On an Approach to Implementing Exact Real Arithmetic in Curry

Christoph Beierle, Udo Lelitko

Dept. of Computer Science, FernUniversität in Hagen, Germany

WFLP 2013
22nd International Workshop on Functional and (Constraint) Logic Programming
27th Workshop on Logic Programming
Kiel, Germany
September 11-13, 2013
Outline

1 Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on $\mathbb{R}$

2 An Abstract View on the Data Type $\text{Real}$

3 Auxiliary Types and Functions

4 Representing Real Numbers as Cauchy Sequences

5 Conclusions and further work
1 Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on $\mathbb{R}$

2 An Abstract View on the Data Type $\mathbb{R}$

3 Auxiliary Types and Functions

4 Representing Real Numbers as Cauchy Sequences

5 Conclusions and further work
Functions on $\mathbb{N}$ (or on finite words)

- well-established concepts of effectively computable functions
- different concepts, all equivalent (e.g., Turing machines)
Computable Functions

- Functions on $\mathbb{N}$ (or on finite words)
  - well-established concepts of effectively computable functions
  - different concepts, all equivalent (eg. Turing machines)

- Functions on $\mathbb{R}$ (or on infinite words)
  - different approaches to computable analysis
  - approaches not equivalent
  - differences in content and in technical details
  - here: exact real arithmetic based on Type-2 Theory of Effectivity [Weihrauch 2000]
(Type-1) Computability Theory

- (partial) functions over finite words:

\[ f : \Sigma^* \rightarrow \Sigma^* \]

- computable function given by Turing machine
- computability on other sets \( M \)
  (e.g., rational numbers, graphs, \ldots)
  - use words as names or codes of elements of \( M \)
  - interpret words computed by Turing machine as elements of \( M \)
real numbers can not be represented by finite words

\[ \pi = 3.14159 \ldots \]
real numbers can not be represented by finite words

\[ \pi = 3.14159 \ldots \]

Type-2 Theory of Effectivity (TTE) [Weihrauch 2000]
- extends Type-1 computability
- infinite words are used as names for real numbers
- (partial) functions over infinite words:

\[ f : \Sigma^\omega \rightarrow \Sigma^\omega \]

computable function given by machine transforming infinite sequences to infinite sequences
Type-2 Machine

Turing machine $M$ with

- $k$ one-way, read-only input tapes
- finitely many (two-way) work tapes
- a single one-way, write-only output tape
function \( f_M \) computed by \( M \)

- \( y_1, \ldots, y_k \in \Sigma^* \cup \Sigma^\omega \) on input tapes

Case 1:

\[
f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*
\]

iff \( M \) halts on input \( y_1, \ldots, y_k \) with \( y_0 \) on the output tape

Note:

\( f_M(y_1, \ldots, y_k) \) is undefined if \( M \) computes forever, but writes only finitely many symbols on the output tape
function $f_M$ computed by $M$

- $y_1, \ldots, y_k \in \Sigma^* \cup \Sigma^\omega$ on input tapes

Case 1:

$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*$$

iff $M$ halts on input $y_1, \ldots, y_k$ with $y_0$ on the output tape

Case 2:

$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^\omega$$

iff $M$ computes forever on input $y_1, \ldots, y_k$ and writes $y_0$ on the output tape
function $f_M$ computed by $M$

- $y_1, \ldots, y_k \in \Sigma^* \cup \Sigma^\omega$ on input tapes

Case 1:

$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*$$

iff $M$ halts on input $y_1, \ldots, y_k$ with $y_0$ on the output tape

Case 2:

$$f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^\omega$$

iff $M$ computes forever on input $y_1, \ldots, y_k$ and writes $y_0$ on the output tape

Note: $f_M(y_1, \ldots, y_k)$ is undefined if $M$ computes forever, but writes only finitely many symbols on the output tape
Definition (computable function)

\[ f : \subseteq Y_1 \times \ldots \times Y_k \rightarrow Y_0 \]

is computable iff it is computed by a Type-2 machine \( M \).
Definition (computable function)

\[ f : \subseteq Y_1 \times \ldots \times Y_k \rightarrow Y_0 \]

is computable iff it is computed by a Type-2 machine \( M \).

Infinite computations can not be finished in reality – but
- finite computations
  - on finite initial parts of inputs
  - producing finite initial parts of outputs
- can be realized
  - up to any arbitrary precision

WFLP 2013
Type-2 Machines for $\mathbb{R}$: Which names?

