$ by (auto simp: cf-pos-def)
also from $p^{\prime}$ have $\ldots=d *$ lead-coeff (cf-pos-poly $p$ ) by (metis lead-coeff-smult)
finally show $0<\ldots$.

```
    show lead-coeff (cf-pos-poly p)>0 using False by (unfold lead-coeff-cf-pos-poly)
    qed
    moreover from p' have ipoly px=of-int d * ipoly (cf-pos-poly p) x
    by (fold poly-smult of-int-hom.map-poly-hom-smult, auto)
    ultimately show ?thesis by (auto simp: sgn-mult[where ' }a='='a]\mathrm{ )
qed
lemma add-rat-1: fixes r1 :: rat assumes inv-y: invariant-1-2 y
    defines z \equivadd-rat-1 r1 y
    shows invariant-1-2 z ^(real-of-1 z=of-rat r1 + real-of-1 y)
proof (cases y)
    case y-def:(fields p2 l2 r2)
    define p}\mathrm{ where p 三cf-pos-poly(poly-add-rat r1 p2)
    obtain l r sr where lr: tighten-poly-bounds-for-x p 0 (l2+r1) (r2+r1) (sgn
(ipoly p (r2+r1))) = (l,r,sr)
    by (metis surj-pair)
    from lr have z:z=(p,l,r) by (auto simp: y-def z-def p-def Let-def)
    from inv-y have ur: unique-root ( p,l2 + r1, r2 + r1)
        by (auto simp: p-def add-rat-root-cond y-def add-rat-unique-root)
    from inv-y[unfolded y-def invariant-1-2-def,simplified] have pc2: poly-cond2 p
        unfolding p-def
        apply (intro poly-cond2I poly-add-rat-irreducible poly-condI, unfold lead-coeff-cf-pos-poly)
        apply (auto elim!: invariant-1E)
        done
    note main = tighten-poly-bounds-for-x[OF ur pc2 lr refl, simplified]
    then have sgn l= sgn r unfolding sgn-if apply simp apply linarith done
    from invariant-1-2-realI[OF main(4)-main(7), simplified, OF this pc2] main(1-3)
ur
    show ?thesis by (auto simp: z p-def y-def add-rat-root-cond ex1-the-shift)
qed
fun tighten-poly-bounds-binary :: int poly }=>\mathrm{ int poly }=>(rat \timesrat \timesrat) > rat \times
rat }\timesrat =>(rat \timesrat \timesrat) > rat \times rat \times 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\equiv\mathrm{ the-unique-root ( }p1,l1,r1)\mathrm{ and }y\equiv\mathrm{ the-unique-root (p2,l2,r2)
    assumes bnd: \bigwedgel1 r1 l2 r2 l r sr1 sr2. I l1 \LongrightarrowIl2 \Longrightarrow root-cond (p1,l1,r1)x
loot-cond (p2,l2,r2) y \Longrightarrow
            bnd ((l1,r1,sr1),(l2,r2,sr2)) = (l,r)\Longrightarrowof-rat l\leqfxy^fxy\leqof-rat r
    and approx: \bigwedgel1 r1 l2 r2l1'r1'l2'r2'l l'r r'sr1 sr2 sr1'sr2'.
    Il1\LongrightarrowIl2 \Longrightarrow
    l1\leqr1 \Longrightarrowl2 \leqr2 \Longrightarrow
    (l,r) = bnd ((l1,r1,sr1), (l2,r2,sr2)) \Longrightarrow
    (l',r') = bnd ((l1',r1',sr1'), (l2',r\mp@subsup{2}{}{\prime},sr2')) \Longrightarrow
    (l1',r1') \in{(l1,(l1+r1)/2),((l1+r1)/2,r1)}\Longrightarrow
```

```
    (l2',r2') 
    (r'}-\mp@subsup{l}{}{\prime})\leq3/4*(r-l)\wedgel\leq\mp@subsup{l}{}{\prime}\wedge\mp@subsup{r}{}{\prime}\leq
    and I-mono: \bigwedgel l '. Il\Longrightarrowl\leq l'\LongrightarrowI l
    and I: I l1 I l2
    and sr:sr1 = sgn (ipoly p1 r1) sr2 = sgn (ipoly p2 r2)
    shows converges-to (\lambda i.bnd ((tighten-poly-bounds-binary p1 p2 ^~i) ((l1,r1,sr1),(l2,r2,sr2))))
        (f x y)
proof -
    let ?upd = tighten-poly-bounds-binary p1 p2
    define upd where upd = ?upd
    define init where init = ((l1,r1,sr1), l2, r2, sr2)
    let ?g = (\lambdai.bnd ((upd ~ i i) init))
    obtain lr where bnd-init: bnd init = (l,r) by force
    note ur1 = unique-rootD[OF ur(1)]
    note ur2 = unique-rootD[OF ur(2)]
    from ur1 (4) ur2(4) x-def y-def
    have rc1: root-cond ( }p1,l1,r1)x\mathrm{ and rc2: root-cond (p2,l2,r2) y by auto
    define g}\mathrm{ where }g=?
    {
    fix i L1 R1 L2 R2 L R j SR1 SR2
    assume ((upd ~ i)) init = ((L1,R1,SR1),(L2,R2,SR2)) g i = (L,R)
    hence I L1 ^I L2 ^ root-cond (p1,L1,R1) x ^ root-cond (p2,L2,R2) y ^
        unique-root (p1,L1, R1) ^ unique-root (p2,L2, R2) ^ in-interval (L,R) (f
x y)^
    (i=Suc j\longrightarrow sub-interval (gi) (g j)^(R-L\leq3/4*(snd (g j) - fst (g
j))))
    \wedge SR1 = sgn (ipoly p1 R1) ^ SR2 = sgn (ipoly p2 R2)
    proof (induct i arbitrary: L1 R1 L2 R2 L R j SR1 SR2)
        case 0
    thus ?case using I rc1 rc2 ur bnd[of l1 l2 r1 r2 sr1 sr2 L R] g-def sr unfolding
init-def by auto
    next
        case (Suc i)
        obtain l1 r1 l2 r2 sr1 sr2 where updi:(upd ~ i) init = ((l1, r1, sr1), l2,
r2, sr2) by (cases (upd ~ i) init, auto)
    obtain lr where bndi: bnd ((l1, r1, sr1), l2, r2, sr2) = (l,r) by force
    hence gi: gi=(l,r) using updi unfolding g-def by auto
    have (upd^^ Suc i) init = upd ((l1,r1,sr1), l2, r2, sr2) using updi by
simp
    from Suc(2)[unfolded this] have upd: upd ((l1,r1, sr1), l2, r2, sr2) = ((L1,
R1,SR1), L2, R2, SR2) .
    from upd updi Suc(3) have bndsi: bnd ((L1, R1,SR1), L2, R2,SR2) =
(L,R) by (auto simp: g-def)
    from Suc(1)[OF updi gi] have I: I l1 I l2
            and rc: root-cond ( p1,l1,r1) x root-cond ( p2,l2,r2) y
            and ur: unique-root (p1,l1, r1) unique-root (p2, l2, r2)
            and sr:sr1 = sgn (ipoly p1 r1) sr2 = sgn (ipoly p2 r2)
            by auto
    from upd[unfolded upd-def]
```

have tight: tighten-poly-bounds p1 l1 r1 sr1 $=($ L1, R1, SR1 $)$ tighten-poly-bounds p2 l2 r2 sr2 $=(L 2, R 2, S R 2)$
by auto
note tight1 $=$ tighten-poly-bounds $[O F \operatorname{tight}(1) \operatorname{ur}(1) \operatorname{pt}(1) \operatorname{sr}(1)]$
note tight2 $=$ tighten-poly-bounds[OF tight(2) $\operatorname{ur}($ (2) $\operatorname{pt}(2) \operatorname{sr}(2)]$
from tight1 have $\operatorname{lr} 1: l 1 \leq r 1$ by auto
from tight2 have lr2: $12 \leq$ r2 by auto
note ur1 $=$ unique-rootD $[$ OF $\operatorname{ur}(1)]$
note ur2 $=$ unique-rootD $[$ OF ur(2)]
from tight1 I-mono[OF I(1)] have I1: I L1 by auto
from tight2 I-mono[OF I(2)] have I2: I L2 by auto
note ur1 $=$ unique-root-sub-interval[OF ur(1) tight1 (1,2,4)]
note $\operatorname{ur2}=$ unique-root-sub-interval $[O F \operatorname{ur}(2) \operatorname{tight2}(1,2,4)]$
from $r c(1)$ ur ur1 have $x: x=$ the-unique-root $(p 1, L 1, R 1)$ by (auto in-tro!:the-unique-root-eqI)
from $\operatorname{rc}(2)$ ur ur2 have $y: y=$ the-unique-root $(p 2, L 2, R 2)$ by (auto in-tro!:the-unique-root-eqI)
from unique-rootD $[$ OF $\operatorname{ur} 1(1)] x$ have $x$ : root-cond $(p 1, L 1, R 1) x$ by auto
from unique-rootD $[$ OF ur2 (1) $] y$ have $y$ : root-cond ( $p 2, L 2, R 2$ ) $y$ by auto
from $\operatorname{tight}(1)$ have half1: $(L 1, R 1) \in\{(l 1,(l 1+r 1) / 2),((l 1+r 1) / 2$, r1) \}
unfolding tighten-poly-bounds-def Let-def by (auto split: if-splits)
from $\operatorname{tight}(2)$ have half2: $(L 2, R 2) \in\{(12,(l 2+r 2) / 2),((l 2+r 2) / 2$, r2) $\}$
unfolding tighten-poly-bounds-def Let-def by (auto split: if-splits)
from approx[OF I lr1 lr2 bndi[symmetric] bndsi[symmetric] half1 half2]
have $R-L \leq 3 / 4 *(r-l) \wedge l \leq L \wedge R \leq r$.

unfolding gi Suc(3) by auto
with bnd[OF I1 I2 $x$ y bndsi]
show ?case using I1 I2 $x$ y ur1 ur2 tight1 (6) tight2(6) by auto
qed
$\}$ note invariants $=$ this
define $L$ where $L=(\lambda i$.fst $(g i))$
define $R$ where $R=(\lambda i$ snd $(g i))$
\{
fix $i$
obtain l1 r1 l2 r2 sr1 sr2 where updi: (upd ^^i) init $=((l 1, r 1, s r 1), l 2, r 2$, sr2) by (cases (upd ~i) init, auto)
obtain $l r$ where $b n d^{\prime}$ : bnd $((l 1, r 1, s r 1), l 2, r 2, s r 2)=(l, r)$ by force
have $g i: g i=(l, r)$ unfolding $g$-def updi bnd ${ }^{\prime}$ by auto
hence $i d: l=L$ ir $=R$ i unfolding $L$-def $R$-def by auto
from invariants[OF updi gi[unfolded id]]
have in-interval ( $L i, R i$ ) ( $f x y$ )
$\wedge j . i=S u c j \Longrightarrow$ sub-interval $(g i)(g j) \wedge R i-L i \leq 3 / 4 *(R j-L j)$
unfolding $L$-def $R$-def by auto
\} note $*=$ this
\{
fix $i$

```
    from *(1)[of i]*(2)[of Suc i,OF refl]
    have in-interval (g i) (fxy) sub-interval (g (Suc i)) (g i)
    R(Suc i)-L(Suc i)\leq3/4*(Ri-L i) unfolding L-def R-def by auto
} note * = this
show ?thesis unfolding upd-def[symmetric] init-def[symmetric] g-def[symmetric]
    unfolding converges-to-def
proof (intro conjI allI impI, rule *(1), rule *(2))
    fix eps :: real
    assume eps: 0<eps
    let ?r = real-of-rat
    define }r\mathrm{ where }r=(\lambdan\mathrm{ . ? r (R n))
    define l where l=( }\lambdan\mathrm{ n. ?r (Ln))
    define diff where diff =(\lambdan.rn-ln)
    {
        fix n
        from *(3)[of n] have ?r (R (Suc n) -L (Suc n)) \leq?r (3 / 4* (Rn-L
n))
            unfolding of-rat-less-eq by simp
        also have ?r (R (Suc n)-L(Suc n)) = (r (Suc n) -l (Suc n))
            unfolding of-rat-diff r-def l-def by simp
        also have ?r (3 / 4* (Rn-Ln))=3 / 4*(rn-ln)
            unfolding r-def l-def by (simp add: hom-distribs)
    finally have diff (Suc n)\leq3/4* diff n unfolding diff-def .
    } note * = this
    {
        fix }
        have diff i\leq(3/4)`i* diff 0
        proof (induct i)
            case (Suc i)
            from Suc*[of i] show ?case by auto
        qed auto
    }
    then obtain c where *: \bigwedgei.diff i\leq(3/4)`i*c by auto
    have \exists n. diff n\leqeps
    proof (cases c \leq 0)
        case True
        with *[of 0] eps show ?thesis by (intro exI[of - 0], auto)
    next
        case False
    hence c:c>0 by auto
    with eps have inverse c*eps>0 by auto
    from exp-tends-to-zero[of 3/4 :: real, OF - this] obtain n where
        (3/4) ^ n \leq inverse c*eps by auto
    from mult-right-mono[OF this, of c] c
    have (3/4) ^ n*c\leqeps by (auto simp: field-simps)
    with *[of n] show ?thesis by (intro exI[of-n], auto)
    qed
    then obtain n where ?r (R n) - ?r (L n) \leqeps unfolding l-def r-def diff-def
by blast
```

thus $\exists n l r . g n=(l, r) \wedge ? r r-? r l \leq e p s$ unfolding $L$-def $R$-def by (intro exI[of - $n]$, force)
qed
qed
fun add-1 :: real-alg-1 $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
add-1 $(p 1, l 1, r 1)(p 2, l 2, r 2)=($
select-correct-factor-int-poly
(tighten-poly-bounds-binary p1 p2)
$(\lambda((l 1, r 1, s r 1),(l 2, r 2, s r 2)) \cdot(l 1+l 2, r 1+r 2))$
((l1, r1, sgn (ipoly p1r1)),(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 $z: z \equiv a d d-1 x y$
shows invariant-2 $z \wedge($ real-of- $2 z=$ real-of- $1 x+$ real-of-1 $y)$
proof (cases $x$ )
case $x$ : (fields p1 l1 r1)
show ?thesis
proof (cases y)
case yt: (fields p2 l2 r2)
let $? x=$ real-of- $1(p 1, l 1, r 1)$
let $? y=$ real-of-1 $(p 2, l 2, r 2)$
let $? p=$ poly-add $p 1 p 2$
note $x=x[$ unfolded $x t]$
note $y=y$ [unfolded $y t$ ]
from $x$ have ax: p1 represents ?x unfolding represents-def by (auto elim!: invariant-1E)
from $y$ have ay: p2 represents ?y unfolding represents-def by (auto elim!: invariant-1E)
let $? b n d=(\lambda((l 1, r 1, s r 1 ~:: ~ r a t), ~ l 2 ~:: ~ r a t, ~ r 2 ~:: ~ r a t, ~ s r 2 ~:: ~ r a t ~) ~ . ~(l 1 ~+~ l 2, ~ r 1 ~$ $+r$ 2))
define $b n d$ where $b n d=? b n d$
have invariant-2 $z \wedge$ real-of- $2 z=? x+$ ? $y$
proof (intro select-correct-factor-int-poly)
from represents-add[OF ax ay]
show $? p \neq 0$ ipoly $? p(? x+? y)=0$ by auto
from $z[$ unfolded $x t y t]$
show sel: select-correct-factor-int-poly
(tighten-poly-bounds-binary p1 p2)
bnd
((l1,r1,sgn (ipoly p1 r1)), (l2,r2, sgn (ipoly p2 r2)))
(poly-add p1 p2) $=z$ by (auto simp: bnd-def)
have ur1: unique-root ( $p 1, l 1, r 1$ ) poly-cond2 $p 1$ using $x$ by auto
have ur2: unique-root ( $p 2,12, r 2$ ) poly-cond2 p2 using $y$ by auto show converges-to
( $\lambda$ i. bnd ( tighten-poly-bounds-binary p1 p2 ~~i)

$$
((l 1, r 1, \operatorname{sgn}(\text { ipoly p1 r1) })),(l 2, r 2, \text { sgn }(\text { ipoly p2 r2 })))))(? x+? y)
$$

```
        by (intro tighten-poly-bounds-binary ur1 ur2; force simp: bnd-def hom-distribs)
    qed
    thus ?thesis unfolding xt yt .
    qed
qed
```

declare add-rat-1.simps[simp del]
declare add-1.simps[simp del]

### 11.2.11 Multiplication

## context

begin
private fun mult-rat-1-pos :: rat $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
 $r 2 * r 1$ )
private fun mult-1-pos :: real-alg-1 $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
mult-1-pos $(p 1, l 1, r 1)(p 2, l 2, r 2)=$
select-correct-factor-int-poly
(tighten-poly-bounds-binary p1 p2)
$(\lambda((l 1, r 1, s r 1),(l 2, r 2, s r 2)) \cdot(l 1 * l 2, r 1 * r 2))$
$((l 1, r 1$, sgn $($ ipoly $p 1 r 1)),(l 2, r 2$, sgn (ipoly p2 r2) $))$ (poly-mult p1 p2)
fun mult-rat-1 :: rat $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
mult-rat-1 $x$ y $=$
(if $x<0$ then uminus-2 (mult-rat-1-pos $(-x) y$ )
else if $x=0$ then Rational 0 else (mult-rat-1-pos $x y$ ))
fun mult-1 :: real-alg-1 $\Rightarrow$ real-alg-1 $\Rightarrow$ real-alg-2 where
mult-1 $x y=($ case $(x, y)$ of $((p 1, l 1, r 1),(p 2, l 2, r 2)) \Rightarrow$
if $r 1>0$ 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 $r 1$ :: rat assumes $r 1: r 1>0$ and $y$ : invariant-1 $y$ defines $z: z \equiv$ mult-rat-1-pos r1 y
shows invariant-2 $z \wedge($ real-of-2 $z=o f$-rat r1 $*$ real-of-1 $y)$
proof -
obtain p2 l2 r2 where $y t: y=(p 2,12, r 2)$ by (cases $y$, auto)
let $? x=$ real-of-rat r1
let $? y=$ real-of- $1(p 2, l 2, r 2)$
let ?p $=$ poly-mult-rat r1 p2
let $? m p=c f$-pos-poly ?p
note $y=y[$ unfolded $y t]$
note $y D=$ invariant $-1 D[$ OF $y]$
from $y D r 1$ have $p: ? p \neq 0$ and $r 10: r 1 \neq 0$ by auto
hence $m p:$ ? $m p \neq 0$ by $\operatorname{simp}$
from $y D(1)$
have rt: ipoly p2 $? y=0$ and bnd: of-rat l2 $\leq ? y ? y \leq o f$-rat r2 by auto
from rt r1 have rt: ipoly ? $m p(? x * ? y)=0$ by (auto simp add: field-simps ipoly-mult-rat[OF r10])
from $y D(5)$ have irr: irreducible p2
unfolding represents-def using $y$ unfolding root-cond-def split by auto
from poly-mult-rat-irreducible[OF this - r10] yD
have irr: irreducible ? mp by simp
from $p$ have mon: cf-pos ? mp by auto
obtain $l r$ where $l r: l=l 2 * r 1 r=r 2 * r 1$ by force
from bnd $r 1$ have bnd: of-rat $l \leq ? x * ? y$ ? $x *$ ? $y \leq$ of-rat $r$ unfolding $l r$ of-rat-mult by auto
with $r$ have $r$ c: root-cond (?mp,l,r) (? $x * ? y$ ) unfolding root-cond-def by auto have ur: unique-root (?mp,l,r)
proof (rule ex1I, rule rc)
fix $z$
assume root-cond (? $m p, l, r$ ) z
from this[unfolded root-cond-def split] have bndz: of-rat $l \leq z z \leq o f$-rat $r$ and $r t$ : ipoly ? $m p z=0$ by auto
have fst (quotient-of r1) $=0$ using quotient-of-div[of r1] r10 by (cases quo-tient-of r1, auto)
with $r t$ have $r$ : ipoly $p 2(z *$ inverse ? $x)=0$ by (auto simp: ipoly-mult-rat $[O F$ r10])
from $b n d z r 1$ have of-rat l2 $\leq z *$ inverse ? $x z *$ inverse ? $x \leq$ of-rat r2 unfolding lr of-rat-mult
by (auto simp: field-simps)
with $r t$ have root-cond ( $p 2,12, r 2$ ) $(z *$ inverse ? $x)$ unfolding root-cond-def by auto
also note invariant-1-root-cond[OF y]
finally have ? $y=z *$ inverse ? $x$ by auto
thus $z=? x * ? y$ using $r 1$ by auto
qed
from $r 1$ have sgnr: sgn $r=\operatorname{sgn} r 2$ unfolding $l r$
by (cases r2 $=0$; cases r2 $<0$; auto simp: mult-neg-pos mult-less- 0 -iff)
from $r 1$ have sgnl: sgn $l=\operatorname{sgn}$ l2 unfolding $l r$
by (cases l2 $=0$; cases $12<0$; auto simp: mult-neg-pos mult-less- 0 -iff)
from the-unique-root-eqI[OF ur rc] have $x y$ : ? $x *$ ? $y=$ the-unique-root (?mp,l,r)
by auto
from $z[$ unfolded $y t$, simplified, unfolded Let-def lr[symmetric] split]
have $z: z=$ real-alg-2 ( $? m p, l, r$ ) by simp
have yp2: p2 represents ? $y$ using $y D$ unfolding root-cond-def split represents-def
by auto
with irr mon have pc: poly-cond ?mp by (auto simp: poly-cond-def cf-pos-def)
have rc: invariant-1 (? $m p, l, r$ ) unfolding $z$ using $y D(2) p c u r$ by (auto simp add: invariant-1-def ur mp sgnr sgnl)
show ?thesis unfolding $z$ using real-alg-2 [OF rc]
unfolding $y t x y$ unfolding $z$ by simp

## qed

lemma mult-1-pos: assumes $x$ : invariant-1-2 $x$ and $y$ : invariant-1-2 $y$
defines $z: z \equiv$ mult-1-pos $x$ y
assumes pos: real-of-1 $x>0$ real-of-1 $y>0$
shows invariant-2 $z \wedge($ real-of- $2 z=$ real-of- $1 x *$ real-of- $1 y)$
proof -
obtain p1 l1 r1 where $x t: x=(p 1, l 1, r 1)$ by (cases $x$, auto)
obtain p2 l2 r2 where $y t: y=(p 2$, l2,r2 $)$ by (cases $y$, auto $)$
let $? x=$ real-of- $1(p 1, l 1, r 1)$
let $? y=$ real-of-1 $(p 2, l 2, r 2)$
let $? r=$ real-of-rat
let $? p=$ poly-mult $p 1$ p2
note $x=x[$ unfolded $x t]$
note $y=y[u n f o l d e d y t]$
from $x y$ have basic: unique-root ( $p 1, l 1, r 1$ ) poly-cond2 p1 unique-root ( $p 2$, l2, r2) poly-cond2 p2 by auto
from basic have irr1: irreducible p1 and irr2: irreducible p2 by auto
from $x$ have ax: p1 represents ? $x$ unfolding represents-def by (auto elim!:invariant-1E)
from $y$ have ay: p2 represents ? y unfolding represents-def by (auto elim!:invariant-1E)
from ax ay pos[unfolded $x t y t]$ have axy: ?p represents (? $x *$ ?y)
by (intro represents-mult represents-irr-non-0[OF irr2], auto)
from represents $D[$ OF this $]$ have $p: ? p \neq 0$ and rt: ipoly ? $p(? x * ? y)=0$.
from $x \operatorname{pos}(1)$ [unfolded $x t$ ] have ? $r ~ r 1>0$ unfolding split by auto
hence sgn $r 1=1$ unfolding sgn-rat-def by (auto split: if-splits)
with $x$ have sgn $l 1=1$ by auto
hence l1-pos: $l 1>0$ unfolding sgn-rat-def by (cases $l 1=0$; cases $l 1<0$; auto)
from $y \operatorname{pos}(2)[$ unfolded $y t]$ have ? $r$ r2 $>0$ unfolding split by auto
hence sgn r2 $=1$ unfolding sgn-rat-def by (auto split: if-splits)
with $y$ have sgn $12=1$ by auto
hence l2-pos: $12>0$ unfolding sgn-rat-def by (cases $12=0$; cases $12<0$; auto)
 r2))
define $b n d$ where $b n d=$ ?bnd
obtain $z^{\prime}$ where sel: select-correct-factor-int-poly
(tighten-poly-bounds-binary p1 p2)
bnd
$((l 1, r 1, \operatorname{sgn}($ ipoly p1 r1) $),($ l2, r2, $\operatorname{sgn}($ ipoly $p 2 r 2)))$ ? $p=z^{\prime}$ by auto
have main: invariant-2 $z^{\prime} \wedge$ real-of- $2 z^{\prime}=? x * ? y$
proof (rule select-correct-factor-int-poly[OF - sel rt p])
\{
fix $11 r 112 r 2 l 1^{\prime} r 1^{\prime} l 2^{\prime} r 2^{\prime} l l^{\prime} r r^{\prime}:: ~ r a t$
let $? m 1=(l 1+r 1) / 2$ let $? m 2=(l 2+r 2) / 2$
define $d 1$ where $d 1=r 1-l 1$
define $d 2$ where $d 2=r 2-12$
let $? M 1=l 1+d 1 / 2$ let $? M 2=12+d 2 / 2$
assume le: $l 1>0 l 2>0 l 1 \leq r 1 l 2 \leq r 2$ and $i d:(l, r)=(l 1 * l 2, r 1 * r 2)$
$\left(l^{\prime}, r\right)=\left(l 1^{\prime} * l 2^{\prime}, r 1^{\prime} * r 2^{\prime}\right)$
and mem: $\left(l 1^{\prime}, r 1^{\prime}\right) \in\{(l 1, ? m 1),(? m 1, r 1)\}$ $\left(l 2^{\prime}, r 2^{\prime}\right) \in\{(l 2, ? m 2),(? m 2, r 2)\}$
hence $i d$ : $l=l 1 * l 2 r=(l 1+d 1) *(l 2+d 2) l^{\prime}=l 1^{\prime} * l 2^{\prime} r^{\prime}=r 1^{\prime} * r 2^{\prime}$ $r 1=l 1+d 1 r 2=12+d 2$ and $i d^{\prime}: ? m 1=? M 1 ? m 2=? M 2$
unfolding d1-def d2-def by (auto simp: field-simps)
define $l 1 d 1$ where $l 1 d 1=l 1+d 1$
from le have ge0:d1 $\geq 0 d 2 \geq 0 l 1 \geq 0 l 2 \geq 0$ unfolding d1-def d2-def by auto
have $4 *\left(r^{\prime}-l^{\prime}\right) \leq 3 *(r-l)$
proof $\left(\right.$ cases $\left.l 1^{\prime}=l 1 \wedge r 1^{\prime}=? M 1 \wedge l 2^{\prime}=l 2 \wedge r 2^{\prime}=? M 2\right)$
case True
hence $i d 2$ : $l 1^{\prime}=l 1 r 1^{\prime}=? M 112^{\prime}=12 r 2^{\prime}=?$ M2 by auto
show ?thesis unfolding id id2 unfolding ring-distribs using ge0 by simp next
case False note $1=$ this
show ?thesis
proof $\left(\right.$ cases $\left.l 1^{\prime}=l 1 \wedge r 1^{\prime}=? M 1 \wedge l 2^{\prime}=? M 2 \wedge r Q^{\prime}=r \mathcal{2}\right)$
case True
hence $i d 2: l 1^{\prime}=l 1 r 1^{\prime}=$ ?M1 $12^{\prime}=$ ?M2 $r 2^{\prime}=r 2$ by auto
show ?thesis unfolding id id2 unfolding ring-distribs using ge0 by simp next
case False note $2=$ this
show ?thesis
proof $\left(\right.$ cases $\left.l 1^{\prime}=? M 1 \wedge r 1^{\prime}=r 1 \wedge l 2^{\prime}=l 2 \wedge r 2^{\prime}=? M 2\right)$
case True
hence $i d 2: l 1^{\prime}=?$ M1 $r 1^{\prime}=r 112^{\prime}=l 2 r 2^{\prime}=? M 2$ by auto
show ?thesis unfolding id id2 unfolding ring-distribs using ge0 by simp next
case False note $3=$ this
from 123 mem have $i d 2: l 1^{\prime}=? M 1 r 1^{\prime}=r 1 l 2^{\prime}=? M 2 r 2^{\prime}=r 2$ unfolding $i d^{\prime}$ by auto
show ?thesis unfolding id id2 unfolding ring-distribs using ge0 by simp qed
qed
qed
hence $r^{\prime}-l^{\prime} \leq 3 / 4 *(r-l)$ by $\operatorname{simp}$
$\}$ note decr $=$ this
show converges-to
( $\lambda$ i. bnd ( $($ tighten-poly-bounds-binary p1 p2 ~ $i)$
$((l 1, r 1, \operatorname{sgn}($ ipoly p1 r1) $),(l 2$, r2, sgn (ipoly p2 r2) $))))(? x * ? y)$
proof (intro tighten-poly-bounds-binary $[$ where $f=(*)$ and $I=\lambda l . l>0]$
basic l1-pos l2-pos, goal-cases)
case (1 L1 R1 L2 R2 L R)
hence $L=L 1 * L 2 R=R 1 * R 2$ unfolding bnd-def by auto
hence id: ?r $L=$ ?r $L 1 *$ ?r L2 ?r $R=$ ?r R1*?r R2 by (auto simp: hom-distribs)
from $1(3-4)$ have $l e$ : ?r $L 1 \leq$ ? $x$ ? $x \leq$ ?r R1 ?r L2 $\leq$ ? $y$ ? $y \leq$ ?r R2
unfolding root-cond-def by auto
from 1 (1-2) have $l t: 0<$ ? L1 $0<$ ? L2 by auto
from mult-mono $[$ OF le $(1,3)$, folded id] lt le have $L$ : ?r $L \leq$ ? $x *$ ? $y$ by linarith
have $R: ? x * ? y \leq$ ? $R$
by (rule mult-mono[OF le(2,4), folded id], insert lt le, linarith + )
show ?case using $L R$ by blast

