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 3

5 A unique representation of real numbers via triples 8

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 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 j: j \neq i shows multiplicity j i = 0
(proof)
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 \leq 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
prime-product-factor-main factor-sq factor-pr limit n i =
(if i \leq limit \land i \geq 2 then
(if i dvd n
then (let n' = n div i in
(if i dvd n' then
let n'' = n' div i in

2

2
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)
    | [] ⇒ 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)) ⟨proof⟩

termination
⟨proof⟩

lemma prime-product-factor-main: assumes ¬ (∃ s. s * s = n)
  and limit = nat (root-nat-floor 3 n)
  and m = factor-sq * factor-sq * factor-pr * n
  and prime-product-factor-main factor-sq factor-pr limit n i = (sq, p)
  and i ≥ 2
  and \( ∀ j. j ≥ 2 \implies j > i \implies \neg j \text{ dvd } n \)
  and \( ∀ j. j < i \implies \text{multiplicity } j \text{ factor-pr} ≤ 1 \)
  and \( ∀ j. j ≥ i \implies \text{multiplicity } j \text{ factor-pr} = 0 \)
  and factor-pr > 0
  shows m = sq * sq * p ∧ prime-product p ⟨proof⟩

definition prime-product-factor :: nat ⇒ nat × nat where
prime-product-factor n = (case sqrt-nat n of
  (Cons s _) ⇒ (s,1)
  | [] ⇒ prime-product-factor-main 1 1 (nat (root-nat-floor 3 n)) n 2)

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

lemma prime-product-factor: assumes pf: prime-product-factor n = (sq,p)
  shows n = sq * sq * p ∧ prime-product p ⟨proof⟩

end

4 A representation of real numbers via triples

theory Real-Impl
imports
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 \( \sqrt{\text{sqrt}} \) is irrational

definition sqrt-irrat = \{ q :: \text{nat}. \neg (\exists \ p. \ p \cdot p = \text{rat-of-nat} \ q) \}

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 \( q = 0 \)
⟨proof⟩

Collect existing code equations for reals, so that they can be deleted.

lemmas real-code-dels =
refl[of op + :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}]
refl[of uminus :: \text{real} \Rightarrow \text{real}]
refl[of op − :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}]
refl[of op * :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}]
refl[of inverse :: \text{real} \Rightarrow \text{real}]
refl[of op / :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real}]
refl[of floor :: \text{real} \Rightarrow \text{int}]
refl[of sqrt]
refl[of HOL.equal :: \text{real} \Rightarrow \text{real} \Rightarrow \text{bool}]
refl[of op ≤ :: \text{real} \Rightarrow \text{real} \Rightarrow \text{bool}]
refl[of op < :: \text{real} \Rightarrow \text{real} \Rightarrow \text{bool}]
refl[of 0 :: \text{real}]
refl[of 1 :: \text{real}]

lemma real-code-unfold-dels:
\text{of-rat} \equiv \text{Ratreal}
\text{of-int} \ a \equiv (\text{of-rat} \ \text{(of-int} \ a)::\text{real})
\text{0} \equiv (\text{of-rat} \ 0::\text{real})
\text{1} \equiv (\text{of-rat} \ 1::\text{real})
\text{numeral} \ k \equiv (\text{of-rat} \ \text{(numeral} \ k)::\text{real})
\text{− numeral} \ k \equiv (\text{of-rat} \ \text{− numeral} \ k::\text{real})
⟨proof⟩

lemma real-standard-impls:
\( \frac{x::\text{real}}{y::\text{real}} = x \cdot \text{inverse} \ y \)
\( x::\text{real} − y::\text{real} = x + (−y) \)
⟨proof⟩

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 \lor b \in sqrt-irrat\}
(proof)

