Implementing field extensions of the form $\mathbb{Q}[\sqrt{b}]$*

René Thiemann

May 28, 2015

Abstract

We apply data refinement to implement the real numbers, where we support all numbers in the field extension $\mathbb{Q}[\sqrt{b}]$, i.e., all numbers of the form $p + q\sqrt{b}$ for rational numbers $p$ and $q$ and some fixed natural number $b$. To this end, we also developed algorithms to precisely compute roots of a rational number, and to perform a factorization of natural numbers which eliminates duplicate prime factors.

Our results have been used to certify termination proofs which involve polynomial interpretations over the reals.

Contents

1 Introduction 1
2 Auxiliary lemmas which might be moved into the Isabelle distribution. 2
3 Prime products 2
4 A representation of real numbers via triples 9
5 A unique representation of real numbers via triples 20

1 Introduction

It has been shown that polynomial interpretations over the reals are strictly more powerful for termination proving than polynomial interpretations over the rationals. To this end, also automated termination prover started to generate such interpretations. [3, 4, 5, 7, 8]. However, for all current implementations, only reals of the form $p + q \cdot \sqrt{b}$ are generated where $b$ is some fixed natural number and $p$ and $q$ may be arbitrary rationals, i.e., we get numbers within $\mathbb{Q}[\sqrt{b}]$.

*This research is supported by FWF (Austrian Science Fund) project P22767-N13.
To support these termination proofs in our certifier Cæta [6], we therefore required executable functions on $\mathbb{Q}[\sqrt{b}]$, which can then be used as an implementation type for the reals. Here, we used ideas from [1, 2] to provide a sufficiently powerful partial implementations via data refinement.

2 Auxiliary lemmas which might be moved into the Isabelle distribution.

theory Real-Impl-Auxiliary
imports
~/src/HOL/Number-Theory/UniqueFactorization
begin

lemma multiplicity-prime: assumes p: prime (i :: nat) and ji: j ≠ i
shows multiplicity j i = 0
proof (rule ccontr)
  assume ¬ thesis
  hence multiplicity j i > 0 by auto
  hence j: j ∈ prime-factors i
    by (metis less-not-refl multiplicity-not-factor-nat)
  hence d: j dvd i
    by (metis p prime-factors-altdef2-nat prime-gt-0-nat)
  then obtain k where i: i = j * k ..
  from j have j ≥ 2
    by (metis prime-factors-prime-nat prime-ge-2-nat)
  hence j1: j ≠ 1 by auto
  from i have j dvd i by auto
  with j1 ji p[unfolded prime-nat-def] show False by auto
qed

end

3 Prime products

theory Prime-Product
imports
  Real-Impl-Auxiliary
  ../Sqrt-Babylonian/Sqrt-Babylonian
begin
  Prime products are natural numbers where no prime factor occurs more than once.

definition prime-product where prime-product (n :: nat) = (\forall p. multiplicity p n ≤ 1)

  The main property is that whenever $b_1$ and $b_2$ are different prime products, then $p_1 + q_1 \sqrt{b_1} = p_2 + q_2 \sqrt{b_2}$ implies $(p_1, q_1, b_1) = (p_2, q_2, b_2)$ for all
rational numbers $p_1, q_1, p_2, q_2$. This is the key property to uniquely represent numbers in $\mathbb{Q}[\sqrt{b}]$ by triples. In the following we develop an algorithm to decompose any natural number $n$ into $n = s^2 \cdot p$ for some $s$ and prime product $p$.

**function** prime-product-factor-main :: $\mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N} \times \mathbb{N}$

**where**

prime-product-factor-main factor-sq factor-pr limit n i =
  (if $i \leq \text{limit} \land i \geq 2$ then
    (if $i \mid n$ then
      let $n' = n \div i \text{ in}$
      (if $i \mid n'$ then
        let $n'' = n' \div i \text{ in}$
        prime-product-factor-main (factor-sq * $i$) factor-pr (nat (root-nat-floor 3 n'')) n'' i
      else
        case sqrt-nat n' of
          Cons sn - ⇒ (factor-sq * sn, factor-pr * $i$) (nat (root-nat-floor 3 n'')) n' (Suc i)
        | [] ⇒ prime-product-factor-main factor-sq (factor-pr * $i$) (nat (root-nat-floor 3 n'')) n' (Suc i)
      )
    else
      prime-product-factor-main factor-sq factor-pr limit n (Suc i))
  else
    (factor-sq, factor-pr * n)) by pat-completeness auto

**termination**

**proof**

let $m1 = \lambda (\text{factor-sq :: } \mathbb{N}, \text{factor-pr :: } \mathbb{N}, \text{limit :: } \mathbb{N}, \text{n :: } \mathbb{N}, i :: \mathbb{N}). n$

let $m2 = \lambda (\text{factor-sq, factor-pr, limit, n, i :: } \mathbb{N}). (\text{Suc limit} - i)$

{ fix $i$
  have $2 \leq i \implies \text{Suc 0 < i} \times i$ using one-less-mult[of i i] by auto
} note $* = \text{this}$

show $?\text{thesis}$

by (rule, rule wf-measures[of [m1, m2]], auto split: if-splits simp:* dvd-eq-mod-eq-0)

**qed**

**lemma** prime-product-factor-main: assumes $\neg (\exists s. s \times s = n)$

and limit = nat (root-nat-floor 3 n)

and $m = \text{factor-sq} \times \text{factor-sq} \times \text{factor-pr} \times n$

and prime-product-factor-main factor-sq factor-pr limit n i = (sq, p)

and $i \geq 2$

and $\forall j. j \geq 2 \implies j < i \implies \neg j \mid n$

and $\forall j. j < i \implies \text{multiplicity} j \mid \text{factor-pr} \leq 1$

and $\forall j. j \geq i \implies \text{multiplicity} j \mid \text{factor-pr} = 0$

and factor-pr > 0

shows $m = \text{sq} \times \text{sq} \times p \land \text{prime-product} p$

3
using assms

proof (induct factor-sq factor-pr limit n i rule: prime-product-factor-main.induct)
  case (1 factor-sq factor-pr limit n i)
  note IH = I (1 - 3)
  note prems = I (4 -)
  note simp = prems (4) [unfolded prime-product-factor-main.simps[of factor-sq factor-pr limit n i]]

show ?case
  proof (cases i ≤ limit)
    case True
    note IH = IH [OF IH]
    note simp = simp [unfolded * if-True]
  show ?thesis
    proof
      from prems (5 - 8) [OF j] show multiplicity j factor-pr ≤ 1 by (cases j = i, auto)
    qed (insert prems (8 - 9) cond, auto)
  next
    case False
    hence *: (i dvd n) = False by simp

next
  case True
  note mod = this

  hence *: (i dvd n) = True by simp

  note IH = I (1, 2) [OF True refl]
  show ?thesis
    proof (cases i dvd n)
      case True
      hence *: (i dvd n) = True by auto
      def n' ≡ n dvd i div i
      from mod True have n: n = n' * i * i by (auto simp: n'-def dvd-eq-mod-eq-0)
      note simp = simp [unfolded * if-True split]
      note IH = IH (1) [OF True refl - refl - simp prems (5 - 7) - prems (7 - 9) - simp prems (7 - 9) - prems (7 - 9)]
      show ?thesis
        proof (rule IH)
          show m = factor-sq * i * (factor-sq * i) * factor-pr * (n dvd i div i)
            unfolding prems (3) n'-def [symmetric]
unfolding $n$ by (auto simp: field-simps)

next
fix $j$
assume $2 \leq j \lessdot i$ from prems(6)[OF this] have $\neg j \mid n$ by auto
thus $\neg j \mid n \div i \div i$
  by (metis dvd-mult $n \cdot n'$-def mult.assoc mult.commute)

next
show $\neg (\exists s. s \cdot s = n \div i \div i)$
proof
  assume $\exists s. s \cdot s = n \div i \div i$
  then obtain $s$ where $s \cdot s = n \div i \div i$ by auto
  hence $(s \cdot i) \cdot (s \cdot i) = n$ unfolding $n$
    by (auto simp: n'-def dvd-eq-mod-eq-0)
with prems(1) show False by blast
qed
qed

