

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	ooooo	oooo	o	
		oooooooooooo		

ERCE: Exact Real Calculator for Everyone

The 18th Asian Technology Conference in Mathematics
Indian Institute of Technology Bombay, Mumbai, India

Weng Kin Ho
wengkin.ho@nie.edu.sg

National Institute of Education, Nanyang Technological University

9 December 2013

Contents

Introduction

Existing problems

Case study 1

Case study 2

Proposed solutions

New challenges

Preliminaries

Finite character of computers

Signed bit representation

Recursive paradigm

Design and implementation

Architecture

Functional programs

Parser

Evaluator

User-interface

Sample applications

Logistic map

Demonstration

Comparative analysis

Conclusion

Disaster

A popular National Geographic documentary ...



Decimals from disaster: Case 1



Figure : Ariane 5 Launcher Failure – 4 June 1996

http://alturl.com/254eg

Decimals from disaster: Case 1

- The horizontal velocity of the rocket relative to the launching platform was code-related to a 64 bit floating point number M .

Decimals from disaster: Case 1

- The horizontal velocity of the rocket relative to the launching platform was code-related to a 64 bit floating point number M .
- $C : M \mapsto K$, where K is a 16 bit signed integer.

Decimals from disaster: Case 1

- The horizontal velocity of the rocket relative to the launching platform was code-related to a 64 bit floating point number M .
- $C : M \mapsto K$, where K is a 16 bit signed integer.
- $K > 2^{16-1} + 1 = 32,767$, the largest integer that can be stored in a 16 bit signed integer.

Decimals from disaster: Case 1

- The horizontal velocity of the rocket relative to the launching platform was code-related to a 64 bit floating point number M .
- $C : M \mapsto K$, where K is a 16 bit signed integer.
- $K > 2^{16-1} + 1 = 32,767$, the largest integer that can be stored in a 16 bit signed integer.
- The conversion C failed.

Decimals from disaster: Case 1

Decimals from disaster: Case 1

- This exception then triggered a system diagnostic, dumping its debugging data into an area of memory being used by the programs guiding the rockets motors.

```

ooo●oooooooooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
ooooo

```

```

o
o
o

```

Decimals from disaster: Case 1

- This exception then triggered a system diagnostic, dumping its debugging data into an area of memory being used by the programs guiding the rockets motors.
- Concurrently, control was switched to a backup computer that (unfortunately) had the same data.

Decimals from disaster: Case 1

Decimals from disaster: Case 1

- The system then misinterpreted this switch as necessitating strong corrective action.

Decimals from disaster: Case 1

- The system then misinterpreted this switch as necessitating strong corrective action.
- The rockets motors then swiveled to the extreme positions of their mountings.

```

○○○○●○○○○○○○○○○○○○○○○○○○○
○○○○
○○

```

```

○○○○○○○○○○
○○○○○○○○○○
○○○○

```

```

○
○○○○○○○○○○○○○○
○○○○○○○○○○
○○○○
○○○○○○○○

```

```

○
○
○
○

```

Decimals from disaster: Case 1

- The system then misinterpreted this switch as necessitating strong corrective action.
- The rockets motors then swiveled to the extreme positions of their mountings.
- Disaster followed.

Cause of disaster


```

○○○○●○○○○○○○○○○○○○○○○○○○○
○○○○
○○

```

```

○○○○○○○○○○
○○○○

```

```

○
○○○○○○○○○○○○○○
○○○○○○○○
○○○○
○○○○○○○○

```

```

○
○
○

```

Decimals from disaster: Case 1

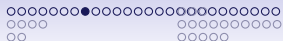
Cause of disaster

- *Realizability*
- *Program correctness*

Decimals from disaster: Case 2



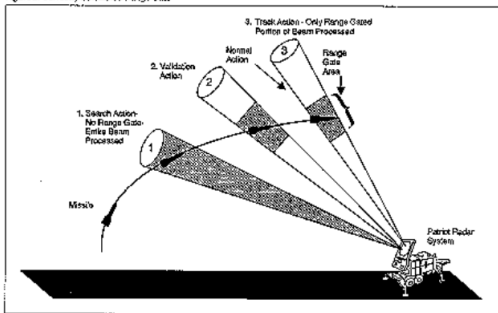
Figure : Patriot Missile Failure – 25 February 1991

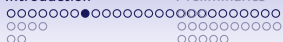


Decimals from disaster: Case 2

Phased Array Tracking Intercepting Of Target

Figure 3: Correctly Calculated Range Gate

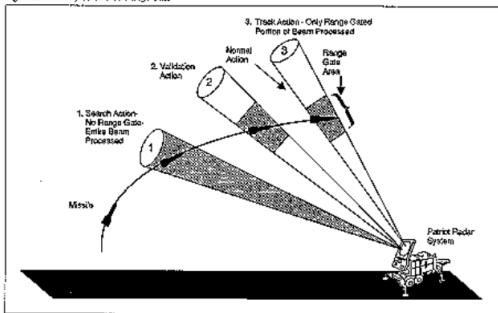




Decimals from disaster: Case 2

Phased Array Tracking Intercepting Of Target

Figure 3: Correctly Calculated Range Gate



- The program was intended to calculate the next location of the incoming target by the range gate.

oooooooo●oooooooooooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
oooo

o
oooooooooooooooo
oooooooooooo
oooo
oooooooo

o
o
o

Decimals from disaster: Case 2

oooooooo●oooooooooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo
 ooooooooooooo

o
 o
 o

Decimals from disaster: Case 2

- The prediction is calculated based in the target's velocity and the time of the last radar detection.

oooooooo●oooooooooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo
 oooooooo

o
 o
 o

Decimals from disaster: Case 2

- The prediction is calculated based in the target's velocity and the time of the last radar detection.
- The bug came into the calculation of the next location of the incoming target.

oooooooo●oooooooooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo
 oooooooo

o
 o
 o

Decimals from disaster: Case 2

- The prediction is calculated based in the target's velocity and the time of the last radar detection.
- The bug came into the calculation of the next location of the incoming target.

oooooooo●oooooooooooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
oooo

o
oooooooooooooooo
oooooooooooo
oooo
oooooooo

o
o
o

Decimals from disaster: Case 2

```

oooooooo●oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
o

```

Decimals from disaster: Case 2

- Velocity v is stored as a whole number and a decimal, and time t is an integer (i.e., the longer the system has been running, the larger the value) measured in $\frac{1}{10}$ second.

```

oooooooo●oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
o

```

Decimals from disaster: Case 2

- Velocity v is stored as a whole number and a decimal, and time t is an integer (i.e., the longer the system has been running, the larger the value) measured in $\frac{1}{10}$ second.
- The algorithm used to predict the next air space to scan by the radar requires that both velocity and time be expressed as real numbers.

oooooooo●oooooooooooooooo
oooo
oo

oooooooooooooooo
oooooooooooo
oooo

o
oooooooooooooooo
oooooooooooo
oooo
oooooooo

o
o
o

Decimals from disaster: Case 2

```

oooooooooooo●oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
oooooooo

```

```

o
o
o

```

Decimals from disaster: Case 2

- The Patriots computer only has 24 bit fixed-point registers.

```

oooooooooooo●oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Decimals from disaster: Case 2

- The Patriots computer only has 24 bit fixed-point registers.
- Because time t was measured as the number of tenth-seconds, the value $\frac{1}{10}$, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point, i.e.,

$$0.0001\dot{1} \approx 0.\underbrace{000110011001100110011001}_{24 \text{ bits}}$$


