URL: http://www.elsevier.nl/locate/entcs/volume85.html 16 pages

A simple canonical representation of rational numbers

Yves Bertot [1,2]

Project Lemme
INRIA Sophia Antipolis
France

Abstract

We propose to use a simple inductive type as a basis to represent the field of rational numbers. We describe the relation between this representation of numbers and the representation as fractions of non-zero natural numbers. The usual operations of comparison, multiplication, and addition are then defined in a naive way. The whole construction is used to build a model of the set of rational numbers as an ordered archimedian field. All constructions have been modeled and verified in the Coq proof assistant.

This work started as a quest to find a simple language to represent strictly positive rational numbers. It started as a reflexion on the proof part of representations of rational numbers as reduced fractions: the proof must then be a proof that the numerator and denominator are respectively prime and this proof can be viewed as a trace of Euclid's algorithm to compute the greatest common divisor of two numbers. Looking further, this trace can be used directly as a data-structure to represent rational numbers.

1 From fractions to `Qplus` and back

We propose to use a very simple formal language, which we will call Q^+, and is given by the following syntax:

$$x := 1 | Nx | Dx$$

This language can easily be encoded as a recursively defined type in a functional programming language or as an inductive type in a theorem prover. For instance, the definition in Coq [5] is the following one:

```
Inductive Qplus : Set :=
```

[1] thanks to Milad Niqui and Loïc Pottier for many discussions on this subject.
[2] Yves.Bertot@sophia.inria.fr

```
One : Qplus | N : Qplus -> Qplus | D : Qplus -> Qplus.
```

Being given a pair of strictly positive natural numbers (p, q), actually representing the fraction $\frac{p}{q}$, we construct the term in our language by recusively applying the following rules:

- if $p = q$ then the term associated to $\frac{p}{q}$ is 1,
- if $p > q$ then the term is Ny where y is the term associated to $\frac{p-q}{q}$ (note that $p - q > 0$),
- if $p < q$ then the term is Dy where y is the term associated to $\frac{p}{q-p}$.

This recursive technique always terminates: if there is a recursive call, then the sum of the two elements in the pair is strictly smaller than the sum of the two initial elements. Thus there is a quantity that decreases strictly at each recursive step: this ensures termination.

There may be several pairs of natural numbers representing the same rational number: in this sense the set of strictly positive rational numbers can be viewed as a quotient set obtained from a partition of the set of pairs of strictly positive rational numbers, but it turns out that the term in Q^+ constructed in this manner does not depend on the pair of numbers that was chosen. Here is another formulation of the same algorithm that shows why. Now this algorithm is described by a function c:

- $c(1) = 1$,
- if $x > 1$, $c(x) = Nc(x - 1)$,
- if $x < 1$ $c(x) = Dc(\frac{1}{\frac{1}{x}-1})$.

It is a simple computation to verify that the two algorithms perform the same steps.

Given a word w in the language Q^+, we can interpret this word as a fraction using the following recursive algorithm.

- if $w = 1$, then the fraction is $\frac{1}{1}$,
- if $w = Ny$ and y can be interpreted as the fraction $\frac{p}{q}$, then w can be interpreted as $\frac{p+q}{q}$,
- if $w = Dy$ then w can be interpreted as $\frac{p}{p+q}$.

The fraction we obtain in this manner is always reduced: the greatest common divisor of p and q is 1. This can be proved by recursion over the length of w.

- Base case: if the length is 1, then $w = 1$, the fraction is $\frac{1}{1}$, which is reduced,
- Let us suppose the length is $n + 1$, where $n \geq 1$, let us suppose any word of length n is interpreted in a reduced fraction. Now w can have one of two forms:

(i) $w = Ny$. In this case y can be interpreted in a reduced fraction $\frac{p}{q}$ and w is interpreted in $\frac{p+q}{q}$. Any divisor common to $p + q$ and q is also common

to $p+q-q$ and q: it is a divisor of p and q. Since $\frac{p}{q}$ is reduced this divisor can only be 1.

(ii) $w = Dy$. In this case we can reason symmetrically to the previous case.