## next

case (2 l1 r1 l2 r2 $\left.11^{\prime} r 1^{\prime} l 2^{\prime} r 2^{\prime} l l^{\prime} r r r^{\prime}\right)$
from 2(5-6) have $l r: l=l 1 * l 2 r=r 1 * r 2 l^{\prime}=l 1^{\prime} * l 2^{\prime} r^{\prime}=r 1^{\prime} * r 2^{\prime}$ unfolding bnd-def by auto
from $2(1-4)$ have le: $0<l 10<l 2 l 1 \leq r 1 l 2 \leq r 2$ by auto
from $2(7-8) l e$ have $l e^{\prime}: l 1 \leq l 1^{\prime} r 1^{\prime} \leq r 1 l 2 \leq l 2^{\prime} r 2^{\prime} \leq r 20<r 2^{\prime} 0<$ r2 by auto
from mult-mono[OF $l e^{\prime}(1,3)$, folded $\left.l r\right] l e l e^{\prime}$ have $l: l \leq l^{\prime}$ by auto
have $r: r^{\prime} \leq r$ by (rule mult-mono[OF le'(2,4), folded lr], insert le le ${ }^{\prime}$, linarith+)
have $r^{\prime}-l^{\prime} \leq 3 / 4 *(r-l)$
by (rule decr $[O F---2(7-8)]$, insert le le ${ }^{\prime} l r$, auto)
thus ?case using $l r$ by blast
qed auto
qed
have $z^{\prime}: z^{\prime}=z$ unfolding $z[$ unfolded $x t y t$, simplified, unfolded bnd-def[symmetric] sel]
by auto
from main[unfolded this] show ?thesis unfolding $x t$ yt by simp qed
lemma mult-1: assumes $x$ : invariant-1-2 $x$ and $y$ : invariant-1-2 $y$ defines $z[$ simp $]: z \equiv$ mult- 1 x $y$
shows invariant-2 $z \wedge$ (real-of-2 $z=$ real-of- $1 x *$ real-of-1 $y$ )
proof -
obtain $p 1 l 1$ r1 where $x t[s i m p]: x=(p 1, l 1, r 1)$ by $($ cases $x)$
obtain p2 l2 r2 where $y t[$ simp $]: y=(p 2$, l2, r2) by $($ cases $y)$
let $? x t=(p 1, l 1, r 1)$
let $? y t=(p 2,12, r 2)$
let $? x=$ real-of-1 ? $x t$
let $? y=$ real-of-1 ?yt
let $? m x t=$ uminus -1 ? $x t$
let $? m y t=$ uminus -1 ? yt
let ? $m x=$ real-of-1 ? $m x t$
let $? m y=$ real-of-1 ?myt
let $? r=$ real-of-rat
from invariant-1-2-of-rat [OF $x$, of 0$]$ have $x 0: ? x<0 \vee ? x>0$ by auto
from invariant-1-2-of-rat [OF y, of 0] have y0: ? $y<0 \vee ? y>0$ by auto
from uminus-1-2[OF $x]$ have $m x$ : invariant-1-2 ? $m x t$ and $[$ simp $]: ~ ? m x=-? x$ by auto
from uminus-1-2[OF y] have my: invariant-1-2 ?myt and $[s i m p]: ? m y=-? y$ by auto

```
have \(i d: r 1>0 \longleftrightarrow\) ? \(x>0 r 1<0 \longleftrightarrow\) ? \(x<0\) r2 \(>0 \longleftrightarrow\) ? \(y>0\) r2 \(<0\)
\(\longleftrightarrow\) ? \(y<0\)
    using \(x y\) by auto
    show ?thesis
    proof (cases ? \(x>0\) )
    case \(x 0\) : True
    show ?thesis
    proof (cases \(? y>0\) )
        case \(y 0\) : True
        with \(x\) y x0 mult-1-pos[OF \(x y]\) show ?thesis by auto
    next
        case False
        with \(y 0\) have \(y 0: ? y<0\) by auto
        with \(x 0\) have \(z: z=\) uminus-2 (mult-1-pos ? \(x t\) ?myt)
            unfolding \(z\) xt yt mult-1.simps split id by simp
        from \(x 0\) y0 mult-1-pos[OF x my] uminus-2 [of mult-1-pos ?xt ?myt]
        show ?thesis unfolding \(z\) by simp
    qed
    next
        case False
        with \(x 0\) have \(x 0\) : ? \(x 0<0\) by simp
    show ?thesis
    proof (cases ?y \(>0\) )
        case y0: True
        with \(x 0 x y\) id have \(z: z=\) uminus-2 (mult-1-pos ?mxt ? yt) by simp
        from \(x 0\) y0 mult-1-pos[OF mx y] uminus-2[of mult-1-pos ?mxt ? \(y t\) ]
        show ?thesis unfolding \(z\) by auto
    next
        case False
        with \(y 0\) have \(y 0: ? y<0\) by \(\operatorname{simp}\)
        with \(x 0 x y\) have \(z: z=\) mult-1-pos ?mxt ? myt by auto
        with \(x 0\) y0 \(x\) y mult-1-pos[OF mx my]
        show ?thesis unfolding \(z\) by auto
    qed
    qed
qed
```

lemma mult-rat-1: fixes $x$ assumes $y$ : invariant-1 $y$
defines $z: z \equiv$ mult-rat-1 $x y$
shows invariant-2 $z \wedge($ real-of-2 $z=o f-r a t \quad x *$ real-of-1 $y)$
proof (cases y)
case yt: (fields p2 l2 r2)
let $? y t=(p 2, l 2, r 2)$
let $? x=$ real-of-rat $x$
let $? y=$ real-of- 1 ? $y t$
let $?$ myt $=$ mult-rat-1-pos $(-x)$ ?yt
note $y=y[$ unfolded $y t]$
note $z=z[$ unfolded $y t]$

```
    show ?thesis
    proof(cases x 0::rat rule:linorder-cases)
    case x: greater
    with z have z:z=mult-rat-1-pos x ?yt by simp
    from mult-rat-1-pos[OF x y]
    show ?thesis unfolding yt z by auto
next
    case less
    then have }x:-x>0\mathrm{ by auto
    hence z: z= uminus-2 ?myt unfolding z by simp
    from mult-rat-1-pos[OF x y] have rc: invariant-2 ?myt
        and rr:real-of-2 ?myt = - ?x * ?y by (auto simp: hom-distribs)
    from uminus-2[OF rc] rr show ?thesis unfolding z[symmetric] unfolding
yt[symmetric]
            by simp
    qed (auto simp:z)
qed
end
declare mult-1.simps[simp del]
declare mult-rat-1.simps[simp del]
```


### 11.2.12 Root

```
definition ipoly-root-delta \(::\) int poly \(\Rightarrow\) real where
```

```
    ipoly-root-delta \(p=\operatorname{Min}(\) insert \(1\{\) abs \((x-y) \mid x y\).ipoly \(p x=0 \wedge\) ipoly \(p y\)
```

    ipoly-root-delta \(p=\operatorname{Min}(\) insert \(1\{\) abs \((x-y) \mid x y\).ipoly \(p x=0 \wedge\) ipoly \(p y\)
    = 0^x\not= y})/4
lemma ipoly-root-delta: assumes p}\not=
shows ipoly-root-delta p>0
2 \leq card (Collect (root-cond ( }p,l,r)))\Longrightarrow\mathrm{ ipoly-root-delta p seal-of-rat (r
-l) / 4
proof -
let ?z=0 :: real
let ?R = {x. ipoly p x=? ?z}
let ?set ={abs (x-y)| x y. ipoly p x =?z ^ ipoly p y=0^x\not=y}
define S where S= insert 1 ?set
from finite-ipoly-roots[OF assms] have finR: finite ?R and fin: finite (?R }\times?R
by auto
have finite ?set
by (rule finite-subset[OF - finite-imageI[OF fin, of \lambda (x,y). abs (x-y)]], force)
hence fin: finite S and ne:S\not={} and pos: \ x. x\inS\Longrightarrowx>0 unfolding
S-def by auto
have delta: ipoly-root-delta p=Min S / \& unfolding ipoly-root-delta-def S-def
have pos: Min S>0 using fin ne pos by auto
show ipoly-root-delta p>0 unfolding delta using pos by auto
let ?S = Collect (root-cond ( }p,l,r)
assume 2 \leq card?S

```
hence 2: Suc (Suc 0) \(\leq\) card ? S by simp
from 2[unfolded card-le-Suc-iff [of - ?S]] obtain \(x T\) where
\(S T: ? S=\) insert \(x T\) and \(x T: x \notin T\) and \(1:\) Suc \(0 \leq\) card \(T\) by auto from 1 [unfolded card-le-Suc-iff \([o f-T]\) ] obtain \(y\) where \(y T: y \in T\) by auto
from \(S T\) xT yT have \(x: x \in ? S\) and \(y: y \in ? S\) and \(x y: x \neq y\) by auto
hence abs \((x-y) \in S\) unfolding \(S\)-def root-cond-def[abs-def] by auto
with fin have Min \(S \leq a b s(x-y)\) by auto
with pos have le: Min \(S / 2 \leq a b s(x-y) / 2\) by auto
from \(x y\) have \(a b s(x-y) \leq o f\)-rat \(r-o f\)-rat \(l\) unfolding root-cond-def[abs-def]
by auto
also have \(\ldots=\) of-rat \((r-l)\) by (auto simp: of-rat-diff)
finally have \(a b s(x-y) / 2 \leq o f-r a t(r-l) / 2\) by auto
with le show ipoly-root-delta \(p \leq\) real-of-rat \((r-l) / 4\) unfolding delta by auto
qed
lemma sgn-less-eq-1-rat: fixes \(a b::\) rat
shows \(\operatorname{sgn} a=1 \Longrightarrow a \leq b \Longrightarrow \operatorname{sgn} b=1\)
by (metis (no-types, opaque-lifting) not-less one-neq-neg-one one-neq-zero or-der-trans sgn-rat-def)
lemma sgn-less-eq-1-real: fixes \(a b\) :: real
shows sgn \(a=1 \Longrightarrow a \leq b \Longrightarrow \operatorname{sgn} b=1\)
by (metis (no-types, opaque-lifting) not-less one-neq-neg-one one-neq-zero or-der-trans sgn-real-def)
definition compare-1-rat \(::\) real-alg-1 \(\Rightarrow\) rat \(\Rightarrow\) order where
compare-1-rat rai \(=\) (let \(p=\) poly-real-alg-1 rai in
if degree \(p=1\) then let \(x=\) Rat.Fract \((-\operatorname{coeff} p 0)(\operatorname{coeff} p 1)\)
in \((\lambda y\). compare \(y x)\)
else ( \(\lambda\) y. compare-rat-1 y rai) \()\)
lemma compare-real-of-rat: compare (real-of-rat \(x)(\) of-rat \(y)=\) compare \(x y\) unfolding compare-rat-def compare-real-def comparator-of-def of-rat-less by auto
lemma compare-1-rat: assumes rc: invariant-1 y shows compare-1-rat y \(x=\) compare (of-rat \(x\) ) (real-of-1 \(y\) )
proof (cases degree (poly-real-alg-1 y) Suc 0 rule: linorder-cases)
case less with invariant-1-degree- \(0\left[\begin{array}{ll}\text { OF rc] show ?thesis by auto }\end{array}\right.\)
next
case deg: greater
with \(r c\) have \(r c\) : invariant-1-2 \(y\) by auto
from deg compare-rat-1[OF rc, of \(x\) ]
show ?thesis unfolding compare-1-rat-def by auto

\section*{next}
case deg: equal
obtain \(p l r\) where \(y: y=(p, l, r)\) by (cases \(y\) )
note \(r c=\) invariant \(-1 D[\) OF rc[unfolded \(y]]\)
from deg have \(p\) : degree \(p=\) Suc 0
and id: compare-1-rat \(y x=\) compare \(x(\) Rat.Fract \((-\operatorname{coeff} p 0)(\) coeff \(p 1))\) unfolding compare-1-rat-def by (auto simp: Let-def y)
from \(r c(1)[\) unfolded split] have ipoly \(p\) (real-of-1 \(y\) ) \(=0\)
unfolding \(y\) by auto
with degree-1-ipoly[OF p, of real-of-1 y]
have id': real-of-1 \(y=\) real-of-rat (Rat.Fract \((-\operatorname{coeff} p 0)(\) coeff \(p 1))\) by simp show ?thesis unfolding id id' compare-real-of-rat ..
qed

\section*{context}
fixes \(n:: n a t\)
begin
private definition initial-lower-bound :: rat \(\Rightarrow\) rat where
initial-lower-bound \(l=(\) if \(l \leq 1\) then l else of-int \((\) root-rat-floor \(n l))\)
private definition initial-upper-bound \(::\) rat \(\Rightarrow\) rat where
initial-upper-bound \(r=(\) of-int (root-rat-ceiling \(n r)\) )

\section*{context}
fixes \(c m p x::\) rat \(\Rightarrow\) order
begin
fun tighten-bound-root ::
rat \(\times r a t \Rightarrow r a t \times r a t\) where
tighten-bound-root \(\left(l^{\prime}, r^{\prime}\right)=(\) let
\(m^{\prime}=\left(l^{\prime}+r^{\prime}\right) / 2 ;\)
\(m=m^{\prime}{ }^{\wedge} n\)
in case cmpx \(m\) of
\[
E q \Rightarrow\left(m^{\prime}, m^{\prime}\right)
\]
| \(L t \Rightarrow\left(m^{\prime}, r^{\prime}\right)\)
\(\left.\mid G t \Rightarrow\left(l^{\prime}, m^{\prime}\right)\right)\)
lemma tighten-bound-root: assumes sgn: sgn il \(=1\) real-of-1 \(x \geq 0\) and
il: real-of-rat il \(\leq\) root \(n\) (real-of-1 \(x\) ) and
\(i r\) : root \(n\) (real-of-1 \(x\) ) real-of-rat ir and
rai: invariant-1 \(x\) and
cmpx: cmpx \(=\) compare-1-rat \(x\) and
\(n: n \neq 0\)
shows converges-to ( \(\lambda i\). (tighten-bound-root ~ \(i)(i l, i r))\)
(root \(n\) (real-of- \(1 x\) )) (is converges-to ?f ? \(x\) )
unfolding converges-to-def
proof (intro conjI impI allI)
\{
fix \(x\) :: real
have \(x \geq 0 \Longrightarrow(\) root \(n x){ }^{\wedge} n=x\) using \(n\) by \(\operatorname{simp}\)
\(\}\) note root-exp-cancel \(=\) this \{
fix \(x\) :: real
have \(x \geq 0 \Longrightarrow\) root \(n\left(x^{\wedge} n\right)=x\) using \(n\)
using real-root-pos-unique by blast
```

} note root-exp-cancel' = this
from il ir have real-of-rat il \leqof-rat ir by auto
hence ir-il: il \leqir by (auto simp: of-rat-less-eq)
from n have n':n>0 by auto
{
fix }
have in-interval (?f i) ?x ^ sub-interval (?f i) (il,ir) ^(i\not=0\longrightarrow sub-interval
(?f i) (?f (i-1)))
^snd (?f i) - fst (?f i)\leq(ir - il) / 2`i
proof (induct i)
case 0
show ?case using il ir by auto
next
case (Suc i)
obtain l' r' where id: (tighten-bound-root ^^ i) (il,ir) = (l',r')
by (cases (tighten-bound-root ~ i) (il, ir), auto)
let ?m' = ( l' + r') / 2
let ?m}=? ? m'^
define m}\mathrm{ where m=?m
note IH = Suc[unfolded id split snd-conv fst-conv]
from IH have sub-interval ( }\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})(il,ir) by aut
hence ill':il\leq l' r'}\leq\mathrm{ ir by auto
with sgn have l'0: l'>0 using sgn-1-pos sgn-less-eq-1-rat by blast
from IH have lr'x: in-interval ( }\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})?x\mathrm{ by auto
hence lr'':real-of-rat l'
hence lr': l'
with l'0 have r'0: r'>0 by auto
note compare = compare-1-rat[OF rai, of ?m, folded cmpx]
from IH have *: r' - l'}\leq(ir - il)/ / ~ ^i by aut
have r}\mp@subsup{r}{}{\prime}-(\mp@subsup{l}{}{\prime}+\mp@subsup{r}{}{\prime})/2=(\mp@subsup{r}{}{\prime}-\mp@subsup{l}{}{\prime})/2 by (simp add: field-simps
also have ... \leq (ir - il) / 2^i / 2 using *
by (rule divide-right-mono, auto)
finally have size: r' - (l'}+\mp@subsup{r}{}{\prime})/2\leq(ir-il)/(2*2^i) by sim
also have }\mp@subsup{r}{}{\prime}-(\mp@subsup{l}{}{\prime}+\mp@subsup{r}{}{\prime})/2=(\mp@subsup{l}{}{\prime}+\mp@subsup{r}{}{\prime})/2-\mp@subsup{l}{}{\prime}\mathrm{ by auto
finally have size': (l' + r')/2 - l'\leq (ir - il)/(2* 2 ^i) by simp
have root n (real-of-rat ?m) = root n ((real-of-rat?m') ^ n) by (simp add:
hom-distribs)
also have ... = real-of-rat ? m'
by (rule root-exp-cancel', insert l'0 lr', auto)
finally have root: root n (of-rat ?m) =of-rat ? m' .
show ?case
proof (cases cmpx ?m)
case Eq
from compare[unfolded Eq] have real-of-1 x =of-rat?m
unfolding compare-real-def comparator-of-def by (auto split: if-splits)
from arg-cong[OF this, of root n] have ?x = root n (of-rat ?m).
also have ... = root n (real-of-rat ?m') ^n
using n real-root-power by (auto simp: hom-distribs)
also have ... = of-rat ?m'

```
```

            by (rule root-exp-cancel, insert IH sgn(2) l'0 r'0, auto)
    finally have }x:?x=of-rat?m' .
    show ?thesis using x id Eq lr' ill' ir-il by (auto simp: Let-def)
    next
    case Lt
    from compare[unfolded Lt] have lt:of-rat ?m \leq real-of-1 x
        unfolding compare-real-def comparator-of-def by (auto split: if-splits)
    have id'\prime: ?f (Suc i)=(?m
        using Lt id by (auto simp add: Let-def)
    from real-root-le-mono[OF n' lt]
    have of-rat?m'\leq?x unfolding root by simp
    with lr'x lr'\prime}\mathrm{ have ineq': real-of-rat l' + real-of-rat r' }\leq?x*2 by (aut
    simp: hom-distribs)
show ?thesis unfolding id"
by (auto simp: Let-def hom-distribs, insert size ineq' lr' ill' lr'x ir-il, auto)
next
case Gt
from compare[unfolded Gt] have lt:of-rat ?m \geq real-of-1 x
unfolding compare-real-def comparator-of-def by (auto split: if-splits)
have id'\prime: ?f (Suc i)=(l',?m}\mp@subsup{m}{}{\prime})\mathrm{ ?f (Suc i - 1) = (l',r')
using Gt id by (auto simp add: Let-def)
from real-root-le-mono[OF n'lt]
have ?x \leq of-rat ?m' unfolding root by simp
with lr'xlr'\prime}\mathrm{ have ineq': ?x * 2 }\leq\mathrm{ real-of-rat l' + real-of-rat r' by (auto
simp: hom-distribs)
show ?thesis unfolding id"
by (auto simp: Let-def hom-distribs, insert size' ineq' lr' ill' lr'x ir-il, auto)
qed
qed
} note main = this
fix }
from main[of i] show in-interval (?f i) ?x by auto
from main[of Suc i] show sub-interval (?f (Suc i)) (?f i) by auto
fix eps :: real
assume eps: 0 < eps
define c where c=eps / (max (real-of-rat (ir - il)) 1)
have c0:c>0 using eps unfolding c-def by auto
from exp-tends-to-zero[OF - this, of 1/2] obtain i where c:(1/2)^i}\leqc\mathrm{ by
auto
obtain l' }\mp@subsup{l}{}{\prime}\mathrm{ where fi: ?f }i=(\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by force
from main[of i, unfolded fi] have le: r' - l' }\leq(ir - il)/ 2 ^ i by aut
have iril: real-of-rat (ir - il)\geq0 using ir-il by (auto simp:of-rat-less-eq)
show \existsn la ra. ?f n = (la, ra) ^ real-of-rat ra - real-of-rat la \leqeps
proof (intro conjI exI, rule fi)
have real-of-rat r' - of-rat l' = real-of-rat ( }\mp@subsup{r}{}{\prime}-\mp@subsup{l}{}{\prime})\mathrm{ by (auto simp: hom-distribs)
also have ... \leqreal-of-rat ((ir - il)/ 2` ` i) using le unfolding of-rat-less-eq
also have ... =(real-of-rat (ir - il))* ((1/2) ^i) by (simp add: field-simps
hom-distribs)

```
```

    also have ... \leq(real-of-rat (ir - il))*c
    by (rule mult-left-mono[OF c iril])
    also have ... \leqeps
    proof (cases real-of-rat (ir - il)\leq1)
    case True
    hence c = eps unfolding c-def by (auto simp: hom-distribs)
    thus ?thesis using eps True by auto
    next
        case False
        hence max (real-of-rat (ir - il)) 1 = real-of-rat (ir - il) real-of-rat (ir - il)
    \not=0
by (auto simp: hom-distribs)
hence (real-of-rat (ir -il))*c=eps unfolding c-def by auto
thus ?thesis by simp
qed
finally show real-of-rat r' - of-rat l'
qed
qed
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)))
(\lambdax. x)
(initial-lower-bound l, initial-upper-bound r)
(poly-nth-root n p)))
fun root-1 :: real-alg-1 }=>\mathrm{ real-alg-2 where
root-1 (p,l,r) = (
if n=0\veer=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\not=0
begin
lemma initial-upper-bound: assumes x: x>0 and xr:x\leq of-rat r
shows sgn(initial-upper-bound r)=1 root n x \leqof-rat (initial-upper-bound r)
proof -
have n: n>0 using n by auto
note d = initial-upper-bound-def
let ?r = initial-upper-bound r
from x xr have r0:r>0 by (meson not-less of-rat-le-0-iff order-trans)
hence of-rat r>(0 :: real) by auto
hence root n (of-rat r)>0 using n by simp
hence 1\leq ceiling (root n (of-rat r)) by auto
hence (1 :: rat) \leq of-int (ceiling (root n (of-rat r))) by linarith

```
```

    also have ... = ?r unfolding d by simp
    finally show sgn ?r = 1 unfolding sgn-rat-def by auto
    have root n x < root n (of-rat r)
    unfolding real-root-le-iff[OF n] by (rule xr)
    also have ... \leqof-rat ?r unfolding d by simp
    finally show root n x s of-rat ?r .
    qed
lemma initial-lower-bound: assumes l: l>0 and lx: of-rat l\leqx
shows sgn (initial-lower-bound l)=1 of-rat (initial-lower-bound l)\leq root n x
proof -
have n: n>0 using n by auto
note d}=\mathrm{ initial-lower-bound-def
let ?l = initial-lower-bound l
from l lx have x0:x>0 by (meson not-less of-rat-le-0-iff order-trans)
have sgn ?l = 1 ^ of-rat ?l \leq root n x
proof (cases l\leq1)
case True
hence ll:?l=l and l0:of-rat l\geq(0 :: real) and l1:of-rat l\leq(1 :: real)
using}l\mathrm{ unfolding True d by auto
have sgn: sgn ?l = 1 using l unfolding ll by auto
have of-rat ?l = of-rat l unfolding ll by simp
also have of-rat l \leq root n (of-rat l) using real-root-increasing[OF - l0 l1, of
1n] n
by (cases n = 1,auto)
also have ... \leqroot n x using lx unfolding real-root-le-iff[OF n] .
finally show ?thesis using sgn by auto
next
case False
hence l:(1 :: real) \leqof-rat l and ll: ?l = of-int (floor (root n (of-rat l)))
unfolding d by auto
hence root n 1 \leq root n (of-rat l)
unfolding real-root-le-iff[OF n] by auto
hence 1\leq root n (of-rat l) using n by auto
from floor-mono[OF this] have 1\leq?l
using one-le-floor unfolding ll by fastforce
hence sgn: sgn ?l = 1 by simp
have of-rat ?l }\leq\mathrm{ root n (of-rat l) unfolding ll by simp
also have ... \leqroot n x using lx unfolding real-root-le-iff[OF n] .
finally have of-rat ?l \leq root n x .
with sgn show ?thesis by auto
qed
thus sgn ?l = 1 of-rat ?l \leq root n x by auto
qed
lemma root-pos-1:
assumes x: invariant-1 x and pos: rai-ub x>0
defines }y:y\equiv\mathrm{ root-pos-1 x
shows invariant-2 y ^ real-of-2 y = root n (real-of-1 x)

```
```

proof (cases x)
case (fields p l r)
let ?l = initial-lower-bound l
let ?r = initial-upper-bound r
from x fields have rai: invariant-1 (p,l,r) by auto
note * = invariant-1D[OF this]
let ?}x=\mathrm{ the-unique-root ( }p,l,r
from pos[unfolded fields] *
have sgnl: sgn l=1 by auto
from sgnl have l0:l>0 by (unfold sgn-1-pos)
hence ll0: real-of-rat l>0 by auto
from * have lx: of-rat l}\leq\mathrm{ ? }x\mathrm{ by auto
with llO have x0:? }x>0\mathrm{ by linarith
note il = initial-lower-bound[OF l0 lx]
from * have ?}x\leqof-rat r by aut
note iu = initial-upper-bound[OF x0 this]
let ?p = poly-nth-root n p
from x0 have id: root n ? x^ n =? x using n real-root-pow-pos by blast
have rc: root-cond (?p,?l, ?r) (root n ?x)
using il iu * by (intro root-condI, auto simp: ipoly-nth-root id)
hence root: ipoly ?p (root n (real-of-1 x)) =0
unfolding root-cond-def fields by auto
from * have p\not=0 by auto
hence }\mp@subsup{p}{}{\prime}:?p\not=0\mathrm{ using poly-nth-root- O[of n p] n by auto
have tbr:0 \leq real-of-1 x
real-of-rat (initial-lower-bound l) \leq root n (real-of-1 x)
root n(real-of-1 x) \leq real-of-rat (initial-upper-bound r)
using x0 il(2) iu(2) fields by auto
from select-correct-factor-int-poly[OF tighten-bound-root[OF il(1)[folded fields]
tbr x refl n] refl root p]
show ?thesis by (simp add: y fields)
qed
end
lemma root-1: assumes x: invariant-1 x
defines y: y \equivroot-1 }
shows invariant-2 y ^(real-of-2 y = root n (real-of-1 x))
proof (cases n=0\vee rai-ub x=0)
case True
with x have n=0\vee real-of-1 x=0 by (cases x, auto)
then have root n(real-of-1 x)=0 by auto
then show ?thesis unfolding y root-1.simps
using x by (cases x, auto)
next
case False with x have n: n\not=0 and x0: real-of-1 x\not=0 by (simp, cases }x\mathrm{ ,
auto)
note rt = root-pos-1
show ?thesis

```
```

proof (cases rai-ub x 0::rat rule:linorder-cases)
case greater
with rt[OF n x this] n show ?thesis by (unfold y, cases x, simp)
next
case less
let ?um = uminus-1
let ?rt = root-pos-1
from n less y x0 have y: y= uminus-2 (?rt (?um x)) by (cases x, auto)
from uminus-1[OF x] have umx: invariant-1 (?um x) and umx2: real-of-1
(?um x) = - real-of-1 }x\mathrm{ by auto
with x less have 0<rai-ub (uminus-1 x)
by (cases x, auto simp: uminus-1.simps Let-def)
from rt[OF n umx this] umx2 have rumx: invariant-2 (?rt (?um x))
and rumx2: real-of-2 (?rt (?um x) ) = root n (- real-of-1 x)
by auto
from uminus-2[OF rumx] rumx2 y real-root-minus show ?thesis by auto
next
case equal with x0 x show ?thesis by (cases x, auto)
qed
qed
end
declare root-1.simps[simp del]

```

\subsection*{11.2.13 Embedding of Rational Numbers}
definition of-rat-1 :: rat \(\Rightarrow\) real-alg-1 where
    of-rat-1 \(x \equiv(\) 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 \(x\)
    unfolding of-rat-1-def
by (atomize(full), intro invariant-1-realI unique-rootI poly-condI, auto )
```

fun info-2 :: real-alg-2 $\Rightarrow$ rat + int poly $\times$ nat where
info-2 $($ Rational $x)=$ Inl $x$
|info-2 $($ Irrational $n(p, l, r))=\operatorname{Inr}(p, n)$

```
lemma info-2-card: assumes rc: invariant-2 \(x\)
    shows info-2 \(x=\operatorname{Inr}(p, n) \Longrightarrow\) poly-cond \(p \wedge\) ipoly \(p(\) real-of-2 \(x)=0 \wedge\) degree
\(p \geq\) 2
    \(\wedge\) card (roots-below \(p(\) real-of-2 \(x))=n\)
        info-2 \(x=\) Inl \(y \Longrightarrow\) real-of-2 \(x=\) of-rat \(y\)
proof (atomize(full), goal-cases)
    case 1
    show ?case
    proof (cases \(x\) )
        case (Irrational m rai)
        then obtain \(q l r\) where \(x: x=\) Irrational \(m(q, l, r)\) by (cases rai, auto)
```

    show ?thesis
    proof (cases q=p^m=n)
        case False
        thus ?thesis using x by auto
    next
        case True
        with }x\mathrm{ have }x\mathrm{ : x= Irrational n ( }p,l,r)\mathrm{ by auto
        from rc[unfolded x, simplified] have inv: invariant-1-2 ( }p,l,r)\mathrm{ and
            n: card (roots-below p (real-of-2 x)) = n and 1: degree p}=
            by (auto simp: x)
        from inv have degree p\not=0 unfolding irreducible-def by auto
        with 1 have degree p \geq2 by linarith
        thus ?thesis unfolding n using inv x by (auto elim!: invariant-1E)
    qed
    qed auto
    qed
lemma real-of-2-Irrational: invariant-2 (Irrational $n$ rai) $\Longrightarrow$ real-of-2 (Irrational $n$ rai) $\neq$ of-rat $x$

```

\section*{proof}
```

