The Role of the Interval Domain in Modern Exact Real Arithmetic

Andrej Bauer    Iztok Kavkler

Faculty of Mathematics and Physics
University of Ljubljana, Slovenia

Domains VIII & Computability over Continuous Data Types
Novosibirsk, September 2007
Teaching theoreticians a lesson

Recently I have been told by an anonymous referee that

“Theoreticians do not like to be taught lessons.”

and by a friend that

“You should stop competing with programmers.”

In defiance of this advice, I shall talk about the lessons I learned, as a theoretician, in programming exact real arithmetic.
## The spectrum of real number computation

<table>
<thead>
<tr>
<th>Slow</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formally verified, extracted from proofs</td>
<td>Cauchy sequences</td>
</tr>
<tr>
<td>Moebius transformations</td>
<td>streams of signed digits</td>
</tr>
<tr>
<td>continued fractions</td>
<td>iRRAM</td>
</tr>
<tr>
<td>RealLib</td>
<td>floating point</td>
</tr>
<tr>
<td>Mathematica</td>
<td></td>
</tr>
</tbody>
</table>

"theoretical" "practical"

- **Common features:**
  - Reals are represented by successive approximations.
  - Approximations may be computed to any desired accuracy.

- **State of the art, as far as speed is concerned:**
  - iRRAM by Norbert Müller,
  - RealLib by Branimir Lambov.
What makes iRRAM and ReaLib fast?

- Reals are represented by sequences of dyadic intervals (endpoints are rationals of the form $m/2^k$).
- The approximating sequences need not be nested chains of intervals.
- No guarantee on speed of converge, but arbitrarily fast convergence is possible.
- Previous approximations are not stored and not reused when the next approximation is computed.
- Each next approximation roughly doubles the amount of work done.
The theory behind iRRAM and RealLib

- Theoretical models used to design iRRAM and RealLib:
  - Type Two Effectivity
  - a version of Real RAM machines
  - Type I representations
- The authors explicitly reject domain theory as a suitable computational model.
  - For example, because representing reals as nested chains of intervals is a bad idea.
  - They seem to have convinced even some high priests of domain theory.
Our goal

- There is usually a significant gap between a theoretical model and the actual practical implementation.
- We wanted exact real arithmetic that was both efficient and easily formalizable:
  - Extract specification from the theory—automatically.
  - Leave the programmer freedom to produce fast implementation.
- We needed a tool that could extract specifications from formal descriptions of mathematical theories.
Representations and realizability

- There are many kinds of representations:
  - Numbered sets (Eršov)
  - Type Two Representations (Weihrauch)
  - Domain representations (Blanck)

These are useful in the theory of computability.

- Programmers use representations by real-world programs.

- In a related project, Chris Stone and Andrej Bauer built a tool RZ which translates first-order constructive theories into Objective Caml representations.

- RZ uses the realizability interpretation, and handles first-order logic, dependent types, and more.
Construction of reals

- We axiomatized the following theories (constructively):
  - the ring of integers $\mathbb{Z}$,
  - the “approximate field” of dyadic rational numbers $\mathbb{D}$,
  - the poset of dyadic intervals $\mathbb{I}D$,
  - $\omega$-cpos as completions of their bases,
  - the interval domain $\mathbb{I}R$ as the completion of $\mathbb{I}D$,
  - the field of real numbers $\mathbb{R}$ as the maximal elements of $\mathbb{I}R$.

- We ran RZ on these to obtain specifications for Objective Caml (given as module signatures with logical assertions).

- We implemented the specifications by hand.

- We hope our implementation satisfies all the assertions . . .

- Logical reverse-engineering: which logical theory does RZ translate to a specification for exact real arithmetic in the style of iRRAM and RealLib?

- Initially, we did not expect domain theory to play a prominent role.
Integers

- The integers are axiomatized as:
  - decidable ordered commutative ring with unit,
  - induction principle for natural numbers,
  - integer division,
  - shift operations $\text{shl}_k(n) = 2^k \cdot n$ and $\text{shr}_k(n) = \lfloor n/2^k \rfloor$,
  - binary logarithm $n \mapsto \lceil \log_2 n \rceil$.