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) (proof)

lift-definition ma-of-rat :: rat ⇒ mini-alg is λ x. (x, 0, 0) (proof)

lift-definition ma-base :: mini-alg ⇒ nat is fst (proof)

lift-definition ma-coeff :: mini-alg ⇒ nat is snd o snd (proof)

lift-definition ma-uminus :: mini-alg ⇒ mini-alg is
λ (p1, q1, b1). (− p1, − q1, b1) (proof)

lift-definition ma-compatible :: mini-alg ⇒ mini-alg ⇒ bool is
λ (p1, q1, b1) (p2, q2, b2). q1 = 0 \lor q2 = 0 \lor b1 = b2 (proof)

definition ma-normalize :: rat × rat × nat ⇒ rat × rat × nat where
  ma-normalize x ≡ case x of (a, b, c) ⇒ if b = 0 then (a, 0, 0) else (a, b, c)

lemma ma-normalize-case[simp]: (case ma-normalize r of (a, b, c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-nat c))
  = (case r of (a, b, c) ⇒ real-of-rat a + real-of-rat b * sqrt (of-nat c))
(proof)

lift-definition ma-plus :: mini-alg ⇒ mini-alg ⇒ mini-alg is
λ (p1, q1, b1) (p2, q2, b2). if q1 = 0 then
  (p1 + p2, q1 + q2, b1) else ma-normalize (p1 + p2, q1 + q2, b1) (proof)

lift-definition ma-times :: mini-alg ⇒ mini-alg ⇒ mini-alg is
λ (p1, q1, b1) (p2, q2, b2). if q1 = 0 then
  ma-normalize (p1 * p2, p1 * q2, b2) else
  ma-normalize (p1 * p2 + of-nat b2 * q1 * q2, p1 * q2 + q1 * p2, b1) (proof)

lift-definition ma-inverse :: mini-alg ⇒ mini-alg is
λ (p, q, b). let d = inverse (p * p − of-nat b * q * q) in
  ma-normalize (p + d, − q * d, b) (proof)

lift-definition ma-floor :: mini-alg ⇒ int is
λ (p, q, b). case (quotient-of p, quotient-of q) of ((z1, n1), (z2, n2)) ⇒
  let z2n1 = z2 * n1; z1n2 = z1 * n2; n12 = n1 * n2; prod = z2n1 * z2n1 * int b in
  (z1n2 + (if z2n1 ≥ 0 then sqrt-int-floor-pos prod else − sqrt-int-ceiling-pos prod)) div n12 (proof)
lift-definition \texttt{ma-sqrt} :: \texttt{mini-alg} \Rightarrow \texttt{mini-alg} is
\begin{align*}
\lambda (p,q,b). \ & \text{let } (a,b) = \text{quotient-of } p; \ a = \text{abs } (a * b) \text{ in } \\
\ & \text{case } \text{sqrt-int } a \text{ of } [] \Rightarrow \ (0, \text{inverse } (\text{of-int } b), \text{nat } a) \ | \ (\text{Cons } s \ ) \Rightarrow \ (\text{of-int } s \ / \ \text{of-int } b, 0, 0) \\
\end{align*}
\langle \text{proof} \rangle

lift-definition \texttt{ma-equal} :: \texttt{mini-alg} \Rightarrow \texttt{mini-alg} \Rightarrow \texttt{bool} is
\begin{align*}
\lambda (p_1,q_1,b_1), (p_2,q_2,b_2). \ & \ p_1 = p_2 \land q_1 = q_2 \land (q_1 = 0 \lor b_1 = b_2) \\
\text{\langle \text{proof} \rangle}
\end{align*}

lift-definition \texttt{ma-ge-0} :: \texttt{mini-alg} \Rightarrow \texttt{bool} is
\begin{align*}
\lambda (p,q,b). \ & \text{let } bqq = \text{of-nat } b * q * q; \ pp = p * p \text{ in } \\
\ & 0 \leq p \land bqq \leq pp \lor 0 \leq q \land pp \leq bqq \text{ \langle \text{proof} \rangle}
\end{align*}

lift-definition \texttt{ma-is-rat} :: \texttt{mini-alg} \Rightarrow \texttt{bool} is
\begin{align*}
\lambda (p,q,b). \ q = 0 \text{ \langle \text{proof} \rangle}
\end{align*}