assume invariant-2 (Irrational $n$ rai) and rat: real-of-2 (Irrational $n$ rai) $=$ real-of-rat $x$
hence real-of-1 rai $\in \mathbb{Q}$ invariant-1-2 rai by auto
from invariant-1-2-of-rat[OF this(2)] rat show False by auto
qed
lemma info-2: assumes
ix: invariant-2 $x$ and iy: invariant-2 $y$
shows info-2 $x=$ info-2 $y \longleftrightarrow$ real-of-2 $x=$ real-of-2 $y$
proof (cases $x$ )
case $x$ : (Irrational n1 rai1)
note $i x=i x[$ unfolded $x]$
show ?thesis
proof (cases y)
case (Rational y)
with real-of-2-Irrational[OF ix, of $y$ ] show ?thesis unfolding $x$ by (cases rai1, auto)
next
case y: (Irrational n2 rai2)
obtain p1 l1 r1 where rai1: rai1 $=(p 1, l 1, r 1)$ by (cases rai1 $)$
obtain $p 2$ l2 r2 where rai2: rai2 $=(p 2, l 2, r 2)$ by $($ cases rai2 $)$
let ? $r x=$ the-unique-root $(p 1, l 1, r 1)$
let ? $r y=$ the-unique-root $(p 2,12, r 2)$
have id: $($ info-2 $x=$ info-2 $y)=(p 1=p 2 \wedge n 1=n 2)$
(real-of-2 $x=$ real-of-2 $y)=(? r x=$ ? $r y)$
unfolding $x$ y rai1 rai2 by auto
from ix[unfolded $x$ rai1]
have ix: invariant-1 $(p 1, l 1, r 1)$ and deg1: degree $p 1>1$ and $n 1: n 1=$ card (roots-below p1 ?rx) by auto

```
note \(I x=\) invariant \(-1 D[\) OF ix]
from \(\operatorname{deg} 1\) have \(p 1-0: p 1 \neq 0\) by auto
from iy[unfolded \(y\) raiQ]
have iy: invariant-1 ( \(p 2,12\), r2) and degree \(p_{2}>1\) and n2: n2 \(=\) card (roots-below p2 ?ry) by auto
note \(I y=\) invariant- \(1 D[\) OF iy \(]\)
show ?thesis unfolding id
proof
assume eq: ? \(r x=\) ? \(r y\)
from \(I x\)
have algx: p1 represents ? \(r x \wedge\) irreducible \(p 1 \wedge\) lead-coeff p1 \(>0\) unfolding represents-def by auto
from iy
have algy: p2 represents ? \(r x \wedge\) irreducible p2 \(\wedge\) lead-coeff p2 \(>0\) unfolding represents-def eq by (auto elim!: invariant-1E)
from algx have algebraic ?rx unfolding algebraic-altdef-ipoly by auto
note unique \(=\) algebraic-imp-represents-unique[OF this]
with algx algy have \(i d: p 2=p 1\) by auto
from eq id \(n 1 n 2\) show \(p 1=p 2 \wedge n 1=n 2\) by auto
next
assume \(p 1=p 2 \wedge n 1=n 2\)
hence \(i d\) : \(p 1=p 2 n 1=n 2\) by auto
hence card: card (roots-below p1 ?rx) = card (roots-below p1 ?ry) unfolding n1 n2 by auto
show ? \(r x=\) ? \(r y\)
proof (cases ?rx ? ry rule: linorder-cases)
case less
have roots-below p1 ? \(r x=\) roots-below p1 ? ry
proof (intro card-subset-eq finite-subset[OF - ipoly-roots-finite] card)
from less show roots-below p1 ? rx \(\subseteq\) roots-below p1 ?ry by auto
qed (insert p1-0, auto)
then show ?thesis using id less unique-rootD(3)[OF Iy(4)] by (auto simp: less-eq-real-def)

\section*{next}
case equal
then show ?thesis by (simp add: id)
next
case greater
have roots-below p1 ?ry \(=\) roots-below \(p 1\) ? \(r x\)
proof (intro card-subset-eq card[symmetric] finite-subset[OF - ipoly-roots-finite[OF p1-0]])
from greater show roots-below p1 ?ry \(\subseteq\) roots-below p1 ?rx by auto qed auto
hence roots-below p2 ?ry = roots-below p2 ?rx unfolding id by auto
thus ?thesis using id greater unique-rootD(3)[OF Ix(4)] by (auto simp: less-eq-real-def)
qed
qed
qed
```

next
case x:(Rational x)
show ?thesis
proof (cases y)
case (Rational y)
thus ?thesis using x by auto
next
case y: (Irrational n rai)
with real-of-2-Irrational[OF iy[unfolded y], of x] show ?thesis unfolding }x\mathrm{ by
(cases rai, auto)
qed
qed
lemma info-2-unique: invariant-2 }x\Longrightarrow\mathrm{ invariant-2 }y
real-of-2 x = real-of-2 }y\Longrightarrow\mathrm{ info-2 }x=\mathrm{ info-2 }
using info-2 by blast
lemma info-2-inj: invariant-2 }x\Longrightarrow\mathrm{ invariant-2 }y>\mathrm{ info-2 }x=\mathrm{ info-2 }y
real-of-2 x = real-of-2 y
using info-2 by blast
context
fixes cr1 cr2 :: rat => rat => nat
begin
partial-function (tailrec) compare-1 :: int poly }=>\mathrm{ int poly }=>\mathrm{ rat }=>\mathrm{ rat }=>\mathrm{ rat }
rat }=>\mathrm{ rat }=>\mathrm{ rat }=>\mathrm{ 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 ( }p2,12,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 -
let ?r = real-of-rat
{
fix dx y
assume d:d}=(r1-l1)+(r2 - l2) and xy:x= the-unique-root ( p1,l1,r1
y= the-unique-root (p2,l2,r2)
define delta where delta =abs (x-y)/4

```
have delta: delta \(>0\) and diff: \(x \neq y\) unfolding delta-def using diff \(x y\) by auto
let \({ }^{2} \mathrm{rel}^{\prime}=\{(x, y) .0 \leq y \wedge\) delta-gt delta \(x y\}\)
let ? rel \(=\) inv-image ? \(\mathrm{rel}^{\prime}\) ? \(r\)
have \(S N\) : \(S N\) ? rel by (rule \(S N\)-inv-image[OF delta-gt-SN[OF delta]])
from d ur1 ur2
have ?thesis unfolding \(x y[\) symmetric \(]\) using \(x y\) sr
proof (induct d arbitrary: l1 r1 l2 r2 sr1 sr2 rule: SN-induct[OF SN])
case (1 d l1 r1 l2 r2)
note \(I H=1(1)\)
note \(d=1\) (2)
note \(u r=1(3-4)\)
note \(x y=1(5-6)\)
note \(s r=1(7-8)\)
note \(\operatorname{simps}=\) compare-1.simps[of p1 p2 l1 r1 sr1 l2 r2 sr2]
note \(\operatorname{urx}=\) unique-root \(D[O F \operatorname{ur}(1)\), folded \(x y]\)
note \(u r y=\) unique-root \(D[O F \operatorname{ur}(2)\), folded \(x y]\)
show ?case (is ?l \(=-\) )
proof (cases r1<l2)
case True
hence \(l: ? l=L t\) and \(l t\) : ? \(r ~ r 1<? r l 2\) unfolding simps of-rat-less by auto show ?thesis unfolding \(l\) using \(l t\) True urx (2) ury (1)
by (auto simp: compare-real-def comparator-of-def)
next
case False note \(l e=\) this
show ?thesis
proof (cases r2 < l1)
case True
with le have \(l: ? l=G t\) and \(l t:\) ?r r2 < ?r \(l 1\) unfolding simps of-rat-less by auto
show ?thesis unfolding \(l\) using \(l t\) True ury(2) urx (1)
by (auto simp: compare-real-def comparator-of-def)
next
case False
obtain \(l 1^{\prime} r 1^{\prime}\) sr1' where tb1: tighten-poly-bounds p1 l1 r1 sr1 = \(\left(l 1^{\prime}, r 1^{\prime}, s r 1{ }^{\prime}\right)\)
by (cases rule: prod-cases3, auto)
obtain \(12^{\prime} r 2^{\prime}\) sr2' where tb2: tighten-poly-bounds p2 12 r2 sr2 \(=\) ( 2 \(^{\prime}\), \(r\) 2 \(^{\prime}\),sr2')
by (cases rule: prod-cases3, auto)
from False le tb1 tb2 have \(l: ? l=\) compare-1 p1 p2 \(l 1^{\prime} r 1^{\prime} s r 1^{\prime} l 2^{\prime} r 2^{\prime}\) sr2' unfolding simps by auto
from tighten-poly-bounds[OF tb1 ur(1) pc(1) sr(1)]
have rc1: root-cond \(\left(p 1, l 1^{\prime}, r 1^{\prime}\right)\) (the-unique-root \(\left.(p 1, l 1, r 1)\right)\)
and bnd1: \(l 1 \leq l 1^{\prime} l 1^{\prime} \leq r 1^{\prime} r 1^{\prime} \leq r 1\) and \(d 1: r 1^{\prime}-l 1^{\prime}=(r 1-l 1) /\)
2
and sr1: sr1' \(=\operatorname{sgn}(\) ipoly p1r1') by auto
from \(p c\) have \(p 1 \neq 0 p 2 \neq 0\) by auto
from unique-root-sub-interval[OF ur(1) rc1 bnd1 (1,3)] xy ur this
have ur1: unique-root \(\left(p 1, l 1^{\prime}, r 1^{\prime}\right)\) and \(x: x=\) the-unique-root ( \(p 1, l 1^{\prime}\), \(r 1^{\prime}\) ) by (auto intro!: the-unique-root-eqI)
from tighten-poly-bounds[OF tb2 ur(2) pc(2) sr(2)]
have rc2: root-cond ( \(p\) 2, l2', r2') (the-unique-root ( \(p 2\), l2, r2))
and \(b n d 2: l 2 \leq l 2^{\prime} l 2^{\prime} \leq r 2^{\prime} r 2^{\prime} \leq r 2\) and \(d 2: r 2^{\prime}-l 2^{\prime}=(r 2-l 2) /\)
2
and \(\operatorname{sr2}\) : \(\operatorname{sr2}^{\prime}=\operatorname{sgn}(\) ipoly p2 r2') by auto
from unique-root-sub-interval[OF ur(2) rc2 bnd2 (1,3)] xy ur pc
have ur2: unique-root ( \(p 2,12^{\prime}, r 2^{\prime}\) ) and \(y: y=\) the-unique-root ( \(p 2, l 2^{\prime}\),
r2') by auto
define \(d^{\prime}\) where \(d^{\prime}=d / 2\)
have \(d^{\prime}: d^{\prime}=r 1^{\prime}-l 1^{\prime}+\left(r 2^{\prime}-l 2^{\prime}\right)\) unfolding \(d^{\prime}\)-def \(d d 1 d 2\) by (simp add: field-simps)
have \(d^{\prime} 0: d^{\prime} \geq 0\) using bnd1 bnd2 unfolding \(d^{\prime}\) by auto
have \(d d\) : \(d-d^{\prime}=d / 2\) unfolding \(d^{\prime}\)-def by simp
have \(a b s(x-y) \leq 2 *\) ? \(r d\)
proof (rule ccontr)
assume \(\neg\) ?thesis
hence \(l t\) : 2 * ? \(r d<a b s(x-y)\) by auto
have \(r 1-l 1 \leq d r 2-l 2 \leq d\) unfolding \(d\) using bnd1 bnd2 by auto
from this[folded of-rat-less-eq[where ' \(a=\) real] \(] l t\)
have ? \(r(r 1-l 1)<a b s(x-y) / 2 \ln (r 2-l 2)<a b s(x-y) / 2\) and \(d d\) : ? \(r\) r \(1-\) ? \(r l 1 \leq\) ? \(r d\) ?r r2 - ?r \(l 2 \leq\) ? \(r d\) by (auto simp:
of-rat-diff)
from le have \(r 1 \geq 12\) by auto hence r1l2: ?r \(r 1 \geq\) ? 12 unfolding of-rat-less-eq by auto
from False have \(r 2 \geq l 1\) by auto hence \(r 2 l 1:\) ? \(r ~ r 2 \geq\) ? \(l l\) unfolding of-rat-less-eq by auto
show False
proof (cases \(x \leq y\) )
case True
from \(\operatorname{urx}(1-2) d d(1)\) have ? \(r 1 \leq x+\) ? \(r d\) by auto
with r1l2 have ? \(12 \leq x+\) ? \(r d\) by auto
with True lt ury(2) dd(2) show False by auto
next
case False
from \(\operatorname{ury}(1-2) d d(2)\) have ? \(r\) r2 \(\leq y+\) ? \(r d\) by auto
with \(r 2 l 1\) have ? \(r l 1 \leq y+? r d\) by auto
with False lt urx(2) \(d d(1)\) show False by auto
qed
qed
hence \(d d^{\prime}\) : delta-gt delta (?r d) (?r d')
unfolding delta-gt-def delta-def using \(d d\) by (auto simp: hom-distribs) show ?thesis unfolding \(l\)
by (rule IH[OF - d'ur1 ur2 \(x\) y sr1 sr2], insert \(d^{\prime} 0 d d^{\prime}\), auto)
qed
qed
qed
```

}
thus ?thesis by auto
qed
end

```
fun real-alg-1 :: real-alg-2 \(\Rightarrow\) real-alg-1 where
    real-alg-1 \((\) Rational \(r)=o f-r a t-1 r\)
\(\mid\) real-alg-1 (Irrational \(n\) rai \()=\) rai
lemma real-alg-1: real-of-1 (real-alg-1 \(x\) ) real-of-2 \(x\)
    by (cases \(x\), auto simp: of-rat-1)
definition root-2 :: nat \(\Rightarrow\) real-alg-2 \(\Rightarrow\) real-alg-2 where
    root-2 \(n x=\) root-1 \(n\) (real-alg-1 \(x\) )
lemma root-2: assumes invariant-2 \(x\)
    shows real-of-2 (root-2 \(n x)=\) root \(n(\) real-of-2 \(x)\)
    invariant-2 (root-2 \(n x\) )
proof (atomize(full), cases \(x\), goal-cases)
    case (1 y)
    from of-rat-1[of y] root-1[of of-rat-1 y n] assms 1 real-alg-2
    show ? case by (simp add: root-2-def)
next
    case (2 i rai)
    from root-1[of rai n] assms 2 real-alg-2
    show ?case by (auto simp: root-2-def)
qed
fun add-2 :: real-alg-2 \(\Rightarrow\) real-alg-2 \(\Rightarrow\) real-alg-2 where
    add-2 (Rational \(r)(\) Rational \(q)=\) Rational \((r+q)\)
    add-2 (Rational r) (Irrational \(n x)=\) Irrational \(n(\) add-rat-1 \(r x)\)
|add-2 (Irrational \(n x)(\) Rational \(q)=\) Irrational \(n(\) add-rat-1 \(q x)\)
add-2 (Irrational \(n x)(\) Irrational \(m y)=a d d-1 x y\)
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)
    using assms add-rat-1 add-1
    by (atomize (full), (cases x; cases y), auto simp: hom-distribs)
fun mult-2 :: real-alg-2 \(\Rightarrow\) real-alg-2 \(\Rightarrow\) real-alg-2 where
    mult-2 (Rational \(r)(\) Rational \(q)=\) Rational \((r * q)\)
\(\mid\) mult-2 (Rational \(r)(\) Irrational \(n y)=\) mult-rat-1 \(r y\)
\(\mid\) mult-2 (Irrational \(n x)(\) Rational \(q)=\) mult-rat-1 \(q x\)
| mult-2 (Irrational \(n x)(\) Irrational \(m y)=\) mult-1 \(x y\)
lemma mult-2: assumes invariant-2 \(x\) invariant-2 y
```

shows real-of-2 (mult-2 x y) = real-of-2 x * real-of-2 y
invariant-2 (mult-2 x y)
using assms
by (atomize(full), (cases x; cases y; auto simp: mult-rat-1 mult-1 hom-distribs))
fun to-rat-2 :: real-alg-2 }=>\mathrm{ 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 隹 then Some (THE q. real-of-2 x =of-rat
q) else None)
proof (cases x)
case (Irrational n rai)
from real-of-2-Irrational[OF rc[unfolded this]] show ?thesis
unfolding Irrational Rats-def by auto
qed simp
fun equal-2 :: real-alg-2 }=>\mathrm{ real-alg-2 }=>\mathrm{ 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=\mathrm{ real-of-2 y)
using info-2[OF rc]
by (cases x; cases y, auto)
fun compare-2 :: real-alg-2 \# real-alg-2 }=>\mathrm{ 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\wedgen=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 rxx)
| compare-2 (Irrational - xx) (Rational r) = (invert-order (compare-rat-1 r xx )
lemma compare-2: assumes rc: invariant-2 x invariant-2 y
shows compare-2 x y compare (real-of-2 x)(real-of-2 y)
proof (cases x)
case (Rational r) note xx = this
show ?thesis
proof (cases y)
case (Rational q) note yy = this
show ?thesis unfolding xx yy by (simp add: compare-rat-def compare-real-def
comparator-of-def of-rat-less)
next
case (Irrational n yy) note yy = this
from compare-rat-1 rc

```
```

    show ?thesis unfolding xx yy by (simp add: of-rat-1)
    qed
    next
case (Irrational n xx) note xx = this
show ?thesis
proof (cases y)
case (Rational q) note yy = this
from compare-rat-1 rc
show ?thesis unfolding xx yy by simp
next
case (Irrational m yy) note yy = this
obtain plr where xxx: xx = (p,l,r) by (cases xx)
obtain q l' r' where yyy:yy =(q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime}) by (cases yy)
note rc = rc[unfolded xx xxx yy yyy]
from rc have I: invariant-1-2 ( p,l,r) invariant-1-2 ( }q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by auto
then have unique-root ( }p,l,r)\mathrm{ unique-root ( }q,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ poly-cond2 p poly-cond2 q
by auto
from compare-1[OF this - refl refl]
show ?thesis using equal-2[OF rc] unfolding xx xxx yy yyy by simp
qed
qed
fun sgn-2 :: real-alg-2 }=>\mathrm{ rat where
sgn-2 (Rational r) = sgn r
| sgn-2 (Irrational n rai) = sgn-1 rai
lemma sgn-2: invariant-2 }x\Longrightarrow\mathrm{ real-of-rat (sgn-2 x) = sgn (real-of-2 x)
using sgn-1 by (cases x, auto simp: real-of-rat-sgn)
fun floor-2 :: real-alg-2 }=>\mathrm{ int where
floor-2 (Rational r) = floor r
|floor-2 (Irrational n rai) = floor-1 rai
lemma floor-2: invariant-2 }x\Longrightarrow\mathrm{ floor-2 }x=\mathrm{ floor (real-of-2 }x\mathrm{ )
by (cases x, auto simp: floor-1)

```

\subsection*{11.2.14 Definitions and Algorithms on Type with Invariant}
```

lift-definition of-rat-3 :: rat $\Rightarrow$ real-alg-3 is of-rat-2
by (auto simp: of-rat-2)
lemma of-rat-3: real-of-3 (of-rat-3 $x$ ) $=$ of-rat $x$
by (transfer, auto simp: of-rat-2)
lift-definition root-3 :: nat $\Rightarrow$ real-alg-3 $\Rightarrow$ real-alg-3 is root-2
by (auto simp: root-2)

```
```

lemma root-3: real-of-3 (root-3 $n x)=$ root $n($ real-of-3 $x)$
by (transfer, auto simp: root-2)
lift-definition equal-3 :: real-alg-3 $\Rightarrow$ real-alg-3 $\Rightarrow$ bool is equal-2 .
lemma equal-3: equal-3 $x y=($ real-of-3 $x=$ real-of-3 $y$ )
by (transfer, auto)
lift-definition compare-3 :: real-alg-3 $\Rightarrow$ real-alg-3 $\Rightarrow$ order is compare-2 .
lemma compare-3: compare-3 $x y=($ compare (real-of-3 $x)($ real-of-3 $y))$
by (transfer, auto simp: compare-2)
lift-definition add-3 :: real-alg-3 $\Rightarrow$ real-alg-3 $\Rightarrow$ real-alg-3 is add-2
by (auto simp: add-2)
lemma add-3: real-of-3 (add-3 $x$ y) real-of-3 $x+$ real-of-3 $y$
by (transfer, auto simp: add-2)
lift-definition mult-3 :: real-alg-3 $\Rightarrow$ real-alg-3 $\Rightarrow$ real-alg-3 is mult-2
by (auto simp: mult-2)
lemma mult-3: real-of-3 (mult-3 $x$ y) $=$ real-of-3 $x *$ real-of-3 $y$
by (transfer, auto simp: mult-2)
lift-definition sgn-3 :: real-alg-3 $\Rightarrow$ rat is sgn-2 .
lemma sgn-3: real-of-rat $(\operatorname{sgn}-3 x)=\operatorname{sgn}($ real-of-3 $x)$
by (transfer, auto simp: sgn-2)
lift-definition to-rat-3 :: real-alg-3 $\Rightarrow$ rat option is to-rat-2 .
lemma to-rat-3: to-rat-3 $x=$
(if real-of-3 $x \in \mathbb{Q}$ then Some (THE q. real-of-3 $x=$ of-rat $q$ ) else None)
by (transfer, simp add: to-rat-2)
lift-definition floor-3 :: real-alg-3 $\Rightarrow$ int is floor-2 .
lemma floor-3: floor-3 $x=$ floor (real-of-3 $x$ )
by (transfer, auto simp: floor-2)

```
lift-definition info-3 :: real-alg-3 \(\Rightarrow\) rat + int poly \(\times\) nat is info-2 .
lemma info-3-fun: real-of-3 \(x=\) real-of-3 \(y \Longrightarrow\) info-3 \(x=\) info-3 \(y\)
    by (transfer, intro info-2-unique, auto)
```

lift-definition info-real-alg :: real-alg }=>\mathrm{ rat + int poly }\times\mathrm{ nat is info-3
by (metis info-3-fun)
lemma info-real-alg:
info-real-alg x = Inr ( }p,n)\Longrightarrowp\mathrm{ represents (real-of }x)\wedge\operatorname{card {y. y \leq real-of x
^ipoly p y = 0} = n ^ irreducible p
info-real-alg x = Inl q\Longrightarrow real-of x =of-rat q
proof (atomize(full), transfer, transfer, goal-cases)
case (1 x p n q)
from 1 have x: invariant-2 }x\mathrm{ by auto
note info = info-2-card[OF this]
show ?case
proof (cases x)
case irr: (Irrational m rai)
from info(1)[of p n]
show ?thesis unfolding irr by (cases rai, auto simp: poly-cond-def)
qed (insert 1 info,auto)
qed
instantiation real-alg :: plus
begin
lift-definition plus-real-alg :: real-alg => real-alg => real-alg is add-3
by (simp add: add-3)
instance ..
end
lemma plus-real-alg: (real-of }x)+(\mathrm{ real-of y) =real-of }(x+y
by (transfer, rule add-3[symmetric])
instantiation real-alg :: minus
begin
definition minus-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ real-alg where
minus-real-alg x y = x + (-y)
instance ..
end
lemma minus-real-alg: (real-of x) - (real-of y) = real-of (x-y)
unfolding minus-real-alg-def minus-real-def uminus-real-alg plus-real-alg .
lift-definition of-rat-real-alg :: rat => real-alg is of-rat-3 .
lemma of-rat-real-alg:real-of-rat x = real-of (of-rat-real-alg x)
by (transfer, rule of-rat-3[symmetric])
instantiation real-alg :: zero

```

\section*{begin}
definition zero-real-alg :: real-alg where zero-real-alg \(\equiv\) of-rat-real-alg 0 instance ..
end
lemma zero-real-alg: \(0=\) real-of 0
unfolding zero-real-alg-def by (simp add: of-rat-real-alg[symmetric])
```

instantiation real-alg :: one
begin
definition one-real-alg :: real-alg where one-real-alg \equivof-rat-real-alg 1
instance ..
end
lemma one-real-alg: 1 = real-of 1
unfolding one-real-alg-def by (simp add:of-rat-real-alg[symmetric])
instantiation real-alg :: times
begin
lift-definition times-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ real-alg is mult-3
by (simp add: mult-3)
instance ..
end
lemma times-real-alg:(real-of x)*(real-of y)=real-of (x*y)
by (transfer, rule mult-3[symmetric])

```
instantiation real-alg :: inverse
begin
lift-definition inverse-real-alg \(::\) real-alg \(\Rightarrow\) real-alg is inverse-3
    by (simp add: inverse-3)
definition divide-real-alg \(::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg where
    divide-real-alg \(x y=x *\) inverse \(y\)
instance ..
end
lemma inverse-real-alg: inverse (real-of \(x)=\) real-of (inverse \(x)\)
    by (transfer, rule inverse- \(3[\) symmetric \(]\) )
lemma divide-real-alg: \((\) real-of \(x) /(\) real-of \(y)=\) real-of \((x / y)\)
    unfolding divide-real-alg-def times-real-alg[symmetric] divide-real-def inverse-real-alg
instance real-alg :: ab-group-add
    apply intro-classes
```

        apply (transfer, unfold add-3, force)
        apply (unfold zero-real-alg-def, transfer, unfold add-3 of-rat-3, force)
        apply (transfer, unfold add-3 of-rat-3, force)
        apply (transfer, unfold add-3 uminus-3 of-rat-3, force)
        apply (unfold minus-real-alg-def, force)
    done
instance real-alg :: field
apply intro-classes
apply (transfer, unfold mult-3, force)
apply (transfer, unfold mult-3, force)
apply (unfold one-real-alg-def, transfer, unfold mult-3 of-rat-3, force)
apply (transfer, unfold mult-3 add-3, force simp: field-simps)
apply (unfold zero-real-alg-def, transfer, unfold of-rat-3, force)
apply (transfer, unfold mult-3 inverse-3 of-rat-3, force simp: field-simps)
apply (unfold divide-real-alg-def, force)
apply (transfer, unfold inverse-3 of-rat-3, force)
done
instance real-alg :: numeral ..
lift-definition root-real-alg :: nat }=>\mathrm{ real-alg }=>\mathrm{ real-alg is root-3
by (simp add: root-3)
lemma root-real-alg: root n (real-of x) = real-of (root-real-alg n x)
by (transfer, rule root-3[symmetric])
lift-definition sgn-real-alg-rat :: real-alg => rat is sgn-3
by (insert sgn-3, metis to-rat-of-rat)
lemma sgn-real-alg-rat:real-of-rat (sgn-real-alg-rat x) = sgn (real-of x)
by (transfer, auto simp: sgn-3)
instantiation real-alg :: sgn
begin
definition sgn-real-alg :: real-alg }=>\mathrm{ real-alg where
sgn-real-alg x = of-rat-real-alg (sgn-real-alg-rat x)
instance ..
end
lemma sgn-real-alg: sgn (real-of x) =real-of ( }\operatorname{sgn}x
unfolding sgn-real-alg-def of-rat-real-alg[symmetric]
by (transfer, simp add: sgn-3)

```
```

instantiation real-alg :: equal
begin
lift-definition equal-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ bool is equal-3
by (simp add: equal-3)
instance
proof
fix x y :: real-alg
show equal-class.equal x y = (x=y)
by (transfer, simp add: equal-3)
qed
end
lemma equal-real-alg: HOL.equal (real-of x) (real-of y)=(x=y)
unfolding equal-real-def by (transfer, auto)
instantiation real-alg :: ord
begin
definition less-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ bool where
[code del]: less-real-alg x y = (real-of }x<\mathrm{ real-of y)
definition less-eq-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ bool where
[code del]:less-eq-real-alg x y = (real-of }x\leq\mathrm{ real-of }y
instance ..
end
lemma less-real-alg:less (real-of x) (real-of y)=(x<y) unfolding less-real-alg-def
lemma less-eq-real-alg: less-eq (real-of x) (real-of y)=(x\leqy) unfolding less-eq-real-alg-def
instantiation real-alg :: compare-order
begin
lift-definition compare-real-alg :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ order is compare-3
by (simp add: compare-3)
lemma compare-real-alg: compare (real-of x) (real-of y) = (compare x y)
by (transfer, simp add: compare-3)
instance
proof (intro-classes, unfold compare-real-alg[symmetric, abs-def])
show le-of-comp ( }\lambdaxy\mathrm{ y. compare (real-of x) (real-of y)) = (土)
by (intro ext, auto simp: compare-real-def comparator-of-def le-of-comp-def
less-eq-real-alg-def)
show lt-of-comp ( }\lambdaxy\mathrm{ . compare (real-of x) (real-of y)) = (<)
by (intro ext, auto simp: compare-real-def comparator-of-def lt-of-comp-def

```
```

less-real-alg-def)
show comparator ( }\lambdaxy.\mathrm{ compare (real-of x) (real-of y))
unfolding comparator-def
proof (intro conjI impI allI)
fix x y z :: real-alg
let ?r = real-of
note rc = comparator-compare[where ' }a=\mathrm{ real, unfolded comparator-def]
from rc show invert-order (compare (?r x) (?r y)) = compare (?r y) (?r x)
by blast
from rc show compare (?r x) (?r y) =Lt \Longrightarrow compare (?r y) (?r z) = Lt \Longrightarrow
compare (?r x) (?r z) = Lt by blast
assume compare (?r x) (?r y) = Eq
with rc have ?r x = ?r y by blast
thus }x=y\mathrm{ unfolding real-of-inj.
qed
qed
end
lemma less-eq-real-alg-code[code]:
(less-eq :: real-alg }=>\mathrm{ real-alg }=>\mathrm{ bool) = le-of-comp compare
(less :: real-alg => real-alg => bool) = lt-of-comp compare
by (rule ord-defs(1)[symmetric], rule ord-defs(2)[symmetric])
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 ..
end
lemma abs-real-alg: abs (real-of x) = real-of (abs x)
unfolding abs-real-alg-def abs-real-def if-distrib
by (auto simp: uminus-real-alg)
lemma sgn-real-alg-sound: sgn x = (if x = 0 then 0 else if 0<real-of x then 1
else - 1)
(is - =?r)
proof -
have real-of (sgn x) = sgn (real-of x) by (simp add: sgn-real-alg)
also have ... = real-of ?r unfolding sgn-real-def if-distrib
by (auto simp: less-real-alg-def
zero-real-alg-def one-real-alg-def of-rat-real-alg[symmetric] equal-real-alg[symmetric]
equal-real-def uminus-real-alg[symmetric])
finally show sgn x = ?r unfolding equal-real-alg[symmetric] equal-real-def by
simp
qed
lemma real-of-of-int:real-of-rat (rat-of-int z) = real-of (of-int z)

```
```