Another way to present the interpretation algorithm is to view it as function returning a rational number and write as a function i. We then have the following presentation:

- $i(1) = 1$,
- $i(Ny) = 1 + i(y)$,
- $i(Dy) = \frac{1}{1+\frac{1}{i(y)}}$.

The functions c and i are clearly inverse to one another: they establish a bijection between the set of strictly positive rational numbers and the language Q^+.

2 The rationale behind rationals

At first sight, this looks like a contrived way to compute the greatest common divisor of two numbers p and q: just compute $c(\frac{p}{q})$ then interpret it as a reduced fraction $\frac{p'}{q'}$ and the greatest common divisor of p and q is the number $\frac{p}{p'} = \frac{q}{q'}$. In fact, we have not done anything else than construct a trace of the decisions made by a simplified form of the usual algorithm to compute the greatest common divisor, known as Euclid's algorithm.

When given two numbers p and q, the greatest common divisor algorithm requires that one divide p by q if $p > q$. If the remainder r is 0 then the greatest common divisor is q, otherwise one should proceed to compute compute the greatest common divisor of r and q. If $q > p$ then one should divide q by p and proceed by computing the greatest common divisor of p and r. If $p = q$ then the greatest common divisor is p.

A simplified form of this algorithm is the algorithm where one subtract q from p when $p > q$ instead of dividing (this is actually the form that was described by Euclid). In the long run, this has the same effect as division: one eventually reaches a point where either $p = q$ (which would correspond to a null remainder in the division) or subtraction has to be done in the other way.

In the simplified form, there is a three way choice that is made based on whether p is larger or smaller than q, or equal to q. The succession of choices made in the algorithm is simply what is recorded in the terms of the Q^+ language.

This is where the representation comes from: the initial motivation was to construct a datastructure to represent rational numbers, so that syntactic equality would coincide with the equality as rational numbers. The usual datastructure, where rational numbers are represented as fractions, that is, as pairs combining a natural number (for the numerator) and a strictly positive natural number (for the denominator) obviously does not fit the requirement:

the two fractions $\frac{22}{10}$ and $\frac{11}{5}$ are not syntactically equal, even though they do represent the same rational number (for this number our representation is $NNDDDD1$.

In type theory, it is usual to manipulate objects that combine data and proofs of properties satisfied by this data. Thus, to have fractions where syntactic equality is meaningful, it would be relevant to consider only reduced fractions, represented by triples where the first two elements would be the usual natural numbers, but the third element would be a proof that the greatest common denominator of the natural numbers is 1. In practice, syntactic equality between proofs is even more problematic to use, but it turned out that one could forget the first two elements, because the proof structure contains enough information to reconstruct them. Hence the idea to represent the rational numbers simply by the trace of the computation of the greatest common divisor.

Still, we could have chosen to use the trace of the computation of the greatest common divisor using the regular algorithm based on division. We will come back to this later. The motivation to take the simplified algorithm was to have all computations easily performed by structural recursion over Peano representations of natural numbers. This will be important when we describe the way Q^+ is implemented in a type-theory based theorem prover like Coq.

Because of its simplicity, this representation is not particularly efficient, when compared to fraction representations, it is still strictly more efficient than a representation where both numerator and denominator are represented as peano numbers, where $\frac{p}{q}$ is represented using $p + q$ symbols, while our representation takes less than $(p/q) + q$ symbols (no gain for natural numbers, obviously); it is probably not as efficient as a fraction representation where both numerator and denominators are represented as binary numbers, especially for rational numbers with a large integer part (or their inverse), where our representation is as inefficient as peano numbers. For a really efficient representation, continued fractions would probably be the best choice.

