A Co-inductive Approach to Real Numbers

Alberto Ciaffaglione and Pietro Di Gianantonio

Dipartimento di Matematica e Informatica, Università di Udine
via delle Scienze, 206 - 33100 Udine (Italy)
e-mail: {ciaffagl,pietro}@dimi.uniud.it

Abstract. We define constructive real numbers in the logical framework Coq using streams, i.e. infinite sequences of digits. Co-inductive types and co-inductive proofs permit to work naturally on this representation. We prove our representation satisfies a set of basic properties which we propose as a set of axioms for constructive real numbers.

1 Introduction

The aim of this work is to experiment a co-inductive representation of Real Numbers in the context of Logical Frameworks, i.e. interactive proof assistants based on Type Theory. This is an effort towards computer assisted formal reasoning on real numbers and it should provide a workbench for specifying and certifying lazy exact algorithms on them.

Several computer aided formalizations of analysis exist in literature. Chirimar and Howe [CH92] have developed analysis following the constructive approach to mathematics of Bishop [Bis67]: they represent real numbers by Cauchy sequences and prove the completeness theorem using the Nuprl system [Con86]. Jones [Jon91] has given some theorems of constructive analysis in Lego [Pol94]. More recently, Harrison [Har96] have presented a significant part of the classical analysis in the context of the Isabelle-HOL system [GM93], introducing the reals by means of a technique closely related to the Cantor's classical method. Cederquist, Coquand and Negri [Ced97] have studied constructive analysis using a point-free topology approach. They prove constructively the Hahn-Banach theorem in an intensional Martin-Löf's type theory and certify such a proof by Half [Mag95].

The main difference between our approach and the previous ones consists both in the representation chosen for reals and in the logical framework we use.

We represent real numbers by potentially infinite sequences —i.e. streams— of 3 digits: $\{0, 1, -1\}$. This representation is close to those adopted in the field of exact computation. In recent years there has been growing interest in exact real number computation [Wei96, PEE97, Sim98]: this approach allows to produce arbitrary precision results from data —thus avoiding round off errors typical of limited precision practice— without having to carry out any independent error analysis. Since exact computation is among other motivated by software reliability reasons, it is important to certify the correctness of the algorithms performing exact computation on real numbers.

We decide to use the logical framework Coq [INR00] to carry out our research. This choice is motivated by the fact that Coq allows a "natural" encoding of streams as infinite objects inhabiting co-inductive types; correspondly, it provides useful techniques for dealing with co-induction proof theory, such as the tactic `Cofix`. Our work is carried out in a constructive logic setting; up to now, it has not been necessary to use the Axiom of Excluded Middle, even if it could probably simplify some of the formal proofs we have developed.

The main line of our research is the following: we start representing real numbers by streams; then we define (co-)inductively on streams the notions of strict order (Less), addition (Add) and multiplication (Mult). We prove formally in Coq that these definitions satisfy a set of properties: we claim such properties can be taken as a set of axioms characterizing the constructive real numbers. We substantiate this claim by proving that most of the basic properties of constructive real numbers can be deduced from our axioms.

The paper is structured as follows. Sections 2 and 3 respectively introduce and justify our representation of real numbers, which is then enriched by means of other notions in section 4. A central importance in the article has the discussion about the setting of a (minimal) set of axioms characterizing constructive real numbers. This topic is undertaken in section 5 and continues after. The last sections present the formalization and the study of the theory in Coq.

Acknowledgement. We wish to thank the anonymous referees for the useful suggestions and the careful reading of the paper. We thank also Furio Honsell and Herman Geuvers for interesting discussions.

2 Real numbers

Many classical constructions of real numbers exist in literature: Cauchy sequences of rational numbers, Cauchy sequences of rational p-adic numbers, Dedekind cuts in the field of rationals, infinite decimal expansions, and so on. All such methods turn out to be equivalent, in the sense that they give rise to isomorphic structures. Many of these constructions can also be formulated in a *constructive* approach to mathematics, but in this case we don't always obtain isomorphic structures —e.g. constructive reals via Cauchy sequences differ from the constructive reals through Dedekind's cuts [TvD88].

In this work we will construct real numbers using infinite sequences of digits, i.e. "infinite expansions". It is a well known phenomenon that standard positional notations make arithmetic operations on real numbers *not computable* [Bro24]. A classical solution to this problem is to adopt *redundant* representations: in a redundant representation a real number enjoys more than one representation. We decide to use here a signed-digit notation: we add the negative digit -1 to the binary digits 0 and 1 of the standard binary notation, maintaining 2 as the value for the base.

We are going now to introduce the basic ingredients of our work. In order to explain and motivate our definitions we will refer to a field of the real numbers

$\mathbb{R}$: this use of an already given field is not essential in our construction, but helps to understand the intended meaning of the definitions. Moreover, in section 3 we will use the same $\mathbb{R}$ to give an *external* proof that the given definitions are correct. In this proof we will make use of simple arithmetic properties that are valid also for the constructive reals. It follows that no matter which field is chosen for $\mathbb{R}$: it could be either the classical or the constructive one [TvD88]. A notational issue: from now on we will use ":" as a justaposition operator for digits.

Definition 1. ***(Ternary streams)***
Let str be the set of the infinite sequences built of ternary digits:

$$str = \{a_1 : a_2 : a_3 : \ldots \mid \forall i \in \mathbf{N}^+.\ a_i \in \{0, 1, -1\}\ \}$$

The elements of str represent the real numbers via the interpretation function $[\![\ ]\!]_{str} : str \to \mathbb{R}$*, defined as:*

$$[\![a_1 : a_2 : a_3 : \ldots\ ]\!]_{str} = \sum_{i \in \mathbf{N}^+} a_i \cdot 2^{-i}$$

We will use $a, b, c, \ldots$ as metavariables ranging on ternary digits and $x, y, z, \ldots$ as metavariables for streams.
Using ternary streams we can represent any real number belonging to the closed interval $[-1, 1]$; it is not difficult to see that any element of the open interval $(-1, 1)$ is represented by an infinite number of different streams. In order to dispose of arbitrarily large reals it is necessary to use a mantissa-exponent notation: we encode a real number by a pair $\langle natural, stream \rangle$, which we call "R-pair".