proof (cases z \geq0)
case True
define }n\mathrm{ where }n=nat
from True have z:z=int n unfolding n-def by simp
show ?thesis unfolding z
by (induct n, auto simp: zero-real-alg plus-real-alg[symmetric] one-real-alg
hom-distribs)
next
case False
define n where n=nat (-z)
from False have z:z=- int n unfolding n-def by simp
show ?thesis unfolding z
by (induct n, auto simp: zero-real-alg plus-real-alg[symmetric] one-real-alg umi-
nus-real-alg[symmetric]
minus-real-alg[symmetric] hom-distribs)
qed
instance real-alg :: linordered-field
apply standard
apply (unfold less-eq-real-alg-def plus-real-alg[symmetric], force)
apply (unfold abs-real-alg-def less-real-alg-def zero-real-alg[symmetric], rule refl)
apply (unfold less-real-alg-def times-real-alg[symmetric], force)
apply (rule sgn-real-alg-sound)
done
instantiation real-alg :: floor-ceiling
begin
lift-definition floor-real-alg :: real-alg => int is floor-3
by (auto simp: floor-3)
lemma floor-real-alg: floor (real-of x) = floor x
by (transfer, auto simp: floor-3)
instance
proof
fix x :: real-alg
show of-int \lfloorx\rfloor\leqx^x< of-int (\lfloorx\rfloor+1) unfolding floor-real-alg[symmetric]
using floor-correct[of real-of x] unfolding less-eq-real-alg-def less-real-alg-def
real-of-of-int[symmetric] by (auto simp: hom-distribs)
hence }x\leq\mathrm{ of-int ( \x\ + 1) by auto
thus }\existsz.x\leqof-int z by blas
qed
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 }=>\mathrm{ -)
definition [simp]: euclidean-size-real-alg = (euclidean-size-field :: real-alg => -)
definition [simp]: division-segment (x :: real-alg) = 1
instance
by standard
(simp-all add: dvd-field-iff field-split-simps split: if-splits)
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 }=>\mathrm{ real-alg }=>\mathrm{ real-alg where
lcm-real-alg = Euclidean-Algorithm.lcm
definition Gcd-real-alg :: real-alg set }=>\mathrm{ real-alg where
Gcd-real-alg = Euclidean-Algorithm.Gcd
definition Lcm-real-alg :: real-alg set }=>\mathrm{ real-alg where
Lcm-real-alg = Euclidean-Algorithm.Lcm
instance by standard (simp-all add: gcd-real-alg-def lcm-real-alg-def Gcd-real-alg-def
Lcm-real-alg-def)
end
instance real-alg :: field-gcd ..
definition min-int-poly-real-alg :: real-alg }=>\mathrm{ int poly where
min-int-poly-real-alg x = (case info-real-alg x of Inl r mpoly-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 (cases info-real-alg x)
case (Inl r)
show ?thesis unfolding info-real-alg(2)[OF Inl] min-int-poly-real-alg-def Inl
by (simp add: min-int-poly-of-rat)
next
case (Inr pair)
then obtain p n where Inr: info-real-alg x = Inr ( p,n) by (cases pair, auto)
hence poly-cond p by (transfer, transfer, auto simp: info-2-card)
hence min-int-poly (real-of x) = p using info-real-alg(1)[OF Inr]
by (intro min-int-poly-unique, auto)
thus ?thesis unfolding min-int-poly-real-alg-def Inr by simp
qed

```
```

lemma min-int-poly-real-code: min-int-poly-real (real-of x) = min-int-poly-real-alg
x
by (simp add: min-int-poly-real-alg-real-of)
lemma min-int-poly-real-of: min-int-poly (real-of x) = min-int-poly x
proof (rule min-int-poly-unique[OF - min-int-poly-irreducible lead-coeff-min-int-poly-pos])
show min-int-poly x represents real-of x oops
definition real-alg-of-real :: real => real-alg where
real-alg-of-real }x=(\mathrm{ if ( }\existsy.x=\mathrm{ real-of y) then (THE y. x = real-of y) else 0)
lemma real-alg-of-real-code[code]: real-alg-of-real (real-of x) =x
using real-of-inj unfolding real-alg-of-real-def by auto
lift-definition to-rat-real-alg-main :: real-alg => rat option is to-rat-3
by (simp add: to-rat-3)
lemma to-rat-real-alg-main: to-rat-real-alg-main x (if real-of x }\in\mathbb{Q}\mathrm{ then
Some (THE q. real-of x =of-rat q) else None)
by (transfer, simp add: to-rat-3)
definition to-rat-real-alg :: real-alg => rat where
to-rat-real-alg x = (case to-rat-real-alg-main x of Some q=>q| None = 0)
definition is-rat-real-alg :: real-alg => bool where
is-rat-real-alg x = (case to-rat-real-alg-main x of Some q => True |None }=>\mathrm{ False)
lemma is-rat-real-alg: is-rat (real-of x) = (is-rat-real-alg x)
unfolding is-rat-real-alg-def is-rat to-rat-real-alg-main by auto
lemma to-rat-real-alg: to-rat (real-of x) = (to-rat-real-alg x)
unfolding to-rat to-rat-real-alg-def to-rat-real-alg-main by auto
lemma algebraic-real-code[code]: algebraic-real (real-of x) = True
proof (cases info-real-alg x)
case (Inl r)
show ?thesis using info-real-alg(2)[OF Inl] by (auto simp: algebraic-of-rat)
next
case (Inr pair)
then obtain p n where Inr: info-real-alg x = Inr ( }p,n)\mathrm{ by (cases pair, auto)
from info-real-alg(1)[OF Inr] have p represents (real-of x) by auto
thus ?thesis by (auto simp: algebraic-altdef-ipoly)
qed

```

\subsection*{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 $\Rightarrow$ real $\Rightarrow$ real
uminus :: real $\Rightarrow$ real
minus :: real $\Rightarrow$ real $\Rightarrow$ real
times :: real $\Rightarrow$ real $\Rightarrow$ real
inverse :: real $\Rightarrow$ real
divide :: real $\Rightarrow$ real $\Rightarrow$ real
floor :: real $\Rightarrow$ int
HOL.equal :: real $\Rightarrow$ real $\Rightarrow$ bool
compare $::$ real $\Rightarrow$ real $\Rightarrow$ order
less-eq :: real $\Rightarrow$ real $\Rightarrow$ bool
less :: real $\Rightarrow$ real $\Rightarrow$ bool
0 :: real
1 :: real
sgn :: real $\Rightarrow$ real
abs :: real $\Rightarrow$ real
min-int-poly-real
root]]

```
declare real-alg-code-eqns [code equation]
lemma Ratreal-code[code]:
    Ratreal \(=\) real-of \(\circ\) of-rat-real-alg
    by (transfer, transfer) (simp add: fun-eq-iff of-rat-2)
lemma real-of-post[code-post]: real-of (Real-Alg-Quotient (Real-Alg-Invariant (Rational \(x)\) )) \(=\) of-rat \(x\) proof (transfer)
fix \(x\)
show real-of-3 (Real-Alg-Invariant \((\) Rational \(x))=\) real-of-rat \(x\)
by (simp add: Real-Alg-Invariant-inverse real-of-3.rep-eq)
qed
end

\section*{12 Real Roots}

This theory contains an algorithm to determine the set of real roots of a rational polynomial. For polynomials with real coefficients, we refer to the AFP entry "Factor-Algebraic-Polynomial".
```

theory Real-Roots
imports
Cauchy-Root-Bound
Real-Algebraic-Numbers
begin
hide-const (open) UnivPoly.coeff
hide-const (open) Module.smult

```
partial-function (tailrec) roots-of-2-main ::
int poly \(\Rightarrow\) root-info \(\Rightarrow(\) rat \(\Rightarrow\) rat \(\Rightarrow\) nat \() \Rightarrow(\) rat \(\times\) rat \()\) list \(\Rightarrow\) real-alg-2 list \(\Rightarrow\) real-alg-2 list where
[code]: roots-of-2-main p ri cr lrs rais \(=\) (case lrs of Nil \(\Rightarrow\) rais
| ( \(l, r\) ) \# lrs \(\Rightarrow\) let \(c=c r l r\) 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 lr \# rais)
else let \(m=(l+r) / 2\) in roots-of-2-main pricr \(((m, r) \#(l, m) \#\) lrs \()\) rais \()\)
definition roots-of-2-irr :: int poly \(\Rightarrow\) real-alg-2 list where
roots-of-2-irr \(p=\) (if degree \(p=1\)
then [Rational (Rat.Fract \((-\operatorname{coeff} p \operatorname{0})(\operatorname{coeff} p 1))]\) else
let ri \(=\) root-info \(p\);
\(c r=\) root-info.l-r ri;
\(B=\) root-bound \(p\)
in (roots-of-2-main p ri cr \([(-B, B)][]))\)
fun pairwise-disjoint :: ' \(a\) set list \(\Rightarrow\) bool where
pairwise-disjoint []\(=\) True
\(\mid\) pairwise-disjoint \((x \# x s)=((x \cap(\bigcup y \in\) set xs. \(y)=\{ \}) \wedge\) pairwise-disjoint xs)
lemma roots-of-2-irr: assumes pc: poly-cond \(p\) and deg: degree \(p>0\)
```

    shows 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 -
note d = roots-of-2-irr-def
from poly-condD[OF pc] have mon: lead-coeff p>0 and irr: irreducible p by
auto
let ?norm = real-alg-2'
have ?one ^ ?two ^ ?three
proof (cases degree p=1)
case True
define c where c= coeff p 0
define d where d}=\mathrm{ coeff p 1
from True have rr: roots-of-2-irr p = [Rational (Rat.Fract (-c) (d))] un-
folding d d-def c-def by auto
from degree1-coeffs[OF True] have p:p=[:c,d:] and d:d\not=0 unfolding
c-def d-def by auto
have *: real-of-int c +x* real-of-int d=0\Longrightarrowx=-(real-of-int c / real-of-int
d) for }
using d by (simp add: field-simps)
show ?thesis unfolding rr using d* unfolding p using of-rat-1 [of Rat.Fract
(-c) (d)]
by (auto simp: Fract-of-int-quotient hom-distribs)
next
case False
let ?r = real-of-rat
let ?rp = map-poly ?r
let ?rr = set (roots-of-2-irr p)
define ri where ri= root-info p
define cr where cr = root-info.l-r ri
define bnds where bnds = [(-root-bound p, root-bound p)]
define empty where empty = (Nil :: real-alg-2 list)
have empty: Ball (set empty) invariant-2 ^ distinct (map real-of-2 empty)
unfolding empty-def by auto
from mon have p:p\not=0 by auto
from root-info[OF irr deg] have ri: root-info-cond ri p unfolding ri-def .
from False
have rr: roots-of-2-irr p = roots-of-2-main p ri cr bnds empty
unfolding d ri-def cr-def Let-def bnds-def empty-def by auto
note root-bound = root-bound[OF refl deg]
from root-bound(2)
have bnds: \bigwedgelr. (l,r) \in set bnds \Longrightarrowl\leqr unfolding bnds-def by auto
have ipoly p x=0\Longrightarrow?r (- root-bound p) \leqx ^x\leq?r (root-bound p) for x
using root-bound(1)[of x] by (auto simp: hom-distribs)
hence rts: {x. ipoly p x=0}
= real-of-2' set empty \cup{x.\existslr. root-cond (p,l,r) x\wedge(l,r) \in set bnds}
unfolding empty-def bnds-def by (force simp: root-cond-def)
define rts where rts lr = Collect (root-cond ( }p,lr))\mathrm{ for lr
have disj: pairwise-disjoint (real-of-2'set empty \# map rts bnds)

```
unfolding empty-def bnds-def by auto
from deg False have deg1: degree \(p>1\) by auto
define delta where delta \(=\) ipoly-root-delta \(p\)
note delta \(=\) ipoly-root-delta[OF p, folded delta-def]
define \(\mathrm{rel}^{\prime}\) where \(\mathrm{rel}^{\prime}=(\{(x, y) .0 \leq y \wedge \text { delta-gt delta } x y\})^{\wedge}-1\)
define \(m m\) where \(m m=(\lambda b n d s . m s e t(\operatorname{map}(\lambda(l, r)\). ?r \(r-\) ?r \(l) b n d s))\)
define rel where rel \(=\) inv-image (mult1 rel') mm
have wf: wf rel unfolding rel-def rel'-def
by (rule wf-inv-image[OF wf-mult1 [OF SN-imp-wf[OF delta-gt-SN[OF delta(1)]]]])
let ? main \(=\) roots-of-2-main pri cr
have real-of-2'set (?main bnds empty) \(=\)
real-of-2' set empty \(\cup\)
\(\{x . \exists l r\) root-cond \((p, l, r) x \wedge(l, r) \in \operatorname{set} b n d s\} \wedge\)
Ball (set (?main bnds empty)) invariant-2 \(\wedge\) distinct (map real-of-2 (?main
bnds empty)) (is ?one' \(\wedge\) ?two' \(\wedge\) ? three')
using empty bnds disj
proof (induct bnds arbitrary: empty rule: wf-induct[OF wf])
case (1 lrss rais)
note rais \(=1\) (2) [rule-format \(]\)
note lrs \(=1\) (3)
note disj \(=1\) (4)
note \(I H=1(1)[\) rule-format \(]\)
note simp \(=\) roots-of-2-main.simps[of p ri cr lrss rais]
show ? case
proof (cases lrss)
case Nil
with rais show ?thesis unfolding simp by auto
next
case (Cons lr lrs)
obtain \(l r\) where \(l r^{\prime}: l r=(l, r)\) by force
\{
fix \(l r^{\prime}\)
assume \(l t: \bigwedge l^{\prime} r^{\prime} .\left(l^{\prime}, r^{\prime}\right) \in\) set \(l r^{\prime} \Longrightarrow\)
\(l^{\prime} \leq r^{\prime} \wedge\) delta-gt delta \((\) ? \(r ~ r-? r l)\left(? r r^{\prime}-? r l^{\prime}\right)\)
have \(l: m m\left(l r^{\prime} @ l r s\right)=m m\) lrs \(+m m l r^{\prime}\) unfolding mm-def by (auto
simp: ac-simps)
have \(r: m m\) lrss \(=m m\) lrs \(+\{\#\) ?r \(r-\) ?r \(l \#\}\) unfolding Cons lr \({ }^{\prime}\)
rel-def mm-def
by auto
have \(\left(m m\left(l r^{\prime} @ l r s\right), m m\right.\) lrss \() \in\) mult1 rel' unfolding \(l\) r mult1-def
proof (rule, unfold split, intro exI conjI, unfold add-mset-add-single[symmetric],
rule refl, rule refl, intro allI impI)
fix \(d\)
assume \(d \in \# m m l r^{\prime}\)
then obtain \(l^{\prime} r^{\prime}\) where \(d: d=? r r^{\prime}-? r l^{\prime}\) and \(l r^{\prime}:\left(l^{\prime}, r^{\prime}\right) \in\) set \(l r^{\prime}\) unfolding mm-def in-multiset-in-set by auto
from \(l t[O F l r]\)
show \((d\), ? \(r ~ r ~-~ ? r ~ l) \in r e l ' ~ u n f o l d i n g ~ d ~ r e l '-d e f ~\)
by (auto simp: of-rat-less-eq)
```

        qed
        hence (lr' @ lrs,lrss) \in rel unfolding rel-def by auto
    } note rel = this
    from rel[of Nil] have easy-rel: (lrs,lrss) \in rel by auto
    define c where cocrlr
    from simp Cons lr' have simp: ?main lrss rais =
        (if c = 0 then ?main lrs rais else if c=1 then
            ?main lrs (real-alg-2' ri plr # rais)
            else let m=(l+r)/2 in ?main ((m,r) # (l,m) # lrs) rais)
    unfolding c-def simp Cons lr' using real-alg-2'[OF False] by auto
    note lrs = lrs[unfolded Cons lr]
    from lrs have lr:l\leqr by auto
    from root-info-condD(1)[OF ri lr, folded cr-def]
    have c:c=card {x. root-cond (p,l,r)x} unfolding c-def by auto
    let ?rt = \lambdalrs. {x. \existslr. root-cond (p,l,r) x^(l,r)\in set lrs}
    have rts:?rt lrss = ?rt lrs }\cup{x.root-cond (p,l,r) x} (is ?rt1 = ?rt2 \cup
    ?rt3)
unfolding Cons lr' by auto
show ?thesis
proof (cases c=0)
case True
with simp have simp: ?main lrss rais = ?main lrs rais by simp
from disj have disj: pairwise-disjoint (real-of-2'set rais \# map rts lrs)
unfolding Cons by auto
from finite-ipoly-roots[OF p] True[unfolded c] have empty: ?rt3 = {}
unfolding root-cond-def[abs-def] split by simp
with rts have rts: ?rt1 = ?rt2 by auto
show ?thesis unfolding simp rts
by (rule IH[OF easy-rel rais lrs disj], auto)
next
case False
show ?thesis
proof (cases c=1)
case True
let ?rai = real-alg-2' ri plr
from True simp have simp: ?main lrss rais = ?main lrs (?rai \# rais)
by auto
from card-1-Collect-ex1[OF c[symmetric, unfolded True]]
have ur: unique-root ( }p,l,r\mathrm{ ) .
from real-alg-2'[OF ur pc ri]
have rai: invariant-2 ?rai real-of-2 ?rai = the-unique-root ( }p,l,r)\mathrm{ by
auto
with rais have rais: \ x. x set (?rai \# rais)\Longrightarrow invariant-2 }
and dist: distinct (map real-of-2 rais) by auto
have rt3: ?rt3 = {real-of-2 ? rai}
using ur rai by (auto intro: the-unique-root-eqI theI')
have real-of-2'set (roots-of-2-main p ri cr lrs (?rai \# rais))=
real-of-2'set (?rai \# rais) \cup ?rt2 ^
Ball (set (roots-of-2-main p ri cr lrs (?rai \# rais))) invariant-2 ^

```
```

        distinct (map real-of-2 (roots-of-2-main p ri cr lrs (?rai # rais)))
        (is ?one ^ ?two ^ ?three)
        proof (rule IH[OF easy-rel, of ?rai # rais, OF conjI lrs])
            show Ball (set (real-alg-2' ri p l r # rais)) invariant-2 using rais by
    auto
have real-of-2 (real-alg-2' ri p l r) \& set (map real-of-2 rais)
using disj rt3 unfolding Cons lr' rts-def by auto
thus distinct (map real-of-2 (real-alg-2' ri plr \# rais)) using dist by
auto
show pairwise-disjoint (real-of-2' set (real-alg-2' ri p l r \# rais) \#
map rts lrs)
using disj rt3 unfolding Cons lr' rts-def by auto
qed auto
hence ?one ?two ?three by blast+
show ?thesis unfolding simp rts rt3
by (rule conjI[OF - conjI[OF〈?two〉<?three>]], unfold <?one>, auto)
next
case False
let ? m = (l+r)/2
let ?lrs = [(?m,r),(l,?m)]@ lrs
from False <c\not=0\rangle have simp: ?main lrss rais = ?main ?lrs rais
unfolding simp by (auto simp: Let-def)
from False {c\not=0\rangle have c\geq2 by auto
from delta(2)[OF this[unfolded c]] have delta: delta \leq?r (r - l)/4
by auto
have lrs: \bigwedgelr. (l,r) \in set ?lrs \Longrightarrowl\leqr
using lr lrs by (fastforce simp: field-simps)
have ?r ?m \in\mathbb{Q unfolding Rats-def by blast}
with poly-cond-degree-gt-1[OF pc deg1, of ?r ?m]
have disj1: ?r ?m \# rts lr for lr unfolding rts-def root-cond-def by
auto
have disj2:rts (?m,r)\caprts (l,?m)={} using disj1[of (l,?m)] disj1[of
(?m,r)]
unfolding rts-def root-cond-def by auto
have disj3: (rts (l,?m) \cuprts (?m,r)) = rts (l,r)
unfolding rts-def root-cond-def by (auto simp: hom-distribs)
have disj4: real-of-2' set rais \cap rts (l,r) ={} using disj unfolding
Cons lr' by auto
have disj: pairwise-disjoint (real-of-2' set rais \# map rts ([(?m,r), (l,
?m)]@ lrs))
using disj disj2 disj3 disj4 by (auto simp: Cons lr')
have (?lrs,lrss) \in rel
proof (rule rel, intro conjI)
fix l' r
assume mem: }(\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\in\operatorname{set}[(?m,r),(l,?m)
from mem lr show l'
from mem have diff: ?r r r - ?r l l}=(?r r - ?r l) / 2 by aut
(metis eq-diff-eq minus-diff-eq mult-2-right of-rat-add of-rat-diff,
metis of-rat-add of-rat-mult of-rat-numeral-eq)

```
```

    show delta-gt delta (?r r - ?r l) (?r r' - ?r l') unfolding diff
    delta-gt-def by (rule order.trans[OF delta], insert lr,
    auto simp: field-simps of-rat-diff of-rat-less-eq)
    qed
note IH = IH[OF this, of rais, OF rais lrs disj]
have real-of-2'set (?main ?lrs rais)=
real-of-2' set rais \cup?rt ?lrs }
Ball (set (?main ?lrs rais)) invariant-2 ^ distinct (map real-of-2 (?main
?lrs rais))
(is ?one ^?two)
by (rule IH)
hence ?one ?two by blast+
have cong: \abc. b=c\Longrightarrowa\cupb=a\cupc by auto
have id: ?rt ?lrs = ?rt lrs \cup ?rt [(?m,r),(l,?m)] by auto
show ?thesis unfolding rts simp 〈?one〉 id
proof (rule conjI[OF cong[OF cong] conjI])
have }\x.root-cond (p,l,r)x=(root-cond (p,l,?m) x\vee root-con
(p,?m,r)x)
unfolding root-cond-def by (auto simp:hom-distribs)
hence id: Collect (root-cond (p,l,r)) ={x. (root-cond (p,l,?m) x\vee
root-cond ( }p,?m,r)x)
by auto
show ?rt [(?m,r),(l,?m)] = Collect (root-cond (p,l,r)) unfolding id
list.simps by blast
show }\foralla\in\mathrm{ set (?main ?lrs rais). invariant-2 a using <?two> by auto
show distinct (map real-of-2 (?main ?lrs rais)) using <?two> by auto
qed
qed
qed
qed
qed
hence idd:?one' and cond:?two' ?three' by blast+
define res where res = roots-of-2-main p ri cr bnds empty
have e: set empty = {} unfolding empty-def by auto
from idd[folded res-def] e have idd:real-of-2'set res ={}\cup{x.\existslr. root-cond
(p,l,r) x^(l,r)\in set bnds}
by auto
show ?thesis
unfolding rr unfolding rts id e norm-def using cond
unfolding res-def[symmetric] image-empty e idd[symmetric] by auto
qed
thus ?one ?two ?three by blast+
qed
definition roots-of-2 :: int poly }=>\mathrm{ 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\not=0\Longrightarrow real-of-2'set (roots-of-2 p)={x. ipoly p x = 0}
    Ball (set (roots-of-2 p)) invariant-2
    distinct (map real-of-2 (roots-of-2 p))
    proof -
let ?rr = roots-of-2 p
note d}=\mathrm{ roots-of-2-def
note frp1 = factors-of-int-poly
{
fix qr
assume q\in set ?rr
then obtain s}\mathrm{ where
s:s
q:q\in set (roots-of-2-irr s)
unfolding d by auto
from frp1(1)[OF refl s] have poly-cond s degree s>0 by (auto simp: poly-cond-def)
from roots-of-2-irr[OF this] q
have invariant-2 q by auto
}
thus Ball (set ?rr) invariant-2 by auto
{
assume p: p\not=0
have real-of-2'set ?rr = (U ((\lambda p.real-of-2'set (roots-of-2-irr p))'
(set (factors-of-int-poly p))))
(is - = ?rrr)
unfolding d set-concat set-map by auto
also have ... = {x. ipoly p x=0}
proof -
{
fix }
assume x }\in\mathrm{ ?rrr
then obtain qs where
s:s}\in\mathrm{ set (factors-of-int-poly p) and
q:q\in set (roots-of-2-irr s) and
x:x = real-of-2 q by auto
from frp1(1)[OF refl s] have s0:s\not=0 and pt: poly-cond s degree s>0
by (auto simp: poly-cond-def)
from roots-of-2-irr[OF pt] q have rt: ipoly s }x=0\mathrm{ unfolding }x\mathrm{ by auto
from frp1(2)[OF refl p, of x] rt s have rt: ipoly p x = 0 by auto
}
moreover
{
fix x :: real
assume rt: ipoly p x = 0
from rt frp1(2)[OF refl p, of x] obtain s where s: s\in set (factors-of-int-poly
p)
and rt: ipoly s x = 0 by auto
from frp1(1)[OF refl s] have s0:s\not=0 and ty: poly-cond s degree s>0
by (auto simp: poly-cond-def)
from roots-of-2-irr(1)[OF ty] rt obtain q}\mathrm{ where

```
```

                    q:q\in set (roots-of-2-irr s) and
                    x:x = real-of-2 q by blast
            have }x\in\mathrm{ ?rrr unfolding x using q s by auto
        }
        ultimately show ?thesis by auto
    qed
    finally show real-of-2'set ?rr = {x. ipoly p x=0} by auto
    }
show distinct (map real-of-2 (roots-of-2 p))
proof (cases p=0)
case True
from factors-of-int-poly-const[of 0] True show ?thesis unfolding roots-of-2-def
by auto
next
case p: False
note frp1 = frp1[OF refl]
let ?fp = factors-of-int-poly p
let ?cc = concat (map roots-of-2-irr ?fp)
show ?thesis unfolding roots-of-2-def distinct-conv-nth length-map
proof (intro allI impI notI)
fix ij
assume ij: i< length ?cc j< length ?cc i\not=j and id: map real-of-2 ?cc!i
= map real-of-2 ?cc!j
from ij id have id: real-of-2 (?cc!i) = real-of-2 (?cc!j) by auto
from nth-concat-diff[OF ij, unfolded length-map] obtain j1 k1 j2 k2 where
*: (j1,k1) \not= (j2,k2)
j1 < length ?fp j2 < length ?fp and
k1<length (map roots-of-2-irr ?fp ! j1)
k2 < length (map roots-of-2-irr ?fp!j2)
?cc!i= map roots-of-2-irr ?fp ! j1!k1
?cc!j = map roots-of-2-irr ?fp ! j2 ! k2 by blast
hence **: k1 < length (roots-of-2-irr (?fp!j1))
k2 < length (roots-of-2-irr (?fp! j2))
?cc!i= roots-of-2-irr (?fp!j1)!k1
?cc!j = roots-of-2-irr (?fp ! j2)!k2
by auto
from * have mem: ?fp!j1 \in set ?fp ?fp ! j2 \in set ?fp by auto
from frp1(1)[OF mem(1)] frp1(1)[OF mem(2)]
have pc1: poly-cond (?fp!j1) degree (?fp! j1)>0 and pc10: ?fp!j1\not=0
and pc2: poly-cond (?fp! j2) degree (?fp! j2)>0
by (auto simp: poly-cond-def)
show False
proof (cases j1 = j2)
case True
with * have neq: k1 \not=k2 by auto
from **[unfolded True] id *
have map real-of-2 (roots-of-2-irr (?fp!j2))!k1 = real-of-2 (?cc! j)
map real-of-2 (roots-of-2-irr (?fp!j2))!k1 = real-of-2 (?cc!j)
by auto

```
```

            hence ᄀ distinct (map real-of-2 (roots-of-2-irr (?fp!j2)))
                    unfolding distinct-conv-nth using *** True by auto
            with roots-of-2-irr(3)[OF pc2] show False by auto
    next
            case neq: False
                            with frp1(4)[of p] * have neq: ?fp ! j1 # ?fp ! j2 unfolding distinct-conv-nth
    by auto
let ?x = real-of-2 (?cc!i)
define }x\mathrm{ where }x=\mathrm{ ? }
from ** have x\in real-of-2' set (roots-of-2-irr (?fp!j1)) unfolding x-def
by auto
with roots-of-2-irr(1)[OF pc1] have x1: ipoly (?fp!j1) x=0 by auto
from ** id have x feal-of-2 'set (roots-of-2-irr (?fp!j2)) unfolding
x-def
by (metis image-eqI nth-mem)
with roots-of-2-irr(1)[OF pc2] have x2: ipoly (?fp! j2) x=0 by auto
have ipoly p x = 0 using x1 mem unfolding roots-of-2-def by (metis
frp1(2) p)
from frp1(3)[OF p this] x1 x2 neq mem show False by blast
qed
qed
qed
qed
lift-definition (code-dt) roots-of-3 :: int poly }=>\mathrm{ real-alg-3 list is roots-of-2
by (insert roots-of-2, auto simp: list-all-iff)
lemma roots-of-3:
shows p\not=0\Longrightarrow real-of-3'set (roots-of-3 p)={x. ipoly p x=0}
distinct (map real-of-3 (roots-of-3 p))
proof -
show p\not=0\Longrightarrow real-of-3'set (roots-of-3 p)={x. ipoly p x = 0}
by (transfer; intro roots-of-2, auto)
show distinct (map real-of-3 (roots-of-3 p))
by (transfer; insert roots-of-2, auto)
qed
lift-definition roots-of-real-alg :: int poly }=>\mathrm{ real-alg list is roots-of-3 .
lemma roots-of-real-alg:
p\not=0\Longrightarrow real-of'set (roots-of-real-alg p)={x. ipoly px=0}
distinct (map real-of (roots-of-real-alg p))
proof -
show }p\not=0\Longrightarrow\mathrm{ real-of'set (roots-of-real-alg p)={x. ipoly p x=0}
by (transfer', insert roots-of-3, auto)
show distinct (map real-of (roots-of-real-alg p))
by (transfer, insert roots-of-3(2), auto)
qed

```
```

definition real-roots-of-int-poly :: int poly }=>\mathrm{ 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 }=>\mathrm{ 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 }=>\mp@subsup{}{}{\prime}a :: field-char-0 = 'a
where rpoly f\equiv poly (map-poly of-rat f)
lemma real-roots-of-int-poly: p =0\Longrightarrow set (real-roots-of-int-poly p)={x. ipoly p
x=0}
distinct (real-roots-of-int-poly p)
unfolding real-roots-of-int-poly-def using roots-of-real-alg[of p] by auto
lemma real-roots-of-rat-poly: p =0\Longrightarrow set (real-roots-of-rat-poly p)={x. rpoly
px=0}
distinct (real-roots-of-rat-poly p)
proof -
obtain cq}\mathrm{ where cq: rat-to-int-poly p=(c,q) by force
from rat-to-int-poly[OF this]
have pq: p = smult (inverse (of-int c)) (of-int-poly q)
and c:c\not=0 by auto
have id: {x. rpoly p x=(0:: real)}={x. ipoly q }x=0
unfolding pq by (simp add: c of-rat-of-int-poly hom-distribs)
show distinct (real-roots-of-rat-poly p) unfolding real-roots-of-rat-poly-def cq
snd-conv
using roots-of-real-alg(2)[of q] .
assume p\not=0
with pq c have q: q\not=0 by auto
show set (real-roots-of-rat-poly p) ={x. rpoly px=0} unfolding id
unfolding real-roots-of-rat-poly-def cq snd-conv using roots-of-real-alg(1)[OF
q]
by auto
qed
end