```

oooooooooooo●oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
ooooo
oooooooooooo

```

```

o
o
o

```

Decimals from disaster: Case 2

- The Patriots computer only has 24 bit fixed-point registers.
- Because time t was measured as the number of tenth-seconds, the value $\frac{1}{10}$, which has a non-terminating binary expansion, was chopped at 24 bits after the radix point, i.e.,

$$0.0001\bar{1} \approx 0.\underbrace{000110011001100110011001}_{24 \text{ bits}}$$

- The error in precision grows as the time value increases, and the inaccuracy resulting from this is directly proportional to the targets velocity.



Decimals from disaster: Case 2

Figure 4: Calculated Range Gate After Approximately 8 Hours

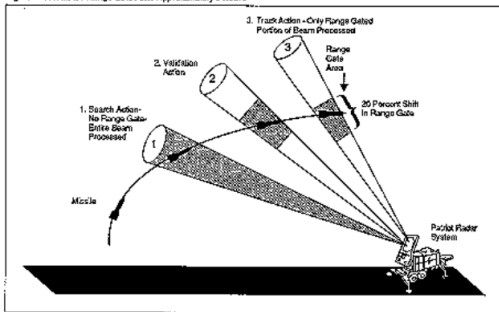
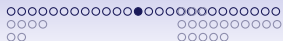


Figure : 20% shift in range gate calculation after 8 consecutive hours of operation



Decimals from disaster: Case 2

Figure 5: Incorrectly Calculated Range Gate

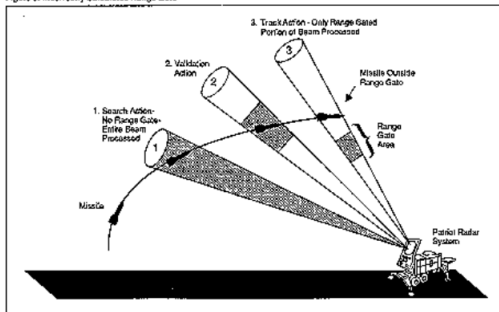


Figure : After 20 consecutive hours, the target is no longer in the range gate area.

oooooooooooooooo●ooooooooooooo
 oooo
 oo

ooooooooooooo
 ooooooooooo
 ooooo

o
 oooooooooooooo
 oooooooooo
 oooo
 oooooooooo

o
 o
 o

Decimals from disaster: Case 2

Let us watch an animation:

<http://www.youtube.com/watch?v=EMVBLg2MrLs>

oooooooooooooooo●ooooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
ooooo

o
oooooooooooooooo
oooooooooooo
oooo
oooooooooooo

o
o
o

Decimals from disaster: Case 2

Cause of disaster

○○○○○○○○○○○○○○●○○○○○○○○○○
 ○○○○
 ○○

○○○○○○○○○○
 ○○○○○○○○
 ○○○○

○
 ○○○○○○○○○○○○
 ○○○○○○○○
 ○○○
 ○○○○○○○○

○
 ○
 ○

Decimals from disaster: Case 2

Cause of disaster

- *Representation of reals*

○○○○○○○○○○○○○○○○●○○○○○○○○○○
 ○○○○ ○○○○○○○○○
 ○○ ○○○○

○
 ○○○○○○○○○○○○
 ○○○○○○○○
 ○○○
 ○○○○○○○○

○
 ○
 ○

Decimals from disaster: Case 2

Cause of disaster

- *Representation of reals*
- *Truncation errors*

oooooooooooooooo●oooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
oooo

o
oooooooooooooooo
oooooooooooo
oooo
oooooooo

o
o
o

Decimals from disaster

oooooooooooooooo●oooooooooooo
 oooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo
 ooooo

o
 o
 o

Decimals from disaster

What is the lesson learnt?

There is an urgent need to calculate **real** numbers **exactly**!

○○○○○○○○○○○○○○○○○○○○●○○○○○○○○○○
 ○○○○
 ○○

○○○○○○○○○○
 ○○○○○○○○○
 ○○○○

○
 ○○○○○○○○○○○○○
 ○○○○○○○○
 ○○○
 ○○○○○○○○

○
 ○
 ○

Floating point system

Definition (FPS)

IEEE 754 single-precision *floating point* is encoded in 32 bits using 1 bit for the sign s , 8 bits for the exponent e and 23 bits for the normalised mantissa m without the leading 1 so that a ‘real’ number can be written in the form

$$(-1)^s \cdot 2^{e-127} (1 : m).$$

Defects of FPS

oooooooooooooooooooo●oooooooo
oooo
oo

oooooooooooo
oooo

o
oooooooooooo
oooooooo
oooo
ooooooo

o
o
o

Defects of FPS

- Truncation errors

oooooooooooooooooooo●oooooooo
oooo
oo

oooooooooooo●oooooooo
oooooooooooo
oooo

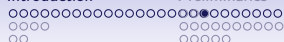
o
oooooooooooooooo
oooooooooooo
oooo
oooo

o
o
o

Defects of FPS

- Truncation errors
- Realizability issues

There are more problematic issues of computability and realizability.



Defects of FPS

There are more problematic issues of computability and realizability.

Theorem

The scaling function $f : \mathbb{R} \longrightarrow \mathbb{R}$ defined by

$$x \mapsto 9x$$

is not realizable.

Defects of FPS

Proof.

oooooooooooooooooooo●oooooooo
 oooo
 oooo
 oo

oooooooooooo
 oooooooooo
 ooooo

o
 ooooooooooooo
 oooooooooo
 oooo
 oooooooo

o
 o
 o

Defects of FPS

Proof.

For simplicity, just consider base 10.

oooooooooooooooooooo●oooooooo
 oooo
 oo

oooooooooooo
 ooooo

o
 ooooooooooooo
 oooooooooo
 oooo
 oooooooooo

o
 o
 o

Defects of FPS

Proof.

For simplicity, just consider base 10.

Suppose that f is realizable by a machine.

Defects of FPS

Proof.

Proof.

We now consider one potential input form for $\frac{1}{9}$:

$$0.\dot{1} := 0.111\dots$$

Proof.

We now consider one potential input form for $\frac{1}{9}$:

$$0.\dot{1} := 0.111\dots$$

How does a machine output the real number 1?

oooooooooooooooooooooooooooo●oooo
 oooo
 oo

oooooooooooo●oooo
 oooooooooo
 ooooo

o
 oooooooooooooo
 oooooooooo
 oooo
 oooooooooo

o
 o
 o

Defects of FPS

Proof.

There are two possibilities:

$$1.\dot{0} := 1.000\dots$$

oooooooooooooooooooooooooooo●oooo
 oooo
 oo

oooooooooooo
 ooooo

o
 oooooooooooooo
 oooooooooo
 oooo
 oooooooooo

o
 o
 o

Defects of FPS

Proof.

There are two possibilities:

$$1.\dot{0} := 1.000\dots$$

or

$$0.\dot{9} := 0.999\dots$$

```

oooooooooooooooooooo●oooo
oooo
oo

```

```

oooooooooooo
oooo

```

```

o
ooooooooooooo
oooooooooooo
oooo
ooooooo

```

```

o
o
o

```

Defects of FPS

Proof.

Suppose the machine chooses the first option for output:

1.000

Suppose the machine chooses the first option for output:

1.000 . . .

0.111...

must the machine read before it decides to print the first digit 1?