next
case False
def $n' \equiv n \div i$
from mod True have $n = n' \cdot i$ by (auto simp: $n'$-def dvd-eq-mod-eq-0)
have prime: prime $i$
  unfolding prime-nat-def
proof (intro conjI allI impI)
  fix $m$
  assume $m: m \mid d i$
  hence $m \mid n \div i$ unfolding $n$ by auto
  with prems(6)[of $m$] have choice: $m \leq 1 \lor m \geq i$ by arith
  from $m$ prems(5) have $m > 0$
  by (metis dvd-0-left-iff le0 le-antisym neq0-conv zero-neq-numeral)
  with choice have choice: $m = 1 \lor m \geq i$ by arith
  from $m$ prems(5) have $m \leq i$
  by (metis False dvd-by-0 dvd.dual-order.refl dvd-imp-le gr0I)
  with choice
  show $m = 1 \lor m = i$ by auto
qed (insert prems(5), auto)
from False have $(i \mid d i \mid n) = False$ by auto
note simp = simp[unfolded this if-False]
note IH = IH(2)[OF False -- refl]
from prime have $i > 0$ by auto
note mult = multiplicity-product-nat[OF prems(9) this]
show $thesis$
proof (cases sqrt-nat $(n \div i)$)
case (Cons $s$
  note simp = simp[unfolded Cons list.simps]
  hence $sq: sq = \text{factor-sq} * s$ and $p: p = \text{factor-pr} * i$ by auto
  from arg-cong[OF Cons, of set] have $s \cdot s = n \div i$ by auto
  have $pp: \text{prime-product} (\text{factor-pr} * i)$
  unfolding prime-product-def
qed
proof
  fix \( m \)
  show \( \text{multiplicity} \ m \ (\text{factor-pr} \ * \ i) \leq 1 \)
    unfolding \( \text{mult} \) using \( \text{prems}(7) \)[of \( m \)] \( \text{prems}(8) \)[of \( m \)] \( \text{mult-i}(1) \)
    \( \text{mult-i}(2) \)[of \( m \)] by fastforce
  qed
  show \( ?\text{thesis} \)
    unfolding \( sq \ p \ \text{prems}(3) \) \( n \) unfolding \( n'\)-def \( s \)[symmetric]
    using \( pp \) by auto
next
  case Nil
  note simp = simp[unfolded Nil list.simps]
  from arg-cong[OF Nil, of set] have \( \neg (\exists \ x. \ x * x = n \ div \ i) \) by simp
  note IH = IH[OF Nil this - simp]
  show \( ?\text{thesis} \)
    proof (rule IH)
      show \( m = \text{factor-sq} \ * \ \text{factor-sq} \ * (\text{factor-pr} \ * \ i) \ * (n \ div \ i) \)
        unfolding \( \text{prems}(3) \) \( n \) by auto
    next
      fix \( j \)
      assume \( \ast: \ 2 \leq j \ < \ Suc \ i \)
      show \( \neg j \ dvd \ n \ \div \ i \)
        proof
          assume \( j: j \ dvd \ n \ \div \ i \)
          with \( False \) have \( j \neq i \) by auto
          with \( \ast \) have \( 2 \leq j \ < i \) by auto
          from \( \text{prems}(6) \)[OF this] \( j \)
          show \( False \)
            unfolding \( n \)
            by (metis dvd-mult \( n \) \( n'\)-def mult.commute)
        qed
    next
      fix \( j \)
      assume \( Suc \ i \leq j \)
      hence \( ij: i \leq j \) and \( j: j \neq i \) by auto
      have \( 0: \ \text{multiplicity} \ j \ i = 0 \) using \( \text{prime} \ j \) by (rule multiplicity-prime)
      show \( \text{multiplicity} \ j \ (\text{factor-pr} \ * \ i) = 0 \) unfolding \( \text{mult} \ \text{prems}(8) \)[OF \( ij \)]
        0 by simp
    next
      fix \( j \)
      assume \( j < Suc \ i \)
      hence \( j < i \ \vee \ j = i \) by auto
      thus \( \text{multiplicity} \ j \ (\text{factor-pr} \ * \ i) \leq 1 \)
        proof
          assume \( j = i \)
          with \( \text{prems}(8) \)[of \( i \)] \( \text{prime} \) show \( ?\text{thesis} \)
            unfolding \( \text{mult} \)
            by (auto)
        next
          assume \( ji: j < Suc \ i \)
          hence \( j \neq i \) by auto
          from \( \text{prems}(7) \)[OF \( ji \)] \( \text{multiplicity-prime} \)[OF \( \text{prime \ this} \)]
show \textit{thesis} unfolding \texttt{mult} by auto
qed
qed (insert prems(5,9), auto)
qed
qed
qed
next
case False
hence \((i \leq \text{limit} \land i \geq 2) = \text{False}\) by auto
note simp = simp[unfolded this if-False]

hence sq: \(sq = \text{factor-sq}\) and p: \(p = \text{factor-pr} \ast n\) by auto
show \textit{thesis}
proof
  show \(m = sq \ast sq \ast p\) unfolding sq p prems(3) by simp
  show \(\text{prime-product} p\) unfolding prime-product-def
  proof
    fix \(m\)
    from prems(1) have \(n1: n > 1\) by (cases \(n\), auto, case-tac nat, auto)
    hence \(n0: n > 0\) by auto
    have \(i > \text{limit}\) using False by auto
    from this[unfolded prems(2)] have less: \(\text{int} \ i \geq \text{root-nat-floor} 3 \ n + 1\) by auto
    have \(\text{int} \ n < (\text{root-nat-floor} 3 \ n + 1) ^ 3\) by (rule root-nat-floor-upper, auto)
    also have \(\ldots \leq \text{int} \ i ^ 3\) by (rule power-mono[OF less, of 3], auto)
    finally have \(n-i3: n < i ^ 3\)
    by (metis zless-int zpower-int)
  
  \{ 
    fix \(m\)
    assume \(m: \text{multiplicity} m \ n > 0\)
    hence \(mp: m \in \text{prime-factors} n\)
    by (metis less-not-refl multiplicity-not-factor-nat)
    hence \(md: m \mid d \ n\)
    by (metis k prime-factors-altdef2-nat)
    then obtain \(k\) where \(n: n = m \ast k\)
    from \(mp\) have \(pm: \text{prime} m\) by auto
    hence \(m2: m > 2\) and \(m0: m > 0\) by auto
    from prems(6)(OF m2) \(md\) have \(mi: m \geq i\) by force
  \}
  \{ 
    assume \(\text{multiplicity} m \ n \neq 1\)
    with \(m\) have \(\exists k, \text{multiplicity} m \ n = 2 + k\) by presburger
    then obtain \(j\) where \(\text{mult: multiplicity} m \ n = 2 + j\)
    from \(n0\) have \(k: k > 0\) by auto
    from \(\text{mult[unfolded n multiplicity-product-nat}(OF m0 k)]\) pm
    have \(\text{multiplicity} m \ k > 0\) by auto
    hence \(mp: m \in \text{prime-factors} k\)
    by (metis less-not-refl multiplicity-not-factor-nat)
    hence \(md: m \mid d \ k\)
    by (metis k prime-factors-altdef2-nat)
then obtain $l$ where $kml: k = m \ast l$.

Note $n = n[unfolded \ kml]$

From $n$ have $l \text{ dvd } n$ by auto

With premis(6)[of $l$] have $l \leq I \lor l \geq i$ by arith

With $n \# 0$ have $l: l = I \lor l \geq i$ by auto

From $n$ premis(1) have $l \neq 1$ by auto

With $l$ have $l \geq 1 \lor l \geq i$ by auto

With $n \# 0$ have $l \neq 1$ by auto

With $l$ have $l \geq i$ by auto

From mult-le-mono[OF mult-le-mono[OF mi mi] $l$]

Have $n \geq i^3$ unfolding $n$ by (auto simp: power3-eq-cube)

With $n \# 3$ have $False$ by auto

Note $n = this$

Have multiplicity $m \# n = 1 \land m \geq i$ by auto

Also have $\ldots \leq 1$

Proof (cases $m < i$)

Case True

From premis(7)[of $m$] $n[\text{of } m]$ True show ?thesis by force

Next

Case False

Hence $m \geq i$ by auto

From premis(8)[OF this] $n[\text{of } m]$ show ?thesis by force

Qed

Finally show multiplicity $m \# p \leq 1$.

Qed

Qed

Qed

Definition prime-product-factor :: nat $\Rightarrow$ nat $\times$ nat where

prime-product-factor $n = (\text{case sqrt-nat } n \text{ of}$

(Cons $s$ $\Rightarrow$ $(s,1)$

| $[] \Rightarrow$ prime-product-factor-main 1 1 (nat (root-nat-floor 3 $n$)) $n$ 2)

Lemma prime-product-one[simp, intro]: prime-product 1

Unfolding prime-product-def multiplicity-one-nat by auto

Lemma prime-product-factor: assumes pf: prime-product-factor $n = (sq,p)$ shows $n = sq \ast sq \ast p \land$ prime-product $p$

Proof (cases sqrt-nat $n$)

Case (Cons $s$)

From pf[unfolded prime-product-factor-def Cons] arg-cong[OF Cons, of set]

prime-product-one

Show ?thesis by auto

definition prime-product-factor :: nat $\Rightarrow$ nat $\times$ nat where

prime-product-factor $n = (\text{case sqrt-nat } n \text{ of}$

(Cons $s$ $\Rightarrow$ $(s,1)$

| $[] \Rightarrow$ prime-product-factor-main 1 1 (nat (root-nat-floor 3 $n$)) $n$ 2)

lemma prime-product-one[simp, intro]: prime-product 1

unfolding prime-product-def multiplicity-one-nat by auto

lemma prime-product-factor: assumes pf: prime-product-factor $n = (sq,p)$ shows $n = sq \ast sq \ast p \land$ prime-product $p$

proof (cases sqrt-nat $n$)

case (Cons $s$)

from pf[unfolded prime-product-factor-def Cons] arg-cong[OF Cons, of set]

prime-product-one

show ?thesis by auto
4 A representation of real numbers via triples

theory Real-Impl
imports
../Sqrt-Babylonian/Sqrt-Babylonian
begin

We represent real numbers of the form $p + q \cdot \sqrt{b}$ for $p, q \in \mathbb{Q}$, $n \in \mathbb{N}$ by triples $(p, q, b)$. However, we require the invariant that $\sqrt{b}$ is irrational. Most binary operations are implemented via partial functions where the common restriction is that the numbers $b$ in both triples have to be identical. So, we support addition of $\sqrt{2} + \sqrt{2}$, but not $\sqrt{2} + \sqrt{3}$.

