Toward verified real computation

Sewon Park

Continuity, Computability, Constructivity From Logic to Algorithms 2021
September 23, 2021

KAIST, South Korea

Motivation: exact real computation

Motivation: exact real computation

- real numbers as primitive data type: the users do not worry about representations
- arithmetical operations $+, \times, -, {}^{-1}$ computed exactly: no rounding errors
- implementable: written programs are actually executable

Motivation: exact real computation

- real numbers as primitive data type: the users do not worry about representations
- arithmetical operations $+, \times, -, \ ^{-1}$ computed exactly: no rounding errors
- implementable: written programs are actually executable
- imperative: variables *store* real numbers exactly
  - E.g., `iRRAM` by Norbert Müller in C++

Motivation: exact real computation

- real numbers as primitive data type: the users do not worry about representations
- arithmetical operations $+, \times, -, \ ^{-1}$ computed exactly: no rounding errors
- implementable: written programs are actually executable
- imperative: variables *store* real numbers exactly
  - E.g., `iRRAM` by Norbert Müller in C++
- functional: lambda terms for constructing function objects
  - E.g., `AERN` by Michal Konečný in Haskell

Motivation: exact real computation

- real numbers as primitive data type: the users do not worry about representations
- arithmetical operations $+, \times, -, \ ^{-1}$ computed exactly: no rounding errors
- implementable: written programs are actually executable
- imperative: variables *store* real numbers exactly
  - E.g., `iRRAM` by Norbert Müller in C++
- functional: lambda terms for constructing function objects
  - E.g., `AERN` by Michal Konečný in Haskell
- how do we write *correct* programs in the languages?

Motivation: exact real computation

- real numbers as primitive data type: the users do not worry about representations
- arithmetical operations $+, \times, -, \ ^{-1}$ computed exactly: no rounding errors
- implementable: written programs are actually executable
- imperative: variables *store* real numbers exactly
  - E.g., `iRRAM` by Norbert Müller in C++
- functional: lambda terms for constructing function objects
  - E.g., `AERN` by Michal Konečný in Haskell
- how do we write *correct* programs in the languages?
- imperative: Hoare-style verification based on

Park, Sewon, Franz Brauße, Pieter Collins, SunYoung Kim, Michal Konečný, Gyesik Lee, Norbert Müller, Eike Neumann, Norbert Preining, and Martin Ziegler "Foundation of Computer (Algebra) ANALYSIS Systems: Semantics, Logic, Programming, Verification." arXiv preprint arXiv:1608.05787 (2021).

- functional: program extraction based on

Michal Knenčný, Sewon Park, and Holger Thies "Axiomatic Reals and Certified Efficient Exact Real Computation." 27th Workshop on Logic, Language, Information and Computation. Springer, 2021 (accepted)

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, *, /$ computed exactly

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, *, /$ computed exactly

Rump's polynomial:

$$R(x, y) = 21y^2 - 2x^2 + 55y^4 - 10x^2y^2 + x/(2y)$$

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, *, /$ computed exactly

Rump's polynomial:

$$R(x, y) = 21y^2 - 2x^2 + 55y^4 - 10x^2y^2 + x/(2y)$$

```
REAL Rump(REAL x, REAL y)

    REAL x2 = x * x;
    REAL y2 = y * y;
    REAL z = 21 * y2 - 2 * x2 + 55 * y2 * y2 - 10 * x2 * y2 + x / (2 * y);

    return z;
```

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, *, /$ computed exactly

Rump's polynomial:

$$R(x, y) = 21y^2 - 2x^2 + 55y^4 - 10x^2y^2 + x/(2y)$$

```
REAL Rump(REAL x, REAL y)
    {x, y ∈ ℝ}
    REAL x2 = x * x;
    REAL y2 = y * y;
    REAL z = 21 * y2 - 2 * x2 + 55 * y2 * y2 - 10 * x2 * y2 + x / (2 * y);
    {z = R(x, y)}
    return z;
```

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, *, /$ computed exactly

Rump's polynomial:

$$R(x, y) = 21y^2 - 2x^2 + 55y^4 - 10x^2y^2 + x/(2y)$$

```
REAL Rump(REAL x, REAL y)
    {x, y ∈ ℝ}
    REAL x2 = x * x;
    REAL y2 = y * y;
    REAL z = 21 * y2 - 2 * x2 + 55 * y2 * y2 - 10 * x2 * y2 + x / (2 * y);
    {z = R(x, y)}
    return z;
```

is this specification correct?

