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

René Thiemann

April 20, 2020

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 7

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 CeTA [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.

```isar
theory Real-Impl-Auxiliary
imports
  HOL-Computational-Algebra.Primes
begin

lemma multiplicity-prime:
  assumes p: prime (i :: nat) and ji: j ≠ i
  shows multiplicity j i = 0
  ⟨proof ⟩
end

3 Prime products

```isar
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. prime 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 ⇒ nat ⇒ nat ⇒ nat ⇒ nat ⇒ nat × nat
  where prime-product-factor-main (factor-sq factor-pr limit n i =
    (if i ≤ limit ∧ i ≥ 2 then
      (if i dvd n
        then (let n' = n div i in
         (if i dvd n' then
          ```
let \( n'' = n' \div i \) in
\[
\text{prime-product-factor-main} \ (\text{factor-sq} \ast i) \ \text{factor-pr} \ (\text{nat} \ (\text{root-nat-floor} \ 3 \ n'')) \ n'' \ i
\]
else
\[
\begin{cases}
\text{case sqrt-nat } n' \ of \\
\quad \text{Cons } s \ n - \Rightarrow (\text{factor-sq} \ast s, \text{factor-pr} \ast i) \\
\quad [] \Rightarrow \text{prime-product-factor-main} \ \text{factor-sq} \ (\text{factor-pr} \ast i) \ (\text{nat} \ (\text{root-nat-floor} \ 3 \ n'')) \ n' \ (\text{Suc } i)
\end{cases}
\]
\]
\]
\]
else
\[
\text{prime-product-factor-main} \ \text{factor-sq} \ \text{factor-pr} \ \text{limit} \ n \ (\text{Suc } i)
\]
\]
else
\[
(\text{factor-sq}, \text{factor-pr} \ast n) \langle \text{proof} \rangle
\]
termination
\[
\langle \text{proof} \rangle
\]

lemma prime-product-factor-main: assumes \( \neg (\exists \ s. \ s \ast s = n) \)
and \( \text{limit} = \text{nat} \ (\text{root-nat-floor} \ 3 \ n) \)
and \( m = \text{factor-sq} \ast \text{factor-sq} \ast \text{factor-pr} \ast n \)
and \( \text{prime-product-factor-main} \ \text{factor-sq} \ \text{factor-pr} \ \text{limit} \ n \ i = (sq, p) \)
and \( i \geq 2 \)
and \( \bigwedge j. \ j \geq 2 \implies j < i \implies \neg j \ \text{dvd} \ n \)
and \( \bigwedge j. \ \text{prime } j \implies j < i \implies \text{multiplicity } j \ \text{factor-pr} \leq 1 \)
and \( \bigwedge j. \ \text{prime } j \implies j \geq i \implies \text{multiplicity } j \ \text{factor-pr} = 0 \)
and \( \text{factor-pr} > 0 \)
shows \( m = sq \ast sq \ast p \land \text{prime-product } p \)
\[
\langle \text{proof} \rangle
\]
definition prime-product-factor :: nat \Rightarrow nat \times nat where
\[
\text{prime-product-factor} \ n = (\text{case sqrt-nat } n \ of \\
\quad \text{Cons } s \ - \Rightarrow (s,1) \\
\quad [] \Rightarrow \text{prime-product-factor-main} \ 1 \ 1 \ (\text{nat} \ (\text{root-nat-floor} \ 3 \ n)) \ n \ 2)
\]
lemma prime-product-one[simp, intro]: prime-product 1
\[
\langle \text{proof} \rangle
\]
lemma prime-product-factor: assumes pf: prime-product-factor \ n = (sq,p)
shows \( n = sq \ast sq \ast p \land \text{prime-product } p \)
\[
\langle \text{proof} \rangle
\]
end

4 A representation of real numbers via triples

theory Real-Impl
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 the 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

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

\[
\text{lemma } \text{sqrt-irrat}: \text{assumes choice: } q = 0 \lor b \in \text{sqrt-irrat}
\text{and eq: } \text{real-of-rat } p + \text{real-of-rat } q \cdot \sqrt{\text{of-nat } b} = 0
\text{shows } q = 0
\langle \text{proof} \rangle
\]

To represent numbers of the form $p + q \cdot \sqrt{b}$, use mini algebraic numbers, i.e., triples $(p, q, b)$ with irrational $\sqrt{b}$.

\[
\text{typedef } \text{mini-alg} = \{(p, q, b) | (p :: \text{rat}) (q :: \text{rat}) (b :: \text{nat}). q = 0 \lor b \in \text{sqrt-irrat}\}
\langle \text{proof} \rangle
\]

\text{setup-lifting type-definition-mini-alg}

\[
\text{lift-definition } \text{real-of :: mini-alg } \Rightarrow \text{real is } \lambda (p, q, b). \text{of-rat } p + \text{of-rat } q \cdot \sqrt{\text{of-nat } b} \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-of-rat :: } \text{rat } \Rightarrow \text{mini-alg is } \lambda x. (x,0,0) \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-rat :: mini-alg } \Rightarrow \text{rat is } \text{fst} \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-base :: mini-alg } \Rightarrow \text{rat is } \text{snd o snd} \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-coeff :: mini-alg } \Rightarrow \text{rat is } \text{fst o snd} \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-uminus :: mini-alg } \Rightarrow \text{mini-alg is } \lambda (p1, q1, b1). (\neg p1, - q1, b1) \langle \text{proof} \rangle
\]

\[
\text{lift-definition } \text{ma-compatible :: mini-alg } \Rightarrow \text{mini-alg } \Rightarrow \text{bool is } \lambda (p1, q1, b1) (p2, q2, b2). q1 = 0 \lor q2 = 0 \lor b1 = b2 \langle \text{proof} \rangle
\]