Example (addition in decimal representation)

Inputs:

\begin{align*}
y_1 &= 0.6666666666... \\
y_2 &= 0.3333333333...
\end{align*}

After reading finitely many input symbols, $M$ must write either $0$ or $1$. ⇒ may be wrong depending on next input symbol. ⇒ there is no Type-2 machine computing addition on $\mathbb{R}$ and using decimal representation.
### Example (addition in decimal representation)

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>$y_1 = 0.6666666666\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$y_2 = 0.3333333333\ldots$</td>
</tr>
</tbody>
</table>
Example (addition in decimal representation)

Inputs: $y_1 = 0.6666666666\ldots$

$y_2 = 0.3333333333\ldots$

After reading \textit{finitely} many input symbols, $M$ must write either

$0.$ or $1.$
Example (addition in decimal representation)

Inputs:

\[ y_1 = 0.6666666666 \ldots \]
\[ y_2 = 0.3333333333 \ldots \]

After reading \textit{finitely} many input symbols, \( M \) must write either

\[ 0. \quad \text{or} \quad 1. \]

\( \Rightarrow \) may be \textit{wrong} depending on next input symbol
Example (addition in decimal representation)

Inputs:

\[ y_1 = 0.6666666666\ldots \]
\[ y_2 = 0.3333333333\ldots \]

After reading \textit{finitely} many input symbols, \( M \) must write either

\[ 0. \quad \text{or} \quad 1. \]

\( \Rightarrow \) may be \textit{wrong} depending on next input symbol

\( \Rightarrow \) there is \textit{no} Type-2 machine computing addition on \( \mathbb{R} \) and using decimal representation
Better names for elements of $\mathbb{R}$

- $x \in \mathbb{R}$
Better names for elements of $\mathbb{R}$

- $x \in \mathbb{R}$

- quickly converging Cauchy sequence of rational numbers $r_0, r_1, r_2, \ldots$

with

$$\lim_{i \to \infty} r_i = x$$

and

$$|r_k - x| \leq 2^{-k}$$
Example (addition using Cauchy sequences as names)

Inputs:  
\[ y = r_0, r_1, r_2, r_3, \ldots \]
\[ y' = r'_0, r'_1, r'_2, r'_3, \ldots \]
Type-2 Machines for $\mathbb{R}$: Computing functions

Example (addition using Cauchy sequences as names)

Inputs: $y = r_0, r_1, r_2, r_3, \ldots$
$y' = r'_0, r'_1, r'_2, r'_3, \ldots$

Addition

Output: $x = r_1 + r'_1, r_2 + r'_2, r_3 + r'_3, r_4 + r'_4, \ldots$
Type-2 Machines for $\mathbb{R}$: Computing functions

Example (addition using Cauchy sequences as names)

Inputs: $y = r_0, r_1, r_2, r_3, \ldots$

$y' = r'_0, r'_1, r'_2, r'_3, \ldots$

Addition

Output: $x = r_1 + r'_1, r_2 + r'_2, r_3 + r'_3, r_4 + r'_4, \ldots$

Multiplication

Output: $x = r_k \times r'_k, r_{k+1} \times r'_{k+1}, r_{k+2} \times r'_{k+2}, \ldots$
### Example (addition using Cauchy sequences as names)

<table>
<thead>
<tr>
<th>Type-2 Machines for $\mathbb{R}$: Computing functions</th>
</tr>
</thead>
</table>
| **Inputs:** $y = r_0, r_1, r_2, r_3, \ldots$
| $y' = r'_0, r'_1, r'_2, r'_3, \ldots$

**Addition**

| Output: $x = r_1 + r'_1, r_2 + r'_2, r_3 + r'_3, r_4 + r'_4, \ldots$

**Multiplication**

| Output: $x = r_k \times r'_k, r_{k+1} \times r'_{k+1}, r_{k+2} \times r'_{k+2}, \ldots$

- componentwise on input sequences
- **look ahead**: $k$ elements dropped from resulting sequence
- depends on function to be computed and on arguments
- look ahead always finite
functions on $\mathbb{R}$ not computable in TTE:

\[
\begin{align*}
  x &= y \\
  x &\leq y \\
  x &\geq y
\end{align*}
\]
Type-2 Machines for \( \mathbb{R} \): Computing functions

- finite initial part of name \( r_0, r_1, r_2, \ldots \) for \( x \in \mathbb{R} \) represents set of possible values
- increasing precision corresponds to use larger input part
- lower and upper bound of denoted set of values converge to \( x \)
- functions using initial parts of names are multi-valued

\[
\begin{align*}
eq & : \mathbb{R} \times \mathbb{R} \Rightarrow \text{Bool} \\
\leq & : \mathbb{R} \times \mathbb{R} \Rightarrow \text{Bool}
\end{align*}
\]
Goal of this work

