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

René Thiemann

September 19, 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 $\mathbb{Q}[\sqrt{b}]$.

\textsuperscript{*}This research is supported by FWF (Austrian Science Fund) project P22767-N13.
To support these termination proofs in our certifier CēTA [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-gt-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) = (∀ 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 \( :: \) nat \( \Rightarrow \) nat \( \Rightarrow \) nat \( \Rightarrow \) nat \( \Rightarrow \) nat \( \Rightarrow \) nat \times nat **where**

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

**termination**

**proof** –

\[
\begin{align*}
\text{let } ?m1 &= \lambda (\text{factor-sq :: nat, factor-pr :: nat, limit :: nat, n :: nat, i :: nat). n} \\
\text{let } ?m2 &= \lambda (\text{factor-sq, factor-pr, limit, n, i}). (\text{Suc limit } - i) \\
\{ \\
\text{fix } i \\
\text{have } 2 \leq i \implies \text{Suc 0} < i \ast i \text{ using one-less-mult[of i i] by auto} \\
\} \text{ note } * = \text{this} \\
\text{show } ?\text{thesis} \\
\text{by (rule, rule wf-measures[of } [?m1, ?m2], \text{ auto split: if-splits simp: * dvd-eq-mod-eq-0)}
\end{align*}
\]

**qed**

**lemma** prime-product-factor-main: assumes \( \exists (s, s \ast s = n) \)

and \( \text{limit} = \text{nat (root-nat-floor 3 n)} \)

and \( m = \text{factor-sq } \ast \text{factor-sq } \ast \text{factor-pr } \ast n \)

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

and \( i \geq 2 \)

and \( \land (i, j \geq 2) \implies j < i \implies \neg j \mid n \)

and \( \land (i, j < i) \implies \text{multiplicity } j \mid \text{factor-pr } \leq 1 \)

and \( \land (j, j \geq i) \implies \text{multiplicity } j \mid \text{factor-pr } = 0 \)

and \( \text{factor-pr } > 0 \)

shows \( m = \text{sq } \ast \text{sq } \ast p \land \text{prime-product p} \)
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 = 1(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 i = this
      with prems(5) have cond: i ≤ limit ∧ i ≥ 2 and *: (i ≤ limit ∧ i ≥ 2) = True by blast+
      note IH = IH[OF cond]
      note simp = simp[unfolded * if-True]
      show ?thesis
        proof (rule IH)
          fix j
          assume 2: 2 ≤ j and j: j < Suc i
          from prems(6)[OF 2] j False
          show ¬j dvd n by (cases j = i, auto)
        next
          fix j
          assume j: j < Suc i
          with prems(7−8)[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
    note simp = simp[unfolded * if-False]
    note IH = IH(3)[OF False prems(1−3) simp]
    show ?thesis
      proof (rule IH)
        fix j
        assume 2: 2 ≤ j and j: j < Suc i
        from prems(6)[OF 2] j False
        show ¬j dvd n by (cases j = i, auto)
      qed (insert prems(8−9) cond, auto)
    qed (insert prems(8−9) cond, auto)
  next
  case True
  note mod = this
  hence simp: (i dvd n) = True by simp
  note simp = simp[unfolded this if-True Let-def]
  note IH = IH(1,2)[OF True refl]
  show ?thesis
    proof (cases i dvd n)
    case True
    hence simp: (i dvd n) = True by auto
    def n' ≡ n div 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) - prems(7−9)]
    show ?thesis
      proof (rule IH)
        show m = factor-sq * i * (factor-sq * i) * factor-pr * (n div i div i)
        unfolding prems(3) n'-def[symmetric]
      qed
unfolding $n$ by \texttt{(auto simp: field-simps)}

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

next
show $\neg (\exists s. s \ast s = n \div i \div i)$
proof
  assume $\exists s. s \ast s = n \div i \div i$
then obtain $s$ where $s \ast s = n \div i \div i$ by \texttt{auto}
hence $(s \ast i) \ast (s \ast i) = n$ unfolding $n$ by \texttt{auto}
with \texttt{prems(1)} show False by \texttt{blast}
qed

qed

next
case False
def $n' \equiv n \div i$
from \texttt{mod True} have $n = n' \ast i$ by \texttt{(auto simp: n'-def dvd-eq-mod-eq-0)}
have $\text{prime}$: $\text{prime } i$
  unfolding \texttt{prime-nat-def}
proof \texttt{(intro conjI allI impI)}
  fix $m$
  assume $m \mid m \div i$
  hence $m \mid n$ unfolding $n$ by \texttt{auto}
  with \texttt{prems(6)[of m]} have choice: $m \leq 1 \lor m \geq i$ by \texttt{arith}
  from $m$ \texttt{prems(5)} have $m > 0$
    by \texttt{(metis dvd-0-left-iff le0 le-antisym neq0-conv zero-neq-numeral)}
  with choice have choice: $m = 1 \lor m \geq i$ by \texttt{arith}
  from $m$ \texttt{prems(5)} have $m \leq i$
    by \texttt{(metis False div-by-0 dvd.dual-order.refl dvd-imp-le gr0I)}
  with choice
  show $m = 1 \lor m = i$ by \texttt{auto}
qed \texttt{(insert prems(5), auto)}