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

\[
\text{lemma } \text{ma-normalize-case[simp]}: \text{case } \text{ma-normalize } r \text{ of } (a,b,c) \Rightarrow \text{real-of-rat } a + \text{real-of-rat } b \cdot \sqrt{\text{of-nat } c})
\text{= } (\text{case } r \text{ of } (a,b,c) \Rightarrow \text{real-of-rat } a + \text{real-of-rat } b \cdot \sqrt{\text{of-nat } c})
\langle \text{proof} \rangle
\]
lemma ma-uminus :: mini-alg \Rightarrow mini-alg \Rightarrow mini-alg is
\lambda (p1,q1,b1) (p2,q2,b2). if q1 = 0 then
(p1 + p2, q2, b2) else ma-normalize (p1 + p2, q1 + q2, b1) \langle \text{proof} \rangle

lift-definition ma-times :: mini-alg \Rightarrow mini-alg \Rightarrow mini-alg is
\lambda (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) \langle \text{proof} \rangle

lift-definition ma-floor :: mini-alg \Rightarrow int is
\lambda (p,q,b). let d = inverse (p * p - of-nat b * q * q) in
ma-normalize (p * d, -q * d, b) \langle \text{proof} \rangle

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

lift-definition ma-equal :: mini-alg \Rightarrow mini-alg \Rightarrow bool is
\lambda (p1,q1,b1) (p2,q2,b2).
p1 = p2 \land q1 = q2 \land (q1 = 0 \lor b1 = b2) \langle \text{proof} \rangle

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

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

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

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

lemma ma-0: 0 = real-of (ma-of-rat 0) \langle \text{proof} \rangle

lemma ma-1: 1 = real-of (ma-of-rat 1) \langle \text{proof} \rangle

lemma ma-uminus:
- (real-of x) = real-of (ma-uminus x)
lemma ma-inverse: inverse (real-of r) = real-of (ma-inverse r)
⟨proof⟩
lemma ma-sqrt-main: ma-rat r ≥ 0 ⇒ ma-coeff r = 0 ⇒ sqrt (real-of r) = real-of (ma-sqrt r)
⟨proof⟩
lemma ma-sqrt: sqrt (real-of r) = (if ma-coeff r = 0 then
  (if ma-rat r ≥ 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") (λ -. sqrt (real-of r)))
⟨proof⟩
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") (λ -. real-of r1 + real-of r2))
⟨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-abbrev]: is-rat x ←→ x ∈ Q
lemma ma-is-rat: is-rat (real-of x) = ma-is-rat x
\begin{proof}
\end{proof}

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

**lemma** \( \text{sqrt-real [simp]} \): assumes \( x : x \in \mathbb{Q} \)

shows \( \text{set } (\text{sqrt-real } x) = \{ y . y \ast y = x \} \)

\begin{proof}
some tests with small numbers. To work on larger number, one should additionally import the theories for efficient calculation on numbers

\begin{itemize}
\item \textbf{value} \[101.1 \ast (3 \ast \text{sqrt } 2 + 6 \ast \text{sqrt } 0.5)\]
\item \textbf{value} \[606.2 \ast \text{sqrt } 2 + 0.001\]
\item \textbf{value} \[101.1 \ast (3 \ast \text{sqrt } 2 + 6 \ast \text{sqrt } 0.5) = 606.2 \ast \text{sqrt } 2 + 0.001\]
\item \textbf{value} \[101.1 \ast (3 \ast \text{sqrt } 2 + 6 \ast \text{sqrt } 0.5) > 606.2 \ast \text{sqrt } 2 + 0.001\]
\item \textbf{value} \((\text{sqrt } 0.1 \in \mathbb{Q}, \text{sqrt } (-0.09) \in \mathbb{Q})\)
\end{itemize}

\end{proof}

\section{A unique representation of real numbers via triples}

theory \( \text{Real-Unique-Impl} \)

imports
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.

```latex
typedef mini-alg-unique = 
\{ \text{r :: mini-alg , ma-coeff r = 0 ∧ ma-base r = 0 ∨ ma-coeff r ≠ 0 ∧ prime-product (ma-base r)} \}
 ⟨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 = (<)

The following code equation terminates if it is started on two different inputs.

**lemma** real-lt [code equation]: 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.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** mau-show-real :: mini-alg ⇒ string is
λ (p,q,b). let sb = shows "sqrt(" ◦ shows b ◦ shows ")";
qb = (if q = 1 then sb else if q = −1 then shows "-" ◦ sb else shows q ◦ shows "-" ◦ sb) in
if q = 0 then shows p [] else
if p = 0 then qb [] else
if q < 0 then ((shows p ◦ qb) []) else ((shows p ◦ shows "-" ◦ qb) [])

 ⟨proof⟩

**lift-definition** mau-show-real :: mini-alg-unique ⇒ string is mau-show-real ⟨proof⟩

**overloading** show-real ≡ show-real begin

**definition** show-real
where show-real x ≡
(if (∃ y. x = real-of-u y) then mau-show-real (THE y. x = real-of-u y) else [])

end

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

**code-datatype** real-of-u
declare [code drop:
  plus :: real ⇒ real ⇒ real
  uminus :: real ⇒ real
  times :: real ⇒ real ⇒ real
  inverse :: real ⇒ real
  floor :: real ⇒ int
  sqrt
  HOL.equal :: real ⇒ real ⇒ bool
  ge-0
  is-rat
  less :: real ⇒ real ⇒ bool
  less-eq :: real ⇒ real ⇒ bool
]

lemmas mau-code-eqns [code] = 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

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 (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


