# Implementing real numbers with RZ

Andrej Bauer Iztok Kavkler

Faculty of mathematics and physics
University of Ljubljana
Slovenia

18 June 2007

# Introduction

Currently, there are two kinds of computable real number implementations.

1. The implementations strictly adhering to the theory, e.g. extracted from formalizations of reals in Coq. They tend to be rather inefficient.
2. Fast implementations based on interval arithmetic, but with only informal theoretical background.

Our goal is to get the best of both: provide real arithmetic that is both efficient and easily formalizable.

1. Extract specification from the theory—automatically, if possible.
2. Leave the programmer some freedom to produce fast implementation.

# RZ

- Converts theories of constructive mathematics to OCaml module type specifications.
- Based on the realizability implementation of logic.
  - The computational content is identified and translated to types and function prototypes.
  - The code is annotated with assertions (that can be proved in classical logic).
  - Allows any implementation, as long as it follows the produced specification—it is possible to use OCaml to its full potential.

# RZ example

The statement that every complex number has a square root can be expressed in RZ as

```
Axiom sqrt:  ∀z:complex, ∃w:complex, z = mul w w.
```

It produces the following OCaml specification.

```
val sqrt :  complex → complex
(** assertion sqrt :  ∀(z:‖complex‖),
  let p=sqrt z in p:‖complex‖ ∧ z ≈complex mul p p *)
```

There is no requirement for extensionality—a multi-valued function realizes the above specification.

# Constructive and computable reals

Usually, real numbers are represented by sequences $\{r_n\}$ with rapid Cauchy convergence property

$$|r_n - r_{n+1}| < 2^{-n}. \tag{1}$$

- The corresponding OCaml implementation is a function `r: nat -> rational` with property (1).
- The problem: every operation has to preserve rapid convergence which usually results in estimates that overshoot the precision. We end up computing much more than is needed.
- Our implementation via the interval domain does not have this drawback.

# Overview of the implementation

We formally axiomatized the following theories:

1. the ring of integers (with natural numbers as subset),
2. the ring of dyadic rationals,
3. the poset of intervals with dyadic endpoints,
4. the interval domain,
5. the field of reals.

These were translated by RZ to OCaml specifications which were implemented by hand.

# Integers

- Integers are defined as a decidable ordered ring with unit whose nonnegative elements are isomorphic to natural numbers (satisfy the axiom of induction).
- For implementation, we use fast integer library Numerix (or GMP).

# Dyadic rationals

- Precise rational operations are costly: the numerator and the denominator grow rapidly with every operation.
- As most other implementations, we use dyadic rationals

$$\mathbb{D} = \left\{ m \cdot 2^{-k} \,\middle|\, m \in \mathbb{Z}, k \in \mathbb{N} \right\}.$$

- Axiomatization. A *dyadic ring* is a decidable Archimedean ordered ring in which 2 is invertible.
- Every dyadic ring admits approximate division.
- In a dyadic ring, every element can be approximated by an element of the form $n \cdot 2^{-k}$ with the error at most $2^{-k}$.

# Dyadic intervals

Axiomatization. A dyadic interval is an interval $[p, q]$ with $p$, $q$ dyadic rationals.

- Define order $[p, q] \sqsubseteq [p', q']$ as $[p, q] \supseteq [p', q']$.
- We also adjoin the bottom element `undefined`.
- Dyadic intervals form a conditional upper semilattice.
- We axiomatize *approximate* interval arithmetic, which allows us to trade precision for efficiency.

# The interval domain

Axiomatization. The interval domain IR is the $\omega$-chain completion of the poset ID of dyadic intervals.

- An element $x \in \mathrm{IR}$ is represented by a chain of dyadic intervals
$$[p_1, q_1] \sqsubseteq [p_2, q_2] \sqsubseteq [p_3, q_3] \sqsubseteq \cdots$$
- $x$ can be thought of as the interval $[a, b]$ where
  - $a = \sup p_i$ is a *lower* real
  - $b = \inf q_i$ is an *upper* real
- Interval arithmetic operations on IR are defined as continuous extensions of the corresponding approximate operations on dyadic intervals.

# Real numbers

Axiomatization. Real numbers form a Cauchy complete Archimedean ordered field.
Implementation. Real numbers are the maximal elements of IR.

- A real $x \in \mathrm{IR}$ is represented as a chain of dyadic intervals. Crucial: no requirement on the speed of convergence.
- Arithmetic operations are inherited from the interval domain.
- Archimedean property is realized by a function

```
val approx_to: real → nat → dyadic
```

  that finds approximations of order $2^{-n}$. Correctness depends on Markov principle.

# Limits

The completeness of the reals is witnessed by the function lim:

```
val lim: (nat → real) → (nat → real) → real
```

The parameters are the sequence of real numbers $(a_n)_n$ and the sequence of Cauchy error bounds $(r_n)_n$.

$$r_n \geq \left|a_i - a_j\right| \quad \forall i, j \geq n$$

This formulation is equivalent but easier to use than the strict Cauchy sequence version.

**Lemma**

*Assume that $r_n \searrow 0$ monotonously. Then the sequence $(c_n)_n$*

$$c_n = \bigvee_{k=0}^{\infty} \left[\, \underline{a}_n^{(k)} - \overline{r}_n^{(k)}, \overline{a}_n^{(k)} + \overline{r}_n^{(k)} \right] \tag{2}$$

*is a chain in* IR *and the limit of $(a_n)_n$ is its supremum.*

# The Markov principle

Markov principle. A loop which does not diverge terminates.

- A real number is represented as a chain of dyadic intervals whose widths are not bounded away from 0.
- We can always find arbitrarily good approximation by unbounded search (equivalent to MP).
- The unbounded search is costly as it makes the time complexity of the program unpredictable. Therefore it is only used in `approx_to` which we (so far) avoid in the implementation of other functions.

# Future directions

- Improve performance and extend current implementation to a useful library.
- Axiomatize and implement other structures in analysis and topology.