iRRAM Example 2

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)

    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }

    return n;
```

iRRAM Example 2

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)
    {x ∈ ℝ}
    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$

$$\text{For any } x \in \mathbb{R}, \text{ compute } n \in \mathbb{Z} \text{ such that } x < 2^n$$

```
int magnitude(REAL x)
    {x ∈ ℝ}
    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)
    {x ∈ ℝ ∧ ∀k ∈ ℕ. x ≠ 2^k}
    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)
    {x ∈ ℝ ∧ ∀k ∈ ℕ. x ≠ 2^k}
    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

is this specification correct?

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$
- nondeterminism $\mathsf{choose}(e_1, \cdots, e_n)$ evaluates to i such that e_i evaluates to `true`
  If non evaluates to `true`, the expression freezes

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)
    {x ∈ ℝ ∧ ∀k ∈ ℕ. x ≠ 2^k}
    REAL y = 1;
    int n = 0;
    while (y < x){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

is this specification correct?

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$
- nondeterminism choose$(e_1, \cdots, e_n)$ evaluates to i such that e_i evaluates to `true`
  If non evaluates to `true`, the expression freezes

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

```
int magnitude(REAL x)
    {x ∈ ℝ}
    REAL y = 1;
    int n = 0;
    while (choose(y < x + 1, x < y) = 1){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

is this specification correct?

iRRAM Example 2

- comparing real numbers $x < y$ *freezes* when $x = y$
- nondeterminism $\mathsf{choose}(e_1, \cdots, e_n)$ evaluates to i such that e_i evaluates to `true`
  If non evaluates to `true`, the expression freezes

$$\text{For any } x \in \mathbb{R}, \text{ compute } n \in \mathbb{Z} \text{ such that } x < 2^n$$