The set of natural numbers whose $\sqrt{}$ is irrational

lemma sqrt-irrat: assumes choice: $q = 0 \lor b \in \text{sqrt-irrat}$
and eq: real-of-rat $p + \text{real-of-rat } q \cdot \sqrt{\text{of-nat } b} = 0$
shows $q = 0$
using choice
proof (cases $q = 0$)
case False
with choice have sqrt-irrat: $b \in \text{sqrt-irrat}$ by blast
from eq have real-of-rat $q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } (\sim p)$
by (auto simp: of-rat-minus)
then obtain $p$ where real-of-rat $q \cdot \sqrt{\text{of-nat } b} = \text{real-of-rat } p$ by blast
from arg-cong[of this, of $\lambda x. x \cdot x$] have real-of-rat $(q * q) * (\sqrt{\text{of-nat } b})$
* sqrt (of-nat b) =
  real-of-rat (p * p) by (auto simp: field-simps of-rat-mult)
also have sqrt (of-nat b) * sqrt (of-nat b) = of-nat b by simp
finally have real-of-rat $(q * q) * \text{rat-of-nat } b = \text{real-of-rat } (p * p)$ by (auto simp: of-rat-mult)
  hence $q * q * \text{rat-of-nat } b = p * p$ by auto
from arg-cong[of this, of $\lambda x. x / (q * q)$]
have $(p / q) * (p / q) = \text{rat-of-nat } b$ using False by (auto simp: field-simps)
with sqrt-irrat show ?thesis unfolding sqrt-irrat-def by blast
qed

Collect existing code equations for reals, so that they can be deleted.
lemmas real-code-dels =
  refl[of op + :: real ⇒ real ⇒ real]
  refl[of uminus :: real ⇒ real]
  refl[of op - :: real ⇒ real ⇒ real]
  refl[of op * :: real ⇒ real ⇒ real]
  refl[of inverse :: real ⇒ real]
  refl[of op / :: real ⇒ real ⇒ real]
  refl[of floor :: real ⇒ int]
  refl[of sqrt]
  refl[of HOL.equal :: real ⇒ real ⇒ bool]
  refl[of op ≤ :: real ⇒ real ⇒ bool]
  refl[of op < :: real ⇒ real ⇒ bool]
  refl[of 0 :: real]
  refl[of 1 :: real]

lemma real-code-unfold-dels:
of-rat ≡ Ratreal
of-int a ≡ (of-rat (of-int a) :: real)
0 ≡ (of-rat 0 :: real)
1 ≡ (of-rat 1 :: real)
numeral k ≡ (of-rat (numeral k) :: real)
− numeral k ≡ (of-rat (− numeral k) :: real)
by simp-all

lemma real-standard-impls:
(x :: real) / (y :: real) = x * inverse y
(x :: real) − (y :: real) = x + (− y)
by (simp-all add: divide-inverse)

To represent numbers of the form \( p + q \sqrt{b} \), use mini algebraic numbers, i.e., triples \((p, q, b)\) with irrational \(\sqrt{b}\).
typedef mini-alg =
\{(p, q, b) | (p :: rat) (q :: rat) (b :: nat).
  q = 0 ∨ b ∈ sqrt-irrat\}
by auto

setup-lifting type-definition-mini-alg

lift-definition real-of :: mini-alg ⇒ real is
  \( \lambda (p,q,b). \ of-rat p + of-rat q * sqrt (of-nat b) \).

lift-definition ma-of-rat :: rat ⇒ mini-alg is \( \lambda x. (x,0,0) \) by auto

lift-definition ma-rat :: mini-alg ⇒ rat is fst .
lift-definition ma-base :: mini-alg ⇒ nat is snd o snd .
lift-definition ma-coeff :: mini-alg ⇒ rat is fst o snd .
lift-definition ma-uminus :: mini-alg ⇒ mini-alg is
  \( \lambda (p1,q1,b1). (− p1, − q1, b1) \) by auto
lift-definition ma-compatible :: mini-alg ⇒ mini-alg ⇒ bool is
  \( \lambda (p1,q1,b1) \ (p2,q2,b2). \ q1 = 0 \lor q2 = 0 \lor b1 = b2 \). 

definition ma-normalize :: \( \text{rat} \times \text{rat} \times \text{nat} \) ⇒ \( \text{rat} \times \text{rat} \times \text{nat} \) where
  ma-normalize \( x = \text{case } x \text{ of } (a,b,c) \Rightarrow \text{if } b = 0 \text{ then } (a,0,0) \text{ else } (a,b,c) \)

lemma ma-normalize-case[simp]: (case ma-normalize \( r \) of \( (a,b,c) \Rightarrow \text{real-of-rat } a + \text{real-of-rat } b \ast \text{sqrt } \) (of-nat \( c \))
  = (case \( r \) of \( (a,b,c) \Rightarrow \text{real-of-rat } a + \text{real-of-rat } b \ast \text{sqrt } \) (of-nat \( c \))
  by (cases \( r \), auto simp: ma-normalize-def)

lift-definition ma-plus :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  \( \lambda (p1,q1,b1) \ (p2,q2,b2). \ \text{if } q1 = 0 \text{ then } \) (\( p1 + p2, q2, b2 \)) else ma-normalize (\( p1 + p2, q1 + q2, b1 \)) by (auto simp: ma-normalize-def)

lift-definition ma-times :: mini-alg ⇒ mini-alg ⇒ mini-alg is
  \( \lambda (p1,q1,b1) \ (p2,q2,b2). \ \text{if } q1 = 0 \text{ then } \) ma-normalize (\( p1\ast p2, p1\ast q2, b2 \)) else ma-normalize (\( p1\ast p2 + \text{of-nat } b2\ast q1\ast q2, p1\ast q2 + q1\ast p2, b1 \)) by (auto simp: ma-normalize-def)

lift-definition ma-inverse :: mini-alg ⇒ mini-alg is
  \( \lambda (p,q,b). \ \text{let } d = \text{inverse } (p \ast p - \text{of-nat } b \ast q \ast q) \text{ in } \) ma-normalize (\( p \ast d, -q \ast d, b \)) by (auto simp: Let-def ma-normalize-def)

lift-definition ma-floor :: mini-alg ⇒ int is
  \( \lambda (p,q,b). \ \text{case } (\text{quotient-of } p, \text{quotient-of } q) \text{ of } (\text{Cons } s \ -) \Rightarrow \) let \( z2n1 = z2 \ast n1; z1n2 = z1 \ast n2; n12 = n1 \ast n2; \) \( \text{prod} = z2n1 \ast z2n1 \ast \) int \( b \) in \( (z1n2 + (\text{if } z2n1 \geq 0 \text{ then } \text{sqrt-int-floor-pos } \text{prod} \text{ else } \text{-sqrt-int-ceiling-pos } \text{prod}) \) \( \text{div} n12 \) .

lift-definition ma-sqrt :: mini-alg ⇒ mini-alg is
  \( \lambda (p,q,b). \ \text{let } (a,b) = \text{quotient-of } p, a = \text{abs } (a \ast b) \) in \( \text{case } \text{sqrt-int } aa \text{ of } [] \Rightarrow (0, \text{inverse } (\text{of-int } b) \text{,nat } aa) | (\text{Cons } s \ -) \Rightarrow (\text{of-int } s / \text{of-int } b,0,0) \)

proof (unfold Let-def)
  fix prod :: \( \text{rat} \times \text{rat} \times \text{nat} \)
  obtain \( p \ q \ b \) where \( \text{prod} = (p,q,b) \) by (cases \( \text{prod} \), auto)
  obtain \( a \ b \) where \( p: \text{quotient-of } p = (a,b) \) by force
  show \( \exists p \ q \ b. \ (\text{case } \text{prod of } (p, q, b) \Rightarrow \text{case } \text{quotient-of } p \text{ of } (a, b) \Rightarrow \) (\( \text{case } \text{sqrt-int } |a \ast b| \text{ of } [] \Rightarrow (0, \text{inverse } (\text{of-int } b), \text{nat } |a \ast b|) | s \ # x \Rightarrow (\text{of-int } s / \text{of-int } b, 0, 0)\)) = 
  (p, q, b) ∧
\[(q = 0 \lor b \in \text{sqrt-irrat})\]

**proof** (cases sqrt-int \((a + b)\))

**case** \(\text{Nil}\)

from sqrt-int[of abs \((a + b)\)] \(\text{Nil}\) **have** \(\text{irrat} : \neg (\exists y. y \ast y = |a + b|)\) **by** auto