Definition 2. ***(R-pairs)***
Let R be the set $(\mathbb{N} \times str)$.
The elements of R represent the real numbers via the interpretation function $[\![\ ]\!]_R : R \to \mathbb{R}$*, defined as:*

$$[\![\langle n, x \rangle]\!]_R = 2^n \cdot [\![x]\!]_{str}$$

We will use $r, s, t, \ldots$ as metavariables ranging on R.

In order to complete our construction of real numbers it is necessary to provide R with an *order* relation and a *field* structure: actually, the real line is completely determined by the binary "strict order" relation ($<$) and the functions of "addition" and "multiplication".

We have considered several different possible definitions for order, addition and multiplication in our research: at the end, we have chosen to describe not only the order, but also addition and multiplication using predicates rather than functions. This choice is due to the fact that the predicates are simpler to define. An intuitive motivation for this is that functions are requested to be "productive" —i.e. they must supply a method to effectively calculate the result, given the input. On the contrary, a predicate just specifies the constrains that the output has to satisfy w.r.t. input: it follows that it is a simpler task to prove the formal

properties of the predicates. Anyway, we will introduce the functions too and we will prove they are coherent with respect to the predicates. One can interpret this fact saying the implementation (algorithms, functions) satisfy the specification (predicates).

We have found that the lenght and the complexity of the formal proofs is greatly affected by the pattern of the definitions: quite frequently the proofs are obtained by structural (co-)induction on the derivations, so the number of the cases to consider increases together with the number of constructors of the predicate involved. In order to simplify the proofs, we have formalized the (co-)inductive predicates using at most two constructors, thus reducing the cases to analyze.

We claim that these considerations about complexity of proofs have general meaning and are not specific to the particular proof assistant we have used: in fact, the proofs developed in Coq are just a completely detailed version of the proofs that we would write with paper and pencil.

Let now resume our goal: to define order, addition and multiplication.
The strict order is defined by *induction*: this is possible because, given two R-pairs, we can semi-decide whether the first is smaller than the second just by examinating a finite number of digits. The binary strict order relation on streams is defined in terms of an auxiliary ternary relation $less\_aux \subseteq (str \times str \times Z)$, whose intended meaning is:

$$less\_aux(x, y, i) \quad \Longleftrightarrow \quad ([\![x]\!]_{str} < [\![y]\!]_{str} + i)$$

This auxiliary predicate permits to define more simply the main predicate of order on streams. In particular, using the integer parameter i we are able to do simpler proofs, because the extensive case analysis on the ternary digits is replaced by proofs over integers.

Definition 3. ***(Order on streams)***
The predicate $less\_aux \subseteq (str \times str \times Z)$ *is defined inductively by the two rules:*

$$\text{LESS-BASE}\ \frac{}{less\_aux(x, y, big)} \qquad where\ big = 32$$

$$\text{LESS-IND}\ \frac{less\_aux(x, y, i) \quad (i + a) \leq\ (2j + b)}{less\_aux(a : x,\ b : y,\ j)}$$

The predicate $less_{str} \subseteq (str \times str)$ *is defined as:*

$$less_{str}(x, y) \quad \equiv_{def} \quad less\_aux(x, y, 0)$$

The above definition requires some discussion. Referring to the intended meaning it is simple to see that the base rule is valid for any value of the parameter *big* greater than 2: a natural choice for *big* would be the integer 3, but any greater value gives rise to an equivalent definition. We have found that greater values

simplify several proofs built by structural induction on the judgments of the kind $less\_aux(x, y, i)$. We have arbitrarily fixed big to 32.

It is immediate to see the base rule is correct.
The induction rule can be informally justified by a simple calculation:

$$\begin{array}{ll} [\![a : x]\!]_{str} \ < \ ([\![b : y]\!]_{str} + j) & \Leftrightarrow \\ a/2 + ([\![x]\!]_{str}/2) \ < \ b/2 + ([\![y]\!]_{str}/2) + j & \Leftrightarrow \\ [\![x]\!]_{str} \ < \ [\![y]\!]_{str} + 2j + b - a & \Leftrightarrow \\ \exists i.\ ([\![x]\!]_{str} < \ [\![y]\!]_{str} + i) \ \wedge \ (i + a \ \leq \ 2j + b) & \end{array}$$

The strict order relation can now be easily extended to R-pairs.

Definition 4. ***(Order on R-pairs)***
The predicate $Less \ \subseteq \ (R \times R)$ *is defined as:*

$$Less(\langle m, x\rangle, \langle n, y\rangle) \quad \equiv_{def} \quad less_{str}(\underbrace{0 : \ldots : 0}_{n} : x, \ \underbrace{0 : \ldots : 0}_{m} : y)$$

On the contrary with respect to the order, the arithmetic predicates are defined by *co-induction*, because the process of adding or multiplying two real numbers is considered non terminating. Co-inductive predicates originate judgements that can be proved by "infinitary" proofs, which are built using infinitely many times the corresponding introduction rules [Coq93, Gim94].

The predicates of addition and multiplication have the following pattern:

$$predicate(operand_1, \ operand_2, \ result).$$

We start from addition: as we have done for the order relation, we first define an auxiliary predicate on streams. The relation $add\_aux \subseteq (str \times str \times str \times Z)$ has intended meaning:

$$add\_aux(x, y, z, i) \quad \Longleftrightarrow \quad ([\![x]\!]_{str} + [\![y]\!]_{str}) = ([\![z]\!]_{str} + i).$$

Definition 5. ***(Addition)***
The predicate $add\_aux \subseteq (str \times str \times str \times Z)$ *is defined by the co-inductive rule:*

$$\text{ADD-COIND} \ \frac{add\_aux(x, y, z, (2i + c - a - b)) \quad (-big \leq i \leq big)}{add\_aux(a : x, \ b : y, \ c : z, \ i)}$$

The addition predicate on streams $add_{str} \subseteq (str \times str \times str)$ *is defined as:*

$$add_{str}(x, y, z) \quad \equiv_{def} \quad add\_aux(x, y, z, 0)$$

The addition predicate on R-pairs $Add \subseteq (R \times R \times R)$ *is defined as:*

$$Add(\langle m, x\rangle, \langle n, y\rangle, \langle p, z\rangle) \quad \equiv_{def} \quad add_{str}(\underbrace{0 \ldots 0}_{n+p} : x, \ \underbrace{0 : \ldots : 0}_{m+p} : y, \ \underbrace{0 : \ldots : 0}_{m+n} : z)$$

The side-condition ($-big \leq i \leq big$) bas been introduced in order the relation *add_aux* is not total —otherwise one can easily prove the judgement $add\_aux(x, y, z, i)$ for any tuple x, y, z, i. Again, values of *big* greater than 3 give rise to equivalent conditions, but lead to simpler proofs.