3 Order

If we note N' the function over rational numbers defined by:

$$N'(x) = i(N(c(x)))$$

and D' the symmetric function, it is obvious that both N' and D' are strictly monotonic functions over the positive rational numbers. Moreover, we have the two following inequalities, for any two strictly positive rational numbers x_1 and x_2:

$$N'(x_1) > 1 > D'(x_2).$$

Combining these two facts, we get the following equivalence:

$$x_1 > x_2 \Leftrightarrow c(x_1) >_{Q^+} c(x_2)$$

where the order $>_{Q^+}$ is defined by:

- for any w_1 and w_2, $Nw_1 >_{Q^+} 1 >_{Q^+} Dw_2$,
- for any w_1 and w_2, $Nw_1 >_{Q^+} Nw_2 \Leftrightarrow w_1 >_{Q^+} w_2$,
- for any w_1 and w_2, $Dw_1 >_{Q^+} Dw_2 \Leftrightarrow w_1 >_{Q^+} w_2$.

To ease notations, we shall often write $>$ for $>_{Q^+}$.

4 Primitive operations

4.1 Inversion

The symetry between numerator and denominator exhibited in the Q^+ language can be exploited to construct the inversion function. For instance, $NDN1$ is $\frac{5}{3}$, while $DND1$ is $\frac{3}{5}$, and $NN1$ is 3 while $DD1$ is $\frac{1}{3}$. We see there is a pattern.

Intuitively, the proof that p and q are relatively prime and the proof that q and p are relatively prime are the same, except that all decisions are symmetric. Thus, constructing the Q^+ representation of the inverse of the number represented by an arbitrary word in Q^+ is simply done with the following inv function:

- $inv(1) = 1$,
- $inv(Nx) = D(inv(x))$,
- $inv(Dx) = N(inv(x))$.

It is simple to prove by induction on the number of N and D that if $i(w)$ returns the fraction p/q, then $i(inv(w))$ returns the fraction q/p.

4.2 Other basic operations

We do not attempt to provide efficient implementations of addition or multiplication. An interesting, probably efficient, algorithm is presented in [8], but we only present naive implementations that use fractions as intermediary data.

We interprete words w and w' in Q^+ as reduced fractions $\frac{p}{q}$ and $\frac{p'}{q'}$, computing the result fraction in the usual manner, and then re-constructing the result word in Q^+ with the c function.

4.2.1 Addition

For addition, the result fraction is

$$\frac{(pq' + p'q)}{qq'.}$$

It is interesting to prove the following theorem:

$$1 + w = Nw$$

Here is a simple proof. If w represents the fraction $\frac{p}{q}$, then the fraction constructed for $1 + w$ is

$$\frac{(1 \times q + p \times 1)}{1 \times q} = \frac{p+q}{q}.$$

When constructing the representation for this number the comparison between numerator and denominator yields that the numerator is bigger and the resulting word is Nw' where w' is the representation of

$$\frac{(p+q-q)}{q} = \frac{p}{q},$$

that is $w' = w$.

This theorem can be used to make addition faster, by adding the integer part of rational numbers before resorting to the more complicated general solution. When adding an integer to a rational number the general solution can simply be avoided.

It is also easy to prove that addition is commutative and associative, simply because addition and multiplication are associative on natural numbers.

4.2.2 Multiplication

For multiplication, the result fraction is pp'/qq'. There is no way to be sure that this fraction is already reduced, so we really have to go the interpretation-reconstruction process. However, we can verify that 1 really acts as a neutral element for multiplication. The result fraction obtained when multiplying with 1 is

$$\frac{1 \times p'}{1 \times q'}$$

and the neutral property is simply inherited from the neutral property of 1 for the multiplication of natural numbers.

Here again, it may be interesting to compute a default approximation of the product of two rational numbers by first computing the product of their integer parts. This will give no gain when multiplying a natural number with an arbitrary rational number, because one still need to resort to the general solution to compute the multiplication of the integer with the fractional part of the other number.

Having both addition and multiplication, it is interesting to verify that we have distributivity. This is done in our formal proof, but we do not describe it in details here.

4.2.3 Subtraction

Subtracting w' from w is meaningful only when w represents a larger rational number than w', this can be checked easily thanks to the comparison procedure outlined in section 3. The result fraction is

$$\frac{(pq' - p'q)}{qq'.}$$

There is a question whether $pq' - p'q$ really is a strictly positive natural number, but this is a simple consequence of the fact that $p/q > p'/q'$ (by multiplying both sides of the inequality by qq').