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

\begin{align*}
\text{lemma } \texttt{ma-ge-0}: \texttt{ge-0 } (\text{real-of } x) = \texttt{ma-ge-0} x \\
\text{\langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-0}: \texttt{0} = \texttt{real-of } (\texttt{ma-of-rat } 0) \text{ \langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-1}: \texttt{1} = \texttt{real-of } (\texttt{ma-of-rat } 1) \text{ \langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-uminus}: \ & \ - (\text{real-of } x) = \texttt{real-of } (\texttt{ma-uminus } x) \\
\text{\langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-inverse}: \texttt{inverse } (\texttt{real-of } r) = \texttt{real-of } (\texttt{ma-inverse } r) \\
\text{\langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-sqrt-main}: \texttt{ma-rat } r \geq 0 \implies \texttt{ma-coeff } r = 0 \implies \texttt{sqrt } (\texttt{real-of } r) = \\
\texttt{real-of } (\texttt{ma-sqrt } r) \text{ \langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-sqrt}: \texttt{sqrt } (\texttt{real-of } r) = (\text{if } \texttt{ma-coeff } r = 0 \text{ then } \\
\text{ if } \texttt{ma-rat } r \geq 0 \text{ then } \texttt{real-of } (\texttt{ma-sqrt } r) \text{ else } \texttt{real-of } (\texttt{ma-sqrt } (\texttt{ma-uminus } r)) \\
\text{ else } \texttt{Code.abort } (\text{STR "cannot represent sqrt of irrational number"}) \ (\lambda \ - \ . \ \texttt{sqrt} \ (\texttt{real-of } r))) \\
\text{\langle \text{proof} \rangle}
\end{align*}

\begin{align*}
\text{lemma } \texttt{ma-plus}: \ & \ (\texttt{real-of } r_1 + \texttt{real-of } r_2) = (\text{if } \texttt{ma-compatible } r_1 \ r_2 \ \text{ then } \texttt{real-of } (\texttt{ma-plus } r_1 \ r_2) \text{ else } \\
\text{ \ Code.abort } (\text{STR "different base"}) \ (\lambda \ - \ . \ \texttt{real-of } r_1 + \texttt{real-of } r_2)) \text{ \langle \text{proof} \rangle}
\end{align*}
proof

lemma ma-times:
(real-of r1 * 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))
(proof)

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)

lemma ma-floor: floor (real-of r) = ma-floor r
(proof)

lemma comparison-impl:
(x :: real) ≤ (y :: real) = ge-0 (y − x)
(x :: real) < (y :: real) = (x ≠ y ∧ ge-0 (y − x))
(proof)

lemma ma-of-rat: real-of-rat r = real-of (ma-of-rat r)
(proof)

definition is-rat :: real ⇒ bool where
[code del]: is-rat x = (x ∈ Q)

lemma [code-unfold]: x ∈ Q ↔ is-rat x (proof)

lemma ma-is-rat: is-rat (real-of x) = ma-is-rat x
(proof)

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)

lemmas ma-code-eqns = ma-ge-0 ma-floor ma-0 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]
Some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

\[101.1 \times (3 \times \sqrt{2} + 6 \times \sqrt{0.5})\]
\[606.2 \times \sqrt{2 + 0.001}\]
\[101.1 \times (3 \times \sqrt{2} + 6 \times \sqrt{0.5}) = 606.2 \times \sqrt{2 + 0.001}\]
\[101.1 \times (3 \times \sqrt{2} + 6 \times \sqrt{0.5}) > 606.2 \times \sqrt{2 + 0.001}\]
\[(\sqrt{0.1} \in \mathbb{Q}, \sqrt{(-0.09)} \in \mathbb{Q})\]