The co-inductive rule can be informally justified by the calculation:

$$\begin{array}{ll} (\llbracket a : x \rrbracket_{str} + \llbracket b : y \rrbracket_{str}) & = \ (\llbracket c : z \rrbracket_{str} + i) \qquad \Leftrightarrow \\ a/2 + (\llbracket x \rrbracket_{str}/2) + b/2 + (\llbracket y \rrbracket_{str}/2) & = \ c/2 + (\llbracket z \rrbracket_{str}/2) + i \Leftrightarrow \\ \llbracket x \rrbracket_{str} + \llbracket y \rrbracket_{str} & = \ \llbracket z \rrbracket_{str} + 2i + c - a - b \end{array}$$

The multiplication predicate is defined in terms of that of addition. This time we need also to define an auxiliary multiplication between digits and streams.

Definition 6. ***(Multiplication)***
The function $times_{d,str} : \{-1, 0, 1\} \times str \to str$ *is defined by co-recursion as:*

$$times_{d,str}(a, (b : x)) \ =_{def} \ (a \cdot b) : (times_{d,str}(a, x))$$

The multiplication predicate on streams $mult_{str} \subseteq (str \times str \times str)$ *is defined by the co-inductive rule:*

$$\text{MULT-COIND} \ \frac{mult_{str}(x, y, w) \quad add_{str}(0 : times_{d,str}(a, y),\ 0 : w,\ z)}{mult_{str}(a : x,\ y,\ z)}$$

The multiplication predicate on R-pairs $Mult \subseteq (R \times R \times R)$ *is defined as:*

$$Mult(\langle m, x\rangle, \langle n, y\rangle, \langle p, z\rangle) \quad \equiv_{def} \quad mult_{str}(\underbrace{0 : \ldots : 0}_{p} : x,\ \underbrace{0 : \ldots : 0}_{p} : y,\ \underbrace{0 : \ldots : 0}_{m+n} : z)$$

The multiplication predicate $mult_{str}$ can be informally justified by calculations similar to those we have detailed for order and addition.

3 Adequacy

In this section we address the question of the *adequacy* of the definitions we have given for order, addition and multiplication. We present here two different approaches to justify our construction: the first argument is internal-axiomatic, the second external-semantic.

Using the first approach we will going to show the definitions of order, addition and multiplication satisfy all the standard properties valid for the real numbers. This proof is carried out in Coq. A limitation of this approach lies in the fact that so far there exists no standard axiomatization for the constructive reals. Another disadvantage relies on the heaviness of the proof editing: the formal proof of all the basic properties is actually a long and tiring job. In our attempt we have almost accomplished this internal proof of adequacy, which will be presented in section 7.

The semantic argument works as follows. In the construction of the previous section we have specified the intended meaning of the sets and the predicates

referring to an external model for the real numbers. It follows that we can justify our predicates by proving that their specification is sound and complete with respect to this external model. This proof makes use of some basic fundamental properties which also hold for the constructive reals. It has not been formalized in Coq, but we conjecture that it could be possible, using an already existing formalization of the real numbers —for example the classical axiomatization provided by the library `Reals`.

This section is just devoted to the proof of the external adequacy. In the proofs below we don't need to specify whether the external model $\mathbb{R}$ is the classical or the constructive one: this omission is motivated by the fact that the order relation is the same for both models.

Proposition 1. ***(Adequacy of order)***
The rules for the order predicate are sound and complete. That is, for any couple of R-pairs r, s: $Less(r, s)$ can be derived if and only if $([\![r]\!]_R < [\![s]\!]_R)$.

Proof. It suffices to prove that the rules for the auxiliary predicate *less_aux* are sound and complete. The proposition then follows easily.

It is quite simple to prove that the rules are sound —i.e. if $less\_aux(x, y, i)$ is derived, then $([\![x]\!]_{str} < [\![y]\!]_{str} + i)$. We need to check the two rules defining the induction:

$$\text{LESS-BASE}\ \frac{}{less\_aux(x, y, big)} \qquad \text{LESS-IND}\ \frac{less\_aux(x, y, i) \quad (i + a) \leq (2j + b)}{less\_aux(a : x,\ b : y,\ j)}$$

The base rule is sound since every stream x represents a real number belonging to the interval $[-1, 1]$: then $[\![x]\!]_{str} < 3 - 1 \leq big - 1 \leq [\![y]\!]_{str} + big$.
The induction rule is sound because from the hypotheses $([\![x]\!]_{str} < [\![y]\!]_{str} + i)$ and $(i + a) \leq (2j + b)$ follows that $[\![a : x]\!]_{str} = a/2 + ([\![x]\!]_{str}/2) < a/2 + ([\![y]\!]_{str} + i)/2 = b/2 + ([\![y]\!]_{str}/2) + (i + a - b)/2 \leq b/2 + ([\![y]\!]_{str}/2) + j = ([\![b : y]\!]_{str} + j)$.