- implement exact real arithmetic based on Type-2-Theory of Effectivity
- use declarative approach close to underlying theory
- use modular approach allowing for different representations (names) of $x \in \mathbb{R}$
- use Curry
  - functional concept
  - lazy evaluation
  - non-determinism
  - ...
Outline

1. Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on $\mathbb{R}$

2. An Abstract View on the Data Type $\text{Real}$

3. Auxiliary Types and Functions

4. Representing Real Numbers as Cauchy Sequences

5. Conclusions and further work
Abstract View on the Data Type \texttt{Real}

\texttt{realq :: Rat \rightarrow Real}
### Abstract View on the Data Type \( \text{Real} \)

<table>
<thead>
<tr>
<th>Function</th>
<th>Type Signature</th>
</tr>
</thead>
</table>
| \( \text{realq} \) | \( \text{Rat} \rightarrow \text{Real} \)
| \( \text{add} \) | \( \text{Real} \rightarrow \text{Real} \rightarrow \text{Real} \)
| \( \text{sub} \) | \( \text{Real} \rightarrow \text{Real} \rightarrow \text{Real} \)
| \( \text{neg} \) | \( \text{Real} \rightarrow \text{Real} \)
| \( \text{mul} \) | \( \text{Real} \rightarrow \text{Real} \rightarrow \text{Real} \)
| \( \text{power} \) | \( \text{Real} \rightarrow \text{Nat} \rightarrow \text{Real} \)
| \( \text{nthroot} \) | \( \text{Nat} \rightarrow \text{Real} \rightarrow \text{Real} \)
Abstract View on the Data Type **Real**

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>realq</code></td>
<td><code>:: Rat -&gt; Real</code></td>
</tr>
<tr>
<td><code>add</code></td>
<td><code>:: Real -&gt; Real -&gt; Real</code></td>
</tr>
<tr>
<td><code>sub</code></td>
<td><code>:: Real -&gt; Real -&gt; Real</code></td>
</tr>
<tr>
<td><code>neg</code></td>
<td><code>:: Real -&gt; Real</code></td>
</tr>
<tr>
<td><code>mul</code></td>
<td><code>:: Real -&gt; Real -&gt; Real</code></td>
</tr>
<tr>
<td><code>power</code></td>
<td><code>:: Real -&gt; Nat -&gt; Real</code></td>
</tr>
<tr>
<td><code>nthroot</code></td>
<td><code>:: Nat -&gt; Real -&gt; Real</code></td>
</tr>
<tr>
<td><code>le</code></td>
<td><code>:: Real -&gt; Real -&gt; Fuzzybool</code></td>
</tr>
<tr>
<td><code>leq</code></td>
<td><code>:: Real -&gt; Real -&gt; Fuzzybool</code></td>
</tr>
<tr>
<td><code>isPositive</code></td>
<td><code>:: Real -&gt; Fuzzybool</code></td>
</tr>
<tr>
<td><code>isZero</code></td>
<td><code>:: Real -&gt; Fuzzybool</code></td>
</tr>
</tbody>
</table>
Outline

1. Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on \( \mathbb{R} \)

2. An Abstract View on the Data Type \( \text{Real} \)

3. Auxiliary Types and Functions

4. Representing Real Numbers as Cauchy Sequences

5. Conclusions and further work
Auxiliary Types and Functions: Rational Numbers

data Rat = Rat Int Int

num :: Rat -> Int
denom :: Rat -> Int
norm :: Rat -> Rat
ratn :: Int -> Rat
ratf :: Int -> Int -> Rat

add :: Rat -> Rat -> Rat
sub :: Rat -> Rat -> Rat
mul :: Rat -> Rat -> Rat
neg :: Rat -> Rat

eq :: Rat -> Rat -> Bool
le :: Rat -> Rat -> Bool
leq :: Rat -> Rat -> Bool
Fuzzybool - result type of e.g. comparing two reals for equality

\[
\text{eq } x \ y = \text{Fuzzy } f
\]

- \text{f: Rat -> Bool}
- nondeterministic function
- depending on precision: \text{f } r \text{ may yield true, false, or both}
Fuzzybool - result type of e.g. comparing two reals for equality

\[
\text{eq } x \ y = \text{Fuzzy } f
\]