**have** \(|a + b| \in \text{sqrt-irrat}\)

**proof** (rule ccontr)

**assume** \(|a + b| \notin \text{sqrt-irrat}\)

then obtain \(x :: \text{rat}\)

where \(x \ast x = \text{rat-of-nat} (\text{nat} \ |a + b|)\) unfolding sqrt-irrat-def **by** auto

**hence** \(x \ast x = \text{rat-of-int} |a + b|\) **by** auto

from sqrt-rat-of-int[\(\text{OF this}\)] \(\text{irrat}\) **show** \(\text{False}\) **by** blast

**qed**

**thus** \(?\text{thesis}\) using \(\text{Nil}\) **by** \((\text{auto simp: prod p})\)

**qed** (auto simp: prod p \(\text{Cons}\))

**qed**

**lift-definition** \(\text{ma-equal} :: \text{mini-alg} \Rightarrow \text{mini-alg} \Rightarrow \text{bool}\) is

\[\lambda (p1.q1.b1) (p2.q2.b2). p1 = p2 \land q1 = q2 \land (q1 = 0 \lor b1 = b2) .\]

**lift-definition** \(\text{ma-ge-0} :: \text{mini-alg} \Rightarrow \text{bool}\) is

\[\lambda (p.q.b). \text{let } bqq = \text{of-nat} b \ast q \ast q; pp = p \ast p \text{ in}\]

\[0 \leq p \land bqq \leq pp \land 0 \leq q \land pp \leq bqq .\]

**lift-definition** \(\text{ma-is-rat} :: \text{mini-alg} \Rightarrow \text{bool}\) is

\[\lambda (p.q.b). q = 0 .\]

**definition** \(\text{ge-0} :: \text{real} \Rightarrow \text{bool}\) where \([\text{code def}]: \text{ge-0} x = (x \geq 0)\)

**lemma** \(\text{ma-ge-0: ge-0} \ (\text{real-of-x}) = \text{ma-ge-0 x}\)

**proof** (transfer, unfold Let-def, clarsimp)

fix \(p.q.q' :: \text{rat}\) and \(b' :: \text{nat}\)

**assume** \(b'. q' = 0 \lor b' \in \text{sqrt-irrat}\)

**def** \(b \equiv \text{real-of-nat} b'\)

**def** \(p \equiv \text{real-of-rat} p'\)

**def** \(q \equiv \text{real-of-rat} q'\)

from \(b'\) **have** \(b: 0 \leq b = 0 \lor b' \in \text{sqrt-irrat}\) unfolding \(\text{b-def} q\text{-def by auto}\)

**def** \(q_b \equiv q \ast \text{sqrt} b\)

from \(b\) **have** \(\text{sqrt}\) \(\sqrt b \geq 0\) **by** auto

from \(b(2)\) **have** \(\text{disj}: q = 0 \lor b \neq 0\) unfolding \(\text{sqrt-irrat-def b-def by auto}\)

**have** \(b\text{def}: b = \text{real-of-rat} (\text{of-nat} b')\) unfolding \(\text{b-def by auto}\)

**have** \(\(0 \leq p + q \ast \text{sqrt} b\) = (0 \leq p + q_b)\) unfolding \(\text{qb-def by simp}\)

**also have** \(\ldots\) \(\leftrightarrow\) \(\(0 \leq p \land \text{abs} q_b \leq \text{abs} p \lor 0 \leq q_b \land \text{abs} p \leq \text{abs} q_b\)\) **by** arith

**also have** \(\ldots\) \(\leftrightarrow\) \(\(0 \leq p \land qb \ast q_b \leq p \ast p \lor 0 \leq q_b \land p \ast p \leq q_b \ast qb\)\)

unfolding \(\text{abs-lesseq-square} \ldots\)

**also have** \(q_b \ast q_b = b \ast q \ast q\) unfolding \(\text{qb-def}\)

using \(b\) **by** auto

**also have** \(0 \leq q_b \leftrightarrow 0 \leq q\) unfolding \(\text{qb-def}\) **using** \(\text{sqrt disj}\)
by (metis le-cases mult-eq-0-iff mult-nonneg-nonneg mult-nonpos-nonneg order-class.order.antisym
qb-def real-sqrt-eq-zero-cancel-iff)
also have \( (0 \leq p \land b \ast q \land q \leq p \ast p \lor 0 \leq q \land p \ast p \leq b \ast q \ast q) \)
\(<\rightarrow (0 \leq p' \land \text{of-nat } b' \ast q' \ast q' \leq p' \ast p' \lor 0 \leq q' \land p' \ast p' \leq \text{of-nat } b' \ast q' \ast q') \)
unfolding qb-def
by (unfold bdef p-def q-def of-rat-mult[symmetric] of-rat-less-eq, simp)
finally
show \( \text{ge-0 } (\text{real-of-rat } p' + \text{real-of-rat } q' \ast \sqrt{\text{of-nat } b'}) = \)
\((0 \leq p' \land \text{of-nat } b' \ast q' \ast q' \leq p' \ast p' \lor 0 \leq q' \land p' \ast p' \leq \text{of-nat } b' \ast q' \ast q') \)
unfolding ge-0-def p-def b-def q-def
by (auto simp: of-rat-add of-rat-mult)
qed

lemma ma-0: \( 0 = \text{real-of } (\text{ma-of-rat } 0) \) by (transfer, auto)

lemma ma-1: \( 1 = \text{real-of } (\text{ma-of-rat } 1) \) by (transfer, auto)

lemma ma-uminus:
\( \neg (\text{real-of } x) = \text{real-of } (\text{ma-uminus } x) \)
by (transfer, auto simp: of-rat-minus)

lemma ma-inverse: inverse (real-of r) = real-of (ma-inverse r)
proof (transfer, unfold Let-def, clarsimp)
fix \( p' q' :: \text{rat and } b' :: \text{nat} \)
assume \( b' : q' \geq 0 \lor b' \in \sqrt{\text{irrat}} \)
def \( b \equiv \text{real-of-nat } b' \)
def \( p \equiv \text{real-of-rat } p' \)
def \( q \equiv \text{real-of-rat } q' \)
from \( b' \) have \( b : b \geq 0 q = 0 \lor b' \in \sqrt{\text{irrat}} \) unfolding b-def q-def by auto
have inverse \( (p + q \ast \sqrt{b}) = (p - q \ast \sqrt{b}) \ast \text{inverse } (p \ast p - b \ast (q \ast q)) \)
proof (cases \( q = 0 \))
case True thus \(?thesis \) by (cases \( p = 0 \), auto simp: field-simps)
next
case False
from \( \sqrt{\text{irrat}}[OF b', \text{of p'}] \) b-def q-def False have \( \text{null: } p + q \ast \sqrt{b} \neq 0 \) by auto
have \(?thesis \leftarrow \rightarrow (p + q \ast \sqrt{b}) \ast \text{inverse } (p + q \ast \sqrt{b}) = \)
\((p + q \ast \sqrt{b}) \ast ((p - q \ast \sqrt{b}) \ast \text{inverse } (p \ast p - b \ast (q \ast q))) \)
unfolding mult-left-cancel[OF null] by auto
also have \((p + q \ast \sqrt{b}) \ast \text{inverse } (p + q \ast \sqrt{b}) = 1 \) using null by auto
also have \((p + q \ast \sqrt{b}) \ast ((p - q \ast \sqrt{b}) \ast \text{inverse } (p \ast p - b \ast (q \ast q))) = \)
\((p \ast p - b \ast (q \ast q)) \ast \text{inverse } (p \ast p - b \ast (q \ast q)) \)
using \( b \) by (auto simp: field-simps)
also have \( ... = 1 \)
proof (rule right-inverse, rule)
assume eq: \( p \ast p - b \ast (q \ast q) = 0 \)
have \( \text{real-of-rat } (p' \ast p' / (q' \ast q')) = p \ast p / (q \ast q) \)
unfolding p-def b-def q-def by (auto simp: of-rat-mult of-rat-divide)
also have ... = b using eq False by (auto simp: field-simps)
also have ... = real-of-rat (of-nat b') unfolding b-def by auto
finally have \((p' / q') * \(p' / q'\) = of-nat b'\)
  unfolding of-rat-eq-iff by simp
with b False show False unfolding sqrt-irrat-def by blast
qed
finally
  show ?thesis by simp
qed

thus inverse (real-of-rat p' + real-of-rat q' * sqrt (of-nat b')) =
  real-of-rat (p' * inverse (p' * p' - of-nat b' * q' * q')) +
  real-of-rat (- (q' * inverse (p' * p' - of-nat b' * q' * q'))) * sqrt (of-nat b')
by (simp add: divide-simps of-rat-mult of-rat-divide of-rat-diff of-rat-minus b-def)
p-def q-def
  split: if-splits
qed