Zero is not element of the set of strictly positive rational numbers, so it is not easy to express that subtraction really is the opposite of addition, still we can express it with a theorem that has the following statement:

$$\forall w, w' \in Q^+. \quad (w + w') - w' = w$$

To prove this theorem, we need to show that

$$\frac{(p''q' - p'q'')}{q''q'}$$

is the same as p/q, where p''/q'' is the reduced fraction of

$$\frac{(pq' + p'q)}{qq'},$$

that is, there exists a natural number a such that $pq' + p'q = ap''$ and $qq' = aq''$ thus the first fraction can also be written

$$\frac{a \times (p''q' - p'q'')}{aq''q'} = \frac{((pq' + p'q)q' - p'qq')}{qq'q'} = \frac{pq'^2}{qq'^2} = \frac{p}{q}.$$

5 Encoding the whole rational field

To encode the whole rational field, we need to add 0 and negative numbers. This is easily done by constructing a disjoint sum. In Coq it will be written as follows:

```
Inductive Q : Set :=
  Qpos : Qplus -> Q
| Qzero : Q
| Qneg : Qplus -> Q.
```

Generalizing inversion on this field is trivial, simply lifting the operation defined in section 4.1. Generalizing addition, multiplication, and subtraction is easily done from the basic operations for strictly positive rationals, taking care of signs almost independently of the computation of significative numbers.

For instance, when adding two positive numbers, the result is positive, and the absolute values must be added. On the other hand, when adding a positive and the negative value, then the absolute values (in Q^+) must be compared. If the absoluve value of the positive argument is larger, then the result will be positive, but the resulting absolute value is going to be the subtraction of the two values.

Of course, a null value may occur among the operands, but this is easily taken care of by expressing the properties of 0 as neutral element for addition and as absorbing element for multiplication. Taking the opposite of a rational number is a simple syntactic operation: just change the sign, when there is one.

Comparison can also be extended to the full field. Here also, it is only a matter of extending comparison for positive numbers given in section 3 with a rule of signs: negative numbers are smaller than 0, which is smaller than positive numbers. For numbers of the same sign, we just compare their absolute values, not forgetting to invert the results when the compared numbers are negative.

6 Implementing the functions in Coq

The calculus of inductive constructions, as implemented in the Coq system, provides good support for describing and proving properties of structural recursive functions. Functions of this kind are easily recognized according to a syntactic pattern when using pattern-matching constructs: recursive calls are only permitted on direct subterms of a special argument and these subterms appear as variables in a pattern.

The function c that we describe above to construct an element of Q^+ from a pair of non-zero natural numbers is not structural recursive. There are several techniques to handle functions that are not structural recursive, several of them include constructing functions that take proofs of termination as arguments [1,2]. Here we have chosen a simpler path: we add a extra artificial argument to the function, whose purpose is only to count the number of allowable recursive calls. The function we obtain has the following form:

```
Fixpoint Qplus_c [p, q, n : nat] : Qplus :=
 Cases n of
   O => One
  | (S n') =>
      Cases (minus p q) of
        O =>
          Cases (minus q p) of
            O => One
          | v => (D (Qplus_c p v n')) end
       | v => (N (Qplus_c v q n'))
      end
 end.
```

In this function we are computing the representation of p/q and the result is correct only for suitable values of n. A simple analysis of the code shows that it suffices that n is larger than the maximum of p and q.

The use of an artificial argument to the `Qplus_c` function makes that it is also defined when its semantics makes no sense. For instance, if numerator or denominator is zero, the value returned is `(D (D ...One))` or `(N (N ...One))`. When stating any proof about this function we need to check that we are talking only about meaningful uses.