The proof of the completeness is less obvious. We need first to choose a natural number k such that $(big + 6 < 2^k)$ and then prove, by induction on n, that if $([\![x]\!]_{str} + 2^{k-n} < [\![y]\!]_{str} + i)$ then the predicate $less\_aux(x, y, i)$ can be derived. In this hypothesis the completeness property follows immediately, since if $([\![x]\!]_{str} < [\![y]\!]_{str} + i)$ then there exists a natural number n such that $([\![x]\!]_{str} + 2^{-n} < [\![y]\!]_{str} + i)$.
- Base step $(n = 0)$. Let $x = (a : x_0)$ and $y = (b : y_0)$: from the hypothesis follows that $(i > 2^k - 2)$; therefore $(big + a) \leq (2^k - 6 + 1) \leq (2^k + 2^k - 4 - 1) \leq (2i + b)$. By the base rule we derive $less\_aux(x_0, y_0, big)$ and by the induction rule we have $less\_aux(a : x_0,\ b : y_0,\ i)$.
- Inductive step. Let $n = m + 1$, $x = (a : x_0)$ and $y = (b : y_0)$.
We need to provide a derivation for $less\_aux(a : x_0,\ b : y_0,\ i)$ in the hypothesis that $([\![a : x_0]\!]_{str} + 2^{k-(m+1)} < [\![b : y_0]\!]_{str} + i)$. From this hypothesis we have that $([\![x_0]\!]_{str} + 2^{k-m}) = (a + [\![x_0]\!]_{str} + 2^{k-m} - a) = (\ 2([\![a : x_0]\!]_{str} + 2^{k-(m+1)}) - a) < (\ 2([\![b : y_0]\!]_{str} + i) - a) = ([\![y_0]\!]_{str} + 2i + b - a)$. By induction hypothesis it is then possible to derive $less\_aux(x_0,\ y_0,\ 2i + b - a)$ and finally, by the application of the induction hypothesis, we can conclude the proof. □

We now consider the arithmetic predicates: it is interesting to notice that the proof technique adopted for these co-inductive predicates is dual with respect to that used for the inductive predicate of order.

Proposition 2. ***(Adequacy of addition)***
The rules for the addition predicate are sound and complete. That is, for any triple of R-pairs r, s, t: Add(r, s, t) can be derived by an infinite derivation if and only if $([\![r]\!]_R + [\![s]\!]_R = [\![t]\!]_R)$ holds in $\mathbb{R}$.

Proof. In this case too it suffices to prove that the rule for the auxiliary predicate *add_aux* is sound and complete: the correspondent properties for the other predicates follow easily. It is interesting to remark that the proof we are giving for the co-inductive addition is almost dual with respect to the case of the inductive order.

Using co-inductive reasoning, it is quite simple to prove that the rule is complete —i.e. if $([\![x]\!]_{str} + [\![y]\!]_{str} = [\![z]\!]_{str} + i)$, then there exists an *infinitary* proof for $add\_aux(x, y, z, i)$. We prove that under the hypothesis $([\![x]\!]_{str} + [\![y]\!]_{str} = [\![z]\!]_{str} + i)$ there exists a rule which permits to derive $add\_aux(x, y, z, i)$ using the judgement $add\_aux(x', y', z', i')$, whose arguments satisfy $([\![x']\!]_{str} + [\![y']\!]_{str} = [\![z']\!]_{str} + i')$. Therefore, using the co-inductive hypothesis, the predicate $add\_aux(x', y', z', i')$ can be derived by means of an infinitary proof.
Let $x = (a : x_0)$, $y = (b : y_0)$ and $z = (c : z_0)$. From the hypothesis $([\![a : x_0]\!]_{str} + [\![b : y_0]\!]_{str} = [\![c : z_0]\!]_{str} + i)$, by a simple calculation we have both $(-3 \leq i \leq 3)$ and $([\![x_0]\!]_{str} + [\![y_0]\!]_{str} = [\![z_0]\!]_{str} + 2i + c - a - b)$. It is then possible to deduce $add\_aux(a : x_0,\ b : y_0,\ c : z_0, i)$ from $add\_aux(x_0,\ y_0,\ z_0, 2i + c - a - b)$: we can conclude carrying out an infinitary proof using the second equation deduced above.

To proof of the soundness is a little more subtle. We cannot simply prove that the rule for *add_aux* is sound, since by using co-inductively a rule that deduces valid conclusions from valid premises it is still possible to derive judgements which are not valid (for example, consider a rule for equality saying that from $(x = y)$ it follows that $(y = x)$, or the rule *add_aux* itself where the premise $(-big \leq i \leq big)$ has been removed). In order to prove the soundness we need to use other arguments: after having chosen a natural number k such that $(big + 3 \leq 2^k)$, we will prove inductively on n that if $add\_aux(x, y, z, i)$ can be derived, then:

$$|([\![x]\!]_{str} + [\![y]\!]_{str}) - ([\![z]\!]_{str} + i)| < 2^{k-n}$$

The soundness follows from the above inequality, since two real numbers arbitrarily close must be equal. The inductive proof proceeds as follows: first of all let $x = (a : x_0)$, $y = (b : y_0)$ and $z = (c : z_0)$; now, if the predicate $add\_aux(x, y, z, i)$ can be derived, it must be deduced via the co-induction rule. It follows that both the constraint $(-big \leq i \leq big)$ and the predicate $add\_aux(x_0,\ y_0,\ z_0,\ 2i + c - a - b)$ can be derived.
The base step (n=0) follows immediately from the hypothesis.
In the case where $n = m + 1$, we derive by inductive hypothesis the disequation $|([\![x_0]\!]_{str} + [\![y_0]\!]_{str}) - ([\![z_0]\!]_{str} + 2i + c - a - b)| < 2^{k-m}$. Then, by means of a simple arithmetic calculation, we conclude the proof. □

Proposition 3. ***(Adequacy of multiplication)***
The rules for the multiplication predicate are sound and complete. That is, for any triple of R-pairs r, s, t: $Mult(r, s, t)$ can be derived by an infinite derivation if and only if $([\![r]\!]_R \cdot [\![s]\!]_R = [\![t]\!]_R)$ holds in $\mathbb{R}$.

Proof. The proof works similarly to the case of addition. □

4 Equivalence and arithmetic functions

In our framework any real number can be represented in infinitely many ways (an infinite choice of R-pairs denoting the same number is actually available): it is then natural to define an *equivalence* predicate on R-pairs. In constructive analysis it is possible to describe the equivalence relation on real numbers by means of the strict order relation.

Definition 7. ***(Inductive equivalence)***
The inductive equivalence predicate $Equal_{ind} \subseteq (R \times R)$ is defined as:

$$Equal_{ind}(r, s) \quad \equiv_{def} \quad \neg(Less(r, s) \ \vee \ Less(s, r))$$

The validity of the above definition motivates our choice of the strict order as a basic notion for constructive real numbers.