```

oooooooooooooooooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo●oo
oooooooooooo
ooooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
ooooooo

```

```

o
o
o

```

Defects of FPS

Proof.

That the machine can print 1 as the first digit implies

$$f(x) \geq 1$$

where the input real number is x .

$$9x \geq 1 \implies x \geq \frac{1}{9}.$$

Thus,

$$9x \geq 1 \implies x \geq \frac{1}{9}.$$

```

ooooooooooooooooooooo
oooooooooooo●o
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
ooooooooooooooooo
oooooooooooo
oooo
ooooo
oooooooooooo

```

```

o
o
o
o

```

Defects of FPS

Proof.

Thus,

$$9x \geq 1 \implies x \geq \frac{1}{9}.$$

Since the input decimal stream representing x is

$$0.\dot{1} := 0.111\dots$$

and since it is impossible to read infinitely many digits before printing, it must be that there is a digit $k \geq 2$ appearing somewhere at some point – which is impossible granted that the input $x = \frac{1}{9}$.

Defects of FPS

Proof.

How about printing the output as

0.999...?

Defects of FPS

Proof.

How about printing the output as

0.999...?

The same problem happens!



Alternative representations of reals

Alternative representations of reals

1. Signed-digit streams

Alternative representations of reals

1. Signed-digit streams
2. Cauchy sequence with fixed rate of convergence

Alternative representations of reals

1. Signed-digit streams
2. Cauchy sequence with fixed rate of convergence
3. Intervals with rational endpoints

Alternative representations of reals

1. Signed-digit streams
2. Cauchy sequence with fixed rate of convergence
3. Intervals with rational endpoints
4. Continued fractions

Alternative representations of reals

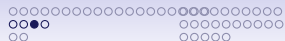
1. Signed-digit streams
2. Cauchy sequence with fixed rate of convergence
3. Intervals with rational endpoints
4. Continued fractions
5. Linear fractional transformations



ERA techniques

Definition (ERA)

Exact Real Arithmetic (ERA, for short) refers to the science of computing real numbers up to arbitrary precision.



ERA techniques

ERA techniques are based on the following theories:



ERA techniques

ERA techniques are based on the following theories:

1. Alternative representations of reals



ERA techniques

ERA techniques are based on the following theories:

1. Alternative representations of reals
2. Recursive data structures



ERA techniques

ERA techniques are based on the following theories:

1. Alternative representations of reals
2. Recursive data structures
3. Functional programming



ERA techniques

ERA techniques are based on the following theories:

1. Alternative representations of reals
2. Recursive data structures
3. Functional programming
4. Topology and domain theory



ERA techniques

ERA techniques are based on the following theories:

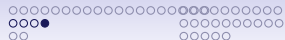
1. Alternative representations of reals
2. Recursive data structures
3. Functional programming
4. Topology and domain theory
5. Real analysis and computable analysis



ERA techniques

ERA techniques are based on the following theories:

1. Alternative representations of reals
2. Recursive data structures
3. Functional programming
4. Topology and domain theory
5. Real analysis and computable analysis
6. Programming semantics



ERA techniques

There are several implementations of ERA:



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality
3. iRRAM



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality
3. iRRAM
4. RealLib



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality
3. iRRAM
4. RealLib
5. xrc: Exact Real in C



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality
3. iRRAM
4. RealLib
5. xrc: Exact Real in C
6. HERA: Haskell Exact Real Arithmetic



ERA techniques

There are several implementations of ERA:

1. D. Plume and M. Escardó's Real Exact Calculator
2. A. Bauer and P. Taylor's use of Dedekind cuts and Abstract Stone Duality
3. iRRAM
4. RealLib
5. xrc: Exact Real in C
6. HERA: Haskell Exact Real Arithmetic
7. RZ

Theory-Practice gap

Drawbacks of available implementations:

Theory-Practice gap

Drawbacks of available implementations:

- ## 1. Heavy mathematical overhead

Theory-Practice gap

Drawbacks of available implementations:

1. Heavy mathematical overhead
2. Need knowledge in functional programming

Theory-Practice gap

Drawbacks of available implementations:

1. Heavy mathematical overhead
2. Need knowledge in functional programming
3. Need knowledge in exact real arithmetic

Theory-Practice gap

Drawbacks of available implementations:

1. Heavy mathematical overhead
2. Need knowledge in functional programming
3. Need knowledge in exact real arithmetic
4. Lack of user-friendly interface

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

has the following advantages:

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

has the following advantages: Users

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

has the following advantages: Users

1. need only to know a hand-held calculator is used,

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

has the following advantages: Users

1. need only to know a hand-held calculator is used,
2. can input one-variable functions $f : \mathbb{R} \rightarrow \mathbb{R}$, and

Bridging the Theory-Practice gap

With the aim of bridging this theory-practice gap, the calculator

ERCE = **E**xact **R**eal **C**alculator for **E**veryone

has the following advantages: Users

1. need only to know a hand-held calculator is used,
2. can input one-variable functions $f : \mathbb{R} \rightarrow \mathbb{R}$, and
3. can perform Exact Real Arithmetic.

```

oooooooooooooooooooo●ooo
oooooooooooooooooooo
oooo
oo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Computability as continuity

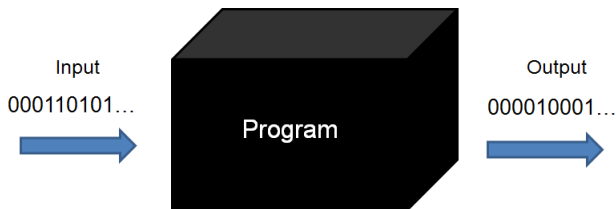


Figure : Continuity of program

Every *finite* part of the output depends only on a *finite* part of the input.

```

oooooooooooooooooooo●●oooooooo
oooooooooooooooooooo
oooo
oo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Computability as continuity

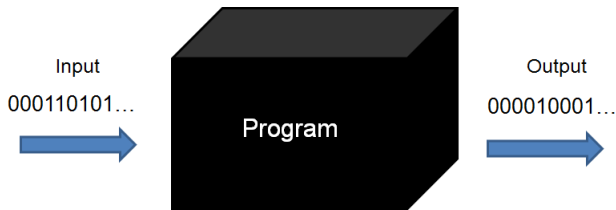


Figure : Continuity of program

$$\forall m \in \mathbb{N}. \exists n \in \mathbb{N}. x =_n x' \implies f(x) =_m f(x').$$

oooooooooooooooooooo●oooooooo
 ooooo●oooooooooooo
 ooooo

o
 oooooooooooooooooo
 oooooooooo
 ooooo
 ooooo

o
 o
 o

Look-aheads

Fact

A working program can only perform finite look-aheads.

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
●oooooooooooooooooooo
oooooo

```

```

o
oooooooooooooooooooo
oooooooooooooooooooo
oooo
oooooo

```

```

o
o
o

```

Signed bit streams

The signed-digit representation of real numbers in the interval $[-1, 1]$ is defined by

$$q : \mathbf{3}^{\omega} \twoheadrightarrow [-1, 1], \quad a_0 a_1 \dots \mapsto \sum_{i=0}^{\infty} a_i 2^{-i-1},$$

where $a_i \in \mathbf{3} := \{\bar{1}, 0, 1\}$. Here $\bar{1} := -1$.