```

\section*{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..
lemma real-poly-real-coeff: assumes set (coeffs p)\subseteq\mathbb{R}
shows coeff p x 价}
proof -
have coeff px\in range (coeff p) by auto
from this[unfolded range-coeff] assms show ?thesis by auto
qed
lemma complex-conjugate-root:
assumes real: set (coeffs p)\subseteq\mathbb{R}\mathrm{ and rt: poly p c=0}0=0
shows poly p (cnj c) = 0
proof -
let ?c = cnj c
{
fix }
have coeff px\in\mathbb{R}
by (rule real-poly-real-coeff[OF real])
hence cnj (coeff p x)= coeff p x by (cases coeff p x, auto)
} note cnj-coeff = this
have poly p ?c = poly ( }\sumx\leq\mathrm{ degree p. monom (coeff p x) x) ?c
unfolding poly-as-sum-of-monoms ..
also have ... = (\sumx\leqdegree p.coeff p x*cnj (c^ x))
unfolding poly-sum poly-monom complex-cnj-power ..
also have ... =( \sumx\leqdegree p.cnj (coeff px*c^^x))
unfolding complex-cnj-mult cnj-coeff ..
also have ... = cnj ( \sumx\leqdegree p.coeff p x*c^^ x)
unfolding cnj-sum ..
also have ( }\sumx\leq\mathrm{ degree p.coeff p x* c^ x)=
poly (\sumx\leqdegree p. monom (coeff p x) x) c
unfolding poly-sum poly-monom ..
also have ... = 0 unfolding poly-as-sum-of-monoms rt ..
also have cnj 0=0 by simp
finally show ?thesis.
qed
context
fixes p :: complex poly
assumes coeffs: set (coeffs p)\subseteq\mathbb{R}
begin
lemma map-poly-Re-poly: fixes }x\mathrm{ :: real

```
```

    shows poly (map-poly Re p) x = poly p (of-real x)
    proof -
have id: map-poly (of-real o Re) p=p
by (rule map-poly-idI, insert coeffs, auto)
show ?thesis unfolding arg-cong[OF id, of poly, symmetric]
by (subst map-poly-map-poly[symmetric], auto)
qed
lemma map-poly-Re-coeffs:
coeffs (map-poly Re p) = map Re (coeffs p)
proof (rule coeffs-map-poly)
have lead-coeff p \in range (coeff p) by auto

```

```

    show (Re (lead-coeff p)=0)=(p=0)
    using of-real-Re[OF x] by auto
    qed
lemma map-poly-Re-0: map-poly Re p=0\Longrightarrowp=0
using map-poly-Re-coeffs by auto
end
lemma real-poly-add:
assumes set (coeffs p)\subseteq\mathbb{R}}\mathrm{ set (coeffs q) }\subseteq\mathbb{R
shows set (coeffs (p+q))\subseteq\mathbb{R}
proof -
define pp where pp=coeffs p
define qq where qq= coeffs q
show ?thesis using assms
unfolding coeffs-plus-eq-plus-coeffs pp-def[symmetric] qq-def[symmetric]
by (induct pp qq rule: plus-coeffs.induct, auto simp: cCons-def)
qed
lemma real-poly-sum:
assumes \ x. x \inS\Longrightarrow set (coeffs (fx))\subseteq\mathbb{R}
shows set (coeffs (sum f S))\subseteq\mathbb{R}
using assms
proof (induct S rule: infinite-finite-induct)
case (insert x S)
hence id: sum f(insert xS)=fx+ sum fS by auto
show ?case unfolding id
by (rule real-poly-add[OF - insert(3)], insert insert, auto)
qed auto
lemma real-poly-smult: fixes p :: ' }a\mathrm{ :: {idom,real-algebra-1} poly
assumes c}\in\mathbb{R}\mathrm{ set (coeffs p)}\subseteq\mathbb{R
shows set (coeffs (smult c p))\subseteq\mathbb{R}
using assms by (auto simp: coeffs-smult)

```
```

lemma real-poly-pCons:
assumes c\in\mathbb{R}}\mathrm{ set (coeffs p)}\subseteq\mathbb{R
shows set (coeffs (pCons c p))\subseteq\mathbb{R}
using assms by (auto simp: cCons-def)
lemma real-poly-mult: fixes }p:: 'a :: {idom,real-algebra-1} poly
assumes p: set (coeffs p)\subseteq\mathbb{R}\mathrm{ and q: set (coeffs q) }\subseteq\mathbb{R}
shows set (coeffs (p*q))\subseteq\mathbb{R}\mathrm{ using p}
proof (induct p)
case (pCons a p)
show ?case unfolding mult-pCons-left
by (intro real-poly-add real-poly-smult real-poly-pCons pCons(2) q,
insert pCons(1,3), auto simp:cCons-def if-splits)
qed simp
lemma real-poly-power: fixes p :: ' }a\mathrm{ :: {idom,real-algebra-1} poly
assumes p: set (coeffs p)\subseteq\mathbb{R}
shows set (coeffs ( }\mp@subsup{p}{}{`}n))\subseteq\mathbb{R
proof (induct n)
case (Suc n)
from real-poly-mult[OF p this]
show ?case by simp
qed simp
lemma real-poly-prod: fixes }f::' 'a=>'b:: {idom,real-algebra-1} poly
assumes \x. x
shows set (coeffs (prod fS))\subseteq\mathbb{R}
using assms
proof (induct S rule: infinite-finite-induct)
case (insert x S)
hence id: prod f(insert x S)=fx* prod fS by auto
show ?case unfolding id
by (rule real-poly-mult[OF - insert(3)], insert insert, auto)
qed auto
lemma real-poly-uminus:
assumes set (coeffs p)\subseteq\mathbb{R}
shows set (coeffs (-p))\subseteq\mathbb{R}
using assms unfolding coeffs-uminus by auto
lemma real-poly-minus:
assumes set (coeffs p)\subseteq\mathbb{R}}\mathrm{ set (coeffs q) }\subseteq\mathbb{R
shows set (coeffs (p-q))\subseteq\mathbb{R}
using assms unfolding diff-conv-add-uminus
by (intro real-poly-uminus real-poly-add, auto)

```
```

lemma fixes $p::$ ' $a$ :: real-field poly
assumes $p$ : set $($ coeffs $p) \subseteq \mathbb{R}$ and $*$ : set $($ coeffs $q) \subseteq \mathbb{R}$
shows real-poly-div: set (coeffs $(q$ div $p)) \subseteq \mathbb{R}$
and real-poly-mod: set $($ coeffs $(q \bmod p)) \subseteq \mathbb{R}$
proof (atomize(full), insert *, induct q)
case 0
thus ?case by auto
next
case ( $p$ Cons a q)
from $p \operatorname{Cons}(1,3)$ have $a: a \in \mathbb{R}$ and $q: \operatorname{set}(\operatorname{coeffs} q) \subseteq \mathbb{R}$ by auto
note res $=p$ Cons
show ?case
proof (cases $p=0$ )
case True
with res $p \operatorname{Cons(3)}$ show ?thesis by auto
next
case False
from $p$ Cons have IH: set $($ coeffs $(q$ div $p)) \subseteq \mathbb{R}$ set $(\operatorname{coeffs}(q \bmod p)) \subseteq \mathbb{R}$ by
auto
define $c$ where $c=\operatorname{coeff}(p$ Cons $a(q \bmod p))($ degree $p) / \operatorname{coeff} p($ degree $p)$
\{
have coeff $(p$ Cons a $(q \bmod p))($ degree $p) \in \mathbb{R}$
by (rule real-poly-real-coeff, insert IH a, intro real-poly-pCons)
moreover have coeff $p$ (degree $p) \in \mathbb{R}$
by (rule real-poly-real-coeff $[O F \quad p]$ )
ultimately have $c \in \mathbb{R}$ unfolding $c$-def by simp
$\}$ note $c=t h i s$
from False
have $r$ : $p$ Cons a $q$ div $p=p$ Cons $c(q$ div $p)$ and $s: p$ Cons a $q \bmod p=p C o n s$
$a(q \bmod p)-$ smult $c p$
unfolding $c$-def div-pCons-eq mod-pCons-eq by simp-all
show ?thesis unfolding $r$ s using apcIH by (intro conjI real-poly-pCons
real-poly-minus real-poly-smult)
qed
qed
lemma real-poly-factor: fixes $p::$ ' $a$ :: real-field poly
assumes set (coeffs $(p * q)) \subseteq \mathbb{R}$
set $($ coeffs $p) \subseteq \mathbb{R}$
$p \neq 0$
shows set (coeffs $q) \subseteq \mathbb{R}$
proof -
have $q=p * q$ div $p$ using $\langle p \neq 0\rangle$ by simp
hence $i d$ : coeffs $q=$ coeffs ( $p * q$ div $p$ ) by simp
show ?thesis unfolding id
by (rule real-poly-div, insert assms, auto)
qed
lemma complex-conjugate-order: assumes real: set (coeffs $p) \subseteq \mathbb{R}$

```
```

    p\not=0
    shows order (cnj c) p = order c p
    proof -
define }n\mathrm{ where }n=\mathrm{ degree }
have degree p}\leqn\mathrm{ unfolding n-def by auto
thus ?thesis using assms
proof (induct n arbitrary: p)
case (0 p)
{
fix }
have order x p \leq degree p
by (rule order-degree[OF 0(3)])
hence order x p=0 using 0 by auto
}
thus ?case by simp
next
case (Suc m p)
note order = order[OF }\langlep\not=0\rangle
let ?c = cnj c
show ?case
proof (cases poly p c = 0)
case True note rt1 = this
from complex-conjugate-root[OF Suc(3) True]
have rt2: poly p ?c = 0.
show ?thesis
proof (cases c \in\mathbb{R})
case True
hence ?c = c by (cases c, auto)
thus ?thesis by auto
next
case False
hence neq:?c f=c by (simp add: Reals-cnj-iff)
let ?fac1 = [:-c, 1:]
let ?fac2 = [:-?c, 1 :]
let ?fac= ?fac1 * ?fac2
from rt1 have ?fac1 dvd p unfolding poly-eq-0-iff-dvd .
from this[unfolded dvd-def] obtain q}\mathrm{ where p: p=?fac1*q by auto
from rt2[unfolded p poly-mult] neq have poly q ?c c=0 by auto
hence ?fac2 dvd q unfolding poly-eq-O-iff-dvd .
from this[unfolded dvd-def] obtain r where q: q=?fac2 * r by auto
have p:p=?fac*r unfolding pq by algebra
from }\langlep\not=0\rangle\mathrm{ have nz:?fac1 }\not=0\mathrm{ ?fac2 }\not=0\mathrm{ ? ?fac }\not=0r\not=0\mathrm{ unfolding p
by auto
have id:?fac = [:?c * c,-(?c + c), 1 :] by simp
have cfac:coeffs ?fac = [?c*c,-(?c+c),1] unfolding id by simp
have cfac: set (coeffs ?fac)\subseteq\mathbb{R}\mathrm{ unfolding cfac by (cases c, auto simp:}
Reals-cnj-iff)
have degree p = degree ?fac + degree r unfolding p
by (rule degree-mult-eq, insert nz, auto)

```
```

            also have degree ?fac = degree ?fac1 + degree ?fac2
            by (rule degree-mult-eq, insert nz, auto)
            finally have degree p=2 + degree r by simp
            with Suc have deg: degree r\leqm by auto
            from real-poly-factor[OF Suc(3)[unfolded p] cfac] nz have set (coeffs r) \subseteq
    R by auto
from Suc(1)[OF deg this 〈r\not=0\rangle] have IH: order ?c r = order c r.
{
fix cc
have order cc p = order cc ?fac + order cc r using < p}\not=0\mathrm{ \ unfolding p
by (rule order-mult)
also have order cc ?fac = order cc ?fac1 + order cc ?fac2
by (rule order-mult, rule nz)
also have order cc ?fac1 = (if cc=c then 1 else 0)
unfolding order-linear' by simp
also have order cc ?fac2 = (if cc=?c then 1 else 0)
unfolding order-linear' by simp
finally have order cc p=
(if cc = c then 1 else 0) + (if cc= cnj c then 1 else 0) + order cc r .
} note order = this
show ?thesis unfolding order IH by auto
qed
next
case False note rt1 = this
{
assume poly p ?c = 0
from complex-conjugate-root[OF Suc(3) this] rt1
have False by auto
}
hence rt2: poly p?c\not=0 by auto
from rt1 rt2 show ?thesis
unfolding order-root by simp
qed
qed
qed
lemma map-poly-of-real-Re: assumes set (coeffs p)\subseteq\mathbb{R}
shows map-poly of-real (map-poly Re p) = p
by (subst map-poly-map-poly, force+, rule map-poly-idI, insert assms, auto)
lemma map-poly-Re-of-real: map-poly Re (map-poly of-real p)=p
by (subst map-poly-map-poly, force+, rule map-poly-idI, auto)
lemma map-poly-Re-mult: assumes p: set (coeffs p)\subseteq\mathbb{R}
and q: set (coeffs q)\subseteq\mathbb{R}\mathrm{ shows map-poly Re (p*q)= map-poly Re p * map-poly}<br>mp@code{m}
Re q
proof -
let ?r = map-poly Re
let ?c = map-poly complex-of-real

```
```

    have ?r ( p*q) = ?r (?c (?r p)* ?c (?r q))
            unfolding map-poly-of-real-Re[OF p] map-poly-of-real-Re[OF q] by simp
    also have ?c (?r p)* ?c (?r q) = ?c (?r p * ?r q) by (simp add: hom-distribs)
    also have ?r ... = ?r p * ?r q unfolding map-poly-Re-of-real ..
    finally show ?thesis.
    qed
lemma map-poly-Re-power: assumes p: set (coeffs p)\subseteq\mathbb{R}
shows map-poly Re (p`n)=(map-poly Re p)^n proof (induct n)     case (Suc n)     let ?r = map-poly Re     have ?r ( p^Suc n) =?r (p* p^n) by simp     also have ... = ?r p * ?r ( p`n)
by (rule map-poly-Re-mult[OF p real-poly-power[OF p]])
also have ?r ( p`n)}=(\mathrm{ ?r p)^n by (rule Suc)
finally show ?case by simp
qed simp
lemma real-degree-2-factorization-exists-complex: fixes p :: complex poly
assumes pR: set (coeffs p)\subseteq\mathbb{R}
shows \exists qs. p= prod-list qs }\wedge(\forallq\in\mathrm{ set qs. set (coeffs q) }\subseteq\mathbb{R}\wedge\mathrm{ degree }q\leq2
proof -
obtain n where degree p=n by auto
thus ?thesis using pR
proof (induct n arbitrary: p rule: less-induct)
case (less n p)
hence pR: set (coeffs p)\subseteq\mathbb{R}\mathrm{ by auto}
show ?case
proof (cases n\leq2)
case True
thus ?thesis using pR
by (intro exI[of - [p]], insert less(2), auto)
next
case False
hence degp: degree p \geq2 using less(2) by auto
hence \neg constant (poly p) by (simp add: constant-degree)
from fundamental-theorem-of-algebra[OF this] obtain x where x: poly p x=
0 by auto
from x have dvd: [:-x, 1 :] dvd p using poly-eq-0-iff-dvd by blast
have }\existsf.f dvd p\wedge set (coeffsf)\subseteq\mathbb{R}\wedge1\leqdegree f ^degree f\leq2
proof (cases x }\in\mathbb{R}\mathrm{ )
case True
with dvd show ?thesis
by (intro exI[of - [:-x, 1:]], auto)
next
case False
let ?x = cnj x
let ?a = ? }x*

```
```

            let ?b = - ? }x-
            from complex-conjugate-root[OF pR x]
            have xx: poly p?x = 0 by auto
            from False have diff: x #=?x by (simp add: Reals-cnj-iff)
            from dvd obtain r where p: p=[:-x,1 :] *r unfolding dvd-def by
    auto
from xx[unfolded this] diff have poly r ?x = 0 by simp
hence [:- ?x, 1 :] dvd r using poly-eq-0-iff-dvd by blast
then obtain s where r:r=[:-?x,1 :]*s unfolding dvd-def by auto
have p=([:-x,1:]*[:-?x,1:])*s unfolding pr by algebra
also have [:-x,1:]*[:-?x, 1:] = [:?a,?b, 1:] by simp
finally have [:?a,?b, 1 :] dvd p unfolding dvd-def by auto
moreover have ?a }\in\mathbb{R}\mathrm{ by (simp add: Reals-cnj-iff)
moreover have ?b }\in\mathbb{R}\mathrm{ by (simp add: Reals-cnj-iff)
ultimately show ?thesis by (intro exI[of - [:?a,?b,1:]], auto)
qed
then obtain f}\mathrm{ where dvd: fdvd p and fR: set (coeffs f)}\subseteq\mathbb{R}\mathrm{ and degf: 1
\leq degree f degree f \leq2 by auto
from dvd obtain r where p: p=f*r unfolding dvd-def by auto
from degp have p0:p}=0\mathrm{ by auto
with p have f0:f\not=0 and r0:r\not=0 by auto
from real-poly-factor[OF pR[unfolded p] fR f0] have rR: set (coeffs r)\subseteq\mathbb{R}.
have deg: degree p=degree f + degree r unfolding p
by (rule degree-mult-eq[OF f0 rO])
with degf less(2) have degr: degree r < n by auto
from less(1)[OF this refl rR] obtain qs
where IH:r = prod-list qs ( }\forallq\in\mathrm{ set qs. set (coeffs q) }\subseteq\mathbb{R}\wedge\mathrm{ degree q < 2)
by auto
from IH(1) have p: p= prod-list (f\#qs) unfolding p by auto
with IH(2) fR degf show ?thesis
by (intro exI[of-f \# qs], auto)
qed
qed
qed
lemma real-degree-2-factorization-exists: fixes p :: real poly
shows \exists qs. p= prod-list qs ^(\forallq\in set qs. degree q}\leq\mp@code{2}
proof -
let ?cp = map-poly complex-of-real
let ?rp = map-poly Re
let ? }p=?cp
have set (coeffs ?p) \subseteq\mathbb{R}}\mathrm{ by auto
from real-degree-2-factorization-exists-complex[OF this]
obtain qs where p: ?p = prod-list qs and
qs: \bigwedgeq. q\in set qs \Longrightarrow set (coeffs q)\subseteq\mathbb{R}\wedge degree q}\leq2\mathrm{ 2 by auto
have p:p=?rp (prod-list qs) unfolding arg-cong[OF p, of ?rp, symmetric]
by (subst map-poly-map-poly, force, rule sym, rule map-poly-idI, auto)
from qs have \exists rs. prod-list qs = ?cp(prod-list rs)}\wedge(\forallr\in set rs. degree r\leq
2)

```
```

proof (induct $q s$ )
case Nil
show ?case by (auto intro!: exI[of - Nil])
next
case (Cons q qs)
then obtain rs where $q s$ : prod-list $q s=? c p$ (prod-list rs)
and $r s: \bigwedge q . q \in$ set $r s \Longrightarrow$ degree $q \leq 2$ by force +
from $\operatorname{Cons}(2)[o f q]$ have $q$ : set $($ coeffs $q) \subseteq \mathbb{R}$ and $d q$ : degree $q \leq 2$ by auto
define $r$ where $r=$ ? $r p q$
have $q: q=$ ? $c p r$ unfolding $r$-def
by (subst map-poly-map-poly, force, rule sym, rule map-poly-idI, insert $q$,
auto)
have $d r$ : degree $r \leq 2$ using $d q$ unfolding $q$ by (simp add: degree-map-poly)
show ?case
by (rule exI[of - r \#rs], unfold prod-list.Cons qs $q$, insert dr rs, auto simp:
hom-distribs)
qed
then obtain $r s$ where $i d$ : prod-list $q s=? c p$ (prod-list rs) and deg: $\forall r \in$ set
$r$ s. degree $r \leq 2$ by auto
show ?thesis unfolding $p$ id
by (intro exI, rule conjI[OF-deg], subst map-poly-map-poly, force, rule map-poly-idI,
auto)
qed
lemma odd-degree-imp-real-root: assumes odd (degree $p$ )
shows $\exists$ x. poly $p x=(0$ :: real $)$
proof -
from real-degree-2-factorization-exists[of $p]$ obtain $q s$ where
$i d: p=$ prod-list $q s$ and $q s: \bigwedge q . q \in$ set $q s \Longrightarrow$ degree $q \leq 2$ by auto
show ?thesis using assms $q s$ unfolding id
proof (induct qs)
case (Cons q qs)
from Cons(3)[of $q$ ] have $d q$ : degree $q \leq 2$ by auto
show ?case
proof (cases degree $q=1$ )
case True
from roots $1[O F$ this] show ?thesis by auto
next
case False
with $d q$ have deg: degree $q=0 \vee$ degree $q=2$ by arith
from Cons(2) have $q *$ prod-list $q s \neq 0$ by fastforce
hence $q \neq 0$ prod-list $q s \neq 0$ by auto
from degree-mult-eq[OF this]
have degree (prod-list $(q \# q s))=$ degree $q+$ degree (prod-list qs) by simp
from Cons(2)[unfolded this] deg have odd (degree (prod-list qs)) by auto
from Cons(1)[OF this Cons(3)] obtain $x$ where poly (prod-list qs) $x=0$
by auto
thus ?thesis by auto

```
```

    qed
    qed simp
    qed
end

```

\subsection*{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 }=>\mp@subsup{}{}{\prime}'
where [simp, code-abbrev]: gcds xs = gcd-list xs
lemma [code]:
gcds xs = fold gcd xs 0
by (simp add: Gcd-fin.set-eq-fold)
definition lcms :: 'a::semiring-gcd list }=>\mp@subsup{}{}{\prime}'
where [simp, code-abbrev]:lcms xs = lcm-list xs
lemma [code]:
lcms xs = fold lcm xs 1
by (simp add: Lcm-fin.set-eq-fold)
lemma in-reals-code [code-unfold]:
x\in\mathbb{R}\longleftrightarrow Im x=0
by (fact complex-is-Real-iff)
definition is-norm-1 :: complex }=>\mathrm{ bool where
is-norm-1 z=((Rez)}\mp@subsup{)}{}{2}+(\operatorname{Im}z\mp@subsup{)}{}{2}=1
lemma is-norm-1[simp]: is-norm-1 x = (norm x = 1)
unfolding is-norm-1-def norm-complex-def by simp
definition is-norm-le-1 :: complex }=>\mathrm{ bool where
is-norm-le-1 z=((Rez)2}+(\operatorname{Im}z\mp@subsup{)}{}{2}\leq1

```
```

lemma is-norm-le-1[simp]: is-norm-le-1 x = (norm x < 1)
unfolding is-norm-le-1-def norm-complex-def by simp
instantiation complex :: finite-UNIV
begin
definition finite-UNIV = Phantom(complex) False
instance
by (intro-classes, unfold finite-UNIV-complex-def, simp add: infinite-UNIV-char-0)
end
instantiation complex :: compare
begin
definition compare-complex :: complex }=>\mathrm{ complex }=>\mathrm{ order where
compare-complex x y = compare (Re x, Im x) (Re y, Im y)
instance
proof (intro-classes, unfold-locales; unfold compare-complex-def)
fix x y z :: complex
let ?c = compare :: (real }\times\mathrm{ real) comparator
interpret comparator ?c by (rule comparator-compare)
show invert-order (?c (Re x, Im x) (Re y, Im y)) = ?c (Re y, Im y) (Re x, Im
x)
by (rule sym)
{
assume ?c (Re x, Im x) (Re y, Im y) = Lt
?c}(\operatorname{Re}y,\operatorname{Im}y)(Rez,\operatorname{Im}z)=L
thus ?c (Re x, Im x) (Re z, Im z)=Lt
by (rule comp-trans)
}
{
assume ?c (Re x, Im x) (Re y, Im y)=Eq
from weak-eq[OF this] show }x=y\mathrm{ unfolding complex-eq-iff by auto
}
qed
end
derive (eq) ceq complex real
derive (compare) ccompare complex
derive (compare) ccompare real
derive (dlist) set-impl complex real
end

```

\section*{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 \(a \equiv\) Interval a a
abbreviation of-int-interval \(::\) int \(\Rightarrow{ }^{\prime} a\) :: ring- 1 interval where
of-int-interval \(x \equiv\) to-interval (of-int \(x\) )

\subsection*{14.1 Syntactic Class Instantiations}
instantiation interval :: (zero) zero begin
definition zero-interval where \(0 \equiv\) Interval 00
instance..
end
instantiation interval :: (one) one begin
definition \(1=\) Interval 11
instance..
end
instantiation interval :: (plus) plus begin
fun plus-interval where Interval lx \(u x+\) Interval ly \(u y=\) Interval \((l x+l y)(u x\) \(+u y\) )
instance..
end
instantiation interval :: (uminus) uminus begin
fun uminus-interval where - Interval \(l u=\) Interval \((-u)(-l)\)
instance..
end
instantiation interval :: (minus) minus begin
fun minus-interval where Interval lx \(u x-\) Interval ly uy \(=\) Interval \((l x-u y)\)
( \(u x-l y\) )
instance..
end
instantiation interval \(::(\{\) ord,times \(\})\) times begin
fun times-interval where
Interval lx \(u x *\) Interval ly \(u y=\)
```

        \((l e t x 1=l x * l y ; x 2=l x * u y ; x 3=u x * l y ; x 4=u x * u y\)
        in Interval \((\min x 1(\min x 2(\min x 3 x 4)))(\max x 1(\max x 2(\max x 3 x 4))))\)
    instance..
    end
instantiation interval :: (\{ord,times,inverse $\}$ ) inverse begin
fun inverse-interval where
inverse ( Interval lu) = Interval (inverse $u$ ) (inverse $l$ )
definition divide-interval :: 'a interval $\Rightarrow$ - where
divide-interval $X Y=X *($ inverse $Y)$
instance..
end

```

\subsection*{14.2 Class Instantiations}
```

instance interval :: (semigroup-add) semigroup-add
proof
fix abc :: 'a interval
show }a+b+c=a+(b+c) by (cases a, cases b, cases c, auto simp: ac-simps
qed
instance interval :: (monoid-add) monoid-add
proof
fix a :: 'a interval
show 0 + a = a by (cases a, auto simp: zero-interval-def)
show }a+0=a\mathrm{ by (cases a, auto simp: zero-interval-def)
qed
instance interval :: (ab-semigroup-add) ab-semigroup-add
proof
fix a b :: 'a interval
show }a+b=b+a\mathrm{ by (cases }a\mathrm{ , cases b, auto simp: ac-simps)
qed
instance interval :: (comm-monoid-add) comm-monoid-add by (intro-classes, auto)

```

Intervals do not form an additive group, but satisfy some properties.
lemma interval-uminus-zero[simp]:
shows -(0 :: 'a :: group-add interval \()=0\)
by (simp add: zero-interval-def)
lemma interval-diff-zero[simp]:
fixes \(a\) :: ' \(a\) :: cancel-comm-monoid-add interval
shows \(a-0=a\) by (cases \(a\), simp add: zero-interval-def)
Without type invariant, intervals do not form a multiplicative monoid, but satisfy some properties.
instance interval :: (\{linorder,mult-zero \(\}\) ) mult-zero
proof
fix \(a\) :: 'a interval
show \(a * 0=00 * a=0\) by (atomize(full), cases a, auto simp: zero-interval-def) qed

\subsection*{14.3 Membership}
fun in-interval :: ' \(a\) :: order \(\Rightarrow\) 'a interval \(\Rightarrow\) bool \(\left(\left(-/ \epsilon_{i}-\right)[51,51] 50\right)\) where \(y \in_{i}\) Interval \(l x u x=(l x \leq y \wedge y \leq u x)\)
lemma in-interval-to-interval[intro!]: \(a \in_{i}\) to-interval \(a\)
by (auto simp: to-interval-def)
lemma plus-in-interval:
fixes \(x y\) :: 'a :: ordered-comm-monoid-add
shows \(x \in_{i} X \Longrightarrow y \in_{i} Y \Longrightarrow x+y \in_{i} X+Y\)
by (cases \(X\), cases \(Y\), auto dest:add-mono)
lemma uminus-in-interval:
fixes \(x::\) ' \(a\) :: ordered-ab-group-add
shows \(x \in_{i} X \Longrightarrow-x \in_{i}-X\)
by (cases \(X\), auto)
lemma minus-in-interval:
fixes \(x\) y :: ' \(a\) :: ordered-ab-group-add
shows \(x \in_{i} X \Longrightarrow y \in_{i} Y \Longrightarrow x-y \in_{i} X-Y\)
by (cases \(X\), cases \(Y\), auto dest:diff-mono)
lemma times-in-interval:
fixes \(x\) : :: ' \(a\) :: linordered-ring
assumes \(x \in_{i} X y \in_{i} Y\)
shows \(x * y \in_{i} X * Y\)
proof -
obtain \(X 1 X 2\) where \(X\) :Interval \(X 1 X 2=X\) by (cases \(X\),auto \()\)
obtain Y1 Y2 where \(Y\) : Interval Y1 Y2 \(=Y\) by (cases \(Y\),auto)
from assms \(X Y\) have assms: \(X 1 \leq x x \leq X 2 Y 1 \leq y y \leq Y 2\) by auto
have \((X 1 * Y 1 \leq x * y \vee X 1 * Y 2 \leq x * y \vee X 2 * Y 1 \leq x * y \vee X 2 * Y 2 \leq\) \(x * y) \wedge\)
\((X 1 * Y 1 \geq x * y \vee X 1 * Y 2 \geq x * y \vee X 2 * Y 1 \geq x * y \vee X 2 * Y 2 \geq\) \(x * y)\)
proof (cases x 0 ::'a rule: linorder-cases)
case \(x 0\) : less
show ?thesis
proof (cases \(y<0\) )
case \(y 0\) : True
from \(y 0 x 0\) assms have \(x * y \leq X 1 * y\) by (intro mult-right-mono-neg, auto) also from \(x 0 y 0\) assms have \(X 1 * y \leq X 1 * Y 1\) by (intro mult-left-mono-neg, auto)
finally have \(1: x * y \leq X 1 * Y 1\).
show ?thesis proof (cases X2 \(\leq 0\) )
```

            case True
            with assms have X2 * Y2 \leqX2 * y by (auto intro: mult-left-mono-neg)
            also from assms y0 have \ldots\leqx*y by (auto intro: mult-right-mono-neg)
            finally have X2 * Y2 \leqx*y.
            with 1 show ?thesis by auto
        next
            case False
            with assms have X2 * Y1 \leqX2 * y by (auto intro: mult-left-mono)
            also from assms y0 have ... \leqx*y by (auto intro: mult-right-mono-neg)
            finally have X2 * Y1 \leqx*y.
            with 1 show ?thesis by auto
        qed
    next
        case False
        then have y0:y\geq0 by auto
    from x0 y0 assms have X1*Y2 \leqx* Y2 by (intro mult-right-mono, auto)
    also from y0 x0 assms have ...\leqx*y by (intro mult-left-mono-neg, auto)
    finally have 1:X1*Y2 \leqx*y.
    show ?thesis
    proof(cases X2 \leq0)
        case X2: True
        from assms y0 have x*y\leqX2 * y by (intro mult-right-mono)
        also from assms X2 have ... \leqX2 * Y1 by (auto intro: mult-left-mono-neg)
            finally have }x*y\leqX2*Y1
        with 1 show ?thesis by auto
        next
        case X2: False
        from assms y0 have }x*y\leqX2*y\mathrm{ by (intro mult-right-mono)
        also from assms X2 have ... \leqX2 * Y2 by (auto intro: mult-left-mono)
        finally have }x*y\leqX2*Y2
        with 1 show ?thesis by auto
        qed
    qed
    next
case [simp]: equal
with assms show ?thesis by (cases Y2 \leq 0, auto intro:mult-sign-intros)
next
case x0: greater
show ?thesis
proof (cases y<0)
case y0: True
from x0 y0 assms have X2 * Y1 \leq X2 * y by (intro mult-left-mono,auto)
also from y0 x0 assms have X2 *y\leqx*y by (intro mult-right-mono-neg,
auto)
finally have 1: X2 * Y1 \leqx*y.
show ?thesis
proof(cases Y2 \leq0)
case Y2: True
from x0 assms have x*y\leqx*Y2 by (auto intro: mult-left-mono)

