Real numbers in type theory

Alex Simpson
Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh

(Based on joint work with Martín Escardó, University of Birmingham)

3-lecture course
Days in Logic, Braga, 23–25 Jan 2014
Desiderata

1. Compute exactly with real numbers, avoiding rounding errors.

2. Program using implementation-independent primitives, so we don’t need to know how real numbers are represented.

3. Reason about programs on real numbers; e.g., prove equations.

We address these by extending type theory with an abstract datatype of real numbers, with associated equational laws.

This approach also addresses another desideratum:

4. Give a foundationally neutral mathematical definition of real number purely in terms of computationally meaningful primitives.
Course structure

Part 1: The intensional approach.
Computing with representations of reals.

Part 2: The extensional approach.
An abstract datatype for the interval \([−1, 1]\).

A mantissa-exponent representation of \(\mathbb{R}\).
Part 1: The intensional approach.
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{c}
0. \\
0. \\
\hline \\
0.
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

$$
\begin{array}{c}
\text{0.} \\
\text{0.} \\
\hline
\text{0.} \\
\text{0.} \\
\end{array}
$$
Representing reals exactly

Natural first attempt: represent a real number in \([0, 1]\) exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in \([0, 1]\) using this representation.

\[
\begin{array}{c}
0. \ 0 \\
0. \ 1 \\
\hline
0. \ ?
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{c}
0. & 0 & 1 \\
0. & 1 & 0 \\
\hline
0. & ? \\
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

```
0.  0  1  0
  0. 1  0  1
------------------
  0.  ?
```
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{cccc}
0. & 0 & 1 & 0 & 1 \\
0. & 1 & 0 & 1 & 0 \\
\hline
0. & ? \\
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{c}
0.0101010 \\
0.1010101 \\
\hline
0.1010101
\end{array}
\]

It is impossible to write such a program! Technically, there is no continuous function $\{0, 1\}^\omega \times \{0, 1\}^\omega \rightarrow \{0, 1\}^\omega$ that computes average.
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{ccccccc}
0. & 0 & 1 & 0 & 1 & 0 & 1 \\
0. & 1 & 0 & 1 & 0 & 1 & 0 \\
\hline
0. & ?
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{cccccccc}
0. & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0. & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
0. & ?
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{cccccccc}
0. & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0. & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
0. & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{ccccccccccc}
0. & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \ldots \\
0. & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \ldots \\
\hline
0. & ?
\end{array}
\]

It is impossible to write such a program!
Representing reals exactly

Natural first attempt: represent a real number in $[0, 1]$ exactly as a lazy stream providing infinite binary expansion.

Let’s see how we might try to write a program computing the average of two real numbers in $[0, 1]$ using this representation.

\[
\begin{array}{c}
0. \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ \ldots \\
0. \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ \ldots \\
\hline \\
0. \ ? \\
\end{array}
\]

It is impossible to write such a program!

Technically, there is no continuous function 
\[\{0, 1\}^\omega \times \{0, 1\}^\omega \rightarrow \{0, 1\}^\omega\] that computes average.
Three implementations of $[-1, 1]$ that do work

Using streams:

$$q_0 : q_1 : q_2 : q_3 : q_4 : q_5 : \ldots$$

1. **Fast Cauchy sequences**
   Require $q_i$ to be rational numbers in $[-1, 1]$ such that $|q_j - q_i| \leq 2^{-i}$, whenever $j \geq i$, The stream represents the real number $\lim_i q_i$.

2. **Dyadic digits**
   Require $q_i$ to be dyadic rational numbers in $[-1, 1]$. The stream represents the real number $\sum_{i \geq 0} 2^{-(i+1)} q_i$.

3. **Signed binary**
   This is the special case of dyadic digits in which we require $q_i \in \{-1, 0, 1\}$.
Three implementations of $[-1, 1]$ that do work

Using streams:

$q_0 : q_1 : q_2 : q_3 : q_4 : q_5 : \ldots$

1. **Fast Cauchy sequences**
   Require $q_i$ to be rational numbers in $[-1, 1]$ such that $|q_j - q_i| \leq 2^{-i}$, whenever $j \geq i$, The stream represents the real number $\lim_{i} q_i$.

2. **Dyadic digits**
   Require $q_i$ to be dyadic rational numbers in $[-1, 1]$. The stream represents the real number $\sum_{i \geq 0} 2^{-(i+1)} q_i$.

3. **Signed binary**
   This is the special case of dyadic digits in which we require $q_i \in \{-1, 0, 1\}$.

These representations are interconvertible.
Simple recursive programs using dyadic digits

Average:

$$\text{avg}(q : r, q' : r') = (q \oplus q') : \text{avg}(r, r')$$

where $q \oplus q' = \frac{q + q'}{2}$.