Partial real number

Definition (Partial real number)

A partial real number is a finite sequence of signed bits of the form

$$\vec{a}_n := a_0 a_1 a_2 \dots a_{n-1}$$

that represents a closed bounded interval

$$\left[\sum_{k=0}^n a_k 2^{-k-1} - 2^{-n}, \sum_{k=0}^n a_k 2^{-k-1} + 2^{-n} \right].$$

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
o●ooooooooooooooooo
ooooo

```

```

o
ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Nested intervals

A real number a is the limit of the sequence of partial numbers $(\vec{a}_n)_{n=0}^{\infty}$; in other words,

$$a \in \bigcap_{n=0}^{\infty} \left[\sum_{k=0}^n a_k 2^{-k-1} - 2^{-n}, \sum_{k=0}^n a_k 2^{-k-1} + 2^{-n} \right]$$

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo●ooooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooooooooooo
oooo
oooo
oooo
oooo

```

```

o
o
o
o

```

Interval domain

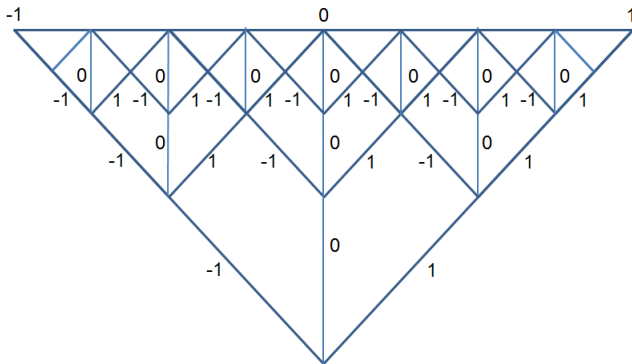
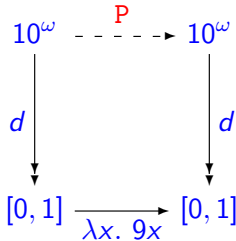
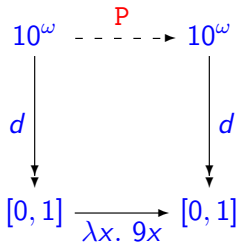


Figure : Interval domain \mathcal{I} : signed bit streams as paths





Solution to the realizability problem



No program P can realize the function $\lambda x. 9x$.

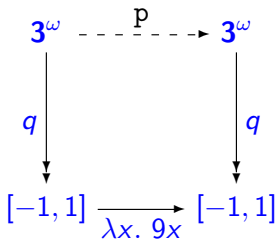
oooooooooooooooooooo
 oooooooooooooooooo
 ooooo
 oo

oooooooooooooooo
 ooooooooooooooooo
 ooooo●oooo
 ooooo

o
 oooooooooooooooooo
 ooooooooooooo
 oooo
 ooooo

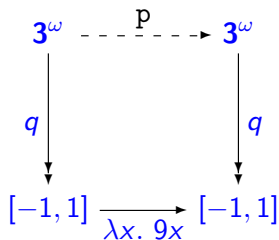
o
 o
 o

Solution to the realizability problem





Solution to the realizability problem



Beware: Conversion back is a little tricky!

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
oooooooo●oooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Solution to the realizability problem

Example

Revisit the function

$$f : x \mapsto 9x$$

acting on the input $x = \frac{1}{9}$.

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
oooooooooooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Solution to the realizability problem

Example

The input stream that represents $x = \frac{1}{9}$ is

$$0.000111000111000111\dots = 0.\dot{0}0011\dot{1}.$$

Solution to the realizability problem

Example

There are two extremes of the possible input that begins with
0.000111:

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Solution to the realizability problem

Example

There are two extremes of the possible input that begins with 0.000111:

- 0.0001111 which represents $\frac{1}{8}$, or


```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Solution to the realizability problem

Example

There are two extremes of the possible input that begins with 0.000111:

- 0.0001111̇ which represents $\frac{1}{8}$, or
- 0.0001111̇ which represents $\frac{3}{32}$.

Solution to the realizability problem

Example

The images of these two extremes that begin with 0.000111 are:

oooooooooooooooooooo
 oooooooooooooooooo
 ooooo
 oo

oooooooooooooooooooo
 oooooooooooooooooo
 ooooooooooooo●
 ooooo

o
 oooooooooooooooooo
 ooooooooooooo
 ooooo
 oooooooooo

o
 o
 o

Solution to the realizability problem

Example

The images of these two extremes that begin with 0.000111 are:

- $0.000111\dot{1} \equiv \frac{1}{8} \mapsto \frac{9}{8} \equiv 1.000\dot{1}$, or



Solution to the realizability problem

Example

The images of these two extremes that begin with 0.000111 are:

- $0.000111\dot{1} \equiv \frac{1}{8} \mapsto \frac{9}{8} \equiv 1.000\dot{1}$, or
- $0.000111\dot{\bar{1}} \equiv \frac{3}{32} \mapsto \frac{27}{32} \equiv 1.\bar{1}1010\dot{1}$.

oooooooooooooooooooo
 ooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooooooooooo●
 ooooo

o
 ooooooooooooo
 ooooooooooooo
 ooooo
 ooooo

o
 o
 o

Solution to the realizability problem

Example

The images of these two extremes that begin with 0.000111 are:

- $0.000111\dot{1} \equiv \frac{1}{8} \mapsto \frac{9}{8} \equiv 1.000\dot{1}$, or
- $0.000111\dot{\bar{1}} \equiv \frac{3}{32} \mapsto \frac{27}{32} \equiv 1.\bar{1}1010\dot{1}$.

Thus, after reading the first six digits 0.000111, the machine is ready to print the first digit 1 and then shift one radix point to the right.

Data types

Since we are working with the signed bit streams, we need a *data type* to store streams.

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
o●oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

List types

Let σ be a data type and define the list type $[\sigma]$ as follows:

$$[\sigma] := 1 + \sigma \times [\sigma].$$

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
o●oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo

```

```

o
o
o

```

List types

Let σ be a data type and define the list type $[\sigma]$ as follows:

$$[\sigma] := 1 + \sigma \times [\sigma].$$

Notice this is a **recursive** definition of data types.


```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
oo●ooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

List types

```

[] :: 1           -- void type
x  :: sigma      -- head
xs :: [sigma]     -- tail
(x:xs) :: [sigma] -- non-empty list

```

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
ooo●o

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o
o

```

Recursive definition of function

Functions are the first-class citizens of functional programming.

Example

ooooooooooooooooooooo
 oooooooooooooo
 ooooo
 oo

ooooooooooooo
 ooooooooooooo
 ooooo●o

o
 oooooooooooooo
 oooooooooo
 oooo
 oooooooo

o
 o
 o

Recursive definition of function

Functions are the first-class citizens of functional programming.

Example

```
repeat :: a -> [a]
repeat x = (x: repeat x)
```

ooooooooooooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
oooo●

o
ooooooooooooooooo
oooooooooooo
oooo
ooooooooo

o
o
o

Functional programming

We use the following data type to handle real numbers:

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo
oo

```

```

oooooooooooooooooooo
oooooooooooooooooooo
oooo●

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Functional programming

We use the following data type to handle real numbers:

```

type SD = Int      -- Signed bits. {-1,0,1}
type I  = [SD]     -- Reals in [-1,1]
type ME = (I,Int)  -- Reals in Mantissa-Exponent form
                  (m,e) := m x 2^e

```

oooooooooooooooooooo●oooooooooooo
oooo
oo

oooooooooooo
oooooooooooo
ooooo

●
oooooooooooooooo
oooooooooooo
oooo
oooooooooooo

o
o
o

Architecture of ERCE

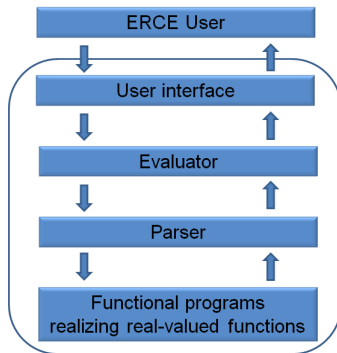


Figure : ERCE

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

○
 ●oooooooooooooooo○
 ooooooooo○
 oooo
 ooooooooo○

○
 ○
 ○

Arithmetic functions

Functional programs are written to perform

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

○
 ●oooooooooooooooo
 ooooooooooooo
 oooo
 ooooooooooooo

○
 ○
 ○

Arithmetic functions

Functional programs are written to perform

- Addition