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 \land ma-base r = 0 \lor ma-coeff r \neq 0 \land prime-product (ma-base r)}
(proof)

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

lift-definition real-of-u :: mini-alg-unique \Rightarrow real is real-of (proof)
lift-definition mau-floor :: mini-alg-unique \Rightarrow int is ma-floor (proof)
lift-definition mau-of-rat :: rat \Rightarrow mini-alg-unique is ma-of-rat (proof)
lift-definition mau-rat :: mini-alg-unique \Rightarrow rat is ma-rat (proof)
lift-definition mau-base :: mini-alg-unique \Rightarrow nat is ma-base (proof)
lift-definition mau-coeff :: mini-alg-unique \Rightarrow rat is ma-coeff (proof)
lift-definition mau-uminus :: mini-alg-unique \Rightarrow mini-alg-unique is ma-uminus (proof)
lift-definition mau-compatible :: mini-alg-unique \Rightarrow mini-alg-unique \Rightarrow bool is ma-compatible (proof)
lift-definition mau-ge-0 :: mini-alg-unique \Rightarrow bool is ma-ge-0 (proof)
lift-definition mau-inverse :: mini-alg-unique ⇒ mini-alg-unique is ma-inverse
⟨proof⟩
lift-definition mau-plus :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
is ma-plus
⟨proof⟩
lift-definition mau-times :: mini-alg-unique ⇒ mini-alg-unique ⇒ mini-alg-unique
is ma-times
⟨proof⟩
lift-definition ma-identity :: mini-alg ⇒ mini-alg ⇒ bool is op =
⟨proof⟩
lift-definition mau-equal :: mini-alg-unique ⇒ mini-alg-unique ⇒ bool is ma-identity
⟨proof⟩
lift-definition mau-is-rat :: mini-alg-unique ⇒ bool is ma-is-rat
⟨proof⟩

lemma mau-floor: floor (real-of-u r) = mau-floor r
⟨proof⟩
lemma mau-inverse: inverse (real-of-u r) = real-of-u (mau-inverse r)
⟨proof⟩
lemma mau-uminus: − (real-of-u r) = real-of-u (mau-uminus r)
⟨proof⟩
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))
⟨proof⟩
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))
⟨proof⟩
lemma real-of-u-inj[simp]: real-of-u x = real-of-u y ←→ x = y
⟨proof⟩

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⟩
lemma sqrt-sgn[simp]: sqrt (of-int (sgn a)) = of-int (sgn a)
⟨proof⟩
lemma mau-sqrt-main: mau-coeff r = 0 =⇒ sqrt (real-of-u r) = real-of-u (mau-sqrt r)
⟨proof⟩
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") (λ -. sqrt (real-of-u r)))
⟨proof⟩

lemma mau-0: 0 = real-of-u (mau-of-rat 0) ⟨proof⟩

lemma mau-1: 1 = real-of-u (mau-of-rat 1) ⟨proof⟩

lemma mau-equal:
HOL.equal (real-of-u r1) (real-of-u r2) = mau-equal r1 r2 ⟨proof⟩

lemma mau-ge-0: ge-0 (real-of-u x) = mau-ge-0 x ⟨proof⟩

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⟩

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))
⟨proof⟩

lemma mau-is-rat: is-rat (real-of-u x) = mau-is-rat x ⟨proof⟩

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 ((shows p o shows "+") o qb) []
else ((shows p o shows "+") o qb) []
⟨proof⟩

lift-definition mau-show-real :: mini-alg-unique ⇒ string is mau-show-real ⟨proof⟩
definition show-real :: real ⇒ string where
  show-real x = (if (∃ y. x = real-of-u y) then mau-show-real (THE y. x = real-of-u y) else [])

lemma mau-show-real: show-real (real-of-u x) = mau-show-real x
⟨proof⟩

lemmas mau-code-eqns = mau-floor mau-0 mau-1 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-dels[code, code del]
declare ma-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 * (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 ∈ ℚ, sqrt (− 0.09) ∈ 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