Multiplication:

$$\text{digit\_mult}(q, q' : r') = (qq') : \text{digit\_mult}(q, r')$$

$$\text{mult}(q : r, q' : r') = \text{avg}( (qq') : \text{mult}(r, r'),$$

$$\text{avg}(\text{digit\_mult}(q, r'),$$

$$\text{digit\_mult}(q', r)))$$
Converting dyadic digits to signed binary

\[ dy_{\text{add}}(q, q': r') = \begin{cases} 
1 : dy_{\text{add}}((2q + q' - 1), r') & \text{if } 2q + q' \geq 1 \\
(2q + q') : r' & \text{if } -1 \leq 2q + q' \leq 1 \\
(-1) : dy_{\text{add}}((2q + q' + 1), r') & \text{if } 2q + q' \leq -1 
\end{cases} \]

\[ \text{convert}(q_0 : q_1 : r) = \begin{cases} 
1 : \text{convert}(dy_{\text{add}}(q_0 - 1, q_1 : r)) & \text{if } q_0 + \frac{q_1}{2} \geq \frac{1}{2} \\
0 : \text{convert}(dy_{\text{add}}(q_0, q_1 : r)) & \text{if } -\frac{1}{2} \leq q_0 + \frac{q_1}{2} \leq \frac{1}{2} \\
(-1) : \text{convert}(dy_{\text{add}}(q_0 + 1, q_1 : r)) & \text{if } q_0 + \frac{q_1}{2} \leq -\frac{1}{2} 
\end{cases} \]
Computing a useful infinite sum

Consider a dyadic digit sequence as given by a function from natural numbers to dyadic rationals.

The program

\[
M((q_0 : q_1 : r) : (q' : r') : s) = \left( \frac{2q_0 + q_1 + q_2}{4} \right) : (M(\text{avg}(r, r') : s))
\]

of type

(dyadic_digit stream) stream → dyadic_digit stream

computes the function

\[
(x_i)_{i \geq 0} \mapsto \sum_{i \geq 0} 2^{-(i+1)} x_i : [-1, 1]^\omega \to [-1, 1]
\]
Two issues

▶ What programming capabilities do we need to write such programs?

▶ Programming in the above style is done at a very nuts-and-bolts level. E.g., we need to take extreme care that programs behave equivalently on different intensional representations of the same real number; e.g., $f(0 : 0 : 0 : 0 : 0 : \ldots)$ and $f((-1) : 1 : 1 : 1 : 1 : \ldots)$ should both result in representations of the same real number.
Two issues

▶ What programming capabilities do we need to write such programs?

We shall see that such programs can be written in the minimal type-theoretic framework of typed $\lambda$-calculus with natural numbers (Gödel’s System T).

▶ Programming in the above style is done at a very nuts-and-bolts level. E.g., we need to take extreme care that programs behave equivalently on different intensional representations of the same real number; e.g.,

$f(0 : 0 : 0 : 0 : 0 : \ldots)$ and $f((-1) : 1 : 1 : 1 : 1 : \ldots)$ should both result in representations of the same real number.
Two issues

▶ What programming capabilities do we need to write such programs?

We shall see that such programs can be written in the minimal type-theoretic framework of typed λ-calculus with natural numbers (Gödel’s System T).

▶ Programming in the above style is done at a very nuts-and-bolts level. E.g., we need to take extreme care that programs behave equivalently on different intensional representations of the same real number; e.g.,

\[ f(0 : 0 : 0 : 0 : 0 : \ldots) \text{ and } f((-1) : 1 : 1 : 1 : 1 : \ldots) \]

should both result in representations of the same real number.

A major goal of this course is to introduce an abstract datatype for real numbers, using which such extensional behaviour is guaranteed.
Gödel’s System T

Types:

\[ \sigma ::= N \mid \sigma \rightarrow \tau \]

Terms:

- Typed variables: \( x : \sigma \) (where \( x \) is a variable of type \( \sigma \))
- Abstractions: if \( t : \tau \) then \( (\lambda x : \sigma. t) : \sigma \rightarrow \tau \)
- Applications: if \( t : \sigma \rightarrow \tau \) and \( u : \sigma \) then \( t u : \tau \).

Constants:

\[
\begin{align*}
0 & : N \\
\text{s} & : N \rightarrow N \\
\text{rec} & : \sigma \rightarrow (\sigma \rightarrow N \rightarrow \sigma) \rightarrow N \rightarrow \sigma
\end{align*}
\]
Computation

Single-step reduction (applicable arbitrarily deep inside terms):

\[
(\lambda x. t) u \longrightarrow t[u/x]
\]

\[
\text{rec } u \text{ v } 0 \longrightarrow u
\]

\[
\text{rec } u \text{ v } (s(t)) \longrightarrow v (\text{rec } u \text{ v } t) t
\]

Theorem (Tait 1960s). Reduction in System T is strongly normalizing.

Reduction is also confluent (Church-Rosser). It follows that every closed term of type N reduces to a unique numeral (a term of the form \(s^k(0)\)).

Strong normalization for System T implies (and is in fact equivalent to) the 1-consistency of Peano Arithmetic.
Set-theoretic semantics

Semantics of types:

\[
\begin{align*}
  \llbracket N \rrbracket &= \mathbb{N} \\
  \llbracket \sigma \rightarrow \tau \rrbracket &= \llbracket \sigma \rrbracket \rightarrow \llbracket \tau \rrbracket
\end{align*}
\]

Every closed term \( t : \tau \) determines an element \( \llbracket t \rrbracket \in \llbracket \tau \rrbracket \) (The semantics routinely generalises to any cartesian-closed category (CCC) with natural numbers object (NNO).)
Set-theoretic semantics

Semantics of types:

\[
\begin{align*}
[N] &= \mathbb{N} \\
[\sigma \to \tau] &= [\sigma] \to [\tau]
\end{align*}
\]

Every closed term \( t : \tau \) determines an element \( [t] \in [\tau] \) (The semantics routinely generalises to any cartesian-closed category (CCC) with natural numbers object (NNO).)