lemma ma-sqrt-main: ma-rat r ≥ 0 =⇒ ma-coeff r = 0 =⇒ sqrt (real-of r) =
  real-of (ma-sqrt r)
proof (transfer, unfold Let-def, clarsimp)
fix p :: rat
assume p: 0 ≤ p
hence abs: abs p = p by auto
obtain a b where ab: quotient-of p = (a,b) by force
hence pab: p = of-int a / of-int b by (rule quotient-of-div)
from ab have b: b > 0 by (rule quotient-of-denom-pos)
with p pab have abpos: a * b ≥ 0
  by (metis of-int-0-le-iff of-int-le-0-iff zero-le-divide-iff zero-le-mult-iff)
have rab: of-nat (nat (a * b)) = real-of-int a * real-of-int b using abpos
  by (metis of-int-mult of-nat-nat)
let ?lhs = sqrt (of-int a / of-int b)
let ?rhs = (case case quotient-of p of
  (a, b) ⇒ (case sqrt-int [a * b] of [] ⇒ (0, inverse (of-int b), nat [a * b])
   | s # x ⇒ (of-int s / of-int b, 0, 0)) of
    (p, q, b) ⇒ of-rat p + of-rat q * sqrt (of-nat b))
have sqrt (real-of-rat p) = ?lhs unfolding pab
  by (metis of-rat-divide of-rat-of-int-eq)
also have ... = ?rhs
proof (cases sqrt-int [a * b])
case Nil
def sb ≡ sqrt (of-int b)
def sa ≡ sqrt (of-int a)
from b sb-def have sb: sb > 0 real-of-int b > 0 by auto
have sbb: sb * sb = real-of-int b unfolding sb-def
  by (rule sqrt-sqrt, insert b, auto)
from Nil have ?thesis = (sa / sb =
  inverse (of-int b) * (sa * sb)) unfolding ab sa-def sb-def using abpos
by (simp add: lub of-rat-divide real-sqrt-mult real-sqrt-divide of-rat-inverse)
also have \ldots = (sa = inverse (of-int b) \ast sa \ast (sb \ast sb)) using sb
  by (metis b divide-real-def eq-divide-imp inverse-divide inverse-inverse-eq
  inverse-mult-distrib less-int-code(1) of-int-eq-0-iff real-sqrt-eq-zero-cancel-iff sb-def
  sbb times-divide-eq-right)
also have \ldots = True using sb(2) unfolding sbb by auto
finally show \textit{thesis} by simp
next
case (Cons s x)
from b have b: real-of-int b > 0 by auto
from Cons sqrt-int\[\text{of abs}(a \ast b)] have s * s = \text{abs}(a \ast b) by auto
with sqrt-int-pos[OF Cons] have sqrt (real-of-int (abs (a \ast b))) = of-int s
  by (metis \text{abs}\-of\-nonneg of-int-real-eq0-real-eq0-cancel-iff
  real-sqrt-abs2)
with abpos have of-int s = sqrt (real-of-int (a \ast b)) by auto
thus \textit{thesis} unfolding ab split using Cons b
  by (auto simp: real-sqrt-minus)
qed
finally show sqrt (real-of-rat p) = \textit{rhs}.
qed

lemma ma-sqrt: sqrt (real-of r) = (if ma-coeff r = 0 then
  (if ma-rat r \geq 0 then real-of (ma-sqrt r) else - real-of (ma-sqrt (ma-uminus r)))
  else Code.abort (STR "cannot represent sqrt of irrational number!") (\\- sqrt (real-of r)))
proof (cases ma-coeff r = 0)
case True note 0 = this
hence 00: ma-coeff (ma-uminus r) = 0 by (transfer, auto)
show \textit{thesis}
proof (cases ma-rat r \geq 0)
case True
from ma-sqrt-main[OF this 0] 0 True show \textit{thesis} by auto
next
case False
hence ma-rat (ma-uminus r) \geq 0 by (transfer, auto)
from ma-sqrt-main[OF this 00, folded ma-uminus, symmetric] False 0
show \textit{thesis} by (auto simp: real-sqrt-minus)
qed
qed auto

lemma ma-plus:
(real-of r1 + real-of r2) = (if ma-compatible r1 r2
  then real-of (ma-plus r1 r2) else
  Code.abort (STR "different base") (\\- real-of r1 + real-of r2))
by transfer (auto split: prod.split simp: field-simps of-rat-add)

lemma ma-times:
(real-of r1 \ast real-of r2) = (if ma-compatible r1 r2
  then real-of (ma-times r1 r2) else
Code.abort (STR "different base") (λ -. real-of-r1 * real-of-r2))
by transfer (auto split: prod.split simp: field-simps of-rat-mult of-rat-add)

lemma ma-equal:
HOL.equal (real-of r1) (real-of r2) = (if ma-compatible r1 r2
then ma-equal r1 r2 else
Code.abort (STR "different base") (λ -. HOL.equal (real-of r1) (real-of r2)))

proof (transfer, unfold equal-real-def, clarsimp)
fix p1 q1 p2 q2 :: rat and b1 b2 :: nat
assume b1: q1 = 0 ∨ b1 ∈ sqrt-irrat
assume b2: q2 = 0 ∨ b2 ∈ sqrt-irrat
assume base: q1 = 0 ∨ q2 = 0 ∨ b1 = b2
let ?l = real-of-rat p1 + real-of-rat q1 * sqrt (of-nat b1) =
real-of-rat p2 + real-of-rat q2 * sqrt (of-nat b2)
let ?m = real-of-rat q1 * sqrt (of-nat b1) = real-of-rat (p2 − p1) + real-of-rat
q2 * sqrt (of-nat b2)
let ?r = p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
have ?l ⇔ real-of-rat q1 * sqrt (of-nat b1) = real-of-rat (p2 − p1) + real-of-rat
q2 * sqrt (of-nat b2)
by (auto simp: of-rat-add of-rat-diff of-rat-minus)
also have ... ⇔ p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
proof
assume ?m
from base have q1 = 0 ∨ q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
by auto
thus p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2)
proof
assume q1: q1 = 0
with (?m) have real-of-rat (p2 − p1) + real-of-rat q2 * sqrt (of-nat b2) = 0
by auto
with sqrt-irrat b2 have q2: q2 = 0 by auto
with q1 (?m) show ?thesis by auto
next
assume q1 ≠ 0 ∧ q2 = 0 ∨ q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
thus ?thesis
proof
assume ass: q1 ≠ 0 ∧ q2 = 0
with (?m) have real-of-rat (p1 − p2) + real-of-rat q1 * sqrt (of-nat b1) = 0
by (auto simp: of-rat-diff)
with b1 have q1 = 0 using sqrt-irrat by auto
with ass show ?thesis by auto
next
assume ass: q1 ≠ 0 ∧ q2 ≠ 0 ∧ b1 = b2
with (?m) have *: real-of-rat (p2 − p1) + real-of-rat (q2 − q1) * sqrt (of-nat b2) = 0
by (auto simp: field-simps of-rat-diff)
have q2 − q1 = 0
by (rule sqrt-irrat[OF *], insert ass b2, auto)
with * ass show ?thesis by auto
qed
qed auto
finally show $?l = ?r by simp
qed

lemma ma-floor: floor (real-of r) = ma-floor r
proof (transfer, unfold Let-def, clarsimp)
fix p :: rat and b :: nat
obtain z1 n1 where qp: quotient-of p = (z1,n1) by force
obtain z2 n2 where qq: quotient-of q = (z2,n2) by force
from quotient-of-denom-pos[OF qp] have n1: 0 < n1 .
from quotient-of-denom-pos[OF qq] have n2: 0 < n2 .
from quotient-of-div[OF qp] have p: p = of-int z1 / of-int n1 .
from quotient-of-div[OF qq] have q: q = of-int z2 / of-int n2 .
have p: p = of-int (z1 * n2) / of-int (n1 * n2) unfolding p using n2 by auto
have q: q = of-int (z2 * n1) / of-int (n1 * n2) unfolding q using n1 by auto
def z1n2 = z1 * n2
def z2n1 = z2 * n1
def n12 = n1 * n2
def r-add = of-int (z2n1) * sqrt (real (int b))
from n1 n2 have n12: n12 > 0 unfolding n12-def by simp
from n1 n2 have n20: n12 > 0 unfolding n12-def by simp
unfolding r-add-def z1n2-def z2n1-def
unfolding p q add-divide-distrib of-rat-divide of-rat-int-eq real-of-int-of-nat-eq
real-eq-of-nat by simp
also have ... = floor (of-int z1n2 + r-add) div n12
  by (rule floor-div-pos-int[OF n120])
also have of-int z1n2 + r-add = r-add + of-int z1n2 by simp
also have floor (...) = floor r-add + z1n2 by simp
also have ... = z1n2 + floor r-add by simp
finally have id: [of-rat p + of-rat q * sqrt (of-nat b)] = (z1n2 + [r-add]) div n12 .
show [of-rat p + of-rat q * sqrt (of-nat b)] =
  (case quotient-of p of
    (z1, n1) \Rightarrow
    case quotient-of q of
    (z2, n2) \Rightarrow
    (z1 * n2 + (if 0 ≤ z2 * n1 then sqrt-int-floor-pos (z2 * n1 * (z2 * n1) * int b) else 
      sqrt-int-ceiling-pos (z2 * n1 * (z2 * n1) * int b))) div (n1 * n2))
unfolding qp qq split id n12-def z1n2-def
proof (rule arg-cong[\(\lambda\) x. \(((z1 * n2) + x) \div (n1 * n2)\)])
have ge-int: z2 * n1 * (z2 * n1) * int b ≥ 0
  by (metis mult-nonneg-nonneg zero-le-square zero-zle-int)
show [r-add] = (if 0 ≤ z2 * n1 then sqrt-int-floor-pos (z2 * n1 * (z2 * n1))
proof (cases \(z_2 \times n_1 \geq 0\))