```
int magnitude(REAL x)
    {x ∈ ℝ}
    REAL y = 1;
    int n = 0;
    while (choose(y < x + 1, x < y) = 1){
        y = y * 2;
        n = n + 1;
    }
    {x < 2^n}
    return n;
```

- y goes $1, 2, 2^2, 2^3, \cdots$. Throughout the loop, $y = 2^n$
- for any y, at least one of two tests $y < x + 1$ and $x < y$ hold
- at some point, $y < x + 1$ must evaluate to `false`. Hence, the loop gets escaped
- when the loop is escaped, $x < y = 2^n$ holds.

Our Goal

A systematic way to verify the correctness of program specifications.

- Design a simple imperative language that can model core fragments of real number computation languages including nondeterministic choose
- Computable semantics in the sense of computable analysis (exact arithmetical operations and partial comparison)
- Convenient-to-use precondition-postcondition-style program verification
- Some thoughts on extending it with further continuous objects

Language Design

Syntax

While-language based on Peano Arithmetic and Boolean logic:

data types	τ	$::=$	B	Boolean
		$\mid$	Z	integer
expressions	e	$::=$	$\texttt{true} \mid \texttt{false} \mid 0 \mid 1 \mid \cdots$	constant
		$\mid$	$e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$	integer arithmetic
		$\mid$	$e_1 = e_2 \mid e_1 \leq e_2$	integer comparison
commands	c	$::=$	$\texttt{skip}$	do nothing
		$\mid$	$c_1 ; c_2$	composition
		$\mid$	$x := e$	assignment
		$\mid$	$\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}$	conditional
		$\mid$	$\texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}$	loop

Syntax

While-language based on Peano Arithmetic and Boolean logic:

$$
\begin{array}{llcll}
\text{data types} & \tau & ::= & \mathsf{B} & \text{Boolean} \\
 & & \mid & \mathsf{Z} & \text{integer} \\
 & & \mid & \mathsf{R} & \text{real number} \\
\text{expressions} & e & ::= & \mathtt{true} \mid \mathtt{false} \mid 0 \mid 1 \mid \cdots & \text{constant} \\
 & & \mid & e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 & \text{integer arithmetic} \\
 & & \mid & e_1 = e_2 \mid e_1 \leq e_2 & \text{integer comparison} \\
 & & \mid & e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 \mid e^{-1} & \text{real arithmetic} \\
 & & \mid & e_1 \lesssim e_2 & \text{partial real comparison} \\
 & & \mid & \mathsf{choose}(e_1, \cdots, e_n) & \text{nondeterminism} \\
 & & \mid & 2^e \mid \iota(e) & \text{coercion from } \mathsf{Z} \text{ to } \mathsf{R} \\
\text{commands} & c & ::= & \mathtt{skip} & \text{do nothing} \\
 & & \mid & c_1 ; c_2 & \text{composition} \\
 & & \mid & x := e & \text{assignment} \\
 & & \mid & \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end} & \text{conditional} \\
 & & \mid & \mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end} & \text{loop}
\end{array}
$$

Denotational Semantics

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \bot$$

Denotational Semantics

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \bot$$

4. Denotation is the set of all nondeterministic evaluations:

$$[\![\mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma = \{1, 2\}$$

Denotational Semantics

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \ \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \ \bot$$

4. Denotation is the set of all nondeterministic evaluations:

$$\begin{aligned} [\![\mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma &= \{1, 2\} \\ [\![\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1]\!]\gamma &= \{0, 1\} \end{aligned}$$

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \bot$$

4. Denotation is the set of all nondeterministic evaluations:

$$\begin{aligned}
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma &= \{1, 2\} \\
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1]\!]\gamma &= \{0, 1\} \\
[\![(\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1)^{-1}]\!]\gamma &= \{\bot, 1\}
\end{aligned}$$

Denotational Semantics

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \bot$$

4. Denotation is the set of all nondeterministic evaluations:

$$\begin{aligned}
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma &= \{1, 2\} \\
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1]\!]\gamma &= \{0, 1\} \\
[\![(\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1)^{-1}]\!]\gamma &= \{\bot, 1\}
\end{aligned}$$

5. Denotation of a command is the set of all nondeterministic resulting states

$$[\![x := \mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma = \{\gamma[x \mapsto 1], \gamma[x \mapsto 2]\}$$

Denotational Semantics

1. Given a state γ, an *expression* evaluates to a value:

$$x + y \rightsquigarrow_\gamma 42 \quad \text{when } x \rightsquigarrow_\gamma 21 \text{ and } y \rightsquigarrow_\gamma 21$$

2. Due to *nondeterminism*, there are several possible evaluations:

$$\mathsf{choose}(\mathtt{true}, \mathtt{true}) \rightsquigarrow_\gamma 1 \text{ or } 2$$

3. There are nonterminating evaluations

$$0^{-1} \rightsquigarrow_\gamma \bot \qquad \pi \lesssim \pi \rightsquigarrow_\gamma \bot$$

4. Denotation is the set of all nondeterministic evaluations:

$$\begin{aligned}
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma &= \{1, 2\} \\
[\![\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1]\!]\gamma &= \{0, 1\} \\
[\![(\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1)^{-1}]\!]\gamma &= \{\bot, 1\}
\end{aligned}$$

5. Denotation of a command is the set of all nondeterministic resulting states

$$\begin{aligned}
[\![x := \mathsf{choose}(\mathtt{true}, \mathtt{true})]\!]\gamma &= \{\gamma[x \mapsto 1], \gamma[x \mapsto 2]\} \\
[\![x := (\mathsf{choose}(\mathtt{true}, \mathtt{true}) - 1)^{-1}]\!]\gamma &= \{\bot, \gamma[x \mapsto 1]\}
\end{aligned}$$

Denotations of expressions

$$[\![e]\!]\gamma = \text{the set of all possible evaluations of } e \text{ under } \gamma$$

Denotations of expressions

$$[\![e]\!]\gamma = \text{the set of all possible evaluations of } e \text{ under } \gamma$$

$$[\![e_1 \lesssim e_2]\!]\gamma = \bigcup_{x \in [\![e_1]\!]\gamma, y \in [\![e_2]\!]\gamma} \begin{cases} \{\texttt{true}\} & \text{if } x < y, \\ \{\texttt{false}\} & \text{if } y < x, \\ \{\bot\} & \text{if } x = y \text{ or } x = \bot \text{ or } y = \bot. \end{cases}$$

$$[\![\mathsf{choose}(e_1, \cdots, e_n)]\!]\gamma = \{i \mid \texttt{true} \in [\![e_i]\!]\gamma\} \cup \{\bot \mid \forall i.\ \bot \text{ or } \texttt{false} \in [\![e_i]\!]\gamma\}$$

Denotational Semantics

Denotations of expressions

$$[\![e]\!]\gamma = \text{the set of all possible evaluations of } e \text{ under } \gamma$$

$$[\![e_1 \lesssim e_2]\!]\gamma = \bigcup_{x \in [\![e_1]\!]\gamma, y \in [\![e_2]\!]\gamma} \begin{cases} \{\texttt{true}\} & \text{if } x < y, \\ \{\texttt{false}\} & \text{if } y < x, \\ \{\bot\} & \text{if } x = y \text{ or } x = \bot \text{ or } y = \bot. \end{cases}$$

$$[\![\mathsf{choose}(e_1, \cdots, e_n)]\!]\gamma = \{i \mid \texttt{true} \in [\![e_i]\!]\gamma\} \cup \{\bot \mid \forall i.\ \bot \text{ or } \texttt{false} \in [\![e_i]\!]\gamma\}$$

Denotations of commands

$$[\![c]\!]\gamma = \text{the set of possible resulting states of executing } c \text{ under } \gamma$$

Denotations of expressions

$$[\![e]\!]\gamma = \text{the set of all possible evaluations of } e \text{ under } \gamma$$

$$[\![e_1 \lesssim e_2]\!]\gamma = \bigcup_{x \in [\![e_1]\!]\gamma, y \in [\![e_2]\!]\gamma} \begin{cases} \{\texttt{true}\} & \text{if } x < y, \\ \{\texttt{false}\} & \text{if } y < x, \\ \{\bot\} & \text{if } x = y \text{ or } x = \bot \text{ or } y = \bot. \end{cases}$$

$$[\![\mathsf{choose}(e_1, \cdots, e_n)]\!]\gamma = \{i \mid \texttt{true} \in [\![e_i]\!]\gamma\} \cup \{\bot \mid \forall i.\ \bot \text{ or } \texttt{false} \in [\![e_i]\!]\gamma\}$$

Denotations of commands

$$[\![c]\!]\gamma = \text{the set of possible resulting states of executing } c \text{ under } \gamma$$

$$[\![\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}]\!]\gamma = \bigcup_{b \in [\![e]\!]\gamma} \begin{cases} [\![c_1]\!]\gamma & \text{if } b = \texttt{true} \\ [\![c_2]\!]\gamma & \text{if } b = \texttt{false} \\ \{\bot\} & \text{if } b = \bot \end{cases}$$

$$[\![\texttt{while } e \texttt{ do } c \texttt{ end}]\!]\gamma = \text{the least fixed point of some operator w.r.t. Plotkin powerdomain}$$

Denotations of expressions

$$[\![e]\!]\gamma = \text{the set of all possible evaluations of } e \text{ under } \gamma$$

$$[\![e_1 \lesssim e_2]\!]\gamma = \bigcup_{x \in [\![e_1]\!]\gamma, y \in [\![e_2]\!]\gamma} \begin{cases} \{\mathtt{true}\} & \text{if } x < y, \\ \{\mathtt{false}\} & \text{if } y < x, \\ \{\bot\} & \text{if } x = y \text{ or } x = \bot \text{ or } y = \bot. \end{cases}$$

$$[\![\mathsf{choose}(e_1, \cdots, e_n)]\!]\gamma = \{i \mid \mathtt{true} \in [\![e_i]\!]\gamma\} \cup \{\bot \mid \forall i.\ \bot \text{ or } \mathtt{false} \in [\![e_i]\!]\gamma\}$$

Denotations of commands

$$[\![c]\!]\gamma = \text{the set of possible resulting states of executing } c \text{ under } \gamma$$

$$[\![\mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end}]\!]\gamma = \bigcup_{b \in [\![e]\!]\gamma} \begin{cases} [\![c_1]\!]\gamma & \text{if } b = \mathtt{true} \\ [\![c_2]\!]\gamma & \text{if } b = \mathtt{false} \\ \{\bot\} & \text{if } b = \bot \end{cases}$$