Say that a function \( f : \mathbb{N}^k \to \mathbb{N} \) is \( T \)-representable if there exists a closed term \( t : \mathbb{N}^k \to \mathbb{N} \) such that, for all \( n_1, \ldots, n_k \in \mathbb{N} \),

\[
t \; n_1 \ldots \; n_k \; \rightarrow^* \; f(n_1, \ldots, n_k)
\]

Theorem (essentially Gödel). A function is \( T \)-representable if and only if it is provably total in Peano Arithmetic.
Equalities (valid in any CCC with NNO)

- If \( t \rightarrow u \) then \( t = u \).
- If \( t : \sigma \rightarrow \tau \) then \( t = \lambda x : \sigma . t x \).
- If \( w : \sigma \rightarrow (\sigma \rightarrow \mathbb{N} \rightarrow \sigma) \rightarrow \mathbb{N} \rightarrow \sigma \) and \( u : \sigma \) and \( v : \sigma \rightarrow \mathbb{N} \rightarrow \sigma \) and \( w u v 0 = u \) and \( w u v (s(x)) = v (w u v x) x \) then \( w u v = \text{rec} u v \).
Equalities (valid in any CCC with NNO)

- If $t \rightarrow u$ then $t = u$.
- If $t : \sigma \rightarrow \tau$ then $t = \lambda x : \sigma. \, t \, x$.
- If $w : \sigma \rightarrow (\sigma \rightarrow N \rightarrow \sigma) \rightarrow N \rightarrow \sigma$ and $u : \sigma$ and $v : \sigma \rightarrow N \rightarrow \sigma$ and $w \, u \, v \, 0 = u$ and $w \, u \, v \, (s(x)) = v \, (w \, u \, v \, x) \, x$ then $w \, u \, v = \text{rec} \, u \, v$.

Exercises on System T.

1. Define the following functions of type $N \rightarrow N \rightarrow N$: addition, multiplication, Ackermann’s function.
2. Prove commutativity and associativity for addition and multiplication. Prove distributivity of multiplication over addition.
Encoding real numbers in System T

We can encode any of our stream representations of $[-1, 1]$ using the type $\mathbb{N} \rightarrow \mathbb{N}$.

E.g., in the case of signed binary, we can consider a function $\alpha : \mathbb{N} \rightarrow \mathbb{N}$ as encoding the signed binary stream

$$(((\alpha(0) \mod 3) - 1) : ((\alpha(1) \mod 3) - 1) : ((\alpha(2) \mod 3) - 1) : \ldots$$

The different representations are all intertranslatable in System T. Moreover, all the programs we gave for computing functions on real numbers using the signed binary and dyadic digits representation are routinely translatable into System T terms of appropriate type.
Encoding functions on reals in System T

For convenience we fix on signed binary representation. Consider the function $\text{real} : (\mathbb{N} \to \mathbb{N}) \to [-1, 1]$ defined by

$$\text{real}(\alpha) = \sum_{i \geq 0} 2^{-(i+1)} ((\alpha(i) \mod 3) - 1)$$

We say that a function $f : [-1, 1]^k \to [-1, 1]$ is $T$-representable if there exists a closed term $t : (\mathbb{N} \to \mathbb{N})^k \to \mathbb{N} \to \mathbb{N}$ making the following diagram commute:

$$\begin{array}{c}
(\mathbb{N} \to \mathbb{N})^k \xrightarrow{[t]} (\mathbb{N} \to \mathbb{N}) \\
\downarrow \text{real}^k \quad \quad \quad \quad \quad \quad \downarrow \text{real} \\
[-1, 1]^k \xrightarrow{f} [-1, 1]
\end{array}$$
Comments

- All T-representable functions are continuous.
- Very many continuous functions are T-representable.
- But programming functions on reals in System T has to be done in the nuts-and-bolts style mentioned earlier.
Comments

- All T-representable functions are continuous.

- Very many continuous functions are T-representable.

- But programming functions on reals in System T has to be done in the nuts-and-bolts style mentioned earlier.

Address last point by introducing an abstract datatype for a closed interval of real numbers.

Programs on real numbers can then be written in an automatically extensional way using the interface for the abstract datatype.
Part 2: The extensional approach.
We extend System T with a new type:

\[ \sigma ::= N \mid \sigma \rightarrow \tau \mid I \]

\(I\) is abstract datatype for the closed interval \([-1, 1]\) of real numbers. Semantics:

\[ [[I]] = [-1, 1] \]

The interface for the type will roughly implement the idea that the closed interval is determined as the free convex set on 2 generators \((-1, 1)\) with respect to affine maps.

Since the very formulation of convexity requires a pre-existing interval, we replace convexity with the existence of iterated midpoints and we replace affineness with preservation of (iterated) midpoints. [Escardó & S., LICS 2001]
Interface and semantics

New constants and their meanings

\begin{align*}
1 : I & \quad [1] = 1 \\
-1 : I & \quad [-1] = -1 \\
m : I \rightarrow I \rightarrow I & \quad [m] x y = \frac{x + y}{2} \\
M : (N \rightarrow I) \rightarrow I & \quad [M] f = \sum_{i \geq 0} \frac{f(i)}{2(i+1)} \\
aff : I \rightarrow I \rightarrow I \rightarrow I & \quad [aff] x y z = \frac{(1 - z)x + (1 + z)y}{2}
\end{align*}
Computation

I is an abstract datatype. We don’t specify a particular concrete implementation, but it is vital that implementation is possible. We have seen three possible implementations of the type I using the type $\mathbb{N} \rightarrow \mathbb{N}$ in System T.