from False have $(i \mid n \div i) = \text{False}$ by \texttt{auto}
note simp = simp[unfolded this if-False]
note $IH = \text{IH(2)[OF False - - refl]}$

from prime have $i > 0$ by \texttt{auto}
note $mult = \text{multiplicity-product-nat[OF prems(9) this]}$
note $mult-i = \text{multiplicity-prime-nat[OF prime] multiplicity-prime[OF prime]}$
show $\text{thesis}$
proof \texttt{(cases sqrt-nat (n div i))}
case (Cons $s$)
  note simp = simp[unfolded Cons list.simps]
  hence sq: $sq = \text{factor-sq} \ast s$ and $p; p = \text{factor-pr} \ast i$ by \texttt{auto}
  from \texttt{arg-cong[OF Cons, of set]} have $s \ast s = n \div i$ by \texttt{auto}
  have $pp$: $\text{prime-product } (\text{factor-pr} \ast i)$
    unfolding \texttt{prime-product-def}
proof
  fix m
  show multiplicity m (factor-pr * i) ≤ 1
  unfolding mult using prems(7)[of m] prems(8)[of m] mult-i(1)
  mult-i(2)[of m] by fastforce
  qed
  show ?thesis unfolding sq p 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 (∃ x. x * x = n div i) by simp
  note IH = IH[OF Nil this - simp]
  show ?thesis
    proof (rule IH)
      show m = factor-sq * factor-sq * (factor-pr * i) * (n div i)
        unfolding prems(3) n by auto
    next
      fix j
      assume *: 2 ≤ j j < Suc i
      show ¬ j dvd n div i
        proof
          assume j: j dvd n div i
          with False have j ≠ i by auto
          with * have 2 ≤ j j < i by auto
          from 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 ≤ j
      hence ij: i ≤ j and j: j ≠ i by auto
      have 0: multiplicity j i = 0 using prime j by (rule multiplicity-prime)
      show multiplicity j (factor-pr * i) = 0 unfolding mult prems(8)[OF ij]
        0 by simp
    next
      fix j
      assume j < Suc i
      hence j < i ∨ j = i by auto
      thus multiplicity j (factor-pr * i) ≤ 1
      proof
        assume j = i
        with prems(8)[of i] prime show ?thesis unfolding mult
          by (auto)
      next
        assume ji: j < i
        hence j ≠ i by auto
        from prems(7)[OF ji] multiplicity-prime[OF prime this]
show ?thesis unfolding mult by auto
qed
qed (insert prems(5,9), auto)
qed
qed
next
case False
hence (i ≤ limit ∧ i ≥ 2) = False by auto
note simp = simp[unfolded this if-False]
hence sq: sq = factor-sq and p: p = factor-pr * n by auto
show ?thesis
proof
show m = sq * sq * p unfolding sq p prems(3) by simp
show prime-product p unfolding prime-product-def
proof
fix m
from prems(1) have n1: n > 1 by (cases n, auto, case-lac nat, auto)
hence n0: n > 0 by auto
have i > limit using False by auto
from this[unfolded prems(2)] have less: int i ≥ root-nat-floor 3 n + 1 by auto
have int n < (root-nat-floor 3 n + 1) ^ 3 by (rule root-nat-floor-upper, auto)
also have ... ≤ 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: multiplicity m n > 0
  hence mp: m ∈ prime-factors n
    by (metis less-not-refl multiplicity-not-factor-nat)
hence md: m dvd n
  by (metis k prime-factors-altdef2-nat)
then obtain k where n: n = m * k ..
from mp have pm: prime m by auto
hence m2: m ≥ 2 and m0: m > 0 by auto
from prems(6)(OF m2] md have mi: m ≥ i by force
{
  assume multiplicity m n ≠ 1
  with m have 3 k. multiplicity m n = 2 + k by presburger
  then obtain j where mult: multiplicity m n = 2 + j ..
from n0 n have k: k > 0 by auto
from mult[unfolded n multiplicity-product-nat[OF m0 k]] pm
have multiplicity m k > 0 by auto
hence mp: m ∈ prime-factors k
  by (metis less-not-refl multiplicity-not-factor-nat)
hence md: m dvd k
  by (metis k prime-factors-altdef2-nat)

7
then obtain \( l \) where \( kml: k = m \ast l \).

note \( n = n \[unfolded kml]\)
from \( n \) have \( l \) \( \text{dvd} \) \( n \) by \( \text{auto} \)
with \( \text{prems}(6)[of } l \] have \( l \leq 1 \lor l \geq i \) by \( \text{arith} \)
with \( n \) \( \not\)\(\text{have} \ l: l = 1 \lor l \geq i \) by \( \text{auto} \)
from \( n \) \( \text{prems}(1) \) have \( l \not= 1 \) by \( \text{auto} \)
with \( \text{l} \) have \( l \geq i \) by \( \text{auto} \)

with \( n \) \( \text{n0} \) have \( l \) \( \text{=} \) \( 1 \lor l \geq i \) by \( \text{auto} \)
from \( n \) \( \text{prems}(1) \) have \( l \not= 1 \) by \( \text{auto} \)
with \( l \) have \( l \geq i \) by \( \text{auto} \)
from \( \text{mult-le-mono[OF } \text{mult-le-mono[OF } mi \ mi \ l] \) have \( n \geq i^3 \) unfolding \( n \) by \( \text{(auto simp: power3-eq-cube)} \)
with \( n-i3 \) have \( \text{False} \) by \( \text{auto} \)