```

oooooooooooooooooooo●oooooooooooo
ooooo                ooooooooooooo
oo                   ooooo
oo

```

```

o
o
●oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o
o

```

Arithmetic functions

Functional programs are written to perform

- Addition
- Multiplication

```

oooooooooooooooooooo●oooooooooooo
oooo●oooooooooooo
oo

```

```

○
○
●oooooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

○
○
○
○

```

Arithmetic functions

Functional programs are written to perform

- Addition
- Multiplication
- Subtraction

oooooooooooooooooooo●oooooooooooo
oooo●oooooooooooo
oo

○
●oooooooooooooooo
oooooooooooo
oooo
oooooooooooo

○
○
○

Arithmetic functions

Functional programs are written to perform

- Addition
- Multiplication
- Subtraction
- Division

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 o
 ●oooooooooooooooo
 ooooooooooooo
 ooooo
 ooooo

o
 o
 o

Arithmetic functions

Functional programs are written to perform

- Addition
- Multiplication
- Subtraction
- Division
- Integral exponentiation

oooooooooooooooooooo●oooooooo
 ooooo●oooooooooooo
 ooooo

○
 ●oooooooooooooooo
 oooooooooo
 oooo
 oooooooooo

○
 ○
 ○

Elementary functions

Standard series are exploited to compute

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

○
 ○●oooooooooooooooo
 ooooooooooooo
 ooooo
 ooooooooooooo

○
 ○
 ○

Elementary functions

Standard series are exploited to compute

- $\sin x$, $\cos x$, $\tan x$, $\exp x$

oooooooooooooooooooo
 oooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo
 ooooo
 ooooooooooooo

o
 o
 o

Elementary functions

Standard series are exploited to compute

- $\sin x, \cos x, \tan x, \exp x$
- $\sin^{-1}, \cos^{-1} x, \tan^{-1} x, \ln x$

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

o
 ●oooooooooooooooo
 oooooooooo
 oooo
 oooooooooo

o
 o
 o

Elementary functions

Standard series are exploited to compute

- $\sin x, \cos x, \tan x, \exp x$
- $\sin^{-1}, \cos^{-1} x, \tan^{-1} x, \ln x$
- \sqrt{x}


```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
o●ooooooooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

Elementary functions

Standard series are exploited to compute

- $\sin x, \cos x, \tan x, \exp x$
- $\sin^{-1}, \cos^{-1} x, \tan^{-1} x, \ln x$
- \sqrt{x}
- $\frac{1}{x}$

and so on for a real number x .

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 ooooo

○
 ○●oooooooooooo
 oooooooooo
 ooooo
 oooooooooo

○
 ○
 ○

Demonstration: factorial function

The functional program which we use is called

HASKELL.

Demonstration: factorial function

The functional program which we use is called

HASKELL.

This is a free ware which you can download at

<http://www.haskell.org/platform/>

Demonstration: factorial function

The functional program which we use is called

HASKELL.

This is a free ware which you can download at

<http://www.haskell.org/platform/>

The front-end editor can be as simple as

NOTEPAD.

Demonstration: factorial function

The functional program which we use is called

HASKELL.

This is a free ware which you can download at

<http://www.haskell.org/platform/>

The front-end editor can be as simple as

NOTEPAD.

We pause for a short demonstration.

Demonstration: addition

The addition function

`add_ME_ME :: ME -> ME -> ME`

takes two real numbers

$(m0, e0)$ and $(m1, e1)$

to produce their sum.

```

oooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooo●oooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Let $x = m_0 \times 2^{e_0}$ and $y = m_1 \times 2^{e_1}$, and assume that $e_0 \leq e_1$ so that $e := \max(e_0, e_1) = e_1$.

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
ooooooooooooooooooooo
ooooo

```

```

o
o
oooo●ooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Let $x = m_0 \times 2^{e_0}$ and $y = m_1 \times 2^{e_1}$, and assume that $e_0 \leq e_1$ so that $e := \max(e_0, e_1) = e_1$. We have

$$x + y = m_0 \times 2^{e_0} + m_1 \times 2^{e_1}$$


```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
ooooooooooooooooooooo
ooooo

```

```

o
o
oooo●ooooooooooooo
oooooooooooo
oooo
ooooo
oooooooooooo

```

```

o
o
o
o

```

Demonstration: addition

Let $x = m_0 \times 2^{e_0}$ and $y = m_1 \times 2^{e_1}$, and assume that $e_0 \leq e_1$ so that $e := \max(e_0, e_1) = e_1$. We have

$$\begin{aligned}
 x + y &= m_0 \times 2^{e_0} + m_1 \times 2^{e_1} \\
 &= \frac{m_0 \times 2^{e_0-e} + m_1}{2} \times 2^{e+1}
 \end{aligned}$$

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
ooooooooooooooooooooo
ooooo

```

```

o
oooo●ooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Let $x = m_0 \times 2^{e_0}$ and $y = m_1 \times 2^{e_1}$, and assume that $e_0 \leq e_1$ so that $e := \max(e_0, e_1) = e_1$. We have

$$\begin{aligned}
 x + y &= m_0 \times 2^{e_0} + m_1 \times 2^{e_1} \\
 &= \frac{m_0 \times 2^{e_0-e} + m_1}{2} \times 2^{e+1} \\
 &= (m_0 \times 2^{-(e-e_0)} \oplus m_1) \times 2^{e+1}
 \end{aligned}$$

```

ooooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooo●oooooooo
ooooooooo
oooo
ooooooooo

```

```

o
o
o

```

Demonstration: addition

The midpoint operator

$$\oplus : [-1, 1] \longrightarrow [-1, 1] \longrightarrow [-1, 1]$$

is realized by the composition

```
mid_I_I :: I -> I -> I
```

```
mid_I_I = div_I_2 . add2
```

```

ooooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
o
oooooooo●oooooooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

```
add2:: I -> I -> I2
```

```
add2 = zipWith (+)
```

where

```
type I2 = [Int] -- List of elements {-2,-1,0,1,2}
```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooo●ooooo
oooooooooooo
oooo
ooooo
oooooooooooo

```

```

o
o
o
o

```

Demonstration: addition

```

div_I_2 :: I2 -> I
div_I_2 (-2:      x) = -1 : div_I_2      x
div_I_2 (-1:(-2):x) = -1 : div_I_2 ( 0:x)
div_I_2 (-1:(-1):x) = -1 : div_I_2 ( 1:x)
div_I_2 (-1:  0 :x) =  0 : div_I_2 (-2:x)
div_I_2 (-1:  1 :x) =  0 : div_I_2 (-1:x)
div_I_2 (-1:  2 :x) =  0 : div_I_2 ( 0:x)
div_I_2 ( 0:      x) =  0 : div_I_2      x
div_I_2 ( 1:(-2):x) =  0 : div_I_2 ( 0:x)
div_I_2 ( 1:(-1):x) =  0 : div_I_2 ( 1:x)
div_I_2 ( 1:  0 :x) =  0 : div_I_2 ( 2:x)
div_I_2 ( 1:  1 :x) =  1 : div_I_2 (-1:x)
div_I_2 ( 1:  2 :x) =  1 : div_I_2 ( 0:x)
div_I_2 ( 2      :x) =  1 : div_I_2      x