In System I, programming on real numbers is performed purely using the interface, irrespective of implementation. The program stays the same even if the implementation changes (e.g., to improve efficiency). Programs on reals are automatically extensional.
Some simple programming

\[ x \oplus y = m \times y = M(x, y, y, y, y, y, y, \ldots) \]

(So not actually necessary to include \( m \) as a primitive, since derivable from \( M \). Nonetheless included for convenience.)

\[ 0 = (-1) \oplus 1 \]
\[ -x = \text{aff } 1 (-1) x \]
\[ xy = \text{aff } (-x) x y \]
\[ \frac{1}{3} = M(1, -1, 1, -1, 1, -1, 1, -1, \ldots) \]

More generally, any rational number is definable using \( M \) applied to an eventually periodic sequence of 1s and \(-1\)s.

Still more generally, any real number with a \( T \)-definable binary expansion is definable.
Equalities

\[
\begin{align*}
m \cdot t \cdot t &= t \\
m \cdot t \cdot u &= m \cdot u \cdot t \\
m \cdot (m \cdot t \cdot u) \cdot (m \cdot v \cdot w) &= m \cdot (m \cdot t \cdot v) \cdot (m \cdot u \cdot w) \\
M_i t_i &= m(t_0, M_i t_{i+1}) \\
\lambda_i \cdot t_i &= \lambda_i \cdot m(u_i, t_{i+1}) \quad \implies \quad t_0 = M_i u_i
\end{align*}
\]

For \( w : I \to I \to I \), if \( w \cdot t \cdot u (-1) = t \) and \( w \cdot t \cdot u 1 = u \) and
\( w \cdot t \cdot u \cdot m(x, y) = m(w \cdot t \cdot u \cdot x, w \cdot t \cdot u \cdot y) \) then
\[
w \cdot t \cdot u = \text{aff} \cdot t \cdot u
\]
Consequences (exercise!)

\[ - - x = x \]
\[ x \oplus -x = 0 \]
\[ x 0 = 0 \]
\[ x y = y x \]
\[ (x y) z = x (y z) \]
\[ x (-y) = -(x y) \]
\[ x (y \oplus z) = (x y) \oplus (x z) \]
More programming, medial power series

Suppose

\[ f(x) = \sum_{n \geq 0} a_n x^n \]

where \( a_n \in [-1, 1] \). Then

\[
\frac{1}{2} f \left( \frac{x}{2} \right) = \frac{1}{2} \sum_{n \geq 0} a_n \left( \frac{x}{2} \right)^n \\
= \sum_{n \geq 0} a_n \left( \frac{x^n}{2^{n+1}} \right) \\
= \sum_{n \geq 0} a_n x^n 
\]
So, for example:

\[
\frac{1}{2 - x} = M \frac{x^n}{n} \\
\frac{1}{2} \exp \left( \frac{x}{2} \right) = M \frac{x^n}{n!} \\
\frac{1}{2} \sin \left( \frac{x}{2} \right) = M \text{parity}(n)(-1)^{\left\lfloor \frac{n-1}{2} \right\rfloor} \frac{x^n}{n!} \\
\frac{1}{2} \ln \left( 1 + \frac{x}{2} \right) = M (-1)^n \frac{x^{n+1}}{(n+1)} \\
\frac{1}{2} \sqrt{1 + \frac{x}{2}} = M \frac{(-1)^n(2n)!}{n (1 - 2n)(n!)^2 4^n} x^n
\]

All these define total continuous functions \([-1, 1] \rightarrow [-1, 1]\).
A function $f : [-1, 1]^k \rightarrow [-1, 1]$ is said to be l-definable if there exists a closed System I term $t : I^k \rightarrow I$ such that

$$\llbracket t \rrbracket x_1 \ldots x_k = f(x_1, \ldots, x_k)$$

Any l-definable function is automatically continuous, indeed computable.

Proposition. If $f$ is l-definable then it is T-representable.
A non-definable function

\[ \text{dbl}(x) = \begin{cases} 
1 & \text{if } \frac{1}{2} \leq x \\
2x & \text{if } -\frac{1}{2} \leq x \leq \frac{1}{2} \\
-1 & \text{if } x \leq -\frac{1}{2} 
\end{cases} \]

Note that \text{dbl} is continuous though not smooth.

Also \text{dbl} is computable, in fact \text{T}-representable.
A non-definable function

\[
dbl(x) = \begin{cases} 
1 & \text{if } \frac{1}{2} \leq x \\
2x & \text{if } -\frac{1}{2} \leq x \leq \frac{1}{2} \\
-1 & \text{if } x \leq -\frac{1}{2}
\end{cases}
\]

Note that \( \text{dbl} \) is continuous though not smooth.

Also \( \text{dbl} \) is computable, in fact \( T \)-representable.

Proposition. The function \( \text{dbl} \) is not definable.

Will prove this later.
A non-definable function

\[
\text{dbl}(x) = \begin{cases} 
1 & \text{if } \frac{1}{2} \leq x \\
2x & \text{if } -\frac{1}{2} \leq x \leq \frac{1}{2} \\
-1 & \text{if } x \leq -\frac{1}{2}
\end{cases}
\]

Note that \text{dbl} is continuous though not smooth.

Also \text{dbl} is computable, in fact \text{T-representable}.