also have \( \ldots \leq 1 \)
proof \( \text{(cases } m < i) \)
  case \( \text{True} \)
  from \( \text{prems}(7)[of } m \] \( n[\text{of } m] \) \( \text{True} \) show \( \text{thesis} \) by \( \text{force} \)
next
  case \( \text{False} \)
  hence \( m \geq i \) by \( \text{auto} \)
  from \( \text{prems}(8)[OF } \text{this} \] \( n[\text{of } m] \) show \( \text{thesis} \) by \( \text{force} \)
qed
finally show \( \text{multiplicity } m \ p \leq 1 \).
qed

definition \( \text{prime-product-factor} :: \text{nat } \Rightarrow \text{nat } \times \text{nat} \) where
\( \text{prime-product-factor } n = (\text{case } \text{sqrt-nat } n \text{ of} \)
\( \text{(Cons } s \text{-) } \Rightarrow (s,1) \)
\( | [] \Rightarrow \text{prime-product-factor-main 1 1 (nat } \text{(root-nat-floor 3 } n)) \text{ n } 2) \)

lemma \( \text{prime-product-one[simp, intro]: prime-product } 1 \)
  unfolding \( \text{prime-product-def multiplicity-one-nat} \) by \( \text{auto} \)

lemma \( \text{prime-product-factor: assumes } pf: \text{prime-product-factor } n = (sq,p) \)
  shows \( n = sq \ast sq \ast p \land \text{prime-product } p \)
proof \( \text{(cases } \text{sqrt-nat } n) \)
  case \( (\text{Cons } s) \)
  from \( pf[unfolded } \text{prime-product-factor-def Cons}] \) \( \text{arg-cong[OF } \text{Cons, of set]} \) \text{prime-product-one} 
  show \( \text{thesis} \) by \( \text{auto} \)
A representation of real numbers via triples

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

definition sqrt-irrat = { q :: nat. ¬ (∃ p. p * p = rat-of-nat q)}

lemma sqrt-irrat: assumes choice: q = 0 ∨ b ∈ sqrt-irrat
    and eq: real-of-rat p + real-of-rat q * sqrt (of-nat b) = 0
    shows q = 0
    using choice
proof (cases q = 0)
case False
    with choice have sqrt-irrat: b ∈ sqrt-irrat by blast
    from eq have real-of-rat q * sqrt (of-nat b) = real-of-rat (− p)
        by (auto simp: of-rat-minus)
    then obtain p where real-of-rat q * sqrt (of-nat b) = real-of-rat p by blast
    from arg-cong[OF this, of λ x. x * x] have real-of-rat (q * q) * (sqrt (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 * rat-of-nat b) = real-of-rat (p * p) by (auto simp: of-rat-mult)
    hence q * q * (rat-of-nat b) = p * p by auto
    from arg-cong[OF this, of λ x. x / (q * q)]
    have (p / q) * (p / q) = 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-imls:
  (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 \(p + q \cdot \sqrt{b}\), 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
  λ (p, q, b). of-rat p + of-rat q * sqrt (of-nat b) .

lift-definition ma-of-rat :: rat ⇒ mini-alg is λ 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
  λ (p1,q1,b1). (− p1, − q1, b1) by auto
lift-definition \textit{ma-compatible} :: \textit{mini-alg} \Rightarrow \textit{mini-alg} \Rightarrow \textit{bool} \text{ is } \\
\lambda (p_1, q_1, b_1) (p_2, q_2, b_2). q_1 = 0 \lor q_2 = 0 \lor b_1 = b_2 .

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

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

lift-definition \textit{ma-plus} :: \textit{mini-alg} \Rightarrow \textit{mini-alg} \Rightarrow \textit{mini-alg} \text{ is } \\
\lambda (p_1, q_1, b_1) (p_2, q_2, b_2). \text{if } q_1 = 0 \text{ then } \\
(p_1 + p_2, q_2, b_2) \text{ else } \textit{ma-normalize} (p_1 + p_2, q_1 + q_2, b_1) \text{ by } \text{(auto simp: } \textit{ma-normalize-def})

lift-definition \textit{ma-times} :: \textit{mini-alg} \Rightarrow \textit{mini-alg} \Rightarrow \textit{mini-alg} \text{ is } \\
\lambda (p_1, q_1, b_1) (p_2, q_2, b_2). \text{if } q_1 = 0 \text{ then } \\
\textit{ma-normalize} (p_1 \times p_2, p_1 \times q_2, b_2) \text{ else } \\
\textit{ma-normalize} (p_1 \times p_2 + \textit{of-nat} b_2 \times q_1 \times q_2, p_1 \times q_2 + q_1 \times p_2, b_1) \text{ by } \text{(auto simp: } \textit{ma-normalize-def})

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

lift-definition \textit{ma-floor} :: \textit{mini-alg} \Rightarrow \textit{int} \text{ is } \\
\lambda (p, q, b). \text{case } \text{quotient-of } p, \text{quotient-of } q \text{ of } ((z_1, n_1), (z_2, n_2)) \Rightarrow \\
\text{let } z_2 n_1 = z_2 \times n_1 ; z_1 n_2 = z_1 \times n_2 ; n_1_2 = n_1 \times n_2 ; \text{prod} = z_2 n_1 \times z_2 n_1 \times \text{int } b \text{ in } \\
(z_1 n_2 + (\text{if } z_2 n_1 \geq 0 \text{ then } \textit{sqrt-int-floor-pos} \text{ prod else } -\textit{sqrt-int-ceiling-pos} \text{ prod})) \div n_1_2 .