```
```

        also from assms Y2 have ... \leqX1*Y2 by (auto intro: mult-right-mono-neg)
            finally have }x*y\leqX1*Y2
            with 1 show ?thesis by auto
        next
            case Y2: False
            from x0 assms have x*y\leqx*Y2 by (auto intro: mult-left-mono)
            also from assms Y2 have ... \leqX2 * Y2 by (auto intro: mult-right-mono)
            finally have }x*y\leqX2*Y2
            with 1 show ?thesis by auto
        qed
    next
        case y0: False
        from x0 y0 assms have }x*y\leqX2*y\mathrm{ by (intro mult-right-mono, auto)
        also from y0 x0 assms have ... \leqX2 * Y2 by (intro mult-left-mono, auto)
        finally have 1:x*y\leqX2 * Y2.
        show ?thesis
        proof(cases X1\leq0)
            case True
            with assms have X1*Y2 \leqX1*y by (auto intro: mult-left-mono-neg)
            also from assms y0 have ...\leqx*y by (auto intro: mult-right-mono)
            finally have X1*Y2 \leqx*y.
            with 1 show ?thesis by auto
        next
            case False
            with assms have X1*Y1\leqX1*y by (auto intro: mult-left-mono)
            also from assms y0 have ...\leqx*y by (auto intro: mult-right-mono)
            finally have X1*Y1\leqx*y.
            with 1 show ?thesis by auto
        qed
    qed
    qed
    hence min:min (X1 * Y1) (min (X1 * Y2) (min (X2 * Y1) (X2 * Y2))) \leqx
    * y
and max:x * y \leqmax (X1 * Y1) (max (X1 * Y2) (max (X2 * Y1) (X2 *
Y2)))
by (auto simp:min-le-iff-disj le-max-iff-disj)
show ?thesis using min max X Y by (auto simp: Let-def)
qed

```

\subsection*{14.4 Convergence}
definition interval-tendsto \(::\left(n a t \Rightarrow{ }^{\prime} a::\right.\) topological-space interval) \(\Rightarrow^{\prime} a \Rightarrow\) bool (infixr \(\longrightarrow{ }_{i} 55\) ) where
\((X \longrightarrow x) \wedge((\) interval.lower \(\circ X) \longrightarrow((\) interval. C\() \longrightarrow\) x)
lemma interval-tendstoI[intro]:
assumes (interval.upper \(\circ X) \longrightarrow x\) and (interval.lower \(\circ X) \longrightarrow x\)
shows \(X \longrightarrow i x\)
using assms by (auto simp:interval-tendsto-def)
lemma const-interval-tendsto: \((\lambda i\). to-interval \(a) \longrightarrow{ }_{i} a\)
by (auto simp: o-def to-interval-def)
lemma interval-tendsto- \(0:(\lambda i .0) \longrightarrow{ }_{i} 0\)
by (auto simp: o-def zero-interval-def)
lemma plus-interval-tendsto:
fixes \(x\) y :: ' \(a\) :: topological-monoid-add
assumes \(X \longrightarrow \longrightarrow_{i} x Y \longrightarrow{ }_{i} y\)
shows \((\lambda i . X i+Y i) \longrightarrow i\)
proof -
have \(*: X i+Y i=\) Interval (interval.lower \((X i)+\operatorname{interval.lower~}(Y i))\)
(interval.upper ( \(X_{i}\) ) + interval.upper \((Y i)\) ) for \(i\) by (cases \(X i\); cases \(Y i\), auto)
from assms show ?thesis unfolding \(*\) interval-tendsto-def o-def by (auto intro: tendsto-intros)
qed
lemma uminus-interval-tendsto:
fixes \(x::\) ' \(a\) :: topological-group-add
assumes \(X \longrightarrow i x\)
shows \((\lambda i .-X i) \longrightarrow i-x\)
proof-
have \(*\) : \(-X i=\) Interval ( - interval.upper \((X i))(-\operatorname{interval.lower~}(X i))\) for \(i\) by (cases \(X\) i, auto)
from assms show ?thesis unfolding o-def \(*\) interval-tendsto-def by (auto intro: tendsto-intros)
qed
lemma minus-interval-tendsto:
fixes \(x y::\) ' \(a\) :: topological-group-add
assumes \(X \longrightarrow{ }_{i} x Y \longrightarrow{ }_{i} y\)
shows \((\lambda i . X i-Y i) \longrightarrow \longrightarrow_{i} x-y\)
proof -
have *: X \(i-Y i=\) Interval (interval.lower \(\binom{X}{i}-\operatorname{interval.upper}\binom{Y}{i}\) (interval.upper ( \(X_{i}\) ) - interval.lower \(\left(Y_{i}\right)\) ) for \(i\)
by (cases \(X i\); cases \(Y i\), auto)
from assms show ?thesis unfolding o-def * interval-tendsto-def by (auto intro: tendsto-intros)
qed
lemma times-interval-tendsto:
fixes \(x\) y :: ' \(a\) :: \{linorder-topology, real-normed-algebra \(\}\)
assumes \(X \longrightarrow i{ }_{i} x Y \longrightarrow i y\)
shows \((\lambda i . X i * Y i) \longrightarrow{ }_{i} x * y\)
proof -
have \(*\) : (interval.lower \((X i * Y i))=(\)
```

    let lx = (interval.lower (X i));ux=(interval.upper ( 
            ly = (interval.lower (Y i));uy=(interval.upper (Y i));
            x1 = lx*ly; x2 =lx*uy; x3 = ux*ly; x4 = ux *uy in
    (min x1 (min x2 (min x3 x4)))) (interval.upper (Xi*Yi)) =(
    let lx = (interval.lower (X i)); ux = (interval.upper (X i));
            ly = (interval.lower (Yi));uy=(interval.upper (Y i));
        x1 =lx*ly; x2 = lx*uy;x3 = ux*ly;x4 = ux*uy in
        (\operatorname{max x1 (max x2 (max x3 x4)))) for i}
    by (cases X i; cases Y i, auto simp: Let-def)+
    have (\lambdai. (interval.lower (Xi*Yi)))\longrightarrow\operatorname{min}(x*y)(\operatorname{min}(x*y)(\operatorname{min}(x
    * y) (x *y)))
using assms unfolding interval-tendsto-def * Let-def o-def
by (intro tendsto-min tendsto-intros, auto)
moreover
have (\lambdai. (interval.upper (Xi*Yi)))\longrightarrow\operatorname{max (x*y)(max (x*y) (max}
(x*y) (x*y)))
using assms unfolding interval-tendsto-def * Let-def o-def
by (intro tendsto-max tendsto-intros, auto)
ultimately show ?thesis unfolding interval-tendsto-def o-def by auto
qed
lemma interval-tendsto-neq:
fixes a b :: real
assumes (\lambda i.fi)\longrightarrow}\mp@subsup{\longrightarrow}{i}{}a\mathrm{ and }a\not=
shows \exists n.\negb \inifn
proof -
let ?d = norm (b-a)/2
from assms have d:?d>0 by auto
from assms(1)[unfolded interval-tendsto-def]
have cvg:(interval.lower of)\longrightarrowa (interval.upper of)\longrightarrowa}\longrightarrowa\mathrm{ by auto
from LIMSEQ-D[OF cvg(1) d] obtain n1 where
n1: \ n. n \geqn1\Longrightarrow norm ((interval.lower \circf) n - a)<?d by auto
from LIMSEQ-D[OF cvg(2) d] obtain n2 where
n2: \ n. n \geq n2 \Longrightarrow norm ((interval.upper \circf) n - a)<?d by auto
define n where n= max n1 n2
from n1[of n] n2[of n] have bnd:
norm ((interval.lower ○f) n-a)<?d
norm ((interval.upper }\circf)n-a)<?
unfolding n-def by auto
show ?thesis by (rule exI[of-n], insert bnd, cases f n, auto,argo)
qed

```

\subsection*{14.5 Complex Intervals}
datatype complex-interval \(=\) Complex-Interval (Re-interval: real interval) (Im-interval: real interval)
definition in-complex-interval \(::\) complex \(\Rightarrow\) complex-interval \(\Rightarrow\) bool \(\left(\left(-/ \epsilon_{c}-\right)\right.\) [51, 51] 50) where
```

    y }\mp@subsup{\epsilon}{c}{}x\equiv(\mathrm{ case x of Complex-Interval r i m Re y }\mp@subsup{\epsilon}{i}{}r\wedge\operatorname{Im}y\mp@subsup{\in}{i}{}i
    instantiation complex-interval :: comm-monoid-add begin
definition 0 \equivComplex-Interval 0 0
fun plus-complex-interval :: complex-interval }=>\mathrm{ complex-interval }=>\mathrm{ complex-interval
where
Complex-Interval rx ix + Complex-Interval ry iy = Complex-Interval (rx + ry)
(ix+iy)
instance
proof
fix a b c :: complex-interval
show }a+b+c=a+(b+c) by (cases a, cases b, cases c, simp add: ac-simps
show }a+b=b+a\mathrm{ by (cases a, cases b, simp add: ac-simps)
show 0 + a=a by (cases a, simp add: ac-simps zero-complex-interval-def)
qed
end
lemma plus-complex-interval: }x\mp@subsup{\epsilon}{c}{}X\Longrightarrowy\mp@subsup{\epsilon}{c}{}Y\Longrightarrowx+y\mp@subsup{\epsilon}{c}{}X+
unfolding in-complex-interval-def using plus-in-interval by (cases X, cases Y,
auto)
definition of-int-complex-interval :: int }=>\mathrm{ complex-interval where
of-int-complex-interval x = Complex-Interval (of-int-interval x) 0
lemma of-int-complex-interval-0[simp]: of-int-complex-interval 0 = 0
by (simp add: of-int-complex-interval-def zero-complex-interval-def to-interval-def
zero-interval-def)
lemma of-int-complex-interval: of-int i }\mp@subsup{\epsilon}{c}{}\mathrm{ of-int-complex-interval i
unfolding in-complex-interval-def of-int-complex-interval-def
by (auto simp: zero-complex-interval-def zero-interval-def)
instantiation complex-interval :: mult-zero begin
fun times-complex-interval where
Complex-Interval rx ix * Complex-Interval ry iy =
Complex-Interval (rx*ry-ix*iy) (rx*iy+ix*ry)
instance
proof
fix a :: complex-interval
show 0*a=0a*0=0 by (atomize(full), cases a,auto simp:zero-complex-interval-def)
qed
end
instantiation complex-interval :: minus begin

```
```

    fun minus-complex-interval where
    Complex-Interval R I - Complex-Interval R' I'=Complex-Interval ( }R-\mp@subsup{R}{}{\prime}\mathrm{ )
    (I-I')
instance..
end
lemma times-complex-interval: }x\mp@subsup{\in}{c}{}X\Longrightarrowy\mp@subsup{\epsilon}{c}{}Y\Longrightarrowx*y\mp@subsup{\epsilon}{c}{}X*
unfolding in-complex-interval-def
by (cases X, cases Y, auto intro: times-in-interval minus-in-interval plus-in-interval)
definition ipoly-complex-interval :: int poly }=>\mathrm{ complex-interval }=>\mathrm{ complex-interval
where
ipoly-complex-interval p x = fold-coeffs (\lambdaa b. of-int-complex-interval a + x*b)
p 0
lemma ipoly-complex-interval-0[simp]:
ipoly-complex-interval 0 x = 0
by (auto simp: ipoly-complex-interval-def)
lemma ipoly-complex-interval-pCons[simp]:
ipoly-complex-interval (pCons a p) x =of-int-complex-interval }a+x*(ipoly-complex-interval
p x)
by (cases p=0; cases a = 0, auto simp: ipoly-complex-interval-def)
lemma ipoly-complex-interval: assumes x: x \inc
shows ipoly p x \inc ipoly-complex-interval p X
proof -
define xs where xs= coeffs p
have 0: in-complex-interval 0 0 (is in-complex-interval ?Z ?z)
unfolding in-complex-interval-def zero-complex-interval-def zero-interval-def
by auto
define Z where Z =? Z
define }z\mathrm{ where }z=?
from 0 have 0: in-complex-interval Z z unfolding Z-def z-def by auto
note x = times-complex-interval[OF x]
show ?thesis
unfolding poly-map-poly-code ipoly-complex-interval-def fold-coeffs-def
xs-def[symmetric] Z-def[symmetric] z-def[symmetric] using 0
by (induct xs arbitrary: Z z, auto intro!: plus-complex-interval of-int-complex-interval
x)
qed
definition complex-interval-tendsto (infix }\longrightarrow\mp@subsup{\longrightarrow}{c}{}55)\mathrm{ where

```

```

Im c)

```
lemma complex-interval-tendstoI[intro!]
(Re-interval \(\circ C) \longrightarrow{ }_{i}\) Re \(c \Longrightarrow\) (Im-interval \(\left.\circ C\right) \longrightarrow{ }_{i}\) Im \(c \Longrightarrow C\)
\(\longrightarrow c c\)
by (simp add: complex-interval-tendsto-def)
lemma of-int-complex-interval-tendsto: ( \(\lambda\) i. of-int-complex-interval \(n\) ) \(\longrightarrow{ }_{c}\) of-int \(n\)
by (auto simp: o-def of-int-complex-interval-def intro!:const-interval-tendsto in-terval-tendsto-0)
lemma Im-interval-plus: Im-interval \((A+B)=\) Im-interval \(A+\) Im-interval \(B\) by (cases \(A\); cases B, auto)
lemma Re-interval-plus: Re-interval \((A+B)=\) Re-interval \(A+\) Re-interval \(B\) by (cases A; cases B, auto)
lemma Im-interval-minus: Im-interval \((A-B)=\) Im-interval \(A-\) Im-interval \(B\)
by (cases \(A\); cases \(B\), auto)
lemma Re-interval-minus: Re-interval \((A-B)=\) Re-interval \(A-R e\)-interval \(B\) by (cases A; cases B, auto)
lemma Re-interval-times: Re-interval \((A * B)=\) Re-interval \(A * R e\)-interval \(B-\) Im-interval \(A *\) Im-interval \(B\)
by (cases \(A\); cases B, auto)
lemma Im-interval-times: Im-interval \((A * B)=\) Re-interval \(A * \operatorname{Im}\)-interval \(B+\) Im-interval \(A * R e\)-interval \(B\)
by (cases \(A\); cases \(B\), auto)
lemma plus-complex-interval-tendsto:
\(A \longrightarrow{ }_{c} a \Longrightarrow B \longrightarrow{ }_{c} b \Longrightarrow(\lambda i . A i+B i) \longrightarrow{ }_{c} a+b\)
unfolding complex-interval-tendsto-def
by (auto intro!: plus-interval-tendsto simp: o-def Re-interval-plus Im-interval-plus)
lemma minus-complex-interval-tendsto:
\(A \longrightarrow c{ }_{c} a \Longrightarrow B \longrightarrow_{c} b \Longrightarrow(\lambda i . A i-B i) \longrightarrow c\)
unfolding complex-interval-tendsto-def
by (auto intro!: minus-interval-tendsto simp: o-def Re-interval-minus Im-interval-minus)
lemma times-complex-interval-tendsto:
\(A \longrightarrow{ }_{c} a \longrightarrow B \longrightarrow{ }_{c} b \Longrightarrow(\lambda i . A i * B i) \longrightarrow c\)
unfolding complex-interval-tendsto-def
by (auto intro!: minus-interval-tendsto times-interval-tendsto plus-interval-tendsto
simp: o-def Re-interval-times Im-interval-times)
lemma ipoly-complex-interval-tendsto:
```

    assumes C \longrightarrow
    shows (\lambdai. ipoly-complex-interval p (Ci)) \longrightarrow}\longrightarrow\mp@subsup{C}{c}{}\mathrm{ ipoly p c
    proof(induct p)
case 0
show ?case by (auto simp: o-def zero-complex-interval-def zero-interval-def com-
plex-interval-tendsto-def)
next
case (pCons a p)
show ?case
apply (unfold ipoly-complex-interval-pCons of-int-hom.map-poly-pCons-hom
poly-pCons)
apply (intro plus-complex-interval-tendsto times-complex-interval-tendsto assms
pCons of-int-complex-interval-tendsto)
done
qed
lemma complex-interval-tendsto-neq: assumes (\lambdai.fi) \longrightarrow\longrightarrow}\mp@subsup{}{c}{}
and }a\not=
shows \exists n.\negb\in\epsilon}f
proof -
from assms(1)[unfolded complex-interval-tendsto-def o-def]
have cvg:( }\lambdax.\mathrm{ Re-interval (f x)) }\longrightarrow\mp@subsup{\longrightarrow}{i}{}\mathrm{ Re a ( }\lambdax\mathrm{ . Im-interval (f x)) }\longrightarrow\mp@subsup{\longrightarrow}{i}{
Im a by auto
from assms(2) have Re a\not=Re b\vee Im a\not= Im b
using complex.expand by blast
thus ?thesis
proof
assume Re a\not=Re b
from interval-tendsto-neq[OF cvg(1) this] show ?thesis
unfolding in-complex-interval-def by (metis (no-types, lifting) complex-interval.case-eq-if)
next
assume Im a\not= Im b
from interval-tendsto-neq[OF cvg(2) this] show ?thesis
unfolding in-complex-interval-def by (metis (no-types,lifting) complex-interval.case-eq-if)
qed
qed
end

```

\section*{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

```

\subsection*{15.1 Complex Roots}
hide-const (open) UnivPoly.coeff
hide-const (open) Module.smult
hide-const (open) Coset.order
abbreviation complex-of-int-poly :: int poly \(\Rightarrow\) complex poly where complex-of-int-poly \(\equiv\) map-poly of-int
abbreviation complex-of-rat-poly :: rat poly \(\Rightarrow\) complex poly where complex-of-rat-poly \(\equiv\) map-poly of-rat
lemma poly-complex-to-real: (poly (complex-of-int-poly \(p)(\) complex-of-real \(x)=0)\)
\(=(\) poly \((\) real-of-int-poly \(p) x=0)\)
proof -
have \(i d\) : of-int \(=\) complex-of-real o real-of-int by auto
interpret cr: semiring-hom complex-of-real by (unfold-locales, auto)
show ?thesis unfolding id
by (subst map-poly-map-poly[symmetric], force+)
qed
```

lemma represents-cnj: assumes $p$ represents $x$ shows $p$ represents ( $c n j x$ )
proof -
from assms have $p: p \neq 0$ and ipoly $p x=0$ by auto
hence rt: poly (complex-of-int-poly $p$ ) $x=0$ by auto
have poly (complex-of-int-poly p) $($ cnj $x)=0$
by (rule complex-conjugate-root[OF - rt], subst coeffs-map-poly, auto)
with $p$ show ?thesis by auto
qed
definition poly-2i :: int poly where
poly-2i $\equiv[: 4,0,1:]$
lemma represents-2i: poly-2i represents (2 $* \mathrm{i}$ )
unfolding represents-def poly-2i-def by simp
definition root-poly-Re :: int poly $\Rightarrow$ int poly where
root-poly-Re $p=c f-$ 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 (\lambda(fi,i).fi* 2 ^ i) (zip fs [0..<k]))))
proof -
have [simp]: quotient-of (1 / 2) = (1,2) by eval
show ?thesis unfolding root-poly-Re-def poly-mult-rat-def poly-mult-rat-main-def
Let-def by simp
qed
definition root-poly-Im :: int poly }=>\mathrm{ int poly list where
root-poly-Im p = (let fs = factors-of-int-poly
(poly-add p (poly-uminus p))
in remdups ((if (\existsf\in set fs.coeff f 0 = 0) then [[:0,1:]] else [])) @
[ cf-pos-poly (poly-div f poly-2i).f}\leftarrowfs, coeff f 0 = 0])
lemma represents-root-poly:
assumes ipoly p x=0 and p: p\not=0
shows (root-poly-Re p) represents (Re x)
and \existsq\in set (root-poly-Im p).q represents (Im x)
proof -
let ?Rep = root-poly-Re p
let ?Imp = root-poly-Im p
from assms have ap: p represents x by auto
from represents-cnj[OF this] have apc: p represents (cnj x).
from represents-mult-rat[OF - represents-add[OF ap apc], of inverse 2]
have ?Rep represents (1 / 2 * (x+cnj x)) unfolding root-poly-Re-def Let-def
by (auto simp: hom-distribs)
also have 1 / 2*(x+cnj x) =of-real (Re x)
by (simp add: complex-add-cnj)
finally have Rep:?Rep \not=0 and rt: ipoly ?Rep (complex-of-real (Re x)) = 0
unfolding represents-def by auto
from rt[unfolded poly-complex-to-real]
have ipoly ?Rep (Re x) = 0 .
with Rep show ?Rep represents (Rex) by auto
let ?q = poly-add p (poly-uminus p)
from represents-add[OF ap, of poly-uminus p - cnj x] represents-uminus[OF apc]
have apq: ?q represents (x-cnj x) by auto
from factors-int-poly-represents[OF this] obtain pi where pi: pi\in set (factors-of-int-poly
?q)
and appi: pi represents (x-cnj x) and irr-pi: irreducible pi by auto
have id: inverse (2* i) * (x-cnj x) =of-real (Im x)
apply (cases x) by (simp add: complex-split imaginary-unit.ctr legacy-Complex-simps)
from represents-2i have 12: poly-2i represents (2 * i) by simp
have \exists qi \in set ?Imp. qi represents (inverse (2 * i) * (x - cnj x))
proof (cases x - cnj x=0)
case False
have poly poly-2i 0}\not=0\mathrm{ unfolding poly-2i-def by auto

```
from represents-div[OF appi 12 this]
represents-irr-non- \(0[\) OF irr-pi appi False, unfolded poly-0-coeff-0] pi
show ?thesis unfolding root-poly-Im-def Let-def by (auto intro: bexI[of -cf-pos-poly (poly-div pi poly-2i)])
next
case True
hence \(i d 2\) : \(\operatorname{Im} x=0\) by (simp add: complex-eq-iff)
from appi[unfolded True represents-def] have coeff pi \(0=0\) by (cases pi, auto)
with pi have mem: \([: 0,1:] \in\) set ?Imp unfolding root-poly-Im-def Let-def by auto
have [:0,1:] represents (complex-of-real \((\operatorname{Im} x)\) ) unfolding id2 represents-def by \(\operatorname{simp}\)
with mem show ?thesis unfolding id by auto
qed
then obtain \(q i\) where \(q i: q i \in\) set ?Imp \(q i \neq 0\) and \(r t\) : ipoly \(q i\) (complex-of-real \((\operatorname{Im} x))=0\)
unfolding id represents-def by auto
from qi rt[unfolded poly-complex-to-real]
show \(\exists q i \in\) set ? Imp. qi represents (Im \(x\) ) by auto
qed
definition complex-poly :: int poly \(\Rightarrow\) int poly \(\Rightarrow\) int poly list where
complex-poly re im \(=(\) let \(i=[: 1,0,1:]\)
in factors-of-int-poly (poly-add re (poly-mult im i)))
lemma complex-poly: assumes re: re represents (Re x)
and \(i m\) : im represents \((\operatorname{Im} x)\)
shows \(\exists f \in\) set (complex-poly re im). f represents \(x \wedge f . f \in\) set (complex-poly
re im) \(\Longrightarrow\) poly-cond \(f\)
proof -
let \(? p=\) poly-add re (poly-mult im [:1, 0, \(1:]\) )
from re have re: re represents complex-of-real (Re x) by simp
from \(i m\) have \(i m\) : im represents complex-of-real ( \(\operatorname{Im} x\) ) by simp
have \([: 1,0,1:]\) represents i by auto
from represents-add[OF re represents-mult[OF im this]]
have ?p represents of-real ( \(\operatorname{Re} x)+\) complex-of-real \((\operatorname{Im} x) *\) i by simp
also have of-real \((\operatorname{Re} x)+\) complex-of-real \((\operatorname{Im} x) * \mathrm{i}=x\)
by (metis complex-eq mult.commute)
finally have \(p\) : ?p represents \(x\) by auto
have factors-of-int-poly ? \(p=\) complex-poly re im
unfolding complex-poly-def Let-def by simp
from factors-of-int-poly(1)[OF this] factors-of-int-poly(2)[OF this, of \(x] p\)
show \(\exists f \in\) set (complex-poly re im). \(f\) represents \(x \bigwedge f . f \in\) set (complex-poly re im\() \Longrightarrow\) poly-cond \(f\)
unfolding represents-def by auto
qed
lemma algebraic-complex-iff: algebraic \(x=(\operatorname{algebraic}(\operatorname{Re} x) \wedge\) algebraic \((\operatorname{Im} x))\)
```

proof
assume algebraic x
from this[unfolded algebraic-altdef-ipoly] obtain p where ipoly p x=0 p\not=0
by auto
from represents-root-poly[OF this] show algebraic (Re x) ^ algebraic (Im x)
unfolding represents-def algebraic-altdef-ipoly by auto
next
assume algebraic (Rex)^ algebraic (Im x)
from this[unfolded algebraic-altdef-ipoly] obtain re im where
re represents (Rex) im represents (Im x) by blast
from complex-poly[OF this] show algebraic x
unfolding represents-def algebraic-altdef-ipoly by auto
qed
definition algebraic-complex :: complex }=>\mathrm{ bool where
[simp]: algebraic-complex = algebraic

```
lemma algebraic-complex-code-unfold[code-unfold]: algebraic \(=\) algebraic-complex
by \(\operatorname{simp}\)
lemma algebraic-complex-code[code]:
    algebraic-complex \(x=(\) algebraic \((\operatorname{Re} x) \wedge\) algebraic \((\operatorname{Im} x))\)
    unfolding algebraic-complex-def algebraic-complex-iff ..

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 \(::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.\) bool \() \Rightarrow\) ' \(a\) list \(\Rightarrow\) ' \(a\) list where
remdups-gen eq [] = []
\(\mid\) remdups-gen eq \((x \# x s)=(\) if \((\exists y \in\) set \(x s\). eq \(x y)\) then remdups-gen eq xs else \(x\) \# remdups-gen eq \(x\) )
lemma real-of-3-remdups-equal-3[simp]: real-of-3'set (remdups-gen equal-3 xs) = real-of-3' set xs
by (induct \(x\) s, auto simp: equal-3)
lemma distinct-remdups-equal-3: distinct (map real-of-3 (remdups-gen equal-3 xs)) by (induct xs, auto, auto simp: equal-3)
lemma real-of-3-code [code]: real-of-3 \(x=\) real-of (Real-Alg-Quotient \(x)\)
by (transfer, auto)
definition real-parts-3 \(p=\) roots-of-3 (root-poly-Re \(p\) )
definition pos-imaginary-parts-3 \(p=\)
remdups-gen equal-3 (filter \((\lambda x\). sgn-3 \(x=1\) ) (concat (map roots-of-3 (root-poly-Im p))))
```