For instance, we proved that the function `Qplus_c` is correct, as stated by

the following theorem (there are more theorems about addition than about maximum in Coq and we chose to use this as a lower bound of acceptable values of n). Here the fact that `p` and `q` are non-zero is ensured by the fact that they are computed by `Qplus_i`.

```
Theorem construct_correct:
 ∀ w : Qplus, p, q, n : nat.
 (Qplus_i w) = (p, q) → (le (plus p q) n) →
 (Qplus_c p q n) = w.
```

We can also define a `Qplus_c'` function that takes only the numerator and denominator of the fraction and adds them before calling `Qplus_c`. Thus, the fraction n/m will be represented by the term `(Qplus_c' (`n`)(`m`))`.

Defining addition and multiplication by converting terms from Q^+ to pairs of natural numbers is then an easy example of structural-recursive programming:

```
Definition Qplus_add : Qplus -> Qplus ->  Qplus :=
   [w, w' : Qplus]
    (Cases (Qplus_i w) of
     (p,q) =>
       (Cases (Qplus_i w') of
        (p',q') =>
          (Qplus_c
           (plus (mult p q') (mult p' q)) (mult q q')
           (plus (plus (mult p q') (mult p' q)) (mult q q')))
         end)
     end).
```

Thanks to the use of pure structural recursive programming, the reductions rules of the calculus of inductive constuctions can always work on closed term, and we can test our addition function on pairs of fractions.

```
Definition Qplus_c' [n,m:nat] := (Qplus_c n m (plus n m)).

Eval Compute in
  (Qplus_i (Qplus_add (Qplus_c' (5)(7))(Qplus_c' (1)(3)))).
   = ((22),(21)) : nat*nat
```

We have followed the same principles for all functions on `Qplus` and on `Q`. all functions are programmed in structural recursive way, sometimes with an extra argument to bound the recursive calls, and the functions have been made total by giving an arbitrary value when they should have been undefined.

7 Constructing the rational number field

In theorem provers, the tradition is to use a definitional approach, where new concepts are defined from old ones. In our case, we want to consider that the

natural numbers are given with the basic operations, addition, multiplication, subtraction, and comparison, the sets Q^+ and Q are defined as above, translation from words in Q^+ to pairs of natural numbers, and the definition of basic operations are also given. From this, we want to show that Q satisfies the properties of an ordered archimedian field. Thus, we have to redo a whole bunch of proofs that were simply solved in the previous section by refereeing to the set of mathematical rational numbers, which we should not be using now.

In fact, we only have to prove the 13 axioms that define an ordered archimedian field [4] (there are 14 axioms for a complete ordered archimedian field, but obviously we cannot expect completeness).

Of course, the fact that some functions are normally not total re-appears in the properties with have proved. For instance, the following property expresses that inversion is the symetric operation to multiplication, but the zero case is clearly avoided in the statement, even though our inversion function does have a value for zero.

```
Q_inv_def: ∀x : Q. x ≠ Zero →
             (Q_mult x (Q_inv x)) = (Qpos One).
```

8 Continued fractions

Readers with enough mathematical background may already have recognized simple continued fractions in the Q^+ language. When considering long sequences of the same symbol, it is possible to use natural numbers, as summarized by the following equalities:

$$N'^k x = k + x \qquad D'^k x = 1/(k + 1/x)$$

Combining these equations to analyze large words, we obtain that the word

$$N^{a_0} D^{a_1} ... N^{a_n} D^{a_{n+1}} 1$$

actually represents the number

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\vdots \quad + \cfrac{1}{a_n + \cfrac{1}{a_{n+1} + 1}}}}$$

This is known as a finite simple continued fraction. In this sense we rediscover a fact that is already known: when looking for canonical representation for rational numbers, continued function can be used, as long as all the a_k's are strictly positive, except for the first one. This representation is actually used in algorithms proposed by Kornerup and Matula in [7] where the representation is also enhanced by looking at the step taken when computing the greatest common divisor, but this time when numbers are represented in binary format. The algorithms proposed in Kornerup and Matula's work are

"on-line" algorithms, which in a functional programing approach we might also want to consider as "lazy" computing algorithms.

If this construction is preferred to the other one for use in a theorem prover or in a functional programming language with recursive types, it is sensible to start by representing the rational numbers that are strictly greater than 1. In this manner, we avoid taking care of the special case for a_0 which does not need to be strictly positive. If we only represent numbers that are greater than 1, then a_0 also needs to be positive.