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

proof (unfold Let-def) \\
\text{fix } \text{prod } :: \textit{rat} \times \textit{rat} \times \textit{nat} \\
\text{obtain } p q b \text{ where } \text{prod} = (p, q, b) \text{ by } \text{(cases } \text{prod}, \text{ auto)}

obtain a b \text{ where } p: \text{quotient-of } p = (a, b) \text{ 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 \times b| \text{ of } [] \Rightarrow (0, \text{inverse } (\textit{of-int } b), \text{nat } |a \times b|) \\
| s \# x \Rightarrow (\text{of-int } s / \text{of-int } b, 0, 0)) = \\
(p, q, b) \land
(q = 0 ∨ b ∈ sqrt-irrat)

proof (cases sqrt-int (abs (a * b)))

case Nil

from sqrt-int[of abs (a * b)] Nil have irrat: ∼ (∃ y. y * y = |a * b|) by auto

have nat |a * b| ∈ sqrt-irrat

proof (rule ccontr)

assume nat |a * b| ∉ sqrt-irrat

then obtain x :: rat

where x * x = rat-of-nat (nat |a * b|) unfolding sqrt-irrat-def by auto

hence x * x = rat-of-int |a * b| by auto

from sqrt-rat-of-int[OF this] irrat show False by blast

qed

thus ?thesis using Nil by (auto simp: prod p)

qed (auto simp: prod p Cons)

qed

lift-definition ma-equal :: mini-alg ⇒ mini-alg ⇒ bool is

λ (p1,q1,b1) (p2,q2,b2).

p1 = p2 ∧ q1 = q2 ∧ (q1 = 0 ∨ b1 = b2).

lift-definition ma-ge-0 :: mini-alg ⇒ bool is

λ (p,q,b). let bqq = of-nat b * q * q; pp = p * p in

0 ≤ p ∧ bqq ≤ pp ∨ 0 ≤ q ∧ pp ≤ bqq.

lift-definition ma-is-rat :: mini-alg ⇒ bool is

λ (p,q,b). q = 0.

definition ge-0 :: real ⇒ bool where [code del]: ge-0 x = (x ≥ 0)

lemma ma-ge-0: ge-0 (real-of x) = ma-ge-0 x

proof (transfer, unfold Let-def, clarsimp)

fix p' q' :: rat and b' :: nat

assume b': q' = 0 ∨ b' ∈ sqrt-irrat

def b ≡ real-of-nat b'

def p ≡ real-of-rat p'

def q ≡ real-of-rat q'

from b' have b: 0 ≤ b q = 0 ∨ b' ∈ sqrt-irrat unfolding b-def q-def by auto

def qb ≡ q * sqrt b

from b have sqrt: sqrt b ≥ 0 by auto

from b have disj: q = 0 ∨ b ≠ 0 unfolding sqrt-irrat-def b-def by auto

have bdef: b = real-of-rat (of-nat b') unfolding b-def by auto

have (0 ≤ p + q * sqrt b) = (0 ≤ p + qb) unfolding qb-def by simp

also have . . . ⟷ (0 ≤ p ∧ abs qb ≤ abs p ∨ 0 ≤ qb ∧ abs p ≤ abs qb) by arith

also have . . . ⟷ (0 ≤ p ∧ qb * qb ≤ p * p ∨ 0 ≤ qb ∧ p * p ≤ qb * qb)

unfolding abs-lesseq-square ..

also have qb * qb = b * q * q unfolding qb-def

using b by auto

also have 0 ≤ qb ⟷ 0 ≤ q unfolding qb-def using sqrt disj

12
by (metis le-cases mult-eq-0-iff mult-nonneg-nonneg mult-npos-nonneg order-class.order.antisym
  qbl-def real-sqrt-eq-zero-cancel-iff)
also have \( (0 \leq p \land b \cdot q \cdot q \leq p \cdot p \lor 0 \leq q \land p \cdot p \leq b \cdot q \cdot q) \)
  \( \iff (0 \leq p' \land \text{of-nat } b' \cdot q' \cdot q' \leq p' \cdot p' \lor 0 \leq q' \land p' \cdot p' \leq \text{of-nat } b' \cdot q' \cdot q') \)
unfolding qbl-def
by (unfold bdef p-def q-def of-rat-mult[symmetric] of-rat-less-eq, simp)
finally
show \( \text{ge-0 } (\text{real-of-rat } p' + \text{real-of-rat } q' \ast \sqrt{\text{of-nat } b'}) = \)
  \( (0 \leq p' \land \text{of-nat } b' \cdot q' \cdot q' \leq p' \cdot p' \lor 0 \leq q' \land p' \cdot p' \leq \text{of-nat } b' \cdot q' \cdot q') \)
unfolding \( \text{ge-0-def } p\text{-def } b\text{-def } q\text{-def} \)
by (auto simp: of-rat-add of-rat-mult)
qed