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooo●oooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Zooming into the second line and denoting `div_I_2` by `f`, we have

$$\begin{aligned}
 \llbracket f(s) \rrbracket &= \frac{\llbracket (1 : -2 : x) \rrbracket}{2} \\
 &= \frac{(-1 \cdot 2^{-1} + (-2) \cdot 2^{-2} + \frac{1}{2^2} \llbracket x \rrbracket)}{2} \\
 &= \frac{(-2 \cdot 2^{-1} + \frac{1}{2^2} \llbracket x \rrbracket)}{2} \\
 &= -1 \cdot 2^{-1} + \frac{1}{2} \frac{(0 \cdot 2^{-1} + \frac{1}{2} \llbracket x \rrbracket)}{2} \\
 &= \llbracket (-1 : f(0 : x)) \rrbracket
 \end{aligned}$$

oooooooooooooooooooo
 oooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 ooooooooooooo●ooo
 ooooooooooooo
 ooooo
 ooooo

o
 o
 o

Demonstration: addition

Based on the previous working

$$x + y = m_0 \times 2^{e_0} + m_1 \times 2^{e_1}$$

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooo●ooo
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Based on the previous working

$$\begin{aligned}
 x + y &= m_0 \times 2^{e_0} + m_1 \times 2^{e_1} \\
 &= \frac{m_0 \times 2^{e_0 - e} + m_1}{2} \times 2^{e+1}
 \end{aligned}$$


```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
ooooo

```

```

o
oooooooooooo●ooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Based on the previous working

$$\begin{aligned}
 x + y &= m_0 \times 2^{e_0} + m_1 \times 2^{e_1} \\
 &= \frac{m_0 \times 2^{e_0-e} + m_1}{2} \times 2^{e+1} \\
 &= (m_0 \times 2^{-(e-e_0)} \oplus m_1) \times 2^{e+1}
 \end{aligned}$$

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooo●ooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

Based on the previous working

$$\begin{aligned}
 x + y &= m_0 \times 2^{e_0} + m_1 \times 2^{e_1} \\
 &= \frac{m_0 \times 2^{e_0-e} + m_1}{2} \times 2^{e+1} \\
 &= (m_0 \times 2^{-(e-e_0)} \oplus m_1) \times 2^{e+1}
 \end{aligned}$$

and the previous program for \oplus , we have:

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooo●ooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Elementary functions

To show you the power of the elementary functions, we recall that

Theorem (John Machin)

$$4 \tan^{-1} \left(\frac{1}{5} \right) - \tan^{-1} \left(\frac{1}{239} \right) = \frac{\pi}{4}.$$

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooooooooooo

```

```

o
o
o

```

Elementary functions

Consider the program

```
cvt_MEtoDECSTRINGraw myPI
```

where

```

i_PIOver4 :: I
i_PIOver4 = mul_I_4 (sub_I_I (my_arctan (cvt_QtoI (2,5)))

-- myPI = pi (derived from pi/4)
myPI :: ME
myPI = mul_ME_INT (cvt_ItoME i_PIOver4) 4

```

```

oooooooooooooooooooo●oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo●
oooooooooooo
oooo
oooo
oooooooooooo

```

```

o
o
o

```

Demonstration: addition

```
add_ME_ME :: ME -> ME -> ME
```

```
add_ME_ME (m0,e0) (m1,e1)
```

```
  = ( mid_I_I (f (m0,e0-e)) (f (m1,e1-e)) , e + 1 )
```

```
  where e          = max e0 e1;
```

```
        f (m2,e2) = (take (abs (e2)) i_ZERO) ++ m2;
```

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

o
 oooooooooooooooooo
 ●oooooooooooo
 oooo
 ooooooooooooo

o
 o
 o

Expression tree

In contrast to most existing calculators, one special feature of ERCE is:

Functional paradigm

Functions are first-class citizens!

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

o
 oooooooooooooooooo
 ●oooooooooooo
 oooo
 ooooooooooooo

o
 o
 o

Expression tree

Roughly speaking, an expression can be regarded as a tree that represents either

oooooooooooooooooooo●oooooooooooo
 oooo●oooooooooooo
 oo

o
 oooooooooooooooooo
 ●oooooooooooo
 oooo
 ooooooooooooo

o
 o
 o

Expression tree

Roughly speaking, an expression can be regarded as a tree that represents either

- closed mathematical expression

oooooooooooooooooooo●oooooooooooo
 ooooo●oooooooooooo
 oo

o
 oooooooooooooooooo
 ●oooooooooooo
 oooo
 oooooooooo

o
 o
 o

Expression tree

Roughly speaking, an expression can be regarded as a tree that represents either

- closed mathematical expression
- open mathematical expression (with a hole X)

```

oooooooooooooooooooo●oooooooooooo
ooooo                ooooooooooooo
oo                  ooooo
oo

```

```

o
oooooooooooooooooooo
●oooooooooooo
oooo
oooooooooooo

```

```

o
o
o
o

```

Expression tree

Roughly speaking, an expression can be regarded as a tree that represents either

- closed mathematical expression
- open mathematical expression (with a hole X)
- function application of a term on another

ooooooooooooooooooooo
 oooooooooooooo
 ooooo
 oo

ooooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 oooooo
 ooooo
 oooooooooo

o
 o
 o

Expression tree

Example (Closed expression)

$1 + \frac{2}{\sin \frac{\pi}{3}} \times 7.13 - 0$ parses to the tree below:

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oo●oooooooo
oooo
oooo
oooo

```

```

o
o
o

```

Expression tree

Example (Closed expression)

$1 + \frac{2}{\sin \frac{\pi}{3}} \times 7.13 - 0$ parses to the tree below:

```

Op Add (Value "1")
  (Op Add (Op Mul (Op Div (Value "2")
    (Fun Sin (Op Div (Value "PI")
      (Value "3"))))
    (Value "7.13")))
    (Op Mul (Value "-1")
      (Value "0"))))

```

ooooooooooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooo●oooo
 oooo
 oooooooooo

o
 o
 o

Expression tree

Example (Open expression)

$4x(1 - x)$ parses to the tree below:

```
Op Mul (Op Mul (Value "4")
                (Var "X"))
      (Op Add (Value "1")
              (Op Mul (Value "-1") (Var "X"))))
```

ooooooooooooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooo●oooo
 oooo
 oooooooooo

o
 o
 o

Expression tree

Example (Application)

$(4x(1 - x))(0.5)$ parses to the tree below:

```
App (Op Mul (Op Mul (Value "4")
                    (Var "X"))
    (Op Add (Value "1")
            (Op Mul (Value "-1") (Var "X"))))
  (Value "0.5")
```

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo●oooo
oooo
oooooooooooo

```

```

o
o
o

```

Syntax data type

```

data Expression =
  Value String | Op Op1 Expression Expression |
  Fun Fun1 Expression | Var String |
  App Expression Expression | Itn Expression String
  deriving Show

```

where

```

data Op1      = Add | Sub | Mul | Div | Pow
  deriving Show

```

```

data Fun1     = Exp | Sin | Cos | Tan | Log | Lon
  deriving Show

```

ooooooooooooooooooooo
 oooooooooooooo
 ooooo
 oo

ooooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooo
 ooooooooooooo
 ooooo●ooo
 oooo
 ooooooooo

o
 o
 o

Function iteration

Because functions are first class citizens, you can execute finite iterations of these to create new functions, i.e., given a function $f : \mathbb{R} \rightarrow \mathbb{R}$ and $n \in \mathbb{N}$,

$$f^{(n)}(x) = \underbrace{f(f(\dots f(x) \dots))}_{n \text{ copies}}.$$


```

oooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo●o
oooo
oooo
oooooooo