A rational number greater than 1 is necessarily an integer greater or equal to 2, or an integer greater or equal to 1 plus the inverse of a natural number, or a number of the form

$$a + \frac{1}{(b + \frac{1}{x})},$$

where a and b are strictly positive natural numbers, and x is a rational number greater than 1. This can be described with the following new inductive definition.

```
Inductive Qplus' : Set :=
 Nat : positive -> Qplus'
| NatInv : positive -> positive -> Qplus'
| R : positive -> positive -> Qplus'.
```

Having this subset of the field of rational numbers, it is a simple matter to add 1 and inverses of rationals greater than 1 to get all strictly positive rational numbers and to add 0 and opposites of positive rational numbers to get all rational numbers, this is done using the following inductive definition:

```
Inductive Q' : Set :=
  G1 : Qplus' -> Q'
| One' : Q'
| IG1 : Qplus' -> Q'
| Zero' : Q'
| OIG1 : Qplus' -> Q'
| OOne' :Q'
| OG1: Qplus' -> Q'.
```

In this description, `G1` is used for numbers larger than 1, `One'` is used for 1, `IG1` (the `I` stands for *inverse*) is used for numbers between 0 and 1, actually `(IG x)` represents the inverse of `(G1 x)`, `Zero'` stands for 0, `OIG1` is used for numbers between -1 and 0, actually `(OIG1 x)` represents the opposite of `(IG1 x)`, `OOne'` is used for -1, and `OG1` is used for numbers lesser than -1, actually `(OG1 x)` represents the opposite of `(G1 x)`.

Basic operations can be defined on this structure by following the guidelines given both by the interpretation of terms in `Qplus'` as finite continued fractions or as compact encodings of terms in Q^+, but this work has not been done yet.

9 Inductive proofs on rational numbers

Having an inductive structure to describe rational numbers, it can be used to guide proofs about these numbers, in the same manner as the peano structure of natural numbers guides proofs by providing the usual induction principle on these numbers. In this section, we show how this leads us into a new way of proving things, that may sometimes turn out to be more efficient.

9.1 A proof that the square root of 2 is not rational

The intuition behind this proof is that the square root of two actually is represented by the following infinite continued fraction:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{\vdots}}}$$

In other terms, if $\sqrt{2}$ were a rational number, then it would be represented by the the term:

$$\sqrt{2} = NDDN\sqrt{2}$$

This is impossible, because it leads to an infinite element in an inductive type.

Let us suppose that $\sqrt{2}$ is rational, and let us show that

$$\sqrt{2} = NDDN\sqrt{2}.$$

The square of 1 is 1 and $1 < 2$, since the square function is increasing, then $\sqrt{2}$ is necessarily of the form Nx, $N1$ is 2 and $2^2 > 2$ then $\sqrt{2}$ is necessarily of the form NDx', $ND1$ is $3/2$ and $(3/2)^2 = 9/4 > 2$, then $\sqrt{2}$ is necessarily of the form $NDDx''$, $NDD1$ is $4/3$ and $(4/3)^2 < 2$ then $\sqrt{2}$ is necessarily of the form $NDDNy$, where y represents a strictly positive rational number which we also denote y. By the definition of interpretation of N and D, we have:

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{1 + y}}.$$

Using a few algebraic transformations that are all licit because y is strictly positive, we get the following equality:

$$\sqrt{2} = \frac{3y + 4}{2y + 3}$$

After squaring both sides of the equality, multiplying by $(2y + 3)^2$ (a strictly positive number), and simplifying, we get:

$$2 = y^2$$

This proves that $y = \sqrt{2}$ and leads to the contradiction we are looking for.