**Proposition.** The function \text{dbl} is not definable.

Will prove this later.
System II

Extend System I with a new constant

\[ \text{dbl} : I \to I \]

denoting the function \( \text{dbl} : [-1, 1] \to [-1, 1] \)

New equations:

\[
\begin{align*}
\text{dbl } m(1, m(1, x)) &= 1 \\
\text{dbl } m(0, x) &= x \\
\text{dbl } m(-1, m(-1, x)) &= -1
\end{align*}
\]

One consequence is cancellation:

\[ m(t, u) = m(t, v) \implies u = v \]
Programming in System II

\[ x + y \] = \text{dbl}(x \oplus y) \\
x \ominus y = x \oplus (-y) \\
[x - y] = \text{dbl}(x \ominus y) \\
\text{max}(0, x) = [[x - 1] + 1] \\
\text{max}(x, y) = \text{dbl}\left(\left[\frac{x}{2}\right] + \text{max}(0, y \ominus x)\right) \\
\text{min}(x, y) = -\text{max}(-x, -y) \\
|x| = \text{max}(-x, x)

Question. Are any of the last 4 functions definable in System I?
A limit-finding function in System II

Define \( \text{fastlim} : (\mathbb{N} \to \mathbb{I}) \to \mathbb{I} \) by:

\[
\text{fastlim}_i x_i = \text{dbl} \left( \text{M} \text{dbl}^n(x_{n+1} \ominus x_n) \right)
\]

Lemma. If \( |x_{n+1} - x_n| \leq 2^{-n} \) then \( \text{fastlim}_i x_i \) is the limit of the (fast) Cauchy sequence \( (x_n)_n \).

Note that \( \text{fastlim}_i x_i \) always returns a value, even if \( (x_n)_n \) is non-convergent.

Also if \( (x_n)_n \) converges slowly, \( \text{fastlim}_i x_i \) need not be the limit.
II-definable functions

A function $f : [-1, 1]^k \to [-1, 1]$ is said to be II-definable if there exists a closed System II term $t : I^k \to I$ such that

$$\llbracket t \rrbracket x_1 \ldots x_k = f(x_1, \ldots, x_k)$$

Proposition. If $f$ is II-definable then it is T-representable.
II-definable functions

A function $f : [-1, 1]^k \to [-1, 1]$ is said to be II-definable if there exists a closed System II term $t : I^k \to I$ such that

$$\langle t \rangle_{x_1 \ldots x_k} = f(x_1, \ldots, x_k)$$

**Proposition.** If $f$ is II-definable then it is T-representable.

**Theorem.** If $f$ is T-representable then it is II-definable.
Outline of proof

1. Use the T-representation $t$ of $f$ to construct a Cauchy sequence of polygon approximations to $f$ in System II.

2. Extract a modulus of uniform continuity for $t$ on the compact subset $(\mathbb{N} \rightarrow \{0, 1, 1\})^k$.

3. Use the modulus of uniform continuity to obtain a fast Cauchy sequence of polygon approximations.

4. A term defining $f$ is then obtained using fastlim.
Define $\text{glue}: (I \to I)^2 \to I \to I$ by

$$\text{glue} f g x = \text{dbl} \left( \text{dbl} \left( \left( f \left( \text{dbl} \left[ x + \frac{1}{2} \right] \right) \oplus g \left( \text{dbl} \left[ x - \frac{1}{2} \right] \right) \right) \oplus \frac{1}{2} f(1) \right) \right)$$

Then glue represents the function:

$$\text{glue} f g x = \left[ f \left( \text{dbl} \left[ x + \frac{1}{2} \right] \right) + g \left( \text{dbl} \left[ x - \frac{1}{2} \right] \right) \right] - f(1)$$

which, whenever $f(1) = g(-1)$, satisfies

$$\text{glue} f g x = \begin{cases} f(2x + 1) & \text{if } -1 \leq x \leq 0 \\ g(2x - 1) & \text{if } 0 \leq x \leq 1 \end{cases}$$
Polygon approximations

For every $k \geq 1$, we define a System II term:

$$\text{appr}_k : \mathbb{N} \rightarrow ((\mathbb{N} \rightarrow \mathbb{N})^k \rightarrow I) \rightarrow (I^k \rightarrow I)$$

The base case $\text{appr}_1$ is defined by recursion on $\mathbb{N}$ to satisfy:

$$\text{appr}_1 \ 0 \ h = \text{aff} (h(-1)) \ (h(1))$$
$$\text{appr}_1 \ (n + 1) \ h = \text{glue} (\text{appr}_1 \ n \ (\lambda \ x. \ h(\text{avg}(x, -1))))$$
$$\hspace{1cm} \text{appr}_1 \ n \ (\lambda \ x. \ h(\text{avg}(x, 1))))$$

Given $\text{appr}_k$, the term $\text{appr}_{k+1}$ is given by

$$\text{appr}_{k+1} \ n \ h \ x_0 \ x_1 \ \ldots \ x_k = \text{appr}_1 \ n \ (\lambda \ y_0. \ \text{appr}_k \ n \ (h \ y_0) \ x_1 \ \ldots \ x_k) \ x_0$$
The application \( \text{appr}_k \ n \ h \) produces a piecewise multilinear approximation to the function \( h \), with the argument types changed from \( \mathbb{N} \rightarrow \mathbb{N} \) to \( \mathbb{I} \).

More precisely, the \( \text{appr}_k \ n \) function uses \( k \)-tuples of values from the set

\[
\mathbb{Q}_n := \{ q^i_n \mid 0 \leq i \leq 2^n \} \quad \text{where} \quad q^i_n := \frac{i}{2^{n-1}} - 1
\]

to form a lattice of \( (2^n + 1)^k \) rational partition points in \([-1, 1]^k\).

The application \( \text{appr}_k \ n \ h \) then results in a function \( \mathbb{I}^k \rightarrow \mathbb{I} \) that agrees with \( h \) at the partition points, and is (separately) affine in each coordinate between partition points. It is also affine in the \( h \) argument.