- We used efficient implementations of large integer arithmetic:
  - Numerix by Gabriel Quercia
  - GNU Multiple Precision Library
Dyadic rationals

- Dyadic rationals $\mathbb{D}$ form a decidable ordered ring.
- Only *approximate* division:

$$\forall k \in \mathbb{N}. \forall x, y \in \mathbb{D}. (y \neq 0 \implies \exists z \in \mathbb{D}. |x/y - z| < 2^{-k})$$

- In fact, we need approximate versions of all ring operations, e.g.:

$$\forall k \in \mathbb{N}. \forall x, y \in \mathbb{D}. \exists z \in \mathbb{D}. |(x + y) - z| < 2^{-k}$$

- This allows us to control the size of dyadic numbers involved in interval arithmetic.
- Implementation:
  - our own Ocaml implementation using integers,
  - with MPFR about threefold speedup.
Dyadic intervals

The poset of dyadic intervals $\mathbb{ID}$:

- A dyadic interval $[c - r, c + r]$ is represented by its center $c \in \mathbb{D}$ and radius $r \in \mathbb{D}$.
- No need to have very precise $c$ when $r$ is large.
- No need to have very precise $r$.
- So we always normalize intervals to have $r$ with small numerator (e.g. 32 bits) and suitably rounded $c$. This trades a little bit of precision for quite a bit of space and time.
We formalize $\omega$-cpos generated from a base.

Take care to get continuous domains, not algebraic ones, and consider only chains, rather than general directed sets.

**Crucial:** how do we say that $P \subseteq D$ is a base for domain $D$?

- Inefficient: every $x \in D$ is the supremum of a chain in $P$.
- Broken: every $x \in D$ is the supremum of a sequence in $P$.
- Right: for every $x \in D$ there exists $(a_k)_k \in P$ such that

\[ \forall k \in \mathbb{N}. \ a_k \leq x \quad \text{and} \quad \lim_k a_k = x \ (\text{in Scott topology}) \]

(For $D = \mathbb{R}$ and $P = \mathbb{D}$ this condition was given by Lambov, without using domains explicitly.)
Reals as maximal elements of the interval domain

- The interval domain $\mathbb{IR}$ is the completion of $\mathbb{ID}$.
- The reals are a Cauchy-complete, Archimedean ordered field.
- $\mathbb{IR}$ is a domain model for $\mathbb{R}$.
- Avoiding explicit rates of convergence:
  - RZ translates the Archimedean axiom:
    \[
    \forall x \in \mathbb{R}. \forall k \in \mathbb{N}. \exists d \in \mathbb{D}. |x - d| < 2^{-k}
    \]
    to specification for 
    \[
    \text{approx} : \text{real} \rightarrow \text{int} \rightarrow \text{dyadic}
    \]
    computing $d$ from $x$ and $k$ such that $|x - d| < 2^{-k}$.
  - The axiom stating that $\mathbb{IR}$ is generated by $\mathbb{ID}$ yields
    \[
    \text{stage} : \text{real} \rightarrow \text{int} \rightarrow \text{dyadicInterval}
    \]
    such that $\text{stage} x$ is a sequence of intervals converging to $x$, without a prescribed speed of convergence.
In conclusion

- Other issues not discussed:
  - Strict linear order $<$ on $\mathbb{R}$ as a map into the $\omega$-cpo of partial booleans.
  - How do we represent continuous real maps?

- Statistics:
  - 503 lines of formal theories in RZ.
  - 1022 lines of Ocaml implementation.
  - 10 times slower than iRRAM for basic arithmetic.

- Lessons learned:
  - Domain theory does play a role in state-of-the-art exact real arithmetic.
  - Theoreticians’ sense of elegance can harm efficiency.
  - Don’t compete with programmers—teach them new ideas!
Future directions

- Implement something new—it will be horribly inefficient:
  - the next big step is to implement locally non-compact spaces, e.g., Banach spaces \( C^{(k)}(\mathbb{R}) \), \( L^p \), \( \ell^p \), ...

- Invent new ways of computing with real numbers, higher types and hyperspaces.

- A promising direction is “computation with quantifiers”:
  - Paul Taylor’s Abstract Stone Duality (ASD) and use of \( \exists x \in \mathbb{R} \) and \( \forall x : [a, b] \) in real analysis.
  - The Russian version of ASD seems to be \( \Sigma \)-definability (with \( \forall K \)), but ASD has more of a programming flavor.
  - Martin Escardó’s implementation of \( \forall_K \) in Haskell is surprisingly efficient.