The same form of reasoning applies to prove that $\sqrt{3}$ is not rational, this time using the fact that if $\sqrt{3}$ were rational, it would have to verify the fol-

lowing equality:

$$\sqrt{3} = NDN\sqrt{3}.$$

It is even possible to re-do this proof by only following the *structure* suggested by the Q^+ language, but without explicitely using the N and D constructs. Here it is:

We prove by induction on n, that there is no pair of non zero numbers p and q such that $p + q \leq n$ and $p^2 = 2q^2$. Let us take an arbirtrary n and, as induction hypothesis, let us suppose that for all $m < n$, there is no pair of non zero numbers p' and q' such that $p' + q' = m$ and $p'^2 = 2q'^2$. Let us suppose we have two non zero numbers p and q such that $p + q = n$ and $p^2 = 2q^2$. Let us prove that there is a contradiction.

Since $1^2 < 2 < 2^2$, we know that $q < p < 2q$, let $p' = p - q$, we know that $p' < q$ and since q is non zero, we have $p' < p$. We also have

(1) $(p' + q)^2 = 2q^2$.

If $q \leq 2p'$ then there exists an $x > 0$ such that $2p' = q + x$, the above equality can be transformed into:

$$(2p' + 2q)^2 = 8q^2.$$

This gives

$$9q^2 + 6qx + x^2 = 8q^2$$

and after simplification:

$$q^2 + qx + x^2 = 0.$$

This is not possible if $q > 0$.

On the other hand, if $q > 3p'$ then there exists an x such that $q = 3p' + x$ and we can simplify the equality 1 into the following one:

$$16p'^2 + 8p'x + x^2 = 18p'^2 + 12p'x + x^2$$

and after simplification

$$0 = 2p'^2 + 12p'x.$$

Again, this is not possible if $p > 0$. Thus, we know that $2p' < q < 3p'$, let q' be the non zero number such that $q = 2p' + q'$ and $q' < p$. We have

(2) $(3p' + q')^2 = 2 \times (2p' + q')^2)$

Now let p'' be the strictly positive number $p'' = p' - q'$ With this number the equation 2 becomes:

$$(3p'' + 4q')^2 = 2 \times (2p'' + 3q')^2$$

and after simplification:

$$p''^2 = 2q'^2$$

By construction $p' < q' < p$ and $q' < q$, thus $p'' + q' < n$ and by using the induction hypothesis, we can deduce that there is a contradiction. The proof is over.

If we analyze the structure of this proof, it follows directly the structure given by Q^+ and the previous proof:

(i) The decision to perform proof by induction on the sum of the numerator and denominator is guided by the fact that the function c terminates because the sum of the numerator and denominators decreases,

(ii) the introduction of the number p' corresponds to the application of N that is the first element of the segment $NDDN$ that is repeated in the continued fraction expansion,

(iii) the introduction of the number q' corresponds to the two applications of the D that occur in $NDDN$,

(iv) the introduction of the number p'' corresponds to the last N occurring in $NDDN$,

(v) the concluding use of the induction hypothesis corresponds to the remark that the continued fraction for $\sqrt{2}$ is infinite.

This proof may look a little more complicated, but we have gone to all these tedious steps to show that we have never used any other operations that multiplication, addition, and subtractions, and comparisons of natural numbers. This is important to show that this proof that square root of 2 is not rational is very simple in the amount of mathematical tools it uses. This is an important point when considered mechanized proofs, where the full extent of mathematical knowledge is rarely available. The usual proof, as proposed initially by Euclid, goes through the argument that if $p^2 = 2q^2$, then p^2 is even, then p is even, then q is even, and the fraction is not reduced. This proof usually requires that one define the concept of even numbers and then show that if the square of a number is even, this number is also even. Euclid's proof carries over to $\sqrt{3}$ only at the expense of defining the property to be a multiple of 3, and with a little efforts it also carries over to a proof that the cubic root of 2 or 3 is not rational. Proofs relying on the Q^+ structure carry easily to the proof that $\sqrt{3}$ is not rational, but they are not adapted for cubic roots.