It is interesting to notice that the equivalence relation on R-pairs could also be defined directly via a co-inductive predicate. Following this approach it is convenient to introduce firstly an auxiliary predicate $equal\_aux \subseteq (str \times str \times Z)$, which has intended meaning:

$$equal\_aux(x, y, i) \quad \Longleftrightarrow \quad ([\![x]\!]_{str} = [\![y]\!]_{str} + i)$$

Definition 8. ***(Co-inductive equivalence)***
The predicate $equal\_aux \subseteq (str \times str \times Z)$ is defined by the co-inductive rule:

$$\text{EQUAL-COIND} \ \frac{equal\_aux(x, y, (2i + b - a)) \quad (-big \leq i \leq big)}{equal\_aux(a : x, \ b : y, \ i)}$$

The equivalence predicate on streams $equal_{str} \subseteq (str \times str)$ is defined as:

$$equal_{str}(x, y) \ \equiv_{def} \ equal\_aux(x, y, 0),$$

The co-inductive equivalence predicate on R-pairs $Equal_{coind} \subseteq (R \times R)$ is defined as:

$$Equal_{coind}(\langle m, x\rangle, \langle n, y\rangle) \ \equiv_{def} \ equal_{str}(\underbrace{0 : \ldots : 0}_{n} : x, \ \underbrace{0 : \ldots : 0}_{m} : y)$$

The above definitions are very similar to those given for the addition predicate: if we fix the first argument of the addition to a stream of "zeros", we can actually see the equivalence as a particular case of addition. It follows that the

co-inductive predicate *equal_aux* can be justified by the same arguments used for *add_aux*.

The main property about equivalence on R-pairs is that the inductive and the co-inductive definitions turn out to be equivalent.

Theorem 1. ***(Inductive and co-inductive equivalence)***
Let r and s be R-pairs. Then: $Equal_{ind}(r,s)$ if and only if $Equal_{coind}(r,s)$.

Proof. *The prove of the implication ($\Rightarrow$) is straightforward. We perform case analysis on the parameters r, s and then we argument by co-induction.*

The case ($\Leftarrow$) is simple too. It is proved by induction and case analysis.

The propostion is proved formally in the Coq system: the full proof is available at the URL http://www.dimi.uniud.it/~ciaffagl. □

We claim that this correspondence between a co-inductive predicate and the negation of an inductive one is just an instance of a more general phenomenon. We conjecture that a large class of co-inductive predicates (but not all [Coq93]) —those defined using decidable predicates— are equivalent to negations of some inductive one.

Another important purpose of this section is to define the addition and multiplication *functions*. The addition for streams makes use of an auxiliary function $+_{aux} : (str \times str \times Z) \to str$, whose intended meaning is:

$$[\![+_{aux}(x,y,i)]\!]_R = \begin{cases} ([\![x]\!]_R + [\![y]\!]_R + i)/4 & \text{if } ([\![x]\!]_R + [\![y]\!]_R + i) \in [-4,4] \\ (-1) & \text{if } ([\![x]\!]_R + [\![y]\!]_R + i) < -4 \\ (+1) & \text{if } ([\![x]\!]_R + [\![y]\!]_R + i) > 4 \end{cases}$$

Using the previous definition we can give addition algorithms for streams and R-pairs. We want here just to remark that the result of the addition between streams must be normalized (divided) by a factor 2, since streams can represent only the real numbers in the limited interval $[-1,1]$.

Definition 9. ***(Addition function)***
The function $+_{aux} : (str \times str \times Z) \to str$ is defined by co-recursion:

$$\begin{aligned} +_{aux}(a:x,\ b:y,\ i) =_{def}\ & \textbf{\textit{let}}\ j := (2i + a + b)\ \textbf{\textit{in}} \\ & \textbf{\textit{Case}}\ j\ \textbf{\textit{of}} \\ & \quad j > 2 :\ (1 : (+_{aux}(x,y,i-4))) \\ & \quad j \in [-2,2] :\ (0 : (+_{aux}(x,y,i))) \\ & \quad j < -2 :\ (-1 : (+_{aux}(x,y,i+4))) \\ & \textbf{\textit{end}} \end{aligned}$$

The addition function on streams $+_{str} : (str \times str) \to str$ is defined as:

$$+_{str}(a:x,\ b:y)\ =_{def}\ +_{aux}(x,\ y,\ a+b)$$

The addition function on R-pairs $+_R : (R \times R) \to R$ is defined as:

$$+_R(\langle m,x\rangle, \langle n,y\rangle)\ =_{def}\ \langle m+n+1,\ +_{str}(\underbrace{0:\ldots:0}_{n}:x,\ \underbrace{0:\ldots:0}_{m}:y)\rangle$$

The multiplication function is defined in terms of the addition one. Also in this case it is convenient to use an auxiliary function $\times_{aux} : (str \times str \times str \times [-2,2]) \to str$, with intended meaning:

$$[\![\times_{aux}(x,y,z,i)]\!]_{str} \;=\; (([\![x]\!]_{str} \times [\![y]\!]_{str}) + [\![z]\!]_{str} + i)/4$$

Definition 10. ***(Multiplication function)***
The function $\times_{aux} : (str \times str \times str \times Z) \to str$ is defined by co-recursion:

$$\begin{array}{l}\times_{aux}(a:x,\ y,\ c:z,\ i) =_{def} \textbf{\textit{let}}\ (d:w) := +_{str}(\times_{d,str}(a,y),z)\ \textbf{\textit{in}}\\ \qquad\qquad\qquad\qquad \textbf{\textit{let}}\ j := (2i+c+d)\ \textbf{\textit{in}}\\ \qquad\qquad\qquad\qquad \textbf{\textit{Case}}\ j\ \textbf{\textit{of}}\\ \qquad\qquad\qquad\qquad\qquad j > 2:\ (1:(\times_{aux}(x,y,w,j-4)))\\ \qquad\qquad\qquad\qquad\qquad j \in [-2,2]:\ (0:(\times_{aux}(x,y,w,j)))\\ \qquad\qquad\qquad\qquad\qquad j < -2:\ (-1:(\times_{aux}(x,y,w,j+4)))\\ \qquad\qquad\qquad\qquad \textbf{\textit{end}}\end{array}$$