case True

hence \(\text{ge} : \text{real-of-int} (z_2 \times n_1) \geq 0\) by (metis \text{of-int-0-le-iff})

have \(\text{radd} : \text{r-add} = \sqrt{\text{of-int} (z_2 \times n_1 \times (z_2 \times n_1) \times \text{int} b)}\)

unfolding \(\text{r-add-def} z_2 n_1\)-def using \(\text{sqrt-sqrt} [\text{OF ge}]\)

by (simp add: \text{ac-simps real-eq-of-int real-sqrt-mult-distrib2})

show \(\text{?thesis}\) unfolding \(\text{radd sq}\)-int-floor-pos [\text{OF ge-int}] \text{real-eq-of-int using True by simp}\n
next

case False

hence \(\text{ge} : \text{real-of-int} (- (z_2 \times n_1)) \geq 0\) by (metis mult-zero-left neg-0-le-iff \text{of-int-0-le-iff order-refl zero-le-mult-iff})

have \(\text{r-add} = - \sqrt{\text{of-int} (z_2 \times n_1 \times (z_2 \times n_1) \times \text{int} b)}\)

unfolding \(\text{r-add-def} z_2 n_1\)-def using \(\text{sqrt-sqrt} [\text{OF ge}]\)

by (metis floor minus-minus \text{minus-mult-commute minus-mult-right of-int-minus of-int-mult real-of-int-def real-sqrt-minus real-sqrt-mult-distrib2 z_2 n_1-def})

hence \(\text{radd} : \text{floor r-add} = - \text{ceiling} (\sqrt{\text{of-int} (z_2 \times n_1 \times (z_2 \times n_1) \times \text{int} b)})\)

by (metis \text{ceiling-def minus-minus})

show \(\text{?thesis}\) unfolding \(\text{radd sq}\)-int-floor-pos [\text{OF ge-int}] \text{real-eq-of-int using False by simp}\n
qed

qed

lemma \text{comparison-impl}:\n
\((x :: \text{real}) \leq (y :: \text{real}) = \text{ge-0} (y - x)\)

\((x :: \text{real}) < (y :: \text{real}) = (x \neq y \land \text{ge-0} (y - x))\)

by (simp-all add: \text{ge-0-def , linarith+})

lemma \text{ma-of-rat} : \text{real-of-rat} r = \text{real-of} (\text{ma-of-rat} r)

by (transfer , auto)

definition \text{is-rat} :: \text{real} \Rightarrow \text{bool where}\n
[code del]: \text{is-rat} x = (x \in \mathbb{Q})

lemma [code-unfold]: \(x \in \mathbb{Q} \longleftrightarrow \text{is-rat x}\) unfolding \text{is-rat-def} by auto

lemma \text{ma-is-rat} : \text{is-rat} (\text{real-of} x) = \text{ma-is-rat} x

proof (transfer , unfold \text{is-rat-def} , clarsimp)

fix \(p \ q :: \text{rat} \land b :: \text{nat}\)

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

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

let \(?b = \text{real-of-nat} b\)

let \(?b' = \text{real-of-rat} (\text{of-nat} b)\)

assume \(b : q = 0 \lor b \in \text{sqrt-irrat}\)

show \((?p + ?q \times ?b \in \mathbb{Q}) = (q = 0)\)

proof (cases \(q = 0\))
\[
\text{case False from False } b \text{ have } b \in \sqrt{-\text{irrat}} \text{ by auto }
\]
\[
\text{assume } ?p + ?q \star \sqrt{?b} \in \mathbb{Q} \text{ from this[unfolded Rats-def] obtain } r \text{ where } r: ?p + ?q \star \sqrt{?b} = \text{real-of-rat } r \text{ by auto }
\]
\[
\text{let } ?r = \text{real-of-rat } r \text{ from } r \text{ have real-of-rat } (p - r) + ?q \star \sqrt{?b} = 0 \text{ by (simp add: of-rat-diff) }
\]
\[
\text{from sqrt-irrat[OF disjI2[OF b] this] False have False by auto }
\]
\[
\text{thus } ?\text{thesis by auto qed auto }
\]

\text{definition sqrt-real } x = (if } x \in \mathbb{Q} \land x \geq 0 \text{ then (if } x = 0 \text{ then } [0] \text{ else (let } sx = \sqrt{x} \text{ in } [sx, -sx]]) \text{ else []) }

\text{lemma sqrt-real[simp]: assumes } x: x \in \mathbb{Q} \text{ shows set (sqrt-real } x) = \{y . y \star y = x\}

\text{proof (cases } x \geq 0\text{) }
\text{case False }
\text{hence } \land y. y \star y \neq x \text{ by auto with False show } ?\text{thesis unfolding sqrt-real-def by auto}
\text{next }
\text{case True }
\text{thus } ?\text{thesis using } x
\text{ by (cases } x = 0\text{, auto simp: Let-def sqrt-real-def)}
\text{qed}

\text{lemmas ma-code-eqns = ma-ge-0 ma-floor ma-0 ma-1 ma-uminus ma-inverse ma-sqrt ma-plus ma-times ma-equal ma-is-rat}
\text{comparison-impl}

\text{code-datatype real-of}
\text{declare real-code-dels[code, code del]}
\text{declare real-code-unfold-dels[code-unfold del]}
\text{declare real-standard-impls[code]}
\text{declare ma-code-eqns[code]}

Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

\text{value } [101.1 \star (3 \star \sqrt{2} + 6 \star \sqrt{0.5})]
\text{value } [606.2 \star \sqrt{2} + 0.001]
\text{value } 101.1 \star (3 \star \sqrt{2} + 6 \star \sqrt{0.5}) = 606.2 \star \sqrt{2} + 0.001
\text{value } 101.1 \star (3 \star \sqrt{2} + 6 \star \sqrt{0.5}) > 606.2 \star \sqrt{2} + 0.001
\text{value } (\sqrt{0.1} \in \mathbb{Q}, \sqrt{-0.09} \in \mathbb{Q})
5 A unique representation of real numbers via triples

theory Real-Unique-Impl
imports
  Prime-Product
  Real-Impl
  ../Show/Show-Instances
begin

  We implement the real numbers again using triples, but now we require an additional invariant on the triples, namely that the base has to be a prime product. This has the consequence that the mapping of triples into $\mathbb{R}$ is injective. Hence, equality on reals is now equality on triples, which can even be executed in case of different bases. Similarly, we now also allow different basis in comparisons. Ultimately, injectivity allows us to define a show-function for real numbers, which pretty prints real numbers into strings.

typedef mini-alg-unique = 
  \{ r :: mini-alg . ma-coeff r = 0 ∧ ma-base r = 0 ∨ ma-coeff r ≠ 0 ∧ prime-product (ma-base r) \}
  by (transfer, auto)

setup-lifting type-definition-mini-alg-unique

lift-definition real-of-u :: mini-alg-unique ⇒ real is real-of .
lift-definition mau-floor :: mini-alg-unique ⇒ int is ma-floor .
lift-definition mau-of-rat :: rat ⇒ mini-alg-unique is ma-of-rat by (transfer, auto)
lift-definition mau-rat :: mini-alg-unique ⇒ rat is ma-rat .
lift-definition mau-base :: mini-alg-unique ⇒ nat is ma-base .
lift-definition mau-coeff :: mini-alg-unique ⇒ rat is ma-coeff .
lift-definition mau-uminus :: mini-alg-unique ⇒ mini-alg-unique is ma-uminus by (transfer, auto)
lift-definition mau-compatible :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is ma-compatible .
lift-definition mau-ge-0 :: mini-alg-unique ⇒ bool is ma-ge-0 .
lift-definition mau-inverse :: mini-alg-unique ⇒ mini-alg-unique is ma-inverse
  by (transfer, auto simp: ma-normalize-def Let-def split: if-splits)
lift-definition mau-plus :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
  is ma-plus
  by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition mau-times :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
  is ma-times
  by (transfer, auto simp: ma-normalize-def split: if-splits)
lift-definition ma-identity :: mini-alg ⇒ mini-alg ⇒ bool is op = .
lift-definition mau-equal :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is ma-identity.

lift-definition mau-is-rat :: mini-alg-unique ⇒ bool is ma-is-rat.