lemma real-parts-3: assumes p: p\not=0 and ipoly p x=0
shows Re x f real-of-3' set (real-parts-3 p)
unfolding real-parts-3-def using represents-root-poly(1)[OF assms(2,1)]
roots-of-3(1) unfolding represents-def by auto
lemma distinct-real-parts-3: distinct (map real-of-3 (real-parts-3 p))
unfolding real-parts-3-def using roots-of-3(2) .
lemma pos-imaginary-parts-3: assumes p: p\not=0 and ipoly p x=0 and Im x>
0
shows Im x f real-of-3' set (pos-imaginary-parts-3 p)
proof -
from represents-root-poly(2)[OF assms(2,1)] obtain q where
q:q\in set (root-poly-Im p) q represents Im x by auto
from roots-of-3(1)[of q] have Im x feal-of-3'set (roots-of-3 q) using q
unfolding represents-def by auto
then obtain i3 where i3: i3 \in set (roots-of-3 q) and id:Im x = real-of-3 i3
by auto
from <Im x > 0` have sgn ( Im x) = 1 by simp
hence sgn: sgn-3 i3 = 1 unfolding id by (metis of-rat-eq-1-iff sgn-3)
show ?thesis unfolding pos-imaginary-parts-3-def real-of-3-remdups-equal-3 id
using sgn i3 q(1) by auto
qed
lemma distinct-pos-imaginary-parts-3: distinct (map real-of-3 (pos-imaginary-parts-3
p))
unfolding pos-imaginary-parts-3-def by (rule distinct-remdups-equal-3)
lemma remdups-gen-subset: set (remdups-gen eq xs) \subseteq set xs
by (induct xs, auto)
lemma positive-pos-imaginary-parts-3: assumes x fet (pos-imaginary-parts-3
p)
shows 0 < real-of-3 x
proof -
from subsetD[OF remdups-gen-subset assms[unfolded pos-imaginary-parts-3-def]]
have sgn-3 x = 1 by auto
thus ?thesis using sgn-3[of x] by (simp add: sgn-1-pos)
qed
definition pair-to-complex ri \equivcase 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\mp@subsup{\in}{i}{}\mathrm{ get-itvl-2 }
    proof (cases x)
case (Irrational n plr)
with assms obtain plr where plr: plr = (p,l,r) by (cases plr,auto)
from assms Irrational plr have inv1: invariant-1 ( p,l,r)
and id: real-of-2 }x=\mathrm{ real-of-1 ( }p,l,r)\mathrm{ by auto
show ?thesis unfolding id using invariant-1D(1)[OF inv1] by (auto simp: plr
Irrational)
qed (insert assms, auto simp: Let-def)
lift-definition get-itvl-3 :: real-alg-3 => real interval is get-itvl-2 .
lemma get-itvl-3: real-of-3 }x\mp@subsup{\in}{i}{}\mathrm{ get-itvl-3 }
by (transfer, insert get-bounds-2, auto)
fun tighten-bounds-2 :: real-alg-2 }=>\mathrm{ real-alg-2 where
tighten-bounds-2 (Irrational n (p,l,r)) = (case tighten-poly-bounds plr (sgn (ipoly
pr))
of (l',r',-) => Irrational n ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})
| 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' }\mp@subsup{r}{}{\prime}\Longrightarrow\mp@subsup{r}{}{\prime}-\mp@subsup{l}{}{\prime}=(r-l)/
proof (atomize(full), cases x)
case (Irrational n plr)
show real-of-2 (tighten-bounds-2 x) = real-of-2 x ^
invariant-2 (tighten-bounds-2 x) ^
(get-itvl-2 x = Interval l r }
get-itvl-2 (tighten-bounds-2 x) = Interval l' r' \longrightarrow r' - l' = (r-l) / 2)
proof -
obtain p l r where plr: plr = ( p,l,r) by (cases plr,auto)
let ?tb = tighten-poly-bounds plr (sgn (ipoly p r))
obtain l' r'sr' where tb: ?tb = ( l', r',sr') by (cases ?tb, auto)
have id: tighten-bounds-2 x = Irrational n ( p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ unfolding Irrational plr}
using tb by auto
from inv[unfolded Irrational plr] have inv: invariant-1-2 ( }p,l,r
n= card {y. y\leq real-of-1 (p,l,r)^ ipoly p y=0} by auto
have rof:real-of-2 }x=\mathrm{ real-of-1 ( }p,l,r
real-of-2 (tighten-bounds-2 x) = real-of-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ using Irrational plr id by
auto
from inv have inv1: invariant-1 ( }p,l,r)\mathrm{ and poly-cond2 p by auto
hence rc: \exists!x. root-cond ( }p,l,r\mathrm{ ) x poly-cond2 p by auto
note t\mp@subsup{b}{}{\prime}}=\mathrm{ tighten-poly-bounds[OF tb rc refl]
have eq: real-of-1 ( }p,l,r)=\mathrm{ real-of-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ using tb' inv1
using invariant-1-sub-interval(2) by presburger
from inv1 tb' have invariant-1 ( }p,\mp@subsup{l}{}{\prime},\mp@subsup{r}{}{\prime})\mathrm{ by (metis invariant-1-sub-interval(1))

```
hence inv2: invariant-2 (tighten-bounds-2 x) unfolding id using inv eq by auto
thus ?thesis unfolding rof eq unfolding id unfolding Irrational plr
using \(t b^{\prime}(1-4)\) arg-cong \(\left[\right.\) OF \(t b^{\prime}(5)\), of real-of-rat \(]\) by (auto simp: hom-distribs) qed
qed (auto simp: Let-def)
lift-definition tighten-bounds-3 :: real-alg-3 \(\Rightarrow\) real-alg-3 is tighten-bounds-2 using tighten-bounds-2 by auto
lemma tighten-bounds-3:
real-of-3 (tighten-bounds-3 \(x\) ) \(=\) real-of- \(3 x\) get-itvl-3 \(x=\) Interval l \(r \Longrightarrow\)
get-itvl-3 (tighten-bounds-3 \(x\) ) \(=\) Interval \(l^{\prime} r^{\prime} \Longrightarrow r^{\prime}-l^{\prime}=(r-l) / 2\)
by (transfer, insert tighten-bounds-2, auto) +
partial-function (tailrec) filter-list-length
\(::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow\left({ }^{\prime} a \Rightarrow\right.\) bool \() \Rightarrow n a t \Rightarrow{ }^{\prime} a\) list \(\Rightarrow{ }^{\prime} a\) list where
[code]: filter-list-length f \(p n x s=(\) let \(y s=\) filter \(p x s\)
in if length ys \(=n\) then ys else
filter-list-length f \(p\) n (map fys))
lemma filter-list-length: assumes length (filter \(P\) xs \()=n\) and \(\bigwedge i x . x \in\) set \(x s \Longrightarrow P x \Longrightarrow p\left(\left(f^{\sim} i\right) x\right)\)
and \(\bigwedge x . x \in\) set \(x s \Longrightarrow \neg P x \Longrightarrow \exists i . \neg p\left(\left(f{ }^{\wedge} i\right) x\right)\)
and \(g: \wedge x . g(f x)=g x\)
and \(P: \bigwedge x . P(f x)=P x\)
shows map \(g\) (filter-list-length \(f p n x s)=\) map \(g(\) filter \(P x s)\)
proof -
from \(\operatorname{assms}(3)\) have \(\forall x . \exists i . x \in \operatorname{set} x s \longrightarrow \neg P x \longrightarrow \neg p((f \sim i) x)\) by auto
from choice \([\) OF this] obtain \(i\) where \(i: \bigwedge x . x \in\) set \(x s \Longrightarrow \neg P x \Longrightarrow \neg p((f\)
\(\sim(i x)) x\) )
by auto
define \(m\) where \(m=\) max-list (map \(i x s\) )
have \(m: \bigwedge x . x \in\) set \(x s \Longrightarrow \neg P x \Longrightarrow \exists i \leq m\). \(\neg p\left(\left(f^{\wedge} i\right) x\right)\)
using max-list [of - map ixs, folded m-def] \(i\) by auto
show ?thesis using assms(1-2) \(m\)
proof (induct \(m\) arbitrary: xs rule: less-induct)
case (less \(m x s\) )
define \(y s\) where \(y s=\) filter \(p x s\)
have \(x s\)-ys: filter \(P\) xs \(=\) filter \(P\) ys unfolding ys-def filter-filter by (rule filter-cong[OF refl], insert less(3)[of-0], auto)
have filter \((P \circ f)\) ys \(=\) filter \(P\) ys using \(P\) unfolding o-def by auto
hence id3: filter \(P(\operatorname{map} f y s)=\operatorname{map} f(\) filter \(P y s)\) unfolding filter-map by \(\operatorname{simp}\)
hence \(i d 2\) : map \(g(\) filter \(P(\) map \(f y s))=\) map \(g(\) filter \(P\) ys) by (simp add: \(g\) ) show ?case
proof (cases length ys \(=n\) )
```

    case True
    hence id: filter-list-length f p n xs=ys unfolding ys-def
    filter-list-length.simps[of--xs] Let-def by auto
    show ?thesis using True unfolding id xs-ys using less(2)
            by (metis filter-id-conv length-filter-less less-le xs-ys)
        next
    case False
    {
        assume m=0
        from less(4)[unfolded this] have Pp: x\in set xs \Longrightarrow\negPx\Longrightarrow \neg px for x
    by auto
with xs-ys False[folded less(2)] have False
by (metis (mono-tags, lifting) filter-True mem-Collect-eq set-filter ys-def)
} note m0 = this
then obtain M where mM:m=Suc M by (cases m,auto)
hence m:M<m by simp
from False have id: filter-list-length f p n xs = filter-list-length f p n (map f
ys)
unfolding ys-def filter-list-length.simps[of - - xs] Let-def by auto
show ?thesis unfolding id xs-ys id2[symmetric]
proof (rule less(1)[OF m])
fix y
assume y \in set (map f ys)
then obtain x where x:x\in set xs px and y:y=fx unfolding ys-def
by auto
{
assume \negPy
hence }\negP<br>mathrm{ unfolding y P.
from less(4)[OF x(1) this] obtain i where i:i\leqm and p:\negp((f~~
i) x) by auto
with x obtain j where ij: i=Suc j by (cases i,auto)
with i have j:j\leqM unfolding mM by auto
have }\negp((f~j) y) using p unfolding ij y funpow-Suc-right by sim
thus }\existsi\leqM.\negp((f~~i)y)\mathrm{ using j by auto
}
{
fix }
assume P y
hence P x unfolding y P .
from less(3)[OF x(1) this, of Suc i]
show p((f~i) y) unfolding y funpow-Suc-right by simp
}
next
show length (filter P (map fys)) = n unfolding id3 length-map using xs-ys
less(2) by auto
qed
qed
qed
qed

```
```

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 $p ;$
$n r=$ length rrts;
crts $=\operatorname{map}(\lambda r$. Complex r 0) rrts
in
if $n=n r$ then crts
else let $n r$-crts $=n-n r$ in if $n r-c r t s=2$ then
let $p p=$ real-of-int-poly $p$ div (prod-list $(\operatorname{map}(\lambda x .[:-x, 1:])$ rrts $)$ );
cpp $=$ map-poly ( $\lambda$ r. Complex r 0) pp
in crts @ croots2 cpp else
let
$n r$-pos-crts $=n r$-crts div 2;
rxs $=$ real-parts-3 $p$;
ixs $=$ pos-imaginary-parts-3 p;
$r t s=[(r x, i x) . r x<-r x s, i x<-i x s] ;$
$c r t s^{\prime}=$ map pair-to-complex
(filter-list-length (map-prod tighten-bounds-3 tighten-bounds-3)
( $\lambda(r, i) .0 \in_{c}$ ipoly-complex-interval $p$ (Complex-Interval (get-itvl-3 $r$ )
(get-itvl-3 i))) nr-pos-crts rts)
in crts @ (concat (map ( $\left.\lambda x .[x, \operatorname{cnj} x]) c r t s^{\prime}\right)$ )
definition complex-roots-of-int-poly-all :: int poly $\Rightarrow$ complex list where
complex-roots-of-int-poly-all $p=$ (let $n=$ degree $p$ in
if $n \geq 3$ 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 $x \in_{i}$ get-itvl-3 ((tighten-bounds-3
~n) $x$ )
proof (induct $n$ arbitrary: $x$ )
case 0
thus ?case using get-itvl-3[of $x$ ] by simp
next
case (Suc n x)
have id: (tighten-bounds-3 ~ $($ Suc $n)) x=($ tighten-bounds-3 ~n $n)($ tighten-bounds-3
x)
by (metis comp-apply funpow-Suc-right)
show ? case unfolding id tighten-bounds-3(1)[of x, symmetric] by (rule Suc)
qed
lemma sandwitch-real:
fixes $l r::$ nat $\Rightarrow$ real
assumes $l a: l \longrightarrow a$ and $r a: r \longrightarrow a$
and $l m: \bigwedge i . l i \leq m i$ and $m r: \bigwedge i . m i \leq r i$

```
```

shows m\longrightarrowa
proof (rule LIMSEQ-I)
fix e :: real
assume 0<e
hence e:0<e / 2 by simp
from LIMSEQ-D[OF la e] obtain n1 where n1: \ n. n\geqn1\Longrightarrow norm (l n
-a)<e/2 by auto
from LIMSEQ-D[OF ra e] obtain n2 where n2: \ n. n \geq n2 \Longrightarrow norm (rn

- a)<e/2 by auto
show \exists no. \foralln\geqno. norm (m n - a)<e
proof (rule exI[of - max n1 n2], intro allI impI)
fix n
assume max n1 n2 \leqn
with n1 n2 have *: norm (ln-a)<e/2 norm (rn-a)<e/2 by auto
from lm[of n] mr[of n] have norm (mn-a) \leqnorm (ln-a) + norm (rn
- a) by simp
with * show norm (m n-a)<e by auto
qed
qed
lemma real-of-tighten-bounds-many[simp]: real-of-3 ((tighten-bounds-3 ~i) x)= real-of-3 $x$
apply (induct i) using tighten-bounds-3 by auto
definition lower-3 where lower-3 $x i \equiv$ interval.lower (get-itvl-3 ((tighten-bounds-3 ~i) $x$ ))
definition upper-3 where upper-3 $x i \equiv$ interval.upper (get-itvl-3 ((tighten-bounds-3
~i)}x\mathrm{ ))
lemma interval-size-3: upper-3 $x i-$ lower-3 $x i=(u p p e r-3 x 0-l o w e r-3 x$ 0)/ 2 ^i $^{\text {i }}$
proof (induct i)
case (Suc i)
have upper-3 x (Suc i) - lower-3 x (Suc i) = (upper-3 x i - lower-3 x i)/2
unfolding upper-3-def lower-3-def using tighten-bounds-3 get-itvl-3 by auto
with Suc show ?case by auto
qed auto
lemma interval-size-3-tendsto-0: (\lambdai. (upper-3 x i - lower-3 xi))}\longrightarrow
by (subst interval-size-3, auto intro: LIMSEQ-divide-realpow-zero)
lemma dist-tendsto-0-imp-tendsto: (\lambdai. |fi - a| :: real) \longrightarrow0\Longrightarrowf\longrightarrowa
using LIM-zero-cancel tendsto-rabs-zero-iff by blast
lemma upper-3-tendsto: upper-3 x \longrightarroweal-of-3 x
proof(rule dist-tendsto-0-imp-tendsto, rule sandwitch-real)
fix }
obtain l r where lr: get-itvl-3 ((tighten-bounds-3 ^~ i) x)= Interval l r
by (metis interval.collapse)

```
```

    with get-itvl-3[of (tighten-bounds-3 ~ i) x]
    show |(upper-3 x) i - real-of-3 x| \leq (upper-3 x i lower-3 xi)
    unfolding upper-3-def lower-3-def by auto
    qed (insert interval-size-3-tendsto-0, auto)
lemma lower-3-tendsto: lower-3 x <ueal-of-3 x
proof(rule dist-tendsto-0-imp-tendsto, rule sandwitch-real)
fix }
obtain lr where lr:get-itvl-3 ((tighten-bounds-3 ~~ i) x)= Interval l r
by (metis interval.collapse)
with get-itvl-3[of (tighten-bounds-3 ^~ i) x]
show |lower-3 x i - real-of-3 x| \leq (upper-3 x i - lower-3 x i)
unfolding upper-3-def lower-3-def by auto
qed (insert interval-size-3-tendsto-0, auto)
lemma tends-to-tight-bounds-3: (\lambdax.get-itvl-3 ((tighten-bounds-3 ^x x) y)) \longrightarrow\longrightarrow
real-of-3 y
using lower-3-tendsto[of y] upper-3-tendsto[of y] unfolding lower-3-def upper-3-def
interval-tendsto-def o-def by auto
lemma complex-roots-of-int-poly3: assumes p: p\not=0 and sf: square-free p
shows set (complex-roots-of-int-poly3 p)={x. ipoly p x=0} (is ?l = ?r)
distinct (complex-roots-of-int-poly3 p)
proof -
interpret map-poly-inj-idom-hom of-real..
define q}\mathrm{ where q}=\mathrm{ real-of-int-poly }
let ?q = map-poly complex-of-real q
from p have q0:q}=0\mathrm{ unfolding q-def by auto
hence q:?q}\not=0\mathrm{ by auto
define }rr\mathrm{ where rr = real-roots-of-int-poly p
define rrts where rrts = map ( }\lambda\mathrm{ r. Complex r 0) rr
note d = complex-roots-of-int-poly3-def[of p, unfolded Let-def, folded rr-def,
folded rrts-def]
have rr: set rr ={x. ipoly px=0} unfolding rr-def
using real-roots-of-int-poly(1)[OF p].
have rrts: set rrts = {x. poly ?q }x=0\wedgex\in\mathbb{R}}\mathrm{ unfolding rrts-def set-map rr
q-def
complex-of-real-def[symmetric] by (auto elim: Reals-cases)
have dist: distinct rr unfolding rr-def using real-roots-of-int-poly(2) .
from dist have dist1: distinct rrts unfolding rrts-def distinct-map inj-on-def
by auto
have lrr: length rr = card {x. poly (real-of-int-poly p) x=0}
unfolding rr-def using real-roots-of-int-poly[of p] p distinct-card by fastforce
have cr: length rr = card {x. poly?q }x=0\wedgex\in\mathbb{R}}\mathrm{ unfolding lrr q-def[symmetric]
proof -
have card {x. poly q x=0}\leq card {x. poly (map-poly complex-of-real q) x=
0\wedgex\in\mathbb{R}}(is ?l \leq ? r)
by (rule card-inj-on-le[of of-real], insert poly-roots-finite[OF q], auto simp:
inj-on-def)

```
```

    moreover have ?l \geq ?r
                            by (rule card-inj-on-le[of Re, OF - poly-roots-finite[OF q0]], auto simp:
    inj-on-def elim!: Reals-cases)
ultimately show ?l=?r by simp
qed
have conv: \ x. ipoly p x=0 \longleftrightarrow poly ?q }x=
unfolding q-def by (subst map-poly-map-poly, auto simp: o-def)
have r: ?r = {x.poly ?q }x=0}\mathrm{ unfolding conv ..

```

```

    proof (cases degree p= length rr)
    case False note oFalse = this
    show ?thesis
    proof (cases degree p-length rr = 2)
        case False
        let ?nr = (degree p - length rr) div 2
        define cpxI where cpxI = pos-imaginary-parts-3 p
        define cpxR where cpxR= real-parts-3 p
        let ?rts = [(rx,ix).rx<- cpxR,ix<-cpxI]
    define cpx where cpx = map pair-to-complex (filter ( }\lambda\mathrm{ c.ipolyp (pair-to-complex
    c) = 0)
?rts)
let ?LL = cpx @ map cnj cpx
let ?LL' = concat (map (\lambda x. [x,cnj x]) cpx)
let ?ll = rrts @ ?LL
let ?ll'=rrts @ ?LL'
have cpx: set cpx\subseteq?r unfolding cpx-def by auto
have ccpx: cnj' set cpx\subseteq?r using cpx unfolding r
by (auto intro!: complex-conjugate-root[of ?q] simp: Reals-def)
have set ?ll \subseteq?r using rrts cpx ccpx unfolding r by auto
moreover
{
fix x :: complex
assume rt: ipoly p x = 0
{
fix }
assume rt: ipoly p x = 0
and gt: Im x > 0
define rx where rx = Re x
let ?x = Complex rx (Im x)
have x: x = ?x by (cases x, auto simp: rx-def)
from rt x have rt': ipoly p ? }x=0\mathrm{ by auto
from real-parts-3[OF p rt, folded rx-def] pos-imaginary-parts-3[OF p rt
gt] rt'
have ?x \in set cpx unfolding cpx-def cpxI-def cpxR-def
by (force simp: pair-to-complex-def[abs-def])
hence }x\in\mathrm{ set cpx using x by simp
} note gt = this
have cases: Im x = 0 \vee Im x>0\vee Im x<0 by auto
from rt have rt': ipoly p (cnj x) = 0 unfolding conv

```
```

            by (intro complex-conjugate-root[of ?q x], auto simp: Reals-def)
        {
            assume Im x > 0
            from gt[OF rt this] have x\in set ?ll by auto
        }
        moreover
        {
            assume Im x < 0
            hence Im (cnj x)>0 by simp
            from gt[OF rt' this] have cnj (cnj x) \in set ?ll unfolding set-append
    set-map by blast
hence x\in set ?ll by simp
}
moreover
{
assume Im x = 0
hence }x\in\mathbb{R}\mathrm{ using complex-is-Real-iff by blast
with rt rrts have x\in set ?ll unfolding conv by auto
}
ultimately have x\in set ?ll using cases by blast
}
ultimately have lr: set ?ll = {x. ipoly p x=0} by blast
let ?rr = map real-of-3 cpxR
let ?pi = map real-of-3 cpxI
have dist2: distinct ?rr unfolding cpxR-def by (rule distinct-real-parts-3)
have dist3: distinct ?pi unfolding cpxI-def by (rule distinct-pos-imaginary-parts-3)
have idd: concat (map (map pair-to-complex) (map (\lambdarx. map (Pair rx) cpxI)
cpxR))
= concat (map ( \lambdar. map ( }\lambda\mathrm{ i. Complex (real-of-3 r) (real-of-3 i)) cpxI)
cpxR)
unfolding pair-to-complex-def by (auto simp:o-def)
have dist4: distinct cpx unfolding cpx-def
proof (rule distinct-map-filter, unfold map-concat idd, unfold distinct-conv-nth,
intro allI impI, goal-cases)
case (1 i j)
from nth-concat-diff[OF 1, unfolded length-map] dist2[unfolded distinct-conv-nth]
dist3[unfolded distinct-conv-nth] show ?case by auto
qed
have dist5: distinct (map cnj cpx) using dist4 unfolding distinct-map by
(auto simp: inj-on-def)
{
fix x :: complex
have rrts: x set rrts \Longrightarrow Im x=0 unfolding rrts-def by auto
have cpx: \bigwedgex. x set cpx\Longrightarrow Im x>0 unfolding cpx-def cpxI-def
by (auto simp: pair-to-complex-def[abs-def] positive-pos-imaginary-parts-3)
have cpx': x \incnj' set cpx \Longrightarrow sgn (Im x)=-1 using cpx by auto
have }x\not\in\mathrm{ set rrts ค set cpx U set rrts ค cnj' set cpx U set cpx ค cnj'set
cpx
using rrts cpx[of x] cpx' by auto

```
\(\}\) note dist6 \(=\) this
have dist: distinct ?ll
unfolding distinct-append using dist6 by (auto simp: dist1 dist4 dist5)
let \(? p=\) complex-of-int-poly \(p\)
have \(p p: ? p \neq 0\) using \(p\) by auto
from \(p\) square-free-of-int-poly[OF \(s f]\) square-free-rsquarefree
have rsf:rsquarefree ? \(p\) by auto
from dist lr have length ?ll = card \(\{x\). poly ? \(p x=0\}\)
by (metis distinct-card)
also have \(\ldots=\) degree \(p\)
using rsf unfolding rsquarefree-card-degree \([O F p p]\) by simp
finally have deg-len: degree \(p=\) length ?ll by simp
let \(? P=\lambda c\). ipoly \(p(\) pair-to-complex \(c)=0\)
let ?itvl \(=\lambda r i\). ipoly-complex-interval \(p(\) Complex-Interval \((\) get-itvl-3r)
(get-itvl-3 i))
let ? itv \(=\lambda(r, i)\). ? itvl \(r i\)
let ? \(p=\left(\lambda(r, i) .0 \in_{c}(\right.\) ? itvl \(\left.r i)\right)\)
let \(? t b=\) tighten-bounds-3
let ?f = map-prod ? tb ? tb
have filter: map pair-to-complex (filter-list-length ?f ?p ?nr ?rts) = map pair-to-complex (filter ?P ?rts)
proof (rule filter-list-length)
have length (filter ?P ?rts) \(=\) length \(c p x\)
unfolding \(c p x\)-def by simp
also have \(\ldots=\) ? \(n r\) unfolding deg-len by (simp add: rrts-def)
finally show length (filter ?P ?rts) \(=\) ? \(n \mathrm{r}\) by auto
next
fix \(n x\)
assume \(x\) : ? \(P x\)
obtain \(r i\) where \(x r i: x=(r, i)\) by force
have \(i d:(? f\) ^^ \(n) x=\left(\left(? t b b^{\wedge} n\right) r,(? t b \sim n) i\right)\) unfolding \(x r i\) by (induct \(n\), auto)
have \(p x\) : pair-to-complex \(x=\) Complex (real-of-3 r) (real-of-3 \(i)\)
unfolding xri pair-to-complex-def by auto
show ? \(p((? f\) ~ \(n) x)\)
unfolding id split
by (rule ipoly-complex-interval[ of pair-to-complex \(x-p\), unfolded \(x\) ], unfold
\(p x\), auto simp: in-complex-interval-def in-real-itvl-get-bounds-tighten)
next
fix \(x\)
assume \(x: x \in\) set ?rts \(\neg\) ? \(P x\)
let \(? x=\) pair-to-complex \(x\)
obtain \(r i\) where \(x r i\) : \(x=(r, i)\) by force
have \(i d:(? f \leadsto n) x=((? t b \leadsto n) r,(? t b \leadsto n) i)\) for \(n\) unfolding \(x r i\) by (induct \(n\), auto)
have \(p x: ? x=\) Complex (real-of-3 \(r)(\) real-of- \(3 i)\)
unfolding xri pair-to-complex-def by auto
have cvg: \((\lambda n\). ?itv \(((? f \sim n) x)) \longrightarrow{ }_{c}\) ipoly \(p\) ? \(x\)
unfolding id split \(p x\)
proof (rule ipoly-complex-interval-tendsto)
show ( \(\lambda i a\). Complex-Interval (get-itvl-3 ((?tb ^^ ia) r)) (get-itvl-3 ((?tb
\(\sim\) ia) \(i))\) ) \(\longrightarrow{ }_{c}\)
Complex (real-of-3 r) (real-of-3 i)
unfolding complex-interval-tendsto-def by (simp add: tends-to-tight-bounds-3
\(o-d e f)\)
qed
from complex-interval-tendsto-neq[OF this x(2)]
show \(\exists i\). \(\neg\) ? \(p((? f \leadsto i) x)\) unfolding id by auto
next
show pair-to-complex (?f \(x\) ) \(=\) pair-to-complex \(x\) for \(x\)
by (cases x, auto simp: pair-to-complex-def tighten-bounds-3(1))
next
show ?P \((\) ?f \(x)=\) ? \(P x\) for \(x\)
by (cases \(x\), auto simp: pair-to-complex-def tighten-bounds-3(1))
qed
have l: complex-roots-of-int-poly3 \(p=\) ? \(l l^{\prime}\)
unfolding \(d\) filter cpx-def[symmetric] cpxI-def[symmetric] cpxR-def[symmetric]
using False oFalse
by auto
have distinct \(? l l^{\prime}=\left(\right.\) distinct rrts \(\wedge\) distinct \(? L L^{\prime} \wedge\) set rrts \(\cap\) set \(\left.? L L^{\prime}=\{ \}\right)\)
unfolding distinct-append ..
also have set ? \(L L^{\prime}=\) set ? \(L L\) by auto
also have distinct \(? L L^{\prime}=\) distinct \(? L L\) by (induct cpx, auto)
finally have distinct ? \(? l^{\prime}=\) distinct ?ll unfolding distinct-append by auto
with dist have distinct ?ll' by auto
with lr \(l\) show ?thesis by auto
next
case True
let ?cr \(=\) map-poly of-real \(::\) real poly \(\Rightarrow\) complex poly
define \(p p\) where \(p p=\) complex-of-int-poly \(p\)
have \(i d\) : \(p p=\) map-poly of-real \(q\) unfolding \(q\)-def \(p p\)-def
by (subst map-poly-map-poly, auto simp: o-def)
let ?rts \(=\operatorname{map}(\lambda x .[:-x, 1:]) r r\)
define rts where rts = prod-list ?rts
let ? \(c 2=\) ? \(c r\) ( \(q\) div rts)
have \(p q: \wedge x\). ipoly \(p x=0 \longleftrightarrow\) poly \(q x=0\) unfolding \(q\)-def by simp
from True have 2: degree \(q\) - card \(\{x\). poly \(q x=0\}=2\) unfolding \(p q[\) symmetric \(]\) lrr
unfolding \(q\)-def by simp
from True have id: degree \(p=\) length \(r r \longleftrightarrow\) False
degree \(p\) - length \(r r=2 \longleftrightarrow\) True by auto
have \(l: ? l=\) of-real' \(\{x\). poly \(q x=0\} \cup\) set (croots2 ?c2)
unfolding \(d\) rts-def id if-False if-True set-append rrts Reals-def
by (fold complex-of-real-def \(q\)-def, auto)
from dist
have len- \(r\) : length \(r r=\) card \(\{x\). poly \(q x=0\}\) unfolding \(r r[\) unfolded \(p q\), symmetric]
```

        by (simp add: distinct-card)
    have }r\mp@subsup{r}{}{\prime}:\r.r\in\mathrm{ set rr # poly q r = 0 using rr unfolding q-def by
    simp
with dist have q=q div prod-list ?rts * prod-list ?rts
proof (induct rr arbitrary: q)
case (Cons r rr q)
note dist = Cons(2)
let ? p=q div [:-r,1:]
from Cons.prems(2) have poly q r = 0 by simp
hence [:-r,1:] dvd q using poly-eq-0-iff-dvd by blast
from dvd-mult-div-cancel[OF this]
have q=?p * [:-r,1:] by simp
moreover have ?p = ?p div (\prodx\leftarrowrr. [:-x, 1:]) * (\prodx\leftarrowrr. [:- x, 1:])
proof (rule Cons.hyps)
show distinct rr using dist by auto
fix }
assume s\in set rr
with dist Cons(3) have s\not=r poly q s=0 by auto
hence poly (?p *[:- 1*r, 1:]) s=0 using calculation by force
thus poly ?p s=0 by (simp add: «s\not=r〉)
qed
ultimately have q:q=?p div (\prodx\leftarrowrr. [:- x, 1:]) * (\prodx\leftarrowrr. [:- x, 1:])