The multiplication function on streams $\times_{str} : (str \times str) \to str$ is defined as:

$$\begin{array}{l}\times_{str}(a:a_1:x,\ b:y) =_{def} \textbf{\textit{let}}\ (c:i) := +_{str}(\times_{d,str}(a,y),\ \times_{d,str}(a_1,\ b:y))\\ \qquad\qquad\qquad\qquad \textbf{\textit{in}}\ \times_{aux}(x,\ b:y,\ i,\ (c+ab))\end{array}$$

The multiplication function on R-pairs $\times_R : (R \times R) \to R$ is defined as:

$$\times_R(\langle m,x\rangle,\langle n,y\rangle) \;=_{def}\; \langle m+n,\ \times_{str}(x,y)\rangle$$

It is fundamental now to prove that the addition and multiplication functions are coherent w.r.t. the corresponding predicates.

Theorem 2. ***(Arithmetic predicates and functions)***
The arithmetic predicates and functions are related by the following properties:

$$\begin{array}{ll}\forall r,s \in R. & Add(r,s,+_R(r,s))\\ \forall r,s,t \in R. & Add(r,s,t) \;\Leftrightarrow\; Equal_{ind}(+_R(r,s),t)\end{array}$$

$$\begin{array}{ll}\forall r,s \in R. & Mult(r,s,\times_R(r,s))\\ \forall r,s,t \in R. & Mult(r,s,t) \;\Leftrightarrow\; Equal_{ind}(\times_R(r,s),t)\end{array}$$

Proof. *The proposition is proved formally in the Coq system.* □

If we consider the arithmetic predicates as a kind of *specification* for the corresponding functions, then the previous proposition states that the implementation we have given by algorithms satisfies the specification. It follows that we can derive the properties of the functions from the properties of the corresponding predicates: this is an advantage, because the specifications are easier to work with.

A final remark: the above proposition can be seen as a case-study for the goal of proving the correctness of functions performing exact real number computation.

5 An axiomatisation of constructive real numbers

In section 3 he have picked out and discussed two different approaches to justify our representation of the constructive real numbers. We start here addressing the internal-axiomatic one.

In order to prove the adequacy of our structure we would need to dispose of a set of properties characterizing abstractly the constructive real numbers, i.e. a set of axioms. As far as we know, there exists no such a standard axiomatization: the only we know is that proposed by the working group of the FTA project [Fta99], whose aim is to formalize and prove in Coq the "Fundamental Theorem of Algebra". Starting from the FTA's axioms we have synthesized a simple and equivalent set of axioms. An advantage of having a small set of axioms is that it is easier to verify whether a given structure satisfies them. This process of minimalization has aided us to understand what are the fundamental notions characterizing the constructive real numbers.

An advantage in presenting and using axioms consists in the re-usability of the proofs: if a certain property has been proved making use only of the axioms —and not considering the actual representation being studied— then we'll be allowed to reuse the proof for any structure satisfying the same axioms.

The axiomatisation we propose is the following —we have used below the customary functional symbolism for the arithmetic operations.

Definition 11. ***(Axioms for constructive real numbers)***

Constants :	$R,\ \{0_R, 1_R\} \in R$	
	$< \subseteq R \times R$	
	$+ :\ R \times R \to R$	
	$\times :\ R \times R \to R$	
Definitions :	$\sim \subseteq\ R \times R$	$\forall x, y \in R.$
		$(x \sim y) \ \Leftrightarrow\ (\neg(x < y) \wedge \neg(y < x))$
	$Near \subseteq R \times R \times R$	$\forall x, y, \epsilon \in R.$
		$Near(x, y, \epsilon) \ \Leftrightarrow\ (x < y + \epsilon) \wedge (y < x + \epsilon)$
Order :	*neuter elements*	$0 < 1$
	$<$ *is asymmetric*	$\forall x, y \in R.\ (x < y) \to \neg(y < x)$
	$<$ *is transitive*	$\forall x, y, z \in R.\ (x < y) \wedge (y < z) \to (x < z)$
	$<$ *is weak-total*	$\forall x, y, z \in R.\ (x < y) \to ((x < z) \vee (z < y))$
Add :	$+$ *is associative*	$\forall x, y, z \in R.\ ((x + y) + z) \sim (x + (y + z))$
	$+$ *is commutative*	$\forall x, y \in R.\ (x + y) \sim (y + x)$
	$+$ *has identity*	$\forall x \in R.\ (x + 0_R) \sim x$
	opposite exists	$\forall x \in R.\ \exists y \in R.\ (x + y) \sim 0_R$
	$+$ *preserves* $<$	$\forall x, y, z \in R.\ (x < y) \to (x + z) < (y + z)$
	$+$ *reflects* $<$	$\forall x, y \in R.\ 0_R < (x + y) \to ((0_R < x) \vee (0_R < y))$

$$
\begin{array}{lll}
\mathit{Mult}: & \times \textit{ is associative} & \forall x,y,z \in R.\ ((x \times y) \times z) \sim (x \times (y \times z)) \\
& \times \textit{ is commutative} & \forall x,y \in R.\ (x \times y) \sim (y \times x) \\
& \times \textit{ has identity} & \forall x \in R.\ (x \times 1_R) \sim x \\
& \textit{inverse exists} & \forall x \in R.\ \neg(x \sim 0_R) \rightarrow \exists y \in R.\ (x \times y) \sim 1_R \\
& \times \textit{ distributes } + & \forall x,y,z \in R. \\
& & (x \times (y + z)) \sim (x \times y) + (x \times z) \\
& \times \textit{ preserves } < & \forall x,y \in R.\ (0_R < x) \wedge (0_R < y) \rightarrow (0_R < (x \times y)) \\
& \times \textit{ reflects } < & \forall x,y \in R. \\
& & (x \times y < 1_R) \rightarrow ((x < 1_R) \vee (y < 1_R)) \\
\\
\mathit{Limit}: & \textit{limit exists} & \forall f : N \rightarrow I\!R. \\
& & (\forall \epsilon > 0.\ \exists n.\ \forall m > n.\ \mathit{Near}(f(n), f(m), \epsilon)) \rightarrow \\
& & (\exists x \in R.\ \forall \epsilon > 0.\ \exists n.\ \forall m > n.\ \mathit{Near}(x, f(m), \epsilon))
\end{array}
$$

It is natural now to raise the fundamental question whether this axiomatisation is complete. In particular, one could require that all the properties provable starting from our representation of real numbers can also be proved using only the above axioms —we don't know yet whether our axioms satisfy this completeness requirement. A weaker and informal request consists in asking that using the above axioms it is possible to derive the standard and fundamental properties of the real numbers: we will discuss this and related aspects in section 8.