lemma mau-floor: floor (real-of-u r) = mau-floor r
using ma-floor by (transfer, auto)

lemma mau-inverse: inverse (real-of-u r) = real-of-u (mau-inverse r)
using ma-inverse by (transfer, auto)

lemma mau-uminus: − (real-of-u r) = real-of-u (mau-uminus r)
using ma-uminus by (transfer, auto)

lemma mau-times:
(real-of-u r1 * real-of-u r2) = (if mau-compatible r1 r2
then real-of-u (mau-times r1 r2) else
Code.abort (STR "different base") (λ -.
real-of-u r1 * real-of-u r2))
using ma-times by (transfer, auto)

lemma mau-plus:
(real-of-u r1 + real-of-u r2) = (if mau-compatible r1 r2
then real-of-u (mau-plus r1 r2) else
Code.abort (STR "different base") (λ -.
real-of-u r1 + real-of-u r2))
using ma-plus by (transfer, auto)

lemma real-of-u-inj[simp]: real-of-u x = real-of-u y ↔ x = y
proof
note field-simps[simp] of-rat-diff[simp]
assume real-of-u x = real-of-u y
thus x = y
proof (transfer)
  fix x y
  assume ma-coeff x = 0 ∧ ma-base x = 0 ∨ ma-coeff x ≠ 0 ∧ prime-product
  (ma-base x)
  and ma-coeff y = 0 ∧ ma-base y = 0 ∨ ma-coeff y ≠ 0 ∧ prime-product
  (ma-base y)
  and real-of x = real-of y
  thus x = y
  proof (transfer, clarsimp)
  fix p1 q1 p2 q2 :: rat and b1 b2
  let ?p1 = real-of-rat p1
  let ?p2 = real-of-rat p2
  let ?q1 = real-of-rat q1
  let ?q2 = real-of-rat q2
  let ?b1 = real-of-rat b1
  let ?b2 = real-of-rat b2
  assume q1: q1 = 0 ∧ b1 = 0 ∨ q1 ≠ 0 ∧ prime-product b1
  and q2: q2 = 0 ∧ b2 = 0 ∨ q2 ≠ 0 ∧ prime-product b2
  and i1: q1 = 0 ∨ b1 ∈ sqrt-irrat
  and i2: q2 = 0 ∨ b2 ∈ sqrt-irrat
  show p1 = p2 ∧ q1 = q2 ∧ b1 = b2
  proof (cases q1 = 0)
case True
have \( q_2 = 0 \)
  by (rule sqrt-irrat[OF i2, of \( p_2 - p_1 \)], insert eq True \( q_1 \), auto)
with True \( q_1 \) \( q_2 \) eq show \(?thesis \) by auto
next
case False
hence 1: \( q_1 \neq 0 \) prime-product \( b_1 \) using \( q_1 \) by auto
{  
  assume *: \( q_2 = 0 \)
  have \( q_1 = 0 \)
    by (rule sqrt-irrat[OF i1, of \( p_1 - p_2 \)], insert eq * \( q_2 \), auto)
  with False have False by auto
}
hence 2: \( q_2 \neq 0 \) prime-product \( b_2 \) using \( q_2 \) by auto
from 1 i1 have \( b_1; \) \( b_1 \neq 0 \) unfolding sqrt-irrat-def by (cases \( b_1 \), auto)
from 2 i2 have \( b_2; \) \( b_2 \neq 0 \) unfolding sqrt-irrat-def by (cases \( b_2 \), auto)
let \(?sq = \lambda \( x \). x \times x \)
def \( q_3 \equiv \( p_2 - p_1 \) \)
let \(?q_3 = \text{real-of-rat} \( q_3 \) \)
let \(?e = \text{of-rat} (\( q_2 \times q_2 \times \text{of-nat} b_2 + ?sq ?q_3 - ?sq q_1 \times \text{of-nat} b_1 \) + \( 2 \times q_2 \times q_3 \) \times \text{sqrt} \( ?b_2 \))
from eq have *: \( ?q_1 \times \text{sqrt} \( ?b_1 \) = ?q_2 \times \text{sqrt} \( ?b_2 \) + ?q_3 \)
  by (simp add: q3-def)
from arg-cong[OF this, of \(?sq \)] have \( \theta = (\text{real-of-rat} 2 \times ?q_2 \times ?q_3 \) \times \text{sqrt} \( ?b_2 \) +
  \( ?sq ?q_2 \times ?b_2 + ?sq ?q_3 - ?sq q_1 \times ?b_1 \))
  by auto
also have \( \ldots = ?c \)
  by (simp add: of-rat-mult of-rat-add of-rat-minus)
finally have eq: \( ?c = 0 \) by simp
from sqrt-irrat[OF - this] 2 i2 have \( q_3 = 0 \) by auto
hence p: \( p_1 = p_2 \) unfolding q3-def by simp
let \( ?b_1 = \text{rat-of-nat} b_1 \)
let \( ?b_2 = \text{rat-of-nat} b_2 \)
from eq[unfolded q3] have eq: \( ?sq q_2 \times ?b_2 = ?sq q_1 \times ?b_1 \) by auto
obtain \( z_1 n_1 \) where d1: quotient-of \( q_1 \) = \( (z_1, n_1) \) by force
obtain \( z_2 n_2 \) where d2: quotient-of \( q_2 \) = \( (z_2, n_2) \) by force
note id = quotient-of-div[OF d1] quotient-of-div[OF d2]
note pos = quotient-of-denom-pos[OF d1] quotident-of-denom-pos[OF d2]
from id(1) 1(1) pos(1) have \( z_1: \) \( z_1 \neq 0 \) by auto
from id(2) 2(1) pos(2) have \( z_2: \) \( z_2 \neq 0 \) by auto
let \(?n_1 = \text{rat-of-int} n_1 \)
let \(?n_2 = \text{rat-of-int} n_2 \)
let \(?z_1 = \text{rat-of-int} z_1 \)
let \(?z_2 = \text{rat-of-int} z_2 \)
from arg-cong[OF eq[unfolded id], of \( \lambda \( x \). x \times ?sq ?n_1 \times ?sq ?n_2 \),
unfolded field-simps]
have \( ?sq (\( ?n_1 \times ?z_2 \)) \times \text{sqrt} \( ?b_2 \) = ?sq (\( ?n_2 \times ?z_1 \)) \times ?b_1 \)
  using pos by auto

22
moreover have ⨂n1 * ⨂n2 ≠ 0 ⨂n2 * ⨂z1 ≠ 0 using ⨂z1 ⨂z2 pos by auto
ultimately obtain i1 i2 where 0: rat-of-int i1 ≠ 0 rat-of-int i2 ≠ 0
and eq: ⨂sq (rat-of-int i2) * ⨂b2 = ⨂sq (rat-of-int i1) * ⨂b1
unfolding of-int-mult[symmetric] by blast
let ⨂b1 = int b1
let ⨂b2 = int b2
from eq have eq: ⨂sq i1 * ⨂b1 = ⨂sq i2 * ⨂b2
by (metis (hide-lams, no-types) of-int-eq-iff of-int-mult of-int-of-nat-eq)
from 0 have 0: i1 ≠ 0 i2 ≠ 0 by auto
from arg-cong[OF eq, of nat] have ⨂sq (nat (abs i1)) * b1 = ⨂sq (nat (abs i2)) * b2
by (metis abs-of-nat eq nat-abs-mult-distrib nat-int)
moreover have nat (abs i1) > 0 nat (abs i2) > 0 using 0 by auto
ultimately obtain n1 n2 where 0: n1 > 0 n2 > 0 and eq: ⨂sq n1 * ⨂b1
= ⨂sq n2 * ⨂b2 by blast
from b1 0 have b1: b1 > 0 n1 > 0 n1 * n1 > 0 by auto
from b2 0 have b2: b2 > 0 n2 > 0 n2 * n2 > 0 by auto
{ fix p
have id1: multiplicity p (⋎n1 * ⨂b1) mod 2 = multiplicity p b1 mod 2
unfolding multiplicity-product-nat[OF b1(2,2)]
by presburger
have id2: multiplicity p (⋎n2 * ⨂b2) mod 2 = multiplicity p b2 mod 2
unfolding multiplicity-product-nat[OF b2(3,1)]
by presburger
from id1 id2 eq have eq: multiplicity p b1 mod 2 = multiplicity p b2 mod 2
by simp
from 1(2) 2(2) have multiplicity p b1 ≤ 1 multiplicity p b2 ≤ 1
unfolding prime-product-def by auto
with eq have multiplicity p b1 = multiplicity p b2 by simp
}
with b1(1) b2(1) have b: b1 = b2 by (rule multiplicity-eq-nat)
from *[unfolded b q3] b1(1) b2(1) have q: q1 = q2 by simp
from p q b show ?thesis by blast
qed
qed
qed simp

lift-definition mau-sqrt :: mini-alg-unique ⇒ mini-alg-unique is
λ ma. let (a,b) = quotient-of (ma-rat ma); (sq,fact) = prime-product-factor
(nat (abs a * b));
ma' = ma-of-rat (of-int (sgn(a)) * of-nat sq / of-int b)
in ma-times ma' (ma-sqrt (ma-of-rat (of-nat fact)))
proof –
fix ma :: mini-alg
let ?num = let (a, b) = quotient-of (ma-rat ma); (sq, fact) = prime-product-factor
(nat (|a| * b));

ma' = ma-of-rat (rat-of-int (sgn a) * of-nat sq / of-int b)
in ma-times ma' (ma-sqrt (ma-of-rat (rat-of-nat fact)))

obtain a b where q: quotient-of (ma-rat ma) = (a,b) by force
obtain sq fact where pff: prime-product-factor (nat (abs a * b)) = (sq,fact)

by force

def asq = rat-of-int (sgn a) * of-nat sq / of-int b

def ma' = ma-of-rat asq

def sqrt = ma-sqrt (ma-of-rat (rat-of-nat fact))

have num: ?num = ma-times ma' sqrt unfolding q pff asq-def Let-def split

ma'-def sqrt-def ..