- \( f: \text{Rat} \rightarrow \text{Bool} \)
- nondeterministic function
- depending on precision: \( f \ r \) may yield \text{true, false}, or both

\[
\text{data Fuzzybool} = \text{Fuzzy} ( \text{Rat} \rightarrow \text{Bool} )
\]

\[
\text{defuzzy} :: \text{Fuzzybool} \rightarrow \text{Rat} \rightarrow \text{Bool}
\]

\[
\text{defuzzy} (\text{Fuzzy } f) \ r = f \ r
\]
**Auxiliary Types and Functions: Fuzzybool**

\[
\text{andf} :: \text{Fuzzybool} \rightarrow \text{Fuzzybool} \rightarrow \text{Fuzzybool} \\
\text{andf}\ a\ b = \text{Fuzzy} (\backslash r \rightarrow (\text{defuzzy}\ r\ a) \&\& (\text{defuzzy}\ r\ b))
\]

\[
\text{orf} :: \text{Fuzzybool} \rightarrow \text{Fuzzybool} \rightarrow \text{Fuzzybool} \\
\text{orf}\ a\ b = \text{Fuzzy} (\backslash r \rightarrow (\text{defuzzy}\ r\ a) \mid\| (\text{defuzzy}\ r\ b))
\]

\[
\text{notf} :: \text{Fuzzybool} \rightarrow \text{Fuzzybool} \\
\text{notf}\ a = \text{Fuzzy} (\backslash r \rightarrow \text{not} (\text{defuzzy}\ r\ a))
\]
Auxiliary Types and Functions: Intervals

data Interval = Interval Rat Rat
lower :: Interval -> Rat
upper :: Interval -> Rat
Auxiliary Types and Functions: Intervals

```
data Interval = Interval Rat Rat
lower :: Interval -> Rat
upper :: Interval -> Rat
```

- `isZero` yields `true` if 0 is in the interval
- `isZero` yields `false` if some x not equal to 0 is in the interval

```
isZero :: Interval -> Bool
isZero arg | q.leq (lower arg) (ratn 0) && q.leq (ratn 0) (upper arg) = True
isZero arg | q.le (lower arg) (ratn 0) || q.le (ratn 0) (upper arg) = False
```
Auxiliary Types and Functions: Intervals

\[
\text{data } \text{Interval} = \text{Interval} \ \text{Rat} \ \text{Rat}
\]

\[
\text{lower} :: \text{Interval} \rightarrow \text{Rat}
\]

\[
\text{upper} :: \text{Interval} \rightarrow \text{Rat}
\]

- **isZero** yields *true* if 0 is in the interval
- **isZero** yields *false* if some \( x \) not equal to 0 is in the interval

\[
\text{isZero} :: \text{Interval} \rightarrow \text{Bool}
\]

\[
\text{isZero arg} | \ q\. \text{leq} (\text{lower arg}) (\text{rat n 0}) \land \ q\. \text{leq} (\text{rat n 0}) (\text{upper arg}) = \text{True}
\]

\[
\text{isZero arg} | \ q\. \text{le} (\text{lower arg}) (\text{rat n 0}) \lor \ q\. \text{le} (\text{rat n 0}) (\text{upper arg}) = \text{False}
\]

- **isPositive** yields *true* if interval contains a positive number
- **isPositive** yields *false* if interval contains a non-positive number

\[
\text{isPositive} :: \text{Interval} \rightarrow \text{Bool}
\]

\[
\text{isPositive arg} | \ q\. \text{le} (\text{rat n 0}) (\text{upper arg}) = \text{True}
\]

\[
\text{isPositive arg} | \ q\. \text{leq} (\text{lower arg}) (\text{rat n 0}) = \text{False}
\]
Outline

1 Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on \( \mathbb{R} \)

2 An Abstract View on the Data Type \( \text{Real} \)

3 Auxiliary Types and Functions

4 Representing Real Numbers as Cauchy Sequences

5 Conclusions and further work
Real Numbers as Cauchy Sequences

data Real :: Cauchy (Int \rightarrow Rat)

realq :: Rat \rightarrow Real
realq a = (Cauchy (\_ \rightarrow a))
data Real :: Cauchy (Int -> Rat)

realq :: Rat -> Real
realq a = (Cauchy (\_ -> a))

add :: Real -> Real -> Real
add a b = Cauchy (\k -> let m=k+1 in q.add (get a m) (get b m))

sub :: Real -> Real -> Real
sub a b = add a (neg b)

neg :: Real -> Real
neg a = Cauchy (\k -> q.neg (get a m))

get :: Real -> Int -> Rat
get (Cauchy x) k = x k

Similar for multiplication and other functions; determine look-ahead
eq :: Real -> Real -> Fuzzybool
eq x y = isZero (sub y x)

le :: Real -> Real -> Fuzzybool
le x y = isPositive (sub y x)