A final remark concerns the arithmetic functions "opposite" ($\lambda x. -x$) and "inverse" ($\lambda x.1/x$): we don't need to require explicitly their existence, provided that —in the statement of the correspondent axioms— we use the existential quantification over *Set* rather than *Prop*. This assumption actually makes the Axiom of Choice provable in Coq.

6 Formalization in Coq

We have already motivated the use of the logical framework Coq to investigate the internal-axiomatic adequacy of our constructive real numbers. We remember here that the logic of Coq is standardly *intuitionistic*.

The formalization of the structure we have developed is simple: we need only to translate our definitions in the specification language of Coq. The first step consists in the encoding of the datatypes (digits, streams and R-pairs).

```
Inductive digit : Set := mino : digit | zero : digit | one : digit.
CoInductive str : Set := cons : digit -> str -> str.
Inductive R : Set := pair : nat -> str -> R.
```

The specification of the other constants of the structure is analogous. Two technical details: we use the function `encod` to map the symbolic names of the ternary digits to their integer values and the function `append0` to "normalize" the streams. We list below these two definitions and some other example of our formalization.

```
Definition encod : digit -> Z :=
     [a:digit] Cases a of mino => `-1` |  zero => `0` |  one  => `1`
               end.
Fixpoint append0 [n:nat] : str -> str :=
     Cases n of (0)   => [x:str] x
             |  (S m) => [x:str] (cons zero (append0 m x))
     end.

Inductive less_aux : str -> str -> Z -> Prop :=
     less_base: (x,y:str) (less_aux x y big)
  |  less_ind : (x,y:str) (a,b:digit) (i,j:Z)
                (less_aux x y i) ->
                (`i + (encod a) <= 2*j + (encod b)`) ->
                (less_aux (cons a x) (cons b y) j).
Definition less_str : str -> str -> Prop :=
     [x,y:str] (less_aux x y `0`).
Definition Less : R -> R -> Prop :=
     [r,s:R] Cases r of (pair m x) => Cases s of (pair n y) =>
                   (less_str (append0 n x) (append0 m y))
             end end.

CoInductive add_aux : str -> str -> str -> Z -> Prop :=
     add_coind: (x,y,z:str) (a,b,c:digit) (i:Z)
                (add_aux x y z `2*i ) + (cod c) - (cod a) - (cod b`) ->
                (`-big <= i`) -> (`i <= big`) ->
                (add_aux (cons a x) (cons b y) (cons c z) i).

CoFixpoint times_d_str : digit -> str -> str :=
     [a:digit][x:str] Cases x of (cons b y) =>
                              (cons (times_digit a b) (times_d_str a y))
                      end.
CoInductive mult_tstr : str -> str -> str -> Prop :=
     mult_coind : (x,y,z,w:str) (a:digit)
          (mult_tstr x y w) ->
          (add_tstr (cons zero (times_d_str a y)) (cons zero w) z) ->
          (mult_tstr (cons a x) y z).
```

It comes natural now to focus briefly on (co-)inductive types and their inhabitants. In Coq, *recursive* types [Gim98a] can be classified into inductive [Coq92, PM93] and co-inductive ones [Gim94]: both of them are described by introduction rules. Elements of the inductive types are constructed by means of a *finite* application of the introduction rules, whereas co-inductive ones can be obtained also by a *potentially infinite* use of the rules.

The user is allowed to construct (lazily) specific infinite objects using co-recursive definitions in the form of "fixed-point" declarations, provided the recursive calls are "guarded" [Coq93, Gim94, AC97, Gim98b]. An example is the

infinite stream of "zeros", which can be obtained by the definition `CoFixpoint zeros : str := (cons zero zeros).` A dual situation holds for inductive objects: in this case recursive definitions can be used as elimination rules performing structural induction. The Coq code listed above shows the two cases just discussed: the function `times_d_str` builds infinite streams; the function `appendO` is defined by induction on a natural parameter.

7 Certification of the constructive real numbers

The aim of this section is to present and discuss the main result of our work.

Theorem 3. ***(Constructive real numbers)***
Our representation of real numbers satisfies the axioms of definition 11.

So far we have not dealt with all the axioms: the properties concerning the order and the addition have been already proved in Coq; the proofs of the other properties are at the moment in progress, but we are optimistic to conclude them in the immediate future. The whole code is available at the URL http://www.dimi.uniud.it/∼ciaffagl.

We want to supply here some remarks about the proof technique used.

Most of the proofs follow a similar pattern: first we prove a lemma for the auxiliary predicate, then a lemma for the predicate defined on streams and finally a main proposition for the R-pairs. As already explained, we have preferred to develop the proofs for addition and multiplication using the predicates rather than the functions. Nevertheless, by the validity of the theorem 2, it is possible to extend the proofs involving the predicates, thus obtaining those of the properties involving the corresponding functions.