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

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

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

lemma \textit{ma-inverse}: \( \text{inverse } (\text{real-of } r) = \text{real-of } (\text{ma-inverse } r) \)
proof (transfer; unfold Let-def, clarsimp)
  fix \( p' \cdot q'::\text{rat} \) \textbf{and} \( b'::\text{nat} \)
  assume \( b': q' = 0 \lor b' \in \sqrt{\text{irrat}} \)
  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 \( b'\text{-have } b: b \geq 0 \land q = 0 \lor b' \in \sqrt{\text{irrat}} \) unfolding b-def q-def by auto
have \( \text{inverse } (p + q \ast \sqrt{b}) = (p - q \ast \sqrt{b}) \ast \text{inverse } (p \ast p - b \ast (q \ast q)) \)
proof (cases \( q = 0 \))
  case False
  thus \( \text{?thesis} \) by (cases \( p = 0 \), auto simp: field-simps)
next
case False
from \( \sqrt{\text{irrat}}\text{-}\text{OF } b', \text{of } p'\) \text{-def } p-def q-def False have \( \text{null: } p + q \ast \sqrt{b} \neq 0 \) by auto
have \( \text{?thesis } \iff (p + q \ast \sqrt{b}) \ast \text{inverse } (p + q \ast \sqrt{b}) = \)
  \( (p + q \ast \sqrt{b}) \ast ((p - q \ast \sqrt{b}) \ast \text{inverse } (p \ast p - b \ast (q \ast q)) \)
unfolding \( \text{null-left-cancel}\text{[OF } \text{null]} \) \textbf{by} auto
also have \( (p + q \ast \sqrt{b}) \ast \text{inverse } (p + q \ast \sqrt{b}) = 1 \) \textbf{using} \( \text{null} \) \textbf{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 \( b \) \textbf{by} (auto simp: field-simps)
also have \( ... = 1 \)
proof (rule right-inverse, rule)
  assume \( \text{eq: } p \ast p - b \ast (q \ast q) = 0 \)
  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 as-def sb-def using abpos
by (simp add: rab of-rat-divide real-sqrt-mult real-sqrt-divide of-rat-inverse)
also have \ldots = (sa = inverse (of-int b) * sa * (sb * 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 \?thesis by simp

next
case (Cons s x)
from b have b: real-of-int b > 0 by auto
from Cons sqrt-int[of abs (a * b)] have s * s = abs (a * b) by auto
with sqrt-int-pos[OF Cons] have sqrt (real-of-int (abs (a * b))) = of-int s
by (metis abs-of-nonneg of-int-mult real-eq-of-int real-of-int-abs real-sqrt-abs2)
with abpos have of-int s = sqrt (real-of-int (abs (a * b))) by auto
thus \?thesis unfolding ab split using Cons b
by (auto simp: of-rat-divide field-simps real-sqrt-divide real-sqrt-mult)
qed
finally show \sqrt (real-of-rat p) = \?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") (\lambda -. 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 \?thesis
proof (cases ma-rat r \geq 0)
case True
from ma-sqrt-main[OF this 0] 0 True show \?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 \?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") (\lambda -. real-of r1 + real-of r2))
by transfer (auto split: prod.split simp: field-simps of-rat-add)

lemma ma-times:
(real-of r1 * real-of r2) = (if ma-compatible r1 r2
then real-of (ma-times r1 r2) else
 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 - 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
qd
qd auto
finally show ?l = ?r by simp
qd

lemma ma-floor: floor (real-of r) = ma-floor r
proof (transfer, unfold Let-def, clarsimp)
  fix p q :: 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 n120: n12 > 0 unfolding n12-def by simp
  unfolding floor [of-rat p + of-rat q * sqrt (real-of-nat b)] = floor ((of-int z1n2 + r-add) / of-int n12)
  unfolding r-add-def n12-def z1n2-def z2n1-def
  also have (...) = 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) ⇒
      case quotient-of q of
        (z2, n2) ⇒
          (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[of - 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 z2 * n1 ≥ 0)
  case True
    hence ge: real-of-int (z2 * n1) ≥ 0 by (metis of-int-0-le-iff)
    have radd: r-add = sqrt (of-int (z2 * n1 * (z2 * n1) * int b))
      unfolding r-add-def z2n1-def using sqrt-sqrt[OF ge]
      by (simp add: ac-simps real-eq-of-int real-sqrt-mult-distrib2)
    show thesis unfolding radd sqrt-int-floor-pos[OF ge-int]
      real-eq-of-int using True by simp
  next
  case False
    hence ge: real-of-int (- (z2 * n1)) ≥ 0
      by (metis mult-zero-left neg-0-le-iff of-int-0-le-iff order-refl zero-le-mult-iff)
    have r-add = - sqrt (of-int (z2 * n1 * (z2 * n1) * int b))
      unfolding r-add-def z2n1-def using sqrt-sqrt[OF ge]
      by (metis minus-minus minus-mult-commute minus-mult-right of-int-mult
        of-int-mult real-of-int-def real-sqrt-minus real-sqrt-mult-distrib2 z2n1-def)
    hence radd: floor r-add = - ceiling (sqrt (of-int (z2 * n1 * (z2 * n1) * int b)))
      by (metis ceiling-def minus-minus)
    show thesis unfolding radd sqrt-int-ceiling-pos[OF ge-int] real-eq-of-int
      using False by simp
qed

lemma comparison-impl:
  (x :: real) ≤ (y :: real) = ge-0 (y - x)
  (x :: real) < (y :: real) = (x ≠ y ∧ ge-0 (y - x))
  by (simp-all add: ge-0-def, linarith+)