The lemma on the next slide formalises this.
Lemma. If $h: (\mathbb{N} \rightarrow \mathbb{N})^k \rightarrow I$ represents $f: [-1, 1]^k \rightarrow [-1, 1]$ then:

1. For all $r_0, \ldots, r_{k-1} \in \mathbb{Q}_n$ we have:

$$\text{appr}_k n h r_0 \ldots r_{k-1} = f r_0 \ldots r_{k-1}$$

2. For $0 \leq j < k$, $0 \leq i < 2^n$, and $0 \leq \lambda \leq 1$

$$\text{appr}_k n h x_0 \ldots x_{j-1} \left( \frac{i + \lambda}{2^n - 1} - 1 \right) x_j \ldots x_{k-1} =$$

$$(1 - \lambda) \text{appr}_k n h x_0 \ldots x_{j-1} q_n^i x_j \ldots x_{k-1} + \lambda \text{appr}_k n h x_0 \ldots x_{j-1} q_{n+1}^i x_{j+1} \ldots x_{k-1}$$

Note that $\left( \frac{i + \lambda}{2^n - 1} - 1 \right) = ((1 - \lambda) q_n^i + \lambda q_{n+1}^i)$.

Also, if $h_1, h_2: (\mathbb{N} \rightarrow \mathbb{N})^k \rightarrow I$ are real-respecting then

3. For $0 \leq \lambda \leq 1$, we have:

$$\text{appr}_k n ((1 - \lambda)h_1 + \lambda h_2) x_0 \ldots x_{k-1} =$$

$$(1 - \lambda) \text{appr}_k n h_1 x_0 \ldots x_{k-1} + \lambda \text{appr}_k n h_2 x_0 \ldots x_{k-1}$$
Continuity in System T

Given a closed System T term

\[ t : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N} \]

it holds that \([t] : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})\) is continuous with respect to the product topology on \(\mathbb{N} \to \mathbb{N}\) (Baire space).

In detail, this means that for any \(\alpha : \mathbb{N} \to \mathbb{N}\), we have:

For all \(e \geq 0\), there exists \(d \geq 0\) such that, for all \(\beta : \mathbb{N} \to \mathbb{N}\) with \(\beta \upharpoonright_d = \alpha \upharpoonright_d\), it holds that

\( [t](\alpha) \upharpoonright_e = [t](\beta) \upharpoonright_e \)

(We write \(\alpha \upharpoonright_n\) for the \(n\)-tuple \((\alpha(0), \ldots, \alpha(n - 1)) \in \mathbb{N}^n\).)
Uniform continuity in System T

Since \( \mathbb{N} \to \{0, 1, 2\} \subseteq \mathbb{N} \to \mathbb{N} \) is compact, it follows that \([t]\) is uniformly continuous on this subspace; i.e.,:

For all \( e \geq 0 \), there exists \( d \geq 0 \) such that, for all \( \alpha, \beta : \mathbb{N} \to \{0, 1, 2\} \) with \( \alpha \upharpoonright_d = \beta \upharpoonright_d \), it holds that
\[
[t](\alpha) \upharpoonright_e = [t](\beta) \upharpoonright_e
\]

A function \( \mu : \mathbb{N} \to \mathbb{N} \) is a modulus of uniform continuity if:

For all \( e \geq 0 \), and all \( \alpha, \beta : \mathbb{N} \to \{0, 1, 2\} \) with \( \alpha \upharpoonright_{\mu(e)} = \beta \upharpoonright_{\mu(e)} \), it holds that
\[
[t](\alpha) \upharpoonright_e = [t](\beta) \upharpoonright_e
\]

Proposition. There exists a closed System T term \( G_t : \mathbb{N} \to \mathbb{N} \) that defines a modulus of uniform continuity for \( t \).
An immediate consequence

Suppose $t : (\mathbb{N} \to \mathbb{N})^k \to \mathbb{N} \to \mathbb{N}$ represents $f : [-1, 1]^k \to [-1, 1]$ (using our System T encoding of signed binary).

Then there exists $G_t : \mathbb{N} \to \mathbb{N}$ such that, for all $e \geq 0$, and for all $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k : \mathbb{N} \to \{0, 1, 2\}$ with $\alpha_i \restriction G_t(d) = \beta_i \restriction G_t(d)$, it holds that

$$|f(\text{real}(\alpha_1), \ldots, \text{real}(\alpha_k)) - f(\text{real}(\beta_1), \ldots, \text{real}(\beta_k))| \leq 2^{-e}$$
Completing proof of theorem

Suppose that \( t: (\mathbb{N} \to \mathbb{N})^k \to \mathbb{N} \to \mathbb{N} \) is a closed term that \( T \)-represents \( f: [-1, 1]^k \to [-1, 1] \).

Let \( G_t \) be a uniform modulus for continuity for \( t \) on \( \mathbb{N} \to \{0, 1, 2\} \), as given on the previous slide.

Let \( g_n: [-1, 1]^k \to [-1, 1] \) be defined by:

\[
\text{appr}_k (G_t(n + 1)) (\text{real} \circ t) : I^k \to I
\]

Then for all \( x_1, \ldots, x_k \in [-1, 1] \),

\[
|f(x_1, \ldots, x_n) - g_n(x_1, \ldots, x_n)| \leq 2^{-n}
\]

Therefore the term below \( I \)-defines \( f \).

\[
\lambda x_0 \ldots x_{k-1}. \text{fastlim} (\lambda n. \text{appr}_k (G_t(n + 1)) (\text{real} \circ t) x_0 \ldots x_{k-1})
\]
We have considered two approaches to computation with real numbers in the interval \([-1, 1]\).