10 Related work

Continued fraction have been used in mathematics for a long time. John Wallis, a professor at Oxford in the 17th century actually introduced the name and described them. Euler showed that simple continued fraction were in 1-1 correspondance with rational numbers. Lagrange showed that roots of quadratic equations were either rational numbers or periodic continued fractions. More recently, a french clock-maker, Achille Brocot, and the german mathematician Moritz Abraham Stern devised a technique to represent rational numbers that turns out to represent the same inductive structure as the rational numbers in Q^+ [9,3] (for an introductory presentation see [6]). Inline algorithms for

the basic operations on continued fractions have been studied by Vuillemin [10] and similar algorithms have been devised by Niqui [8] for the structures described by Stern and Brocot, which are the same as ours. Milad Niqui and the author of these lines plan to collaborate to construct the proofs that the algorithms described by Niqui compute the same values as the algorithms described naively in this work.

11 Conclusion

All proofs described in this paper have been performed using the Coq system and are available from the author on demand. These proofs include a proof that `Q` has a field structure and a new presentation of the proof that $\sqrt{2}$ is not rational.

We have given a quotient free representation of rational numbers. There exists several other such representations, and actually continued fractions, with which our representation is related also provide such a quotient free representation. Another example is where positive rational numbers are represented by finite lists of relative numbers, where the k^{th} element describes the power of the k^{th} prime number. Such lists may be of practical use if multiplication plays a more important role than comparison. However, the mathematical background needed to ascertain the validity of this representation is much more important than for our notation, as it relies on the fundamental theorem of arithmetics (unicity of decomposition of any natural number as a product of powers of prime numbers).

The beauty of our representation is in its simplicity. It is remarkable that the positive rational numbers, such a dense set, can be obtained from the natural numbers by virtually adding only one inductive constructor. The constructor N corresponds to the successor function of peano arithmetics, the constructor we add is simply the D constructor, which is simply presented as a symmetric to the N constructor.

Practical applications to this representation seem hard to find, mainly because the basic operations are so clumsy. We have shown that the inductive structure it gives to the set of rational numbers is well adapted to certain kinds of proofs. For instance, proofs that π is not rational may possibly be made easier thanks to this structure, since some of the known proofs rely on the fact that the rational numbers whose sum of numerator and denominator is bounded never get close enough to π. Also this presentation of rational numbers can be used as an intermediary step to prove the correctness of efficient algorithms for exact computation on rational numbers and this will be used in future collaboration with M. Niqui.

As a last remark, I would like to point out that the whole elaboration of this representation comes directly from a reflection on proof as proof objects in type-theory based theorem provers. Although all the statements given in this paper can easily be expressed in a wide variety of theorem provers,

the guideline for elaborating the data-structure is provided by a study of the structure of proofs that two numbers are relatively prime, in other words, a study of Euclid's algorithm to compute the greatest common divisor of two numbers.

References

[1] Antonia Balaa and Yves Bertot. Fix-point equations for well-founded recursion in type theory. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2000.

[2] Ana Bove and Venanzio Capretta. Nested general recursion and partiality in type theory. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135. Springer-Verlag, September 2001.

[3] Achille Brocot. Calcul des rouages par approximation, nouvelle méthode. *Revue chronométrique. Journal des horlogers, scientifique et pratique*, 3:186–194, 1861.

[4] David Delahaye and Micaela Mayero. Field: une procédure de décision pour les nombres réels en coq. In *Proceedings of JFLA'2001*. INRIA, 2001.

[5] Bruno Barras et al. *The Coq Proof Assistant Reference Manual, Version 7.3*. INRIA, `http://coq.inria.fr/doc/main.html`, oct 2002.

[6] Brian Hayes. On the teeth of wheels. *American Scientist*, 88(4):296–300, july-august 2000.

[7] Peter Kornerup and David Matula. LCF: A lexicographic binary representation of the rationals. *Journal of Universal Computer Science*, 1(7):484–503, july 1995.

[8] Milad Niqui. Exact Arithmetic on Stern-Brocot Tree. *submitted*, jan 2003.

[9] Moritz Abraham Stern. Ueber eine zahlentheoretische Funktion. *Journal für die Reine und angewandte Mathematik*, 55:193–220, 1858.

[10] Jean E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, aug 1990.