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

definition is-rat :: real ⇒ bool where
  [code del]: is-rat x = (x ∈ Q)
lemma [code-unfold]: x ∈ Q ⇔ is-rat x unfolding is-rat-def by auto

lemma ma-is-rat: is-rat (real-of x) = ma-is-rat x
proof (transfer, unfold is-rat-def, clarsimp)
  fix p q :: rat and b :: nat
  let ?p = real-of-rat p
  let ?q = real-of-rat q
  let ?b = real-of-nat b
  let ?b' = real-of-rat (of-nat b)
  assume b: q = 0 ∨ b ∈ sqrt-irrat
  show (?p + ?q * sqrt ?b ∈ Q) = (q = 0)
  proof (cases q = 0)
case False
from False b have b: b ∈ sqrt-irrat by auto
{
  assume ?p + ?q * sqrt ?b ∈ Q
  from this[unfolded Rats-def] obtain r where r: ?p + ?q * sqrt ?b = real-of-rat r
  by auto
  let ?r = real-of-rat r
  from r have real-of-rat (p − r) + ?q * sqrt ?b = 0 by (simp add: of-rat-diff)
  from sqrt-irrat[OF disjI2[OF b] this] False have False by auto
} thus ?thesis by auto
qed

definition sqrt-real x = (if x ∈ Q ∧ x ≥ 0 then (if x = 0 then [0] else (let sx = sqrt x in [sx,—sx])) else [ ])

lemma sqrt-real(simp): assumes x: x ∈ Q
shows set (sqrt-real x) = {y . y * y = x}
proof (cases x ≥ 0)
  case False
  hence ∨ y. y * y ≠ x by auto
  with False show ?thesis unfolding sqrt-real-def by auto
next
  case True
  thus ?thesis using x
  by (cases x = 0, auto simp: Let-def sqrt-real-def)
qed

lemmas ma-code-eqns = ma-code-eqns[of add ma-floor ma-0 ma-1 ma-uminus ma-inverse ma-sqrt ma-plus ma-times ma-equal ma-is-rat comparison-impl

code-datatype real-of

declare real-code-dels[code, code del]
declare real-code-unfold-dels[code-unfold del]
declare real-standard-impls[code]
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

value [101.1 * (3 * sqrt 2 + 6 * sqrt 0.5)]
value [606.2 * sqrt 2 + 0.001 ]
value 101.1 * (3 * sqrt 2 + 6 * sqrt 0.5) = 606.2 * sqrt 2 + 0.001
value 101.1 * (3 * sqrt 2 + 6 * sqrt 0.5) > 606.2 * sqrt 2 + 0.001
value (sqrt 0.1 ∈ Q, sqrt (− 0.09) ∈ 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 \wedge ma-base r = 0 \vee ma-coeff r \neq 0 \wedge prime-product
  (ma-base r)\}

by (transfer, auto)

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

lift-definition real-of-u :: mini-alg-unique \Rightarrow real is real-of .

lift-definition mau-floor :: mini-alg-unique \Rightarrow int is ma-floor .

lift-definition mau-of-rat :: rat \Rightarrow mini-alg-unique is ma-of-rat by (transfer, auto)

lift-definition mau-rat :: mini-alg-unique \Rightarrow rat is ma-rat .

lift-definition mau-base :: mini-alg-unique \Rightarrow nat is ma-base .

lift-definition mau-coeff :: mini-alg-unique \Rightarrow rat is ma-coeff .

lift-definition mau-uminus :: mini-alg-unique \Rightarrow mini-alg-unique is ma-uminus
by (transfer, auto)

lift-definition mau-compatible :: mini-alg-unique \Rightarrow mini-alg-unique \Rightarrow bool is ma-compatible .

lift-definition mau-ge-0 :: mini-alg-unique \Rightarrow bool is ma-ge-0 .