Normally, the main difficulty is to prove the lemma at the "aux" level. In order to exemplify the whole process, let us consider the "associativity of addition". In this case we need to prove the following hierarchy of judgements:

$$\begin{aligned}
add_assoc_{aux} : & \forall x,y,z,w,u,v \in \mathrm{str},\ \forall i,j,k \in Z. \\
& add_{aux}(x,y,w,i) \rightarrow add_{aux}(w,z,u,j) \rightarrow \\
& add_{aux}(y,z,v,\text{`}i+j-k\text{`}) \rightarrow add_{aux}(x,v,u,k)
\end{aligned}$$

$$\begin{aligned}
add_assoc_{str} : & \forall x,y,z,w,u,v \in \mathrm{str}. \\
& add_{str}(x,y,w) \rightarrow add_{str}(w,z,u) \rightarrow add_{str}(y,z,v) \rightarrow \\
& add_{str}(x,v,u)
\end{aligned}$$

$$\begin{aligned}
add_assoc_{R} : & \forall r,s,t,l,p,q \in R. \\
& Add(r,s,l) \rightarrow Add(l,t,p) \rightarrow Add(s,t,q) \rightarrow \\
& Add(r,q,p)
\end{aligned}$$

$$add\_assoc : \quad \forall r,s,t \in R.\ Equiv((x+y)+z,\ x+(y+z))$$

Further details concern the two tactics we have mainly used: `Cofix` and `Omega`. The tactic `Cofix` is specific for co-inductive reasoning: it is the main

tool for proving co-inductive assertions. This tactic allows to develop top-down (infinitary) proofs assuming the conclusion as a premisse, provided it is later used only within introduction rules. The tactic `Omega` automaticly proves all the judgements expressed in the language of Presburger's arithmetic: it has been very useful to avoid repeated case analyses on the values of the ternary digits. The use of this tactic and the introduction of the auxiliary predicates have permitted us to write quite simple proofs: almost all the propositions are proved invoking at most 50 strategies.

8 Consequences of the axioms

In order the axiomatisation of definition 11 can be considered (sufficiently) complete, most of the properties of the real numbers should follow from it.

A first step is to consider the axioms proposed by the FTA working group [Fta99]. We conjecture that all the properties presented there can be deduced using our axioms: we have proved formally this fact for all the axioms concerning the order relation and addition; we are confident that the others can be proved too.

Moreover, it is interesting to compare our axioms with the basic properties of the constructive reals presented by Troelstra and van Dalen [TvD88]. In that work, real numbers are constructed as Cauchy sequences of rationals; the formal properties of this construction are then studied in depth. So far, we have investigated the properties concerning the order relation and addition. We have proved informally that they follow from our axioms and we are at the moment formalizing those proofs. At present, we have proved the following facts, among other more technical.

- The relations "equal" ($\sim$), "less-equal" ($\leq$) and "apart" ($\#$) can be defined in terms of our "less" ($<$) relation.
- The relation "$\sim$" is an equivalence. The relation "$\leq$" is an order.
- The addition operation ($+$) preserves the order relation.
- The real numbers, equipped with the relation "$\sim$" and the operation "$+$", form a group.

As the reader can see, for some aspects this is still a work in progress. In the future we will going to conclude our basic formulation of analysis and then to develop it. We are also interested in the use of our framework for testing the correctness of algorithms performing exact computation on real numbers. This and further work will be documented at the URL http://www.dimi.uniud.it/~ciaffagl.

References

[AC97] R. Amadio and S. Coupet. *"Analysis of a guard condition in Type Theory"*. Technical report, INRIA, 1997.

[Bis67] E. Bishop. *"Foundations of constructive analysis"*. McGraw-Hill, New York, 1967.

[Bro24] L.E.J. Brouwer. *"Beweis, dass jede volle Funktion gleichmässig stetig ist"*. In *Proc. Amsterdam 27*, pages 189–194, 1924.

[Ced97] J. Cederquist. *"A pointfree approach to Constructive Analysis in Type Theory"*. PhD thesis, Göteborg University, 1997.

[CH92] J. Chirimar and D.J. Howe. *"Implementing constructive real analysis: preliminar report"*. *LNCS*, 613, 1992.

[Con86] R.L. Constable. *"Implementing mathematics with the Nuprl development system"*. Prentice-Hall, 1986.

[Coq92] T. Coquand. *"Pattern-matching in Type Theory"*. In *Informal Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.

[Coq93] T. Coquand. *"Infinite objects in Type Theory"*. In *1st Workshop on Types for Proofs and Programs*, LNCS 806, 1993.

[Fta99] *The "Fundamental theorem of Algebra" Project*, 1999. Computing Science Institute Nijmegen, http://www.cs.kun.nl/∼freek/fta/index.html.

[Gim94] E. Giménez. *"Codifying guarded definitions with recursion schemes"*. In *5th Workshop on Types for Proofs and Programs*, LNCS 996, 1994.

[Gim98a] E. Giménez. *"A tutorial on recursive types in Coq"*. Technical report, INRIA, 1998.

[Gim98b] E. Giménez. *"Structural recursive definitions in Type Theory"*. In *Proceedings of ICALP'98*, LNCS 1443, 1998.

[GM93] M.J.C. Gordon and T.F. Melham. *"Introduction to HOL: a theorem proving environment for higher order logic"*. Cambridge University Press, 1993.

[Har96] J.R. Harrison. *"Theorem proving with real numbers"*. PhD thesis, Universitiy of Cambridge, 1996.

[INR00] INRIA, project Coq. *"The Coq proof assistant - Reference manual V6.3.1"*, May 2000.

[Jon91] C. Jones. *"Completing the rationals and metric spaces in Lego"*. In Cambridge Universitiy Press, editor, *Logical Frameworks*, pages 209–222. 1991.

[Mag95] L. Magnusson. *"The implementation of Alf"*. PhD thesis, Chalmers University of Technology, Göteborg, 1995.

[PEE97] P.J. Potts, A. Edalat, and M.H. Escardo. *"Semantics of exact real arithmetic"*. In *IEEE Symposium on Logic in Computer Science*, 1997.

[PM93] C. Paulin-Mohring. *"Inductive definitions in the system Coq: rules and properties"*. In *Proceedings of the TLCA*, 1993.

[Pol94] R. Pollack. *"The theory of Lego, a proof checker for the Extended Calculus of Constructions"*. PhD thesis, University of Edimburgh, 1994.

[Sim98] A. Simpson. *"Lazy functional algorithms for exact real functionals"*. In *MFCS 1998*. Springer Verlag, 1998.

[TvD88] A.S. Troelstra and D. van Dalen. *"Constructivism in Mathematics"*. North-Holland, 1988.

[Wei96] K. Weihrauch. *"A foundation for computable analysis"*. In *Proceedings of DMTCS 1996*, 1996.