```

```

o
o
o

```

Function iteration

The calculator admits the syntax

`(exp1 @ exp2)(exp3)`

where

`exp1` -- function `f`

`exp2` -- non-negative integer `n`

`exp3` -- seed of iteration `x`

```

ooooooooooooooooooooo
ooooooooooooooooooooo
oooo
oo

```

```

ooooooooooooooooooooo
ooooooooooooooooooooo
ooooo

```

```

o
ooooooooooooooooooooo
oooooooooooo●
oooo
oooooooooooo

```

```

o
o
o

```

Function iteration

Example (Application of function iteration)

The function $f = \lambda x. 4x(1 - x)$ applied 5 times on the seed x is parsed to the expression tree:

```

App (Itn (Op Mul (Op Mul (Value "4") (Var "X"))
              (Op Add (Value "1") (Op Mul (Value "-1") (Var "X"))
                    "5")
      (Value "0.5"))

```

oooooooooooooooooooo●oooooooooooo
 ooooo
 oo

oooooooooooo
 ooooooooooooo
 ooooo

o
 oooooooooooooooooo
 ooooooooooooo
 ●ooo
 ooooooooooooo

o
 o
 o

Evaluator

The evaluator reads the parsed expression tree and converts it to a function.

- Function expression \mapsto a function of type $\text{ME} \rightarrow \text{ME}$
- Non-function \mapsto a constant function of type $\text{ME} \rightarrow \text{ME}$

Evaluator

```
-- "Evaluating" an application to a function
evaltofn :: Expression -> ME -> ME
evaltofn (Var x) r = r
evaltofn (Value x) r = if x == "PI" then myPI else cvt_DECSTRINGtoME x
evaltofn (Itm e1 e2) r = (repeat_MEfunction (evaltofn e1) (read e2 :: Int)) r
evaltofn (Op Add e1 e2) r = add_ME_ME ((evaltofn e1)r) ((evaltofn e2)r)
evaltofn (Op Mul e1 e2) r = mul_ME_ME ((evaltofn e1)r) ((evaltofn e2)r)
evaltofn (Op Div e1 e2) r = div_ME_ME ((evaltofn e1)r) ((evaltofn e2)r)
evaltofn (Fun Exp x) r = e_power_ME ((evaltofn x)r)
evaltofn (Fun Sin x) r = sin_ME ((evaltofn x)r)
evaltofn (Fun Cos x) r = cos_ME ((evaltofn x)r)
evaltofn (Fun Tan x) r = tan_ME ((evaltofn x)r)
```

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oo●oo
ooooo

```

```

o
o
o

```

Evaluator

```

-- Evaluating parsed trees in mantissa exponent form
evalme :: Expression -> ME
evalme (Value x)      = if x == "PI" then myPI else cvt_DECSTRINGtoME x
evalme (App e1 e2)    = (evaltofn e1)(evalme e2)
evalme (Op Add e1 e2) = add_ME_ME (evalme e1) (evalme e2)
evalme (Op Mul e1 e2) = mul_ME_ME (evalme e1) (evalme e2)
evalme (Op Div e1 e2) = div_ME_ME (evalme e1) (evalme e2)
evalme (Fun Exp x)    = e_power_ME (evalme x)
evalme (Fun Sin x)    = sin_ME (evalme x)
evalme (Fun Cos x)    = cos_ME (evalme x)
evalme (Fun Tan x)    = tan_ME (evalme x)

```

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo●	o	
		oooooooooooo		

Evaluator

For human consumption, the output signed-bit stream is then converted into readable form, i.e., decimal representation correct to a default of 20 decimal places.

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	o	o	
oooo	oooooooooooo	oooooooooooooooooooo	o	
oo	oooo	oooooooooooo	o	
		oooo		
		oooo		
		●oooooooo		

Monadic programming

To simulate the I/O interaction between user and program,
HASKELL makes use of

Monadic Programming.

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooooooo	oooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		oooo		
		o●oooooooo		

Actions

The module `Main` is the standard platform by which *actions-sequence* can be scripted.

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		oo●oooooooo		

Actions

To link with the underlying calculator, we invoke the commands:

```
import Eval
import Parse
import Graphics.UI.Gtk
```

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	o	o	
oooo	oooooooooooo	oooooooooooooooooooo	o	
oo	ooooo	oooooooooooo	o	
		oooo		
		ooo●ooooo		

Actions

Actions are of an abstract data type **IO a**, where

Actions

Actions are of an abstract data type $\text{IO } a$, where

- IO is a fixed type constructor that flags I/O action,

Actions

Actions are of an abstract data type $\text{IO } a$, where

- IO is a fixed type constructor that flags I/O action,
- a is any data type.

```

ooooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo●oooo

```

```

o
o
o

```

Actions

Give a simple instruction for setting the dimensions of the calculator:

```
main :: IO ()
```

```
main = do
```

```
    initGUI
```

```
    window <- windowNew
```

```
    set window [windowTitle := "ERCE", containerBorderWidth
                  windowDefaultWidth := 350,
                  windowDefaultHeight := 400]
```

```
    table <- tableNew 10 10 True
```

```
    containerAdd window table
```

Buttons

There are two different kinds of buttons:

Buttons

There are two different kinds of buttons:

- Symbolic

Buttons

There are two different kinds of buttons:

- Symbolic
- Executive

When a symbolic button is pressed, a single mathematical symbol that matches the button label (e.g., `button1`) is inserted at the end of the existing string of symbols previously keyed in. This is stored at `label12`.

```

ooooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
ooooooooo●o

```

```

o
o
o

```

Symbolic button

When a symbolic button is pressed, the `buttonSwitch` function below is invoked:

```

buttonSwitch :: Button -> Label -> Label -> IO()
buttonSwitch b l z= do
  txt <- get b buttonLabel
  lb  <- get l labelText
  lbs <- get z labelText

  if txt == "="
    then labelSetText l( show(eval (parse lbs)))
    else labelSetText z(lbs ++txt)

```

```

ooooooooooooooooooooo
oooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
ooooooooooooooooo
oooooooooooo
oooo
oooo
ooooooooo●

```

```

o
o
o

```

ERCE user-interface

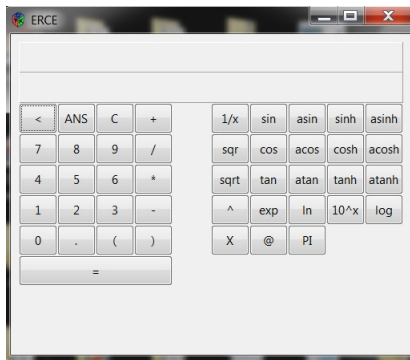


Figure : A screen-capture of ERCE user-interface

oooooooooooooooooooo●●●oooooooo
 oooo●oooooooooooo
 oo

o
 oooooooooooooo
 oooooooooo
 oooo
 oooooooooo

●
 o
 o

Orbit of logistic map

We now proceed to calculate x_{100} of the sequence $(x_n)_{n=0}^{\infty}$ defined by

$$x_0 = 0.125$$

$$x_{n+1} = 4x_n(1 - x_n)$$

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
oooo