lift-definition mau-inverse :: mini-alg-unique \Rightarrow 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 \Rightarrow mini-alg-unique \Rightarrow mini-alg-unique
is ma-plus
by (transfer, auto simp: ma-normalize-def split: if-splits)

lift-definition mau-times :: mini-alg-unique \Rightarrow mini-alg-unique \Rightarrow mini-alg-unique
is ma-times
by (transfer, auto simp: ma-normalize-def split: if-splits)

lift-definition ma-identity :: mini-alg \Rightarrow mini-alg \Rightarrow 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)
    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 q2 = 0
  by (rule sqrt-irrat[OF i2, of p2 − p1], insert eq True q1, auto)
with True q1 q2 eq show ?thesis by auto
next
case False
hence 1: q1 ≠ 0 prime-product b1 using q1 by auto
  |
  |  assume *: q2 = 0
  |      have q1 = 0
  |        by (rule sqrt-irrat[OF i1, of p1 − p2], insert eq * q2, auto)
  |  with False have False by auto
  |
  hence 2: q2 ≠ 0 prime-product b2 using q2 by auto
from 1 i1 have b1: b1 ≠ 0 unfolding sqrt-irrat-def by (cases b1, auto)
from 2 i2 have b2: b2 ≠ 0 unfolding sqrt-irrat-def by (cases b2, auto)
let ?sq = λ x. x * x
def q3 ≡ p2 − p1
let ?e = rat_of_rat (2 * q2 * q2 * of_nat b2 + ?sq q3 − ?sq q1 * of_nat b1) +
of_rat ((2 * q2 * q3) * sqrt ?b2)
from eq have *: ?q1 * sqrt ?b1 = ?q2 * sqrt ?b2 + ?q3
  by (simp add: q3-def)
from arg-cong[OF this, of ?sq] have θ = (real_of_rat 2 * ?q2 * ?q3) * sqrt ?b2 +
  by auto
also have ... = ?e
  by (simp add: of_rat_mult of_rat_add of_rat_minus)
finally have eq: ?e = 0 by simp
from sqrt-irrat[OF - this] 2 i2 have q3: q3 = 0 by auto
hence p: p1 = p2 unfolding q3-def by simp
let ?b1 = rat_of_nat b1
let ?b2 = rat_of_nat b2
from eq unfolded q3] have eq: ?sq q2 * ?b2 = ?sq q1 * ?b1 by auto
obtain z1 n1 where d1: quotient_of q1 = (z1, n1) by force
obtain z2 n2 where d2: quotient_of q2 = (z2, n2) by force
note pos = quotient_of_denom_pos[OF d1] quotient_of_denom_pos[OF d2]
from id(1) 1(1) pos(1) have z1: z1 ≠ 0 by auto
from id(2) 2(1) pos(2) have z2: z2 ≠ 0 by auto
let ?n1 = rat_of_int n1
let ?n2 = rat_of_int n2
let ?z1 = rat_of_int z1
let ?z2 = rat_of_int z2
from arg_cong[OF eq unfold q3] have λ x. x * ?sq ?n1 * ?sq ?n2, unfolded field_simps
have ?sq (?n1 * ?z2) * ?b2 = ?sq (?n2 * ?z1) * ?b1
  using pos by auto

22
moreover have \(?n1 * ?z2 \neq 0 \Rightarrow ?n2 * ?z1 \neq 0\) using \(z1 z2 pos\) by auto
ultimately obtain \(i1 i2\) where \(0: \text{rat-of-int } i1 \neq 0 \Rightarrow \text{rat-of-int } i2 \neq 0\)
and eq: \(?sq (\text{rat-of-int } i2) * ?b2 = ?sq (\text{rat-of-int } i1) * ?b1\)
unfolding \(\text{of-int-mult}[\text{symmetric}]\) by blast+
let \(?b1 = \text{int } b1\)
let \(?b2 = \text{int } b2\)
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: \(i1 \neq 0 \Rightarrow i2 \neq 0\) by auto
from \(\text{arg-cong}[\text{OF eq, of nat}]\) have \(?sq (\text{nat } (\text{abs } i1)) * b1 = ?sq (\text{nat } (\text{abs } i2)) * b2\)
by (metis \(\text{abs-of-nat}\) eq \(\text{nat-abs-mult-distrib}\) \(\text{nat-int}\))
moreover have \(\text{nat } (\text{abs } i1) > 0 \Rightarrow \text{nat } (\text{abs } i2) > 0\) using 0 by auto
ultimately obtain \(n1 n2\) where \(0: n1 > 0 \Rightarrow n2 > 0\) and eq: \(?sq n1 * b1 = ?sq n2 * b2\) by blast
from \(b1 0\) have \(b1: b1 > 0 \Rightarrow n1 > 0 \Rightarrow n1 * n1 > 0\) by auto
from \(b2 0\) have \(b2: b2 > 0 \Rightarrow n2 > 0 \Rightarrow n2 * n2 > 0\) by auto
\{ fix \(p\)
have \(\text{id1: multiplicity } p (\text{sq } n1 * b1) \mod 2 = \text{multiplicity } p b1 \mod 2\)
unfolding \(\text{multiplicity-product-nat}[\text{OF } b1(2,2)]\) by presburger
have \(\text{id2: multiplicity } p (\text{sq } n2 * b2) \mod 2 = \text{multiplicity } p b2 \mod 2\)
unfolding \(\text{multiplicity-product-nat}[\text{OF } b2(3,1)]\) by presburger
\}
with \(\text{multiplicity } p b1 = \text{multiplicity } p b2\) by simp
with \(\text{b1(1) b2(1)}\) have \(b: b1 = b2\) by (rule \(\text{multiplicity-eq-nat}\))
from \(*[\text{unfolded } b q3]\) \(\text{b1(1) b2(1)}\) have \(q: q1 = q2\) by simp
from \(p q b\) show \(\text{thesis}\) by blast
qed
qed
qed simp