let ?inv = λ ma. ma-coeff ma = 0 ∨ ma-base ma = 0 ∨ ma-coeff ma ≠ 0 ∧

prime-product (ma-base ma)

have ma': ?inv ma' unfolding ma'-def

by (transfer, auto)

have id: ∃ i. i * I = i ∃ i :: rat. i / I = i rat-of-int I = 1 inverse (I ::

rat) = 1

∧ n. nat | int n = n by auto

from prime-product-factor[OF pff] have prime-product fact by auto

hence sqrt: ?inv sqrt unfolding sqrt-def

by (transfer, unfold split quotient-of-nat Let-def id, case-tac sqrt-int |int facta|,

auto)

show ?inv ?num unfolding num using ma' sqrt

by (transfer, auto simp: ma-normalize-def split: if-splits)

qed

lemma sqrt-sgn[simp]: sqrt (of-int (sgn a)) = of-int (sgn a)

by (cases a ≥ 0, cases a = 0, auto simp: real-sqrt-minus)

lemma mau-sqrt-main: mau-coeff r = 0 ⇒ sqrt (real-of-u r) = real-of-u (mau-sqrt r)

proof (transfer)

fix r

assume ma-coeff r = 0

hence rr: real-of r = of-rat (ma-rat r) by (transfer, auto)

obtain a b where q: quotient-of (ma-rat r) = (a,b) by force

from quotient-of-div[OF q] have r: ma-rat r = of-int a / of-int b by auto

from quotient-of-denom-pos[OF q] have b: b > 0 by auto

obtain sq fact where pff: prime-product-factor (nat (|a| * b)) = (sq, fact) by force

from prime-product-factor[OF pff] have ab: nat (|a| * b) = sq * sq * fact ..

have sqrt (real-of r) = sqrt((of-int a / of-int b) unfolding rr r

by (metis of-rat-divide of-rat-of-int-eg)

also have real-of-int a / of-int b = of-int a * of-int b / (of-int b * of-int b)

using b by auto

also have sqrt (..) = sqrt (of-int a * of-int b) / of-int b using sqrt-sqrt[of

real-of-int b] b

by (metis less-eq-real-def of-int-0-less-iff real-sqrt-divide real-sqrt-mult-distrib2)

also have real-of-int a * of-int b = real-of-int (a * b) by auto
also have \( a \times b = \text{sgn } a \times (\text{abs } a \times b) \) by (simp, metis mult-sgn-abs)
also have real-of-int (...) = of-int (sgn a) \times real-of-int ([|a|] \times b)
    unfolding of-int-mult[of sgn a] ..
also have real-of-int ([|a|] \times b) = of-nat (nat (abs a \times b)) using b
    by (metis abs-sgn mult-pos-pos mult-zero-left nat-int of-int-of-nat-eq of-nat-0
    zero-less-abs-iff zero-less-imp-eq-int)
also have \( \ldots = \text{of-nat } \text{fact} \times (\text{of-nat } \text{sq} \times \text{of-nat } \text{sq}) \) unfolding ab of-nat-mult
by simp
also have \( \sqrt{\text{of-int} (\text{sgn } a \times (\text{of-nat } \text{fact} \times \text{of-nat } \text{sq} \times \text{of-nat } \text{sq})})} = \)
    of-nat fact \times (of-nat sq \times of-nat sq)
    unfolding real-sqrt-mult-distrib by simp
finally have \( \sqrt{\text{of-int} (\text{sgn } a \times \text{of-nat } \text{fact})} = \)
    of-int (sgn a) \times \sqrt{(\text{of-nat } \text{fact})}
    by (simp add : of-rat-divide of-rat-mult)
qed

lemma mau-sqrt: \( \sqrt{\text{real-of-u } r} = (\text{if mau-coeff } r = 0 \text{ then real-of-u (mau-sqrt } r) \) else Code.abort (STR "cannot represent sqrt of irrational number") (λ -. sqrt (real-of-u r))\)
    using mau-sqrt-main[of r] by (cases mau-coeff r = 0, auto)

lemma mau-0: \( 0 = \text{real-of-u (mau-of-rat 0)} \) using ma-0 by (transfer, auto)

lemma mau-1: \( 1 = \text{real-of-u (mau-of-rat 1)} \) using ma-1 by (transfer, auto)

lemma mau-equal:
    HOL.equal (real-of-u r1) (real-of-u r2) = mau-equal r1 r2 unfolding equal-real-def
    using real-of-a-inj[of r1 r2]
    by (transfer, transfer, auto)

lemma mau-ge-0: \( \text{ge-0 } (\text{real-of-u } x) = \text{mau-ge-0 } x \) using mau-ge-0
The following code equation terminates if it is started on two different inputs.

**lemma** `real-lt[code]`: `real-lt x y = (let fx = floor x; fy = floor y in
  (if fx < fy then True else if fx > fy then False else real-lt (x * 1024) (y * 1024)))`

**proof** (cases floor x < floor y)
  case True
  thus `?thesis` by (auto simp: real-lt-def floor-less-cancel)
  next
  case False
  note nless = this
  show `?thesis`
    proof (cases floor x > floor y)
      case True
      from floor-less-cancel[OF this] True nless show `?thesis`
      by (simp add: real-lt-def)
    next
    case False
    with nless show `?thesis`
      unfolding real-lt-def by auto
  qed
  qed

For comparisons we first check for equality. Then, if the bases are compatible we can just compare the differences with 0. Otherwise, we start the recursive algorithm `real-lt` which works on arbitrary bases. In this way, we have an implementation of comparisons which can compare all representable numbers.

Note that in `Real-Impl` we did not use `real-lt` as there the code-equations for equality already require identical bases.

**lemma** `comparison-impl`:
  `real-of-u x ≤ real-of-u y` ⟷ `real-of-u x = real-of-u y` ∨
  (if mau-compatible x y then ge-0 (real-of-u y − real-of-u x) else real-lt (real-of-u x) (real-of-u y))
  `real-of-u x < real-of-u y` ⟷ `real-of-u x ≠ real-of-u y` ∧
  (if mau-compatible x y then ge-0 (real-of-u y − real-of-u x) else real-lt (real-of-u x) (real-of-u y))
  unfolding ge-0-def real-lt-def by (auto simp del: real-of-u-inj)

**lemma** `mau-is-rat`: `is-rat (real-of-u x) = mau-is-rat x` using `mau-rat`
  by (transfer, auto)

**lift-definition** `ma-show-real :: mini-alg ⇒ string` is
  `λ (p,q,b). let sb = shows "sqrt" o shows b o shows ")";;
  qb = (if q = 1 then sb else if q = -1 then shows "-" o sb else shows q o
  shows "+" o sb) in
  if q = 0 then shows p [] else
  if p = 0 then qb [] else
if \( q < 0 \) then \((\text{shows } p \circ \text{qb} \ [\])\) 
else \((\text{shows } p \circ \text{shows }' + ' \circ \text{qb} \ [\])\).

**lift-definition** mau-show-real :: mini-alg-unique \(\Rightarrow\) string is mau-show-real.

**definition** show-real :: real \(\Rightarrow\) string where

\[
\text{show-real } x = (\text{if } (\exists \ y. x = \text{real-of-u } y) \text{ then mau-show-real } \text{THE } y. x = \text{real-of-u } y \text{ else } [])
\]

**lemma** mau-show-real: show-real (real-of-u \( x \)) = mau-show-real \( x \)

**unfolding** show-real-def by simp

**lemmas** mau-code-eqns = mau-floor mau-0 mau-uminus mau-inverse mau-sqrt mau-plus mau-times mau-equal mau-ge-0 mau-is-rat mau-show-real comparison-impl

**code-datatype** real-of-u

**declare** real-code-del[\text{code, code del}]

**declare** mau-code-eqns[\text{code del}]

**declare** real-code-unfold-del[\text{code-unfold del}]

**declare** real-standard-impls[\text{code}]

**declare** mau-code-eqns[\text{code}]

Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

**value** [101.1 * (sqrt 18 + 6 * sqrt 0.5)]

**value** [324 * sqrt 7 + 0.001]

**value** 101.1 * (sqrt 18 + 6 * sqrt 0.5) = 324 * sqrt 7 + 0.001

**value** 101.1 * (sqrt 18 + 6 * sqrt 0.5) > 324 * sqrt 7 + 0.001

**value** show-real (101.1 * (sqrt 18 + 6 * sqrt 0.5))

**value** (sqrt 0.1 \( \in \mathbb{Q} \)), sqrt \((-0.09) \in \mathbb{Q})

end

**Acknowledgements**

We thank Bertram Felgenhauer for interesting discussions and especially for mentioning Cauchy’s mean theorem during the formalization of the algorithms for computing roots.

**References**