```

```

o
●
o

```

Orbit of logistic map

```

cvt_MEtoDECSTRINGtrunc (cvt_ItoME ((repeat_Ifunction logistic_I 100)(cvt_QtoI (1,8)))) 20

```

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
●

```

Orbit of logistic map

n	ERCE	MATLAB
0	0.12500000000000000000	0.12500000000000000000
10	0.38367583854736609603	0.38367583854736526000
20	0.55150781744159181178	0.55150781744258315000
30	0.29059706649102177619	0.29059706556439530000
40	0.94723756671816869896	0.94723803391889350000
50	0.97984857115056995132	0.98014818012213323000
100	0.99971849434213872830	0.78273071414107154000

Figure : Orbits of the logistic map produced by ERCE and MATLAB

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		oooooooooooo		

Concluding remarks and future directions

Areas of improvement:

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		oooooooooooo		

Concluding remarks and future directions

Areas of improvement:

1. Efficiency

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		ooooooo		

Concluding remarks and future directions

Areas of improvement:

1. Efficiency
2. Improved capabilities

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		ooooooo		

Concluding remarks and future directions

Areas of improvement:

1. Efficiency
2. Improved capabilities
 - calculus (supremum and Riemann integral)

Introduction	Preliminaries	Design and implementation	Sample applications	Conclusion
oooooooooooooooooooo	oooooooooooo	oooooooooooooooooooo	o	
oooo	oooooooooooo	oooooooooooo	o	
oo	oooo	oooo	o	
		oooooooooooo		

Concluding remarks and future directions

Areas of improvement:

1. Efficiency
2. Improved capabilities
 - calculus (supremum and Riemann integral)
 - complex analysis

```

ooooooooooooooooooooo
oooo
ooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
o
o

```

References I



A. Avizienis. Binary-computable signed-digit arithmetic. In AFIPS Conference Proceedings, volume 26, pages 663–672, 1964.



A. Bauer and I. Kavkler. Implementing real numbers with RZ. In Ruth Dillhage, Tanja Grubba, Andrea Sorbi, Klaus Weihrauch, and Ning Zhong, editors, Proceedings of the Fourth International Conference on Computability and Complexity in Analysis, volume 202 of Electronic Notes in Theoretical Computer Science, pages 365–384, Siena, Italy, June 2007. CCA 2007, Elsevier.



A. Bauer and P. Taylor. The Dedekind reals in abstract Stone duality. Mathematical Structures in Computer Science, 19(4):757–838, July 2009.

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
o

```

References II



E. Bishop. Foundation of constructive analysis. McGraw-Hill, New York, 1967.



H. J. Boehm, R. Cartwright, M. Riggle, and M .J. O'Donell. Exact real arithmetic: a case study in higher programming. In ACM Symposium on lisp and functional programming, 1986.



K. Briggs. XRC (eXact Reals in C). Available at the website <http://keithbriggs.info/xrc.html>, 2013.



L. E. J. Brouwer. Besitzt jede reelle zahl eine dezimalbruchentwicklung? Math. Ann., 83(3-4):201–210, 1921.



A. Edalat and M. H. Escardó. Integration in Real PCF. Information and Computation, 160(1):128–166, July 2000.

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
ooooo
ooooo

```

```

o
o
o

```

References III



A. Edalat and P. J. Potts. A new representation for exact real numbers. *Electronic Notes in Theoretical Computer Science*, 6:119–132, 1997.



A. Edalat and R. Heckmann. Computing with real numbers - I. The LFT Approach to Real Number Computation - II. A Domain Framework for Computational Geometry. In *PROC APPSEM SUMMER SCHOOL IN PORTUGAL*, pages 193–267. Springer Verlag, 2002.



A. Edalat and P. Sunderhauf. A domain-theoretic approach to computability on the real line. *Theor. Comput. Sci.*, 210(1):73–98, January 1999.







M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.

-



References V

-  J. Gibbons and G. Hutton. Proof Methods for Corecursive Programs. *Fundamentae Informaticae*, 20:1–14, 2005.
-  W. K. Ho. An operational domain-theoretic treatment of recursive types. *Mathematical Structures in Computer Science*, FirstView:1–59, 6 2013.
-  IEEE. IEEE standard 754 for binary oating-point arithmetic. *SIGPLAN*, 22(2):9–25, 1985.
-  W. Kramer. A priori worst-case error bounds for floating-point computations. In Tomas Lang, Jean-Michel Muller, and Naofumi Takagi, editors, 13th IEEE Symposium on Computer Arithmetic, volume 13, pages 64–73, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, July 1997. IEEE Computer Society Press.


```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
ooooo
ooooo

```

```

o
o
o

```

References VI



D. Lacombe. Constructivity in mathematics, chapter Quelques procédés de définitions en topologie récursif, pages 129–158. North-Holland, 1959.



Branimir Lambov. Reallib: An efficient implementation of exact real arithmetic. Mathematical. Structures in Comp. Sci., 17(1):81–98, February 2007.







P. Martin-Lof. Note on Constructive Mathematics. Almqvist and Wiksell, 1970.



R. M. May. Simple mathematical models with very complicated dynamics. Nature, 261(5560):459–467, 1976.



N. T. Muller. The iRRAM: Exact arithmetic in C++. Lecture Notes in Computer Science, 2064:222–252, 2001.

-  D. Plume. A Calculator for Exact Real Number Computation. 4th year project report, University of Edinburgh, 1998.
-  P. J. Potts and A. Edalat. Exact real computer arithmetic. Draft available from www.doc.ic.ac.uk/~ae/papers/computerarithmetic.ps.gz, March 1997.
-  D. S. Scott. Outline of the mathematical theory of computation. In Proceedings of the 4th Princeton Conference on Information Science, 1970.
-  A.K. Simpson. Lazy functional algorithms for exact real functionals. Lecture Notes in Computer Science, (1450):323–342, 1998.

```

ooooooooooooooooooooo
oooo
oo

```

```

oooooooooooo
oooooooooooo
ooooo

```

```

o
oooooooooooooooooooo
oooooooooooo
oooo
ooooo
ooooo

```

```

o
o
o

```

References VIII



[30] E. Specker. Nicht konstruktiv beweisbare satze der analysis. J. Symbolic Logic, 14:145–158, 1949.



A. S. Troelstra and D. van Dalen. Constructivism in Mathematics. North-Holland, Amsterdam, 1988.



A. M. Turing. On computable numbers with an application to the entscheidungsproblem. Proc. Lond. Math. Soc., Ser. 2, 42:230–265, 1937.



C. Vuik. Some disasters caused by numerical errors. Website at <http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html>.



J. Vuillemin. Exact real computer arithmetic with continued fractions. In Proc. A.C.M. conference on Lisp and functional Programming, pages 14–27, 1988.

```

oooooooooooooooooooo
oooo
oo
oo

```

```

oooooooooooo
oooooooooooo
oooo

```

```

o
oooooooooooooooo
oooooooooooo
oooo
oooo
oooo

```

```

o
o
o

```

References IX



J. Vuillemin. Exact real computer arithmetic with continued fractions. IEEE Transactions on computers, 39(8):1087–1105, August 1990.



K. Weihrauch. A simple introduction to computable analysis. Technical Report 171-7/1995, FernUniversität, 1995.



K. Weihrauch and C. Kreitz. Representations of the real numbers and of the open subsets of the set of real numbers. Annals of Pure and Applied Logic, 35:247–260, 1987.



K. Weihrauch and U. Schreiber. Embedding metric spaces into cpo's. Theoret. Comp. Sci., 16:5–34, 1981.