* [:-r,1:]
by auto
also have ... = (?p div (\prodx\leftarrowrr. [:-x, 1:])) * (\prodx\leftarrowr\# rr. [:- x, 1:])
unfolding mult.assoc by simp
also have ?p div (\prodx\leftarrowrr. [:-x, 1:]) = q div (\prodx\leftarrowr\# rr. [:- x, 1:])
unfolding poly-div-mult-right[symmetric] by simp
finally show ?case .
qed simp
hence q-div: q = q div rts * rts unfolding rts-def .
from q-div q0 have q div rts }\not=0\mathrm{ rts }\not=0\mathrm{ by auto
from degree-mult-eq[OF this] have degree q = degree ( }q\mathrm{ div rts) + degree rts
using q-div by simp
also have degree rts = length rr unfolding rts-def by (rule degree-linear-factors)
also have ... = card {x. poly q x = 0} unfolding len-rr by simp
finally have deg2: degree ?c2 = 2 using 2 by simp
note croots2 = croots2[OF deg2, symmetric]
have ? q = ?cr ( }q\mathrm{ div rts * rts) using q-div by simp
also have ... = ?cr rts * ?c2 unfolding hom-distribs by simp
finally have q-prod: ?q = ?cr rts * ?c2 .
from croots2l
have l:?l=of-real' {x.poly q x = 0} \cup {x.poly ?c2 x=0} by simp
from r[unfolded q-prod]
have r: ?r }={x.poly(?cr rts) x=0}\cup{x. poly?c2 x=0} by aut
also have ?cr rts = (\prodx\leftarrowrr. ?.cr [:- x, 1:]) by (simp add: rts-def o-def
of-real-poly-hom.hom-prod-list)
also have {x.poly ... x=0} =of-real' set rr
unfolding poly-prod-list-zero-iff by auto

```
```

    also have set rr ={x.poly qx=0} unfolding rr q-def by simp
    finally have lr:?l=?r unfolding l by simp
    show ?thesis
    proof (intro conjI[OF lr])
    from sf have sf: square-free q unfolding q-def by (rule square-free-of-int-poly)
        {
            interpret field-hom-0' complex-of-real ..
            from sf have square-free ?q unfolding square-free-map-poly .
            } note sf = this
            have l: complex-roots-of-int-poly3 p = rrts @ croots2 ?c2
    unfolding d rts-def id if-False if-True set-append rrts q-def complex-of-real-def
    by auto
have dist2: distinct (croots2 ?c2) unfolding croots2-def Let-def by auto
{
fix }
assume x: x\in set (croots2 ?c2) x cet rrts
from x(1)[unfolded croots2] have x1: poly ?c2 }x=0\mathrm{ by auto
from x(2) have x2: poly (?cr rts) x=0
unfolding rrts-def rts-def complex-of-real-def[symmetric]
by (auto simp: poly-prod-list-zero-iff o-def)
from square-free-multD(1)[OF sf[unfolded q-prod], of [: -x, 1:]]
x1 x2 have False unfolding poly-eq-0-iff-dvd by auto
} note dist3 = this
show distinct (complex-roots-of-int-poly3 p) unfolding l distinct-append
by (intro conjI dist1 dist2, insert dist3, auto)
qed
qed
next
case True
have card {x. poly ?q x = 0} \leq degree ?q by (rule poly-roots-degree[OF q])
also have ... = degree p unfolding q-def by simp
also have ... = card {x. poly ?q }x=0\wedgex\in\mathbb{R}}\mathrm{ using True cr by simp
finally have le:card {x. poly ?q }x=0}\leq\operatorname{card {x. poly ?q x=0^x\in\mathbb{R}}
by auto
have {x.poly ?q x=0^x\in\mathbb{R}}={x.poly ?q }x=0
by (rule card-seteq[OF - le], insert poly-roots-finite[OF q], auto)
with True rrts dist1 show ?thesis unfolding rd by auto
qed
thus distinct (complex-roots-of-int-poly3 p) ?l = ?r by auto
qed
lemma complex-roots-of-int-poly-all: assumes sf:degree p\geq3\Longrightarrow square-free p
shows p\not=0\Longrightarrow set (complex-roots-of-int-poly-all p)={x. ipoly p x = 0} (is -
let ?l = ?r)
and distinct (complex-roots-of-int-poly-all p)
proof -
note d = complex-roots-of-int-poly-all-def Let-def
have }(p\not=0\longrightarrow\mathrm{ set ?l = ?r) ^(distinct (complex-roots-of-int-poly-all p))

```
```

    proof (cases degree p \geq3)
    case True
    hence p: p\not=0 by auto
    from True complex-roots-of-int-poly3[OF p] sf show ?thesis unfolding d by
    auto
next
case False
let ?p = map-poly (of-int :: int }=>\mathrm{ complex) p
have deg: degree ?p = degree p
by (simp add: degree-map-poly)
show ?thesis
proof (cases degree p=1)
case True
hence l:?l = [roots1 ?p] unfolding d by auto
from True have degree ?p = 1 unfolding deg by auto
from roots1[OF this] show ?thesis unfolding l roots1-def by auto
next
case False
show ?thesis
proof (cases degree p=2)
case True
hence l: ?l = croots2 ?p unfolding d by auto
from True have degree ?p = 2 unfolding deg by auto
from croots2[OF this] show ?thesis unfolding l by (simp add: croots2-def
Let-def)
next
case False
with <degree p\not=1\rangle\langledegree p\not=2\rangle\langle\neg(degree p \geq 3)> have True: degree
p=0 by auto
hence l: ?l = [] unfolding d by auto
from True have degree ?p = 0 unfolding deg by auto
from rootsO[OF - this] show ?thesis unfolding l by simp
qed
qed
qed
thus }p\not=0\Longrightarrow\mathrm{ set ?l = ?r distinct (complex-roots-of-int-poly-all p) by auto
qed

```

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

definition complex-roots-of-int-poly $::$ int poly $\Rightarrow$ complex list where
complex-roots-of-int-poly $p=($
let $p s=($ if degree $p \geq 3$ then factors-of-int-poly $p$ else $[p])$
in concat (map complex-roots-of-int-poly-all ps))
definition complex-roots-of-rat-poly :: rat poly $\Rightarrow$ complex list where
complex-roots-of-rat-poly $p=$ complex-roots-of-int-poly (snd (rat-to-int-poly p))

```
```

lemma complex-roots-of-int-poly:
shows }p\not=0\Longrightarrow\mathrm{ set (complex-roots-of-int-poly p)={x. ipoly p x=0}(is - \#
?l=?r)
and distinct (complex-roots-of-int-poly p)
proof -
have ( }p\not=0\longrightarrow?l=?r)\wedge(\mathrm{ distinct (complex-roots-of-int-poly p))
proof (cases degree p \geq3)
case False
hence complex-roots-of-int-poly p = complex-roots-of-int-poly-all p
unfolding complex-roots-of-int-poly-def Let-def by auto
with complex-roots-of-int-poly-all[of p] False show ?thesis by auto
next
case True
{
fix q
assume q}\in\operatorname{set (factors-of-int-poly p)
from factors-of-int-poly(1)[OF refl this] irreducible-imp-square-free[of q]
have 0:q\not=0 and sf: square-free q by auto
from complex-roots-of-int-poly-all(1)[OF sf 0] complex-roots-of-int-poly-all(2)[OF
sf]
have set (complex-roots-of-int-poly-all q) ={x. ipoly q x = 0}
distinct (complex-roots-of-int-poly-all q) by auto
} note all = this
from True have
?l = (U ((\lambda p. set (complex-roots-of-int-poly-all p))' set (factors-of-int-poly
p)))
unfolding complex-roots-of-int-poly-def Let-def by auto
also have ···. = (U ((\lambda p.{x. ipoly p x = 0})'set (factors-of-int-poly p)))
using all by blast
finally have l: ?l=(\bigcup((\lambdap.{x. ipoly px=0})'set (factors-of-int-poly p)))
have lr: p\not=0\longrightarrow?l=?r using l factors-of-int-poly(2)[OF refl, of p] by
auto
show ?thesis
proof (rule conjI[OF lr])
from True have id: complex-roots-of-int-poly p=
concat (map complex-roots-of-int-poly-all (factors-of-int-poly p))
unfolding complex-roots-of-int-poly-def Let-def by auto
show distinct (complex-roots-of-int-poly p) unfolding id distinct-conv-nth
proof (intro allI impI, goal-cases)
case (1 ij)
let ?fp = factors-of-int-poly p
let ?rr = complex-roots-of-int-poly-all
let ?cc = concat (map ?rr (factors-of-int-poly p))
from nth-concat-diff[OF 1, unfolded length-map]
obtain j1 k1 j2 k2 where
*: (j1,k1) = (j2,k2)
j1 < length ?fp j2 < length ?fp and
k1<length (map ?rr ?fp!j1)

```
```

    k2 < length (map ?rr ?fp!j2)
    ?cc!i= map ?rr ?fp!j1!k1
    ?cc!j = map ?rr ?fp! j2 ! k2 by blast
    hence **: k1 < length (?rr (?fp!j1))
k2 < length (?rr (?fp! j2))
?cc!i=?rr (?fp!j1)!k1
?cc!j = ?rr (?fp!j2)!k2
by auto
from * have mem: ?fp!j1 \in set ?fp ?fp ! j2 \in set ?fp by auto
show ?cc!i\not=?cc!j
proof (cases j1 = j2)
case True
with * have k1\not=k2 by auto
with all(2)[OF mem(2)] **(1-2) show ?thesis unfolding **(3-4)
unfolding True
distinct-conv-nth by auto
next
case False
from 〈degree p\geq3` have p: p\not=0 by auto
note fip = factors-of-int-poly(2-3)[OF refl this]
show ?thesis unfolding **(3-4)
proof
define }x\mathrm{ where }x=
assume id: ?rr (?fp!j1)!k1 = ?rr (?fp ! j2)!k2
from ** have x1:x\in set (?rr (?fp! j1)) unfolding x-def id[symmetric]
by auto
from ** have x2: x \in set (?rr (?fp!j2)) unfolding x-def by auto
from all(1)[OF mem(1)] x1 have x1: ipoly (?fp!j1) x=0 by auto
from all(1)[OF mem(2)] x2 have x2: ipoly (?fp! j2) x=0 by auto
from False factors-of-int-poly(4)[OF refl, of p] have neq: ?fp!j1 \not= ?fp
! j2
using * unfolding distinct-conv-nth by auto
have poly (complex-of-int-poly p) x=0 by (meson fip(1)mem(2) x2)
from fip(2)[OF this] mem x1 x2 neq
show False by blast
qed
qed
qed
qed
qed
thus p\not=0\Longrightarrow ?l = ?r distinct (complex-roots-of-int-poly p) by auto
qed
lemma complex-roots-of-rat-poly:
p\not=0\Longrightarrow set (complex-roots-of-rat-poly p)={x.rpoly px=0} (is - "?l=
?r)
distinct (complex-roots-of-rat-poly p)
proof -

```
obtain \(c q\) where \(c q\) : rat-to-int-poly \(p=(c, q)\) by force
from rat-to-int-poly[OF this]
have \(p q: p=\) smult (inverse (of-int c)) (of-int-poly \(q\) )
and \(c: c \neq 0\) by auto
show distinct (complex-roots-of-rat-poly p) unfolding complex-roots-of-rat-poly-def
using complex-roots-of-int-poly(2) .
assume \(p: p \neq 0\)
with \(p q c\) have \(q: q \neq 0\) by auto
have id: \(\{x\). rpoly \(p x=(0::\) complex \()\}=\{x\). ipoly \(q x=0\}\)
unfolding \(p q\) by (simp add: c of-rat-of-int-poly hom-distribs)
show ?l \(=? r\) unfolding complex-roots-of-rat-poly-def cq snd-conv id complex-roots-of-int-poly(1)[OF q] ..
qed
lemma min-int-poly-complex-of-real[simp]: min-int-poly (complex-of-real \(x)=\) min-int-poly \(x\)
proof (cases algebraic \(x\) )
case False
hence \(\neg\) algebraic (complex-of-real x) unfolding algebraic-complex-iff by auto
with False show ?thesis unfolding min-int-poly-def by auto
next
case True
from min-int-poly-represents[OF True]
have min-int-poly \(x\) represents \(x\) by auto
thus ?thesis
by (intro min-int-poly-unique, auto simp: lead-coeff-min-int-poly-pos)
qed
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 \(f x=\left(0:^{\prime} a\right)\), but because of complex-algebraic-number arithmetic, I think that search would be slower than the current one via " \(x \in\) set (complex-roots-of-int-poly \(f\) )
definition min-int-poly-complex :: complex \(\Rightarrow\) int poly where
min-int-poly-complex \(x=\) (if algebraic \(x\) then if \(\operatorname{Im} x=0\) then min-int-poly-real (Re \(x\) )
else the (find \((\lambda f . x \in\) set (complex-roots-of-int-poly \(f)\) ) (complex-poly (min-int-poly \((\operatorname{Re} x))(\) min-int-poly \((\operatorname{Im} x))))\)
else [:0, 1:])
lemma min-int-poly-complex[code-unfold]: min-int-poly \(=\) min-int-poly-complex proof (standard)
fix \(x\)
define \(f s\) where \(f s=\) complex-poly (min-int-poly \((\operatorname{Re} x))(\) min-int-poly \((\operatorname{Im} x))\)
let ?f \(=\) min-int-poly-complex \(x\)
show min-int-poly \(x=\) ?f
proof (cases algebraic \(x\) )
case False
thus ?thesis unfolding min-int-poly-def min-int-poly-complex-def by auto
```

    next
    case True
    show ?thesis
    proof (cases Im x = 0)
        case *: True
        have id: ?f = min-int-poly-real (Re x) unfolding min-int-poly-complex-def *
    using True by auto
show ?thesis unfolding id min-int-poly-real-code-unfold[symmetric] min-int-poly-complex-of-real[symmetr
using * by (intro arg-cong[of - min-int-poly] complex-eqI, auto)
next
case False
from True[unfolded algebraic-complex-iff] have algebraic (Re x) algebraic (Im
x) by auto
from complex-poly[OF min-int-poly-represents[OF this(1)] min-int-poly-represents[OF
this(2)]]
have fs: \existsf\in set fs.ipoly fx=0 \f.f\in set fs \Longrightarrow poly-cond f unfolding
fs-def by auto
let ?fs = find ( }\lambda\mathrm{ f. ipoly fx=0) fs
let ?fs' = find ( }\lambdaf.x\in\mathrm{ set (complex-roots-of-int-poly f)) fs
have ?f = the ?fs' unfolding min-int-poly-complex-def fs-def
using True False by auto
also have ?fs' = ?fs
by (rule find-cong[OF refl], subst complex-roots-of-int-poly, insert fs, auto)
finally have id: ?f = the ?fs .
from fs(1) have ?fs }\not=\mathrm{ None unfolding find-None-iff by auto
then obtain f}\mathrm{ where Some:?fs = Some f by auto
from find-Some-D[OF this] fs(2)[of f]
show ?thesis unfolding id Some
by (intro min-int-poly-unique, auto)
qed
qed
qed
end

```

\section*{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
by (rule show-lawI) (simp add: showsp-real-alg-def show-law-simps)
lemma showsp-real-alg-append [show-law-simps]:
showsp-real-alg pr(x@y)= showsp-real-alg prx @ y
by (intro show-lawD show-law-intros)
local-setup <
Show-Generator.register-foreign-showsp @{typ real-alg} @{term showsp-real-alg}
@{thm show-law-real-alg}
,
derive show real-alg
We now define show-real.
overloading show-real \equiv show-real
begin
definition show-real \equiv show-real-alg o real-alg-of-real
end
end

```

\section*{17 Show for Real (Algebraic) Numbers - Approximate Representation}

We implement the show-function for real (algebraic) numbers by calculating the number precisely for three digits after the comma.
```

theory Show-Real-Approx
imports
Show-Real-Alg
Show.Show-Instances
begin
overloading show-real-alg \equiv show-real-alg
begin
definition show-real-alg[code]: show-real-alg x let
x1000' = floor (1000 * x);
(x1000,s)=(if x1000' < 0 then (-x1000', "-'') else (x1000', '"''));
(bef,aft) = divmod-int x1000 1000;
a}=\mathrm{ show aft;
a= replicate (3-length a')(CHR '"}\mp@subsup{0}{}{\prime\prime})@ @ a
in

```
end
end

\section*{18 Show for Real (Algebraic) Numbers - Unique Representation}

We implement the show-function for real (algebraic) numbers by printing them uniquely via their monic irreducible polynomial with a special cases for polynomials of degree at most 2 .
```

theory Show-Real-Precise
imports
Show-Real-Alg
Show.Show-Instances
begin
datatype real-alg-show-info = Rat-Info rat | Sqrt-Info rat rat | Real-Alg-Info int
poly nat

```
```

fun convert-info :: rat + int poly $\times$ nat $\Rightarrow$ real-alg-show-info where

```
fun convert-info :: rat + int poly \(\times\) nat \(\Rightarrow\) real-alg-show-info where
    convert-info \((\) Inl \(q)=\) Rat-Info \(q\)
    convert-info \((\) Inl \(q)=\) Rat-Info \(q\)
| convert-info \((\operatorname{Inr}(f, n))=(\) if degree \(f=2\) then \((\) let \(a=\operatorname{coeff} f 2 ; b=\operatorname{coeff} f 1\);
| convert-info \((\operatorname{Inr}(f, n))=(\) if degree \(f=2\) then \((\) let \(a=\operatorname{coeff} f 2 ; b=\operatorname{coeff} f 1\);
\(c=\) coeff f 0;
\(c=\) coeff f 0;
        b2a \(=\) Rat.Fract (-b) \((2 * a)\);
        b2a \(=\) Rat.Fract (-b) \((2 * a)\);
        below \(=\) Rat.Fract \((b * b-4 * a * c)(4 * a * a)\)
        below \(=\) Rat.Fract \((b * b-4 * a * c)(4 * a * a)\)
        in Sqrt-Info b2a (if \(n=1\) then -below else below))
        in Sqrt-Info b2a (if \(n=1\) then -below else below))
        else Real-Alg-Info f \(n\) )
```

        else Real-Alg-Info f \(n\) )
    ```
definition real-alg-show-info :: real-alg \(\Rightarrow\) real-alg-show-info where
    real-alg-show-info \(x=\) convert-info (info-real-alg \(x)\)

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

lemma real-alg-show-info: real-alg-show-info $x=$ Rat-Info $r \Longrightarrow$ real-of $x=o f$-rat
$r$
real-alg-show-info $x=$ Sqrt-Info $r s q \Longrightarrow$ real-of $x=o f$-rat $r+s q r t(o f$-rat $s q)$
real-alg-show-info $x=$ Real-Alg-Info $p n \Longrightarrow p$ represents (real-of $x) \wedge n=$ card
$\{y . y \leq$ real-of $x \wedge$ ipoly p $y=0\}$
(is ? $l \Longrightarrow$ ? $r$ )
proof (atomize(full), goal-cases)
case 1
note $d=$ real-alg-show-info-def
show ?case
proof (cases info-real-alg $x$ )
case (Inl q)

```
```

from info-real-alg(2)[OF this] this show ?thesis unfolding $d$ by auto

``` next
case ( \(\operatorname{Inr} q m\) )
then obtain \(p n\) where \(i d\) : info-real-alg \(x=\operatorname{Inr}(p, n)\) by (cases qm,auto)
from info-real-alg(1)[OF id]
have ap: \(p\) represents (real-of \(x\) ) and \(n: n=\) card \(\{y . y \leq\) real-of \(x \wedge\) ipoly \(p y\) \(=0\}\)
and irr: irreducible \(p\) by auto
note \(i d^{\prime}=\) real-alg-show-info-def id convert-info.simps Fract-of-int-quotient Let-def
have last: ?l \(\Longrightarrow\) ?r unfolding \(i d^{\prime}\) using ap \(n\) by (auto split: if-splits)
\{
assume \(*\) : real-alg-show-info \(x=\) Sqrt-Info r sq
from this[unfolded id'] have deg: degree \(p=2\) by (auto split: if-splits)
from degree2-coeffs[OF this] obtain \(a b c\) where \(p: p=[: c, b, a:]\) and \(a: a \neq\) 0 by auto
hence coeffs: coeff p \(0=c\) coeff \(p 1=b\) coeff \(p(S u c(\) Suc 0\())=a 2=\) Suc (Suc 0) by auto
let \(? a=\) real-of-int \(a\)
let \(? b=\) real-of-int \(b\)
let ?c \(=\) real-of-int \(c\)
define \(A\) where \(A=? a\)
define \(B\) where \(B=? b\)
define \(C\) where \(C=? c\)
let ? \(r=-(B /(2 * A))\)
define \(R\) where \(R=\) ? \(r\)
let ? \(s q=(B * B-4 * A * C) /(4 * A * A)\)
let \(? p=\) real-of-int-poly \(p\)
let ?disc \(=(B /(2 * A)) \wedge\) Suc \((S u c 0)-C / A\)
define \(D\) where \(D=\) ?disc
from arg-cong[OF p, of map-poly real-of-int]
have \(r p: ? p=[: C, B, A:]\)
using \(a\) by (auto simp: \(A\)-def \(B\)-def \(C\)-def)
from \(a\) have \(A: A \neq 0\) unfolding \(A\)-def by auto
from \(*\left[\right.\) unfolded \(i d^{\prime}\) deg, unfolded coeffs of-int-minus of-int-minus of-int-mult of-int-diff, simplified]
have \(r\) : real-of-rat \(r=R\) and sq: sqrt (of-rat \(s q)=(\) if \(n=1\) then - sqrt?sq else sqrt?sq)
by (auto simp: \(A\)-def B-def \(C\)-def \(R\)-def real-sqrt-minus hom-distribs)
note \(s q\)
also have ? \(s q=D\) using \(A\) by (auto simp: field-simps \(D\)-def)
finally have sq: sqrt (of-rat sq) \(=(\) if \(n=1\) then - sqrt \(D\) else sqrt \(D)\) by simp
with rp have coeffs': coeff ?p \(0=C\) coeff ?p \(1=B\) coeff ?p (Suc (Suc 0)) \(=A 2=S u c(S u c ~ 0)\) by auto
from \(r p A\) have degree (real-of-int-poly \(p\) ) 22 by auto
note roots \(=\) rroots2[OF this, unfolded rroots2-def Let-def coeffs', folded D-def \(R\)-def]
from ap[unfolded represents-def] have root: ipoly \(p(\) real-of \(x)=0\) by auto
```

from root roots have D:(D<0) = False by auto
note roots = roots[unfolded this if-False, folded R-def]
have real-of x =of-rat r + sqrt (of-rat sq)
proof (cases D=0)
case True
show ?thesis using roots root unfolding sq r True by auto
next
case False
with D have D: D>0 by auto
from roots False have roots: {x. ipoly p x = 0} ={R+ sqrt D,R - sqrt

```
\(D\}\) by auto
    let ?Roots \(=\{y . y \leq\) real-of \(x \wedge\) ipoly \(p y=0\}\)
    have \(x\) : real-of \(x \in\) ?Roots using root by auto
    from root roots have choice: real-of \(x=R+\) sqrt \(D \vee\) real-of \(x=R-\) sqrt
\(D\) by auto
    hence small: \(R-\) sqrt \(D \in\) ?Roots using roots \(D\) by auto
    show ?thesis
    proof (cases \(n=1\) )
        case True
        from card-1-singleton \(E[\) OF \(n[\) symmetric, unfolded this \(]\) ] obtain \(y\) where
id: ?Roots \(=\{y\}\) by auto
    from \(x\) small show ?thesis unfolding sq rid using True by auto
    next
            case False
            from \(x\) obtain \(Y\) where \(Y\) : ?Roots \(=\) insert (real-of \(x)(Y-\{\) real-of
\(x\}\) ) by auto
            with False[unfolded \(n\) ] obtain \(z Z\) where \(Z: Y-\{\) real-of \(x\}=\) insert \(z\)
\(Z\) by (cases \(Y-\{\) real-of \(x\}=\{ \}\), auto)
            from \(Y[\) unfolded \(Z] Z\) have sub: \(\{\) real-of \(x, z\} \subseteq\) ?Roots and \(z: z \neq\) real-of
\(x\) by auto
                with roots choice \(D\) have real-of \(x=R+\) sqrt \(D\) by force
                thus ?thesis unfolding sq \(r\) id using False by auto
            qed
        qed
    \}
    with last show ?thesis unfolding \(d\) by (auto simp: id Let-def)
    qed
qed
fun show-rai-info :: int \(\Rightarrow\) real-alg-show-info \(\Rightarrow\) string where
    show-rai-info fl (Rat-Info \(r\) ) \(=\) show \(r\)
| show-rai-info fl (Sqrt-Info r sq) = (let sqrt = "'sqrt(" @ show (abs sq) @ ") "
        in if \(r=0\) then (if \(s q<0\) then \({ }^{\prime \prime}-{ }^{\prime \prime}\) else []) @ sqrt
                else ('"(" @ show \(r\) @ (if sq < 0 then \({ }^{\prime \prime}-{ }^{\prime \prime}\) else \({ }^{\prime \prime}+^{\prime \prime}\) ) @ sqrt @ \(\left.\left.{ }^{\prime \prime}\right)^{\prime \prime}\right)\) )
| show-rai-info fl (Real-Alg-Info p \(n\) ) \(=\)
    '(root \#'" @ shown @ " of "' @ show p@ ", in (" @ show fl @ "," @ show (fl
+1) @ 'л) \()^{\prime \prime}\)
overloading show-real-alg \(\equiv\) 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

```

\section*{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

```

\subsection*{19.1 Stand-Alone Examples}
abbreviation (input) show-lines \(x \equiv\) shows-lines \(x\) Nil
```

fun show-factorization :: 'a :: \{semiring-1,show $\} \times(($ 'a poly $\times$ nat $)$ list $) \Rightarrow$ string
where
show-factorization $(c,[])=$ show $c$
| show-factorization $(c,((p, i) \# p s))=$ show-factorization $(c, p s) @{ }^{\prime \prime} *\left({ }^{\prime \prime} @\right.$ show
p@ '")" @
(if $i=1$ then [] else "ハヘ1 @ show $i$ )
definition show-sf-factorization :: 'a :: \{semiring- 1, show $\} \times\left(\left(\begin{array}{l}\text { a }\end{array}\right.\right.$ poly $\times$ nat $)$ list $)$
$\Rightarrow$ 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 \(34>\operatorname{sqrt}(\) root 43\()+\lfloor 1 / 10 *\) root 37\(\rfloor\) by eval
lemma csqrt \((4+3 *\) i) \(\notin \mathbb{R}\) by eval
value \([\) code] show \((\operatorname{csqrt}(4+3 * \mathrm{i}))\)
value \([\) code \(]\) show \((\operatorname{csqrt}(1+\mathrm{i}))\)

\subsection*{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 \(\Rightarrow\) 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 \(\leftarrow\) eigenvalues testmat]))
value [code] char-poly testmat
value [code] spectral-radius-test
end

\section*{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

\subsection*{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 \(\Rightarrow\) real-alg \()\)
definition of-rational-ra \(=((\lambda\) (num, denom). of-rat-real-alg (Rat.Fract (int-of-integer num) (int-of-integer denom)))
:: integer \(\times\) integer \(\Rightarrow\) real-alg)
definition plus-ra \(=((+)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition minus-ra \(=((-)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition uminus-ra \(=(\) uminus \(::\) real-alg \(\Rightarrow\) real-alg \()\)
definition times-ra \(=((*)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition divide-ra \(=((/)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition inverse-ra \(=(\) inverse \(::\) real-alg \(\Rightarrow\) real-alg \()\)
definition \(a b s-r a=(a b s::\) real-alg \(\Rightarrow\) real-alg \()\)
definition floor-ra \(=\) (integer-of-int o floor :: real-alg \(\Rightarrow\) integer \()\)
definition ceiling-ra \(=(\) integer-of-int o ceiling \(::\) real-alg \(\Rightarrow\) integer \()\)
definition minimum-ra \(=(\min ::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition maximum-ra \(=(\max ::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition equals-ra \(=((=)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) bool \()\)
definition less-ra \(=((<)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) bool \()\)
definition less-equal-ra \(=((\leq)::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) bool \()\)
definition compare-ra \(=(\) compare \(::\) real-alg \(\Rightarrow\) real-alg \(\Rightarrow\) order \()\)
definition roots-of-poly-ra \(=(\) roots-of-real-alg o poly-of-list o map int-of-integer \(::\) integer list \(\Rightarrow\) real-alg list)
definition root-ra \(=(\) root-real-alg o nat-of-integer \(::\) integer \(\Rightarrow\) real-alg \(\Rightarrow\) real-alg \()\)
definition show-ra \(=((\) String.implode o show) :: real-alg \(\Rightarrow\) String.literal \()\)
definition is-rational-ra \(=(\) is-rat-real-alg :: real-alg \(\Rightarrow\) bool \()\)
definition to-rational-ra \(=\) (decompose-rat o to-rat-real-alg \(::\) real-alg \(\Rightarrow\) integer \(\times\) integer)
definition sign-ra \(=(\) fst o to-rational-ra o sgn :: real-alg \(\Rightarrow\) 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 \(\Rightarrow\) integer \(\times\) integer + integer list \(\times\) integer \()\)

\subsection*{20.2 Operations on Complex Algebraic Numbers}
definition zero-ca \(=(0::\) complex \()\)
definition one-ca \(=(1::\) complex \()\)
definition imag-unit-ca \(=(\mathrm{i}::\) complex \()\)
definition of-integer-ca \(=(\) of-int o int-of-integer \(::\) integer \(\Rightarrow\) complex \()\)
definition of-rational-ca \(=((\lambda\) (num, denom) . of-rat (Rat.Fract (int-of-integer
num) (int-of-integer denom)))
\(::\) integer \(\times\) integer \(\Rightarrow\) complex)
definition of-real-imag-ca \(=((\lambda\) (real, imag). Complex (real-of real) \()(\) real-of imag \())\)
:: real-alg \(\times\) real-alg \(\Rightarrow\) complex \()\)
definition plus-ca \(=((+)\) :: complex \(\Rightarrow\) complex \(\Rightarrow\) complex \()\)
definition minus-ca \(=((-)::\) complex \(\Rightarrow\) complex \(\Rightarrow\) complex \()\)
definition uminus-ca \(=(\) uminus \(::\) complex \(\Rightarrow\) complex \()\)
definition times-ca \(=((*)::\) complex \(\Rightarrow\) complex \(\Rightarrow\) complex \()\)
definition divide-ca \(=((/)::\) complex \(\Rightarrow\) complex \(\Rightarrow\) complex \()\)
definition inverse-ca \(=(\) inverse \(::\) complex \(\Rightarrow\) complex \()\)
definition equals-ca \(=((=)::\) complex \(\Rightarrow\) complex \(\Rightarrow\) bool \()\)
definition roots-of-poly-ca \(=\) (complex-roots-of-int-poly o poly-of-list o map int-of-integer
:: integer list \(\Rightarrow\) complex list)
definition csqrt-ca \(=(\) csqrt \(::\) complex \(\Rightarrow\) complex \()\)
definition show-ca \(=((\) String.implode o show \()::\) complex \(\Rightarrow\) String.literal \()\)
definition real-of-ca \(=\) (real-alg-of-real o Re \(::\) complex \(\Rightarrow\) real-alg \()\)
definition imag-of-ca \(=(\) real-alg-of-real o Im :: complex \(\Rightarrow\) real-alg \()\)

\subsection*{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

```

\section*{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.```


[^0]:    *Supported by FWF (Austrian Science Fund) project Y757.