1. **Intensional** approach.
   Encode a real as a stream/function.
   Elaborated in the context of System T.

2. **Extensional** approach.
   Encapsulate real numbers in an abstract datatype.
   Elaborated in the context of Systems I and II.

Main result: both approaches equivalent in power.
Summary of what we have done so far

We have considered two approaches to computation with real numbers in the interval $[-1, 1]$.

1. **Intensional** approach.
   Encode a real as a stream/function.
   Elaborated in the context of System T.

2. **Extensional** approach.
   Encapsulate real numbers in an abstract datatype.
   Elaborated in the context of Systems I and II.

Main result: both approaches equivalent in power.

Pressing question: What about the full real line?
Mantissa-exponent representation of real line

We use System I (no need for II!).

For convenience we include product types $\sigma \times \tau$.

We use $\mathbb{N}$ to encode integers including negative integers. We write $\mathbb{Z}$ when doing this, and use evident related notation.

Represent arbitrary real numbers using the type $I \times \mathbb{Z}$.

$(x, z)$ represents the real number $2^z x$.

This is a semi-extensional representation of real numbers, combining: a discrete scaling $z$, which is intensional; and a continuous value $x$, which is extensional.

(An alternative choice would be to use $(x, z)$ to represent $z + x$. But this doesn’t work, not even if we use System II!)
Equivalence between reals

We reason in a setting in which we have intuitionistic first-order logic over System I extending the equational logic we have considered so far.

(Actually would be better to reformulate what follows, as far as possible, to use only equational logic.)

Equality between reals is the smallest equivalence relation on \( I \times \mathbb{Z} \) satisfying

\[
(x, m) \sim \left( \frac{x}{2}, m + 1 \right)
\]

This is can be defined equationally by:

\[
(x, m) \sim (y, n) \iff \frac{x}{2^{\max(m,n)} - m} = \frac{y}{2^{\max(m,n)} - n}
\]

N.B., we need to assume the cancellation law

\[x \oplus y = x \oplus z \implies y = z\]
Basic arithmetic

\[ 0 = (0, 0) \]

\[ 1 = (1, 0) \]

\[ -(x, m) = (-x, m) \]

\[ (x, m) + (y, n) = \left( \frac{x}{2^{\max(m,n)}-m} \oplus \frac{y}{2^{\max(m,n)}-n}, \max(m,n) + 1 \right) \]

\[ (x, m) \times (y, n) = (xy, m+n) \]

Can also define any rational number.

So can define polynomials with rational coefficients.
Basic equations

The operations $-, +, \times$ respect $\sim$.

The expected equations are all provable, e.g.:

\[
\begin{align*}
0 + x & \sim x \\
x + y & \sim y + x \\
(x + y) + z & \sim x + (y + z) \\
0 \times x & \sim 0 \\
1 \times x & \sim 1 \\
x \times y & \sim y \times x \\
(x \times y) \times z & \sim x \times (y \times z) \\
x \times (y + z) & \sim (x \times y) + (x \times z) \\
\end{align*}
\]
Basic definitions

Define
\[ |(x, m)| \leq 2^n \iff \exists y. (x, m) \sim (y, n) \]
(N.B., \(|x| \leq 2^n\) is defined as a primitive relation between \(x: I \times \mathbb{Z}\) and \(n: \mathbb{N}\). Is it possible to define \(|\cdot|: I \times \mathbb{Z} \rightarrow I \times \mathbb{Z}\) in System I?)

Lemma. The following are equivalent

1. For all integers \(n\), \(|(x, m)| \leq 2^n\)
2. \((x, m) \sim 0\)
3. \(x = 0\)

Define
\[ 0 \leq (x, m) \iff \exists y. x = 1 \oplus y \]
\[ x \leq y \iff 0 \leq y - x \]
\[ x < y \iff \exists q \in \mathbb{Q}. 0 < q \text{ and } x + q \leq y \]
Limits of fast Cauchy sequences

Suppose we have a sequence \((x_i)_i\) given by:

\[ x_{(-)} : \mathbb{N} \to \mathbb{I} \times \mathbb{Z} \]

such that the inequalities

\[ |x_{i+1} - x_i| \leq 2^{-(i+1)} \]

are witnessed by \(d_{(-)} : \mathbb{N} \to \mathbb{I}\) satisfying

\[ x_{i+1} - x_i \sim (d_i, -(i + 1)) \]

Then we define

\[ \lim_{i} x_i = x_0 + (\lim_i d_i, 0) \]
Question. Is every System T representable function \( \mathbb{R}^k \to \mathbb{R} \) representable in System I by a term \((I \times \mathbb{Z})^k \to I \times \mathbb{Z}\)?

Given that we can define rational polynomials and Cauchy limits, it is not implausible that a positive answer to the question might be obtained following a suitable constructive proof of the Stone-Weierstrass theorem.

Question. If we add \( I \) as an abstract datatype to dependent type theory, does this support the development of constructive analysis, based on the semi-extensional real numbers defined via mantissa-exponent?
Definability in System I revisited

Given that System I now seems more interesting than Part 2 suggested, we end by returning to questions of definability and undefinability in System I.

We address the following.

▶ All system I functions we have defined so far have been smooth.

   Are all I-definable functions smooth?

▶ How do we prove that \texttt{dbl} is not I-definable?

