Università degli Studi di Udine
Dipartimento di Matematica e Informatica

A Co-inductive approach to Real Numbers

Alberto Ciaffaglione, Pietro Di Gianantonio

Workshop on Certified Arithmetics
École Normale Supérieure de Lyon (France)
Lyon, November 16, 2000

Motivations

Computer assisted reasoning on real numbers:

- formalization of mathematics
- verification of computer systems where the "continous" is involved (applications: hybrid systems, floating point hw, etc.)
- specification and certification of programs for exact real computation (software reliability)

Exact real computation:
- infinite sequences (streams) as real number generators
- lazy evaluation of initial segments of streams
- computations start from the most significant digits

$\Rightarrow$ Experimenting Coq with a co-inductive theory:
Coq mechanizes powerful definition and proof principles

Co-induction and Reals in Type Theory

Coq's primitive features are naturally well suited w.r.t. our aims:
- co-inductive types (potentially infinite objects, as streams-Reals)
- lazy representation of infinite objects (functional implementation of exact computation: names as real numbers "generators")
- co-recursive definitions introducing specific infinite objects; finite initial segments are lazily evaluated
- proofs by co-induction are implemented via the `Cofix` tactic, corresponding to a "guarded" proof principle

⇒ Choice: to construct Reals using intuitionistic logic

- real numbers via streams (and redundant notations)
- fundamental notions and algorithms
- adequacy (fundamental properties)

Notation

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

Interpretation: $[\![\ ]\!]_{str} : str \to I\!R$

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

Let $R \equiv (I\!N \times str)$ $(R - pairs)$

Interpretation: $[\![\ ]\!]_R : R \to I\!R$

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

```
Inductive digit : Set := mino : digit | zero : digit | one : digit.
Definition encod : digit -> Z := [a:digit]
      Cases a of mino => ‘-1‘ | zero => ‘0‘ | one  => ‘1‘
      end.
CoInductive str : Set := cons : digit -> str -> str.
Inductive R : Set := pair : nat -> str -> R.
```

Primitive notions: strict order (<)

We introduce an auxiliary predicate $less\_aux \subseteq (str \times str \times Z)$:

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

It is defined inductively by the two rules:

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

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

Then we define the main predicates on streams and R-pairs:

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

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

Primitive notions: addition (+)

It is simpler to reason on arithmetic predicates than on functions.

We will prove that functions match specifications given by predicates.

(+): we introduce an auxiliary predicate $add\_aux \subseteq (str \times str \times str \times Z)$:

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

It is defined co-inductively by the rule:

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

The main predicates on streams and R-pairs follow straightforwardly:

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

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

Primitive notions: multiplication (×)

We need an auxiliary function $times_{d,str} : \{-1, 0, 1\} \times str \to str$, defined by co-recursion:

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

Then we introduce the multiplication predicate on streams $mult_{str} \subseteq (str \times str \times str)$ via the co-inductive rule:

$$\textsc{mult-coind}\ \frac{mult_{str}(x, y, w) \quad add_{str}(0 : times_{d,str}(a, y),\ 0 : w,\ z)}{mult_{str}(a : x,\ y,\ z)}$$

We can gain the level of R-pairs as usual:

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

A derived notion: equivalence (quotient)

It is natural to define an equivalence predicate on R-pairs.

1. We introduce an auxiliary predicate $equal\_aux \subseteq (str \times str \times Z)$:

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

It is defined co-inductively by the rule (expand it: synthetic):

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

2. We introduce directly an inductive predicate on R-pairs using $<$:

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

Theorem 1 ***(Inductive and co-inductive equivalence)***
$Equal_{ind}(r, s)$ if and only if $Equal_{coind}(r, s)$.
Proof. *In Coq: (⇒) By co-induction (⇐) By induction*

Arithmetic functions

We introduce addition and multiplication functions.

(+): an auxiliary co-recursive function $+_{aux} : (str \times str \times Z) \rightarrow str$ is needed:

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

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

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

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

Arithmetic functions (cont'd)

If we consider the arithmetic predicates as a specification for the corresponding functions, then it is significant to prove that the implementation given by algorithms satisfies the specification described by predicates.