lift-definition \(\text{mau-sqrt :: mini-alg-unique} \Rightarrow \text{mini-alg-unique}\) is
\(\lambda ma. \text{let } (a,b) = \text{quotient-of } (\text{ma-rat } ma); (\text{sq,fact}) = \text{prime-product-factor}\)
\(\text{nat } (\text{abs } a * b)\);
\(\text{ma}' = \text{ma-of-rat } (\text{of-int } (\text{sgn}(a)) \ast \text{of-nat } sq \ast \text{of-int } b)\)
in \(\text{ma-times } ma'(\text{ma-sqrt } (\text{ma-of-rat } (\text{of-nat } \text{fact})))\)
proof
fix \(ma:: \text{mini-alg}\)
let \(?num = \text{let } (a,b) = \text{quotient-of } (\text{ma-rat } ma); (\text{sq,fact}) = \text{prime-product-factor}\)
(nat (|a| * b)); 
  ma' = ma-of-rat (rat-of-int (sgn a) * rat-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 ppf: prime-product-factor (nat (abs a * b)) = (sq,fact)
by force
  def asq = rat (prime-product (b
  using real-of-int b
  force (rat
  def sqrt = def ma = def asq
  def ma' = ma-of-rat asq
  def sqrt = def ma = ma-of-rat (rat-of-nat fact))
  have num: ?num = ma-times ma' sqrt unfolding q ppf 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. int i * 1 = i \∧ i :: rat. i / 1 = i rat-of-int 1 = 1 inverse (1 ::
  rat) = 1
  \∧ n. nat |int n| = n by auto
  from prime-product-factor[OF ppf] 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 \implies 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-denom-pos[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 ppf: prime-product-factor (nat (|a| * b)) = (sq, fact) by
  force
  from prime-product-factor[OF ppf] 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-eq)
  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

24
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 ... = of-nat fact \times (of-nat sq \times of-nat sq) unfolding ab of-nat-mult
  by simp
also have sqrt (of-int (sgn a) \times (of-nat fact \times (of-nat sq \times of-nat sq))) =
  of-int (sgn a) \times sqrt (of-int fact) \times of-int (sq)
  unfolding real-sqrt-mult-distrib by simp
finally have r: sqrt (real-of r) = real-of-int (sgn a) \times real-of-int sq / real-of-int b
  unfolding ma-times by simp
let \(?asqb\) = ma-of-rat (rat-of-int (sgn a) \times rat-of-int sq / rat-of-int b)
let \(?f\) = ma-of-rat (rat-of-int fact)
let \(?sq\) = ma-sqrt \(?f\)
have sq: 0 \leq ma-rat \(?f\) \leq 0 by ((transfer, simp)+)
have compat: \(\forall m. (\text{ma-compatible } \text{of } \text{asqb } m) = True\)
  by (transfer, auto)
show sqrt (real-of r) =
  real-of
    (let (a, b) = quotient-of (ma-rat r); (sq, fact) = prime-product-factor (nat (|a| \times b))):
    ma' = ma-of-rat (rat-of-int (sgn a) \times rat-of-int sq / rat-of-int b)
    in ma-times ma' (ma-sqrt (ma-of-rat (rat-of-int fact))))
  unfolding q ppf Let-def split
  unfolding r
  unfolding ma-times[symmetric, of \text{asqb, unfolded compat if-True}]
  unfolding ma-sqrt-main[symmetric]
  unfolding ma-of-rat[symmetric]
  by (simp add: of-rat-divide of-rat-mult)
Qed

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

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

lemma mau-1: 1 = 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: ge-0 (real-of-u x) = mau-ge-0 x using mau-ge-0
by (transfer, auto)

**Definition** real-lt :: real ⇒ real ⇒ bool where real-lt = op <

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*

with nless show ?thesis unfolding real-lt-def by auto

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-is-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 $(\text{shows } p \circ qb) []$
else $(\text{shows } p \circ \text{shows } "+" \circ qb) []$.

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

**definition** show-real :: real $\Rightarrow$ string where
  show-real $x = (\text{if } (\exists \ y. \ x = \text{real-of-u } y) \text{ then } \text{mau-show-real (THE } y. \ x = \text{real-of-u } y) \text{ else []})$

**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
decode datatype real-of-u
declare real-code-dels[code, code del]
declare mau-code-eqns[code del]
declare real-code-unfold-dels[code-unfold del]
declare real-standard-impls[code]
declare mau-code-eqns[code]

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

**value** $[101.1 \ast (\text{sqrt } 18 + 6 \ast \text{sqrt } 0.5)]$
**value** $[324 \ast \text{sqrt } 7 + 0.001]$
**value** $101.1 \ast (\text{sqrt } 18 + 6 \ast \text{sqrt } 0.5) = 324 \ast \text{sqrt } 7 + 0.001$
**value** $101.1 \ast (\text{sqrt } 18 + 6 \ast \text{sqrt } 0.5) > 324 \ast \text{sqrt } 7 + 0.001$
**value** show-real $(101.1 \ast (\text{sqrt } 18 + 6 \ast \text{sqrt } 0.5))$
**value** $(\text{sqrt } 0.1 \in \mathbb{Q}, \text{sqrt } (-0.09) \in \mathbb{Q})$

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**


