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

Outline

Computable Functions

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

Computable Functions

(Type-1) Computability Theory

- (partial) functions over finite words:

$$f : \Sigma^* \to \Sigma^*$$

- computable function given by Turing machine
- computability on other sets M
  (e.g., rational numbers, graphs, ...)
  - use words as names or codes of elements of M
  - interpret words computed by Turing machine as elements of M

Computable Functions

- real numbers can not be represented by finite words

  $\pi = 3.14159\ldots$

Computable Functions

- 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:

    $$f : \Sigma^\omega \to \Sigma^\omega$$

  - computable function given by machine transforming infinite sequences to infinte 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

Type-2 Machine

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

Type-2 Machine

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

Type-2 Machine

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

Type-2 Machine

Definition (computable function)

$$f :\subseteq Y_1 \times \ldots \times Y_k \to Y_0$$

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

Type-2 Machine

Definition (computable function)

$$f :\subseteq Y_1 \times \ldots \times Y_k \to Y_0$$

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

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

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

Example (addition in decimal representation)

$$\text{Inputs:} \quad y_1 = 0.6666666666\ldots$$
$$y_2 = 0.3333333333\ldots$$

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

Example (addition in decimal representation)

$$\text{Inputs:} \quad y_1 = 0.6666666666\ldots$$
$$y_2 = 0.3333333333\ldots$$

After reading finitely many input symbols, M must write either

$$0. \qquad \text{or} \qquad 1.$$

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

Example (addition in decimal representation)

$$\text{Inputs:}\quad y_1 = 0.6666666666\ldots$$
$$y_2 = 0.3333333333\ldots$$

After reading finitely many input symbols, M must write either

$$0. \qquad \text{or} \qquad 1.$$

$\Rightarrow$ may be wrong depending on next input symbol

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

Example (addition in decimal representation)

$$\text{Inputs:} \quad y_1 = 0.6666666666\ldots$$
$$y_2 = 0.3333333333\ldots$$

After reading finitely many input symbols, M must write either

0. or 1.

$\Rightarrow$ may be wrong depending on next input symbol

$\Rightarrow$ there is no Type-2 machine computing addition on $\mathbb{R}$ and using decimal representation

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

Better names for elements of $\mathbb{R}$

- $x \in \mathbb{R}$

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

Better names for elements of $\mathbb{R}$

- $x \in \mathbb{R}$
- quickly converging Cauchy sequence of rational numbers

  $$r_o, r_1, r_2, \ldots$$

  with

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

  and

  $$|r_k - x| \leqslant 2^{-k}$$

Type-2 Machines for $\mathbb{R}$: Computing functions

Example (addition using Cauchy sequences as names)

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

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

Type-2 Machines for $\mathbb{R}$: Computing functions

Example (addition using Cauchy sequences as names)

Addition

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

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

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

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

- 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

Type-2 Machines for $\mathbb{R}$: Computing functions

functions on $\mathbb{R}$ not computable in TTE:

$$x = y$$

$$x \leqslant y$$

$$x \geqslant y$$

Type-2 Machines for $\mathbb{R}$: Computing functions

- finite initial part of name $r_o, r_1, r_2, \ldots$ 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

$$eq : \mathbb{R} \times \mathbb{R} \rightrightarrows Bool$$
$$le : \mathbb{R} \times \mathbb{R} \rightrightarrows Bool$$

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

Abstract View on the Data Type Real

```
realq :: Rat -> Real
```

Abstract View on the Data Type Real

```
realq :: Rat -> Real
```

```
add     :: Real -> Real -> Real
sub     :: Real -> Real -> Real
neg     :: Real -> Real
mul     :: Real -> Real -> Real
power   :: Real -> Nat  -> Real
nthroot :: Nat  -> Real -> Real
```

Abstract View on the Data Type `Real`

```
realq :: Rat -> Real
```

```
add     :: Real -> Real -> Real
sub     :: Real -> Real -> Real
neg     :: Real -> Real
mul     :: Real -> Real -> Real
power   :: Real -> Nat  -> Real
nthroot :: Nat  -> Real -> Real
```

```
le         :: Real -> Real -> Fuzzybool
leq        :: Real -> Real -> Fuzzybool
isPositive :: Real -> Fuzzybool
isZero     :: Real -> Fuzzybool
```

Outline

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
```

Auxiliary Types and Functions: Fuzzybool

Fuzzybool - result type of e.g. comparing two reals for equality

```
eq x y = Fuzzy f
```

- `f: Rat -> Bool`
- nondeterministic function
- depending on precision: `f r` may yield *true*, *false*, or both

Auxiliary Types and Functions: Fuzzybool

Fuzzybool - result type of e.g. comparing two reals for equality

```
eq x y = Fuzzy f
```

- `f: Rat -> Bool`
- nondeterministic function
- depending on precision: `f r` may yield *true*, *false*, or both

```
data Fuzzybool = Fuzzy (Rat -> Bool)

defuzzy ::  Fuzzybool -> Rat -> Bool
defuzzy (Fuzzy f) r = f r
```

Auxiliary Types and Functions: Fuzzybool

```
andf :: Fuzzybool -> Fuzzybool -> Fuzzybool
andf a b  = Fuzzy (\r -> (defuzzy r a) && (defuzzy r b))

orf :: Fuzzybool -> Fuzzybool -> Fuzzybool
orf a b  = Fuzzy (\r -> (defuzzy r a) || (defuzzy r b))

notf :: Fuzzybool -> Fuzzybool
notf a  = Fuzzy (\r -> not (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

```
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
```

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

```
isPositive :: Interval -> Bool
isPositive arg | q.le (ratn 0) (upper arg) = True
isPositive arg | q.leq (lower arg) (ratn 0) = False
```

Outline

Real Numbers as Cauchy Sequences

```
data Real :: Cauchy (Int -> Rat)
```

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

Real Numbers as Cauchy Sequences

```
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

Real Numbers as Cauchy Sequences

```
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.notf . 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
isPositive 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`.

Real Numbers as Cauchy Sequences

Given p of type `Rat` and x of type `Real`:

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

```
toInterval :: Rat -> Real -> Interval
toInterval p x = let y = approx p x in
    interval (q.sub y p) (q.add y p)

approx :: Rat -> Real -> Rat
approx p x = get x (prec p)

prec :: Rat -> Int
prec x | q.le (ratn 0) x = minexp q.leq x (ratf 1 2)
```

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

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

```
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

```
dec :: Real -> Int -> String
```

dec x k

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

```
real> dec 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

sign function on $\mathbb{R}$

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

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

With additional precision parameter p:

```
sgn :: Rat -> Real -> Int
sgn p x | defuzzy p (r.isPositive x) == True = 1
sgn p x | defuzzy p (r.isZero x) == True = 0
sgn p x | defuzzy p (notf (r.isPositive x)) == True = -1
```

Outline

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, ...)
  - alternative representations (Cauchy sequences, Cauchy sequences with rounding, intervals)

- applications
- efficiency
- complexity issue
- ...