Theorem 2 ***(Arithmetic predicates and functions)***
The arithmetic predicates and functions are related as follows:

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

Proof. *In Coq*

⇒ We can work directly on the properties of predicates (it is simpler)

- We have presented a case-study about the correctness of functions performing exact real number computation

The Coq code

```
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_tstr : tstr -> tstr -> Prop := [x,y:tstr] (less_aux x y ‘0‘).
Definition less : real -> real -> Prop := [r,s:real] Cases r of (pair m x) =>
                Cases s of (pair n y) => (less_tstr (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 <= big‘) -> (add_aux (cons a x) (cons b y) (cons c z) i).
CoFixpoint r_plus_aux : tstr -> tstr -> Z -> tstr :=
     [x,y:tstr] [i:Z] let j = ‘2*i + (cod (head x)) + (cod (head y))‘ in Cases (‘j + 1‘)
                of (NEG k) => (cons mino (r_plus_aux (tail x) (tail y) (‘j + 4‘)))
                |  (ZERO)  => (cons zero (r_plus_aux (tail x) (tail y) j))
                |  (POS k) => Cases (‘j - 1‘)
                   of (POS k) => (cons one (r_plus_aux (tail x) (tail y) (‘j - 4‘)))
                   |  ______  => (cons zero (r_plus_aux (tail x) (tail y) j)) end end.
```

Adequacy

Two approaches to justify our construction: external-semantic; internal-axiomatic

Theorem 3 ***(Adequacy of order, addition and multiplication)***
The rules for the order, addition and multiplication predicates are sound and complete. That is, for any triple of R-pairs r, s, t*:*
$Less(r, s)$ *can be derived if and only if* $(\llbracket r \rrbracket_R < \llbracket s \rrbracket_R)$,
$Add(r, s, t)$ *can be derived by an infinite derivation if and only if* $(\llbracket r \rrbracket_R + \llbracket s \rrbracket_R = \llbracket t \rrbracket_R)$,
$Mult(r, s, t)$ *can be derived by an infinite derivation if and only if* $(\llbracket r \rrbracket_R \cdot \llbracket s \rrbracket_R = \llbracket t \rrbracket_R)$

Theorem 4 ***(Constructive real numbers)***
Our representation of real numbers satisfies a set of axioms

Internal adequacy (constructive axioms)

Consts : $R,\ \{0_R, 1_R\} \in R,\ < \ \subseteq R \times R, \quad + :\ R \times R \to R, \quad \times :\ R \times R \to R$

Defs : $\sim\ \subseteq\ R \times R : \quad \forall x, y \in R.\ (x \sim y)\ \Leftrightarrow\ (\neg(x < y) \wedge \neg(y < x))$

$Near \subseteq R \times R \times R : \quad \forall x, y, \varepsilon \in R.\ Near(x, y, \varepsilon)\ \Leftrightarrow\ (x < y + \varepsilon) \wedge (y < x + \varepsilon)$

Order : neuter elements : $0 < 1$, $<$ is asymmetric and transitive

$<$ is weak-total : $\forall x, y, z \in R.\ (x < y) \to ((x < z) \vee (z < y))$

Add : $+$ is associative, commutative, has identity and opposite exists

$+$ preserves $<$: $\forall x, y, z \in R.\ (x < y) \to (x + z) < (y + z)$

$+$ reflects $<$: $\forall x, y \in R.\ 0_R < (x + y) \to ((0_R < x) \vee (0_R < y))$

Mult : $\times$ is associative, commutative, has identity and inverse exists

$\times$ distributes $+$: $\forall x, y, z \in R.\ (x \times (y + z)) \sim (x \times y) + (x \times z)$

$\times$ preserves $<$: $\forall x, y, z \in R.\ (0_R < x) \wedge (0_R < y) \to (0_R < (x \times y))$

$\times$ reflects $<$: $\forall x, y, z \in R.\ (x \times y < 1_R) \to ((x < 1_R) \vee (y < 1_R))$

Limit : limit exists : $\forall f : \mathbb{N} \to R.\ (\forall \varepsilon > 0.\ \exists n.\ \forall m > n.\ Near(f(n), f(m), \varepsilon)) \to$
$(\exists x \in R.\ \forall \varepsilon > 0.\ \exists n.\ \forall m > n.\ Near(x, f(m), \varepsilon))$

Related and future work

- to further develop analysis
- to compare our representation to other constructive approaches
  - FTA[a] in Nijmegen
- to certify exact real algorithms
  (analytic functions: sin, cos, exp, log, . . .)

---

[a] "The Fundamental Theorem of Algebra" project