   And what does this tell us about I-definable functions more generally?
Non-smooth I-definable functions

\[
\begin{align*}
times^*(x, y) &= \text{aff} \left( -1 \right) \times y \\
sq^*(x) &= \times^*(x, x) \\
g(x) &= \times^* \left( \frac{7}{9}, \ sq^*(-sq^*(-x)) \right) \\
h(x) &= \bigwedge_i g^{3(i+1)}(x) \\
H(x) &= \bigwedge_i (g^{3(i+1)}(x))^2
\end{align*}
\]

N.B., \( g \) defines the function

\[
g(x) = \frac{1}{9}x^4 - \frac{4}{9}x^3 - \frac{2}{9}x^2 + \frac{4}{3}x
\]

which satisfies:

\[
g(0) = 0 \quad g'(0) = \frac{4}{3}
\]
- $h$ has derivative $+\infty$ at 0.
- $H$ at 0 has derivative $-\infty$ from below and $+\infty$ from above.

Graph of $H$ (thanks to Yitong Li):
Proposition. The function $\text{dbl}$ is not $I$-definable.

We prove this using logical relations

For every type $\tau$ we define a binary relation

$$\Delta_\tau \subseteq \llbracket \tau \rrbracket \times \llbracket \tau \rrbracket$$

by:

$$\Delta_N(m, n) \iff m = n$$
$$\Delta_I(x, y) \iff \text{if } x \in \{-1, 1\} \text{ or } y \in \{-1, 1\} \text{ then } x = y$$
$$\Delta_{\sigma \rightarrow \tau}(f, g) \iff \forall x, y \in \llbracket \sigma \rrbracket. \Delta_\sigma(x, y) \implies \Delta_\tau(f(x), g(y))$$
Lemma. For every System I constant $c : \tau$, it holds that $\Delta_\tau([c], [c])$.

E.g., to show $\Delta_{I \to I \to I}(m, m)$.

Suppose $\Delta_I(x, x')$ and $\Delta_I(y, y')$. \hfill (⋆)

Need to show $\Delta_I(x \oplus y, x' \oplus y')$.

Suppose $x \oplus y \in \{-1, 1\}$. W.l.o.g., assume it is 1.

Then $x = y = 1$. So $x' = 1$ and $y' = 1$, by (⋆).

Thus indeed $x' \oplus y' = 1 = x \oplus y$. And similarly for the case when $x' \oplus y' \in \{-1, 1\}$.
Lemma. For every System I constant $c : \tau$, it holds that $\Delta_\tau([c], [c])$.

E.g., to show $\Delta_{I \rightarrow I \rightarrow I \rightarrow I}(\text{aff}, \text{aff})$.

Suppose $\Delta_I(x, x')$ and $\Delta_I(y, y')$ and $\Delta_I(z, z')$.

Need to show $\Delta_I\left(\frac{(1-z)x+(1+z)y}{2}, \frac{(1-z')x'+(1+z')y'}{2}\right)$

Suppose $\frac{(1-z)x+(1+z)y}{2} \in \{-1, 1\}$. W.l.o.g., assume it is 1.

There are three possibilities: (i) $x = 1$ and $z = -1$; (ii) $y = z = 1$; (iii) $x = y = 1$.

In each case, the corresponding equations hold for $x', y', z'$, by $(\star)$.

Thus indeed $\frac{(1-z')x'+(1+z')y'}{2} = 1 = \frac{(1-z)x+(1+z)y}{2}$.

And similarly for the case when $x' \oplus y' \in \{-1, 1\}$.
Using the previous lemma, the fundamental lemma of logical relations gives us

**Lemma.** For every closed System I term \( t : \tau \), \( \Delta_\tau([t], [t]) \).
(The direct proof is by induction on term structure.)

But \( \Delta_{\rightarrow}(\text{dbl}, \text{dbl}) \) does not hold.

Indeed we have \( \Delta_{\rightarrow}(0, \frac{1}{2}) \) but not \( \Delta_{\rightarrow}(\text{dbl}(0), \text{dbl}(\frac{1}{2})) \)
(Since \( \text{dbl}(0) = 0 \) and \( \text{dbl}(\frac{1}{2}) = 1 \))

Therefore \( \text{dbl} \) is not \( \rightarrow \)-definable.
Using the previous lemma, the fundamental lemma of logical relations gives us

Lemma. For every closed System I term \( t : \tau \), \( \Delta_{\tau}([t], [t]) \).
(The direct proof is by induction on term structure.)

But \( \Delta_{I \rightarrow I}(\text{dbl}, \text{dbl}) \) does not hold.

Indeed we have \( \Delta_{I}(0, \frac{1}{2}) \) but not \( \Delta_{I}(\text{dbl}(0), \text{dbl}(\frac{1}{2})) \)
(Since \( \text{dbl}(0) = 0 \) and \( \text{dbl}(\frac{1}{2}) = 1 \))

Therefore \( \text{dbl} \) is not \( I \)-definable.

More generally, for any \( I \)-definable \( f : [-1, 1] \rightarrow [-1, 1] \), if
\( f(x) \in \{-1, 1\} \) for some \( x \in (0, 1) \), then \( f \) is a constant function.
Course summary

1. Gödel’s System T serves as a minimal functional programming language for implementing exact computation on real numbers via intensional representations.

2. Systems I and II implement an abstract datatype for a closed interval, offering an extensional approach to exact programming on the interval. System II is equivalent in power to the intensional approach.

3. A mantissa-exponent representation offers a semi-extensional approach to computing on the full real line. Apparently, the dbl function is not needed for this, so we can use System I.