leq :: Real -> Real -> Fuzzybool
leq x y = (f.nof . isPositive) (sub x y)

isZero and isPositive reduced to corresponding functions on intervals:

isPositive :: Real -> Fuzzybool
isPositive x = f.fuzzy (\r -> i.isPositive (toInterval r x))

isZero :: Real -> Fuzzybool
isZero x = f.fuzzy (\r -> i.isZero (toInterval r x))

function yielding interval realizing any given precision with respect to the given x of type Real.
Given \( p \) of type \( \text{Rat} \) and \( x \) of type \( \text{Real} \):

toInterval determines an interval containing \( \tilde{x} \in \mathbb{R} \) represented by \( x \) and approximating \( \tilde{x} \) with precision \( p \).

\[
\begin{align*}
toInterval & :: \text{Rat} \to \text{Real} \to \text{Interval} \\
toInterval p x &= \text{let } y = \text{approx } p x \text{ in} \\
& \quad \text{interval } (q.\text{sub} y p) (q.\text{add} y p) \\
\text{approx} & :: \text{Rat} \to \text{Real} \to \text{Rat} \\
\text{approx } p x &= \text{get } x (\text{prec } p) \\
\text{prec} & :: \text{Rat} \to \text{Int} \\
\text{prec } x \mid q.\text{le} (\text{ratn } 0) x &= \text{minexp } q.\text{leq } x (\text{ratf } 1 2) \\
\end{align*}
\]

\( \text{approx } p x \) approximates \( \tilde{x} \) with precision \( p \).
Example: Square Root

\[ x_0 = 2 \]
\[ x_{k+1} = \frac{1}{2} \left( x_n + \frac{2}{x_k} \right) \]

has the limit

\[ \lim_{k \to \infty} x_k = \sqrt{2}. \]

```wflp
sqrt2 :: Real
sqrt2 = Cauchy (\k -> sqrt2sub (ratf 0 1) (ratf 2 1) (q.power (ratf 1 2) k))

sqrt2sub :: Rat -> Rat -> Rat -> Rat
sqrt2sub x1 x2 e =
  let u = q.max x1 x2
      l = q.min x1 x2
  in if q.leq (q.sub u l) e then x2
     else sqrt2sub x2 (q.mul (ratf 1 2) (q.add x2 (q.dvd (ratf 2 1) x2))) e
```
Example: Decimal Representation

\[ \text{dec} :: \text{Real} \rightarrow \text{Int} \rightarrow \text{String} \]

\[ \text{dec} \ x \ k \]

returns value of \( \tilde{x} \) as a string containing \( k \) decimal places
(no rounding)

\[ \text{real} \gg \text{dec} \ \text{sqrt2} \ 10 \]
Result: "1,4142135623"
More Solutions? [Y(es) n(o) a(ll)]
Result: "1,4142135624"
More Solutions? [Y(es) n(o) a(ll)]
No more Solutions
Example: Decision Functions

\textit{sign} function on \( \mathbb{R} \)

\[
\text{sign}(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{if } x = 0 \\
-1 & \text{if } x < 0
\end{cases}
\]

not exactly computable
⇒ multi-function
⇒ nondeterministic function in Curry

With additional precision parameter \( p \):

\[
\begin{align*}
\text{sgn} &: \text{ Rat } \to \text{ Real } \to \text{ Int }
\text{sgn } p \ x \ | \ \text{defuzzy } p \ (r.\text{isPositive } x) = \text{ True } &= 1 \\
\text{sgn } p \ x \ | \ \text{defuzzy } p \ (r.\text{isZero } x) = \text{ True } &= 0 \\
\text{sgn } p \ x \ | \ \text{defuzzy } p \ (\lnot f (r.\text{isPositive } x)) = \text{ True } &= -1
\end{align*}
\]
1 Motivation and Background
   Computable Functions
   Type-2 Machines
   Type-2 Machines for Functions on \( \mathbb{R} \)

2 An Abstract View on the Data Type \( \mathbb{R} \)

3 Auxiliary Types and Functions

4 Representing Real Numbers as Cauchy Sequences

5 Conclusions and further work
Conclusions and further work

- Type-2 Theory of Effectivity (TTE) [Weihrauch 2000]
  - computation on infinite objects
  - multi-functions
- exact real arithmetic in Curry based on TTE
- high-level declarative approach using features of Curry
  - functional concept
  - lazy evaluation
  - non-determinism
- implemented system
  - rich set of functions (including exp, log, \( \ln \), sin, cos, \ldots)
  - alternative representations (Cauchy sequences, Cauchy sequences with rounding, intervals)

- applications
- efficiency
- complexity issue
- \ldots