$$[\![\mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end}]\!]\gamma = \text{the least fixed point of some operator w.r.t. Plotkin powerdomain}$$

Theorem
The language is approximately complete in that to any computable (partial) real function, there is a program that rigorously approximates it.

Formal Verification

Specification

$$\{\phi\}\ c\ \{\psi\}$$

All states satisfying ϕ make c terminate and results states satisfying ψ.

$$\{\phi\}\ c\ \{\psi\}$$

All states satisfying ϕ make c terminate and results states satisfying ψ.

Theorem

The first order logic over the structures of integers and reals connected via $\mathbb{Z} \ni z \mapsto z \in \mathbb{R}$ and $\mathbb{Z} \ni z \mapsto 2^z \in \mathbb{R}$ is expressive for the expression language;

for any expression e, there is a predicate $(\!|e|\!)(y)$ that defines the denotation of e

$$(\!|e|\!)(v) \textit{ iff } v \textit{ is in (but } \bot \textit{ is not in) the denotation of } e$$

$$\{\phi\}\ c\ \{\psi\}$$

All states satisfying ϕ make c terminate and results states satisfying ψ.

Theorem
The first order logic over the structures of integers and reals connected via $\mathbb{Z} \ni z \mapsto z \in \mathbb{R}$ and $\mathbb{Z} \ni z \mapsto 2^z \in \mathbb{R}$ is expressive for the expression language;

for any expression e, there is a predicate $(\!|e|\!)(y)$ that defines the denotation of e

$$(\!|e|\!)(v) \textit{ iff } v \textit{ is in (but } \bot \textit{ is not in) the denotation of } e$$

For example, (in the simplified form)

$$(\!|x \lesssim y|\!)(v) \equiv (v = \mathtt{true} \land x < y) \lor (v = \mathtt{false} \land y < x)$$
$$(\!|\mathsf{choose}_n(e_1, \cdots, e_n)|\!)(v) \equiv (v = 1 \land (\!|e_1|\!)(\mathtt{true})) \lor \cdots \lor (v = n \land (\!|e_n|\!)(\mathtt{true}))$$

Specification

$$\{\phi\}\ c\ \{\psi\}$$

All states satisfying ϕ make c terminate and results states satisfying ψ.

Theorem
The first order logic over the structures of integers and reals connected via $\mathbb{Z} \ni z \mapsto z \in \mathbb{R}$ and $\mathbb{Z} \ni z \mapsto 2^z \in \mathbb{R}$ is expressive for the expression language;

for any expression e, there is a predicate $(\!|e|\!)(y)$ that defines the denotation of e

$$(\!|e|\!)(v) \textit{ iff } v \textit{ is in (but } \bot \textit{ is not in) the denotation of } e$$

For example, (in the simplified form)

$$(\!|x \lesssim y|\!)(v) \equiv (v = \mathtt{true} \land x < y) \lor (v = \mathtt{false} \land y < x)$$
$$(\!|\mathsf{choose}_n(e_1, \cdots, e_n)|\!)(v) \equiv (v = 1 \land (\!|e_1|\!)(\mathtt{true})) \lor \cdots \lor (v = n \land (\!|e_n|\!)(\mathtt{true}))$$

$$\begin{aligned} \exists v.\ (\!|e|\!)(v) \quad &\text{iff} \quad \bot \notin [\![e]\!]\gamma \\ \forall v.\ (\!|e|\!)(v) \Rightarrow \psi(v) \quad &\text{iff} \quad \forall v \in [\![e]\!]\gamma \text{ satisfy } \psi \end{aligned}$$

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \land \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\bot \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \land \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\bot \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Example:

$$b := x \lesssim y\ \left\{b = \mathtt{true}\right\}$$

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \wedge \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\perp \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Example:

$$\left\{(\exists v.\ (\!|x \lesssim y|\!)(v)) \wedge \forall v.\ (\!|x \lesssim y|\!)(v) \Rightarrow (b = \texttt{true})[v/b]\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \land \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\bot \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Example:

$$\left\{(\exists v.\ (\!|x \lesssim y|\!)(v)) \land \forall v.\ (\!|x \lesssim y|\!)(v) \Rightarrow (b = \texttt{true})[v/b]\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

$$\begin{aligned}&\left\{(\exists v.\ (v = \texttt{true} \land x < y) \lor (v = \texttt{false} \land y < x))\right.\\ &\quad \left.\land \forall v.\ ((v = \texttt{true} \land x < y) \lor (v = \texttt{false} \land y < x)) \Rightarrow v = \texttt{true}\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}\end{aligned}$$

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \land \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\bot \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Example:

$$\left\{(\exists v.\ (\!|x \lesssim y|\!)(v)) \land \forall v.\ (\!|x \lesssim y|\!)(v) \Rightarrow (b = \mathtt{true})[v/b]\right\}\ b := x \lesssim y\ \left\{b = \mathtt{true}\right\}$$

$$\left\{(\exists v.\ (v = \mathtt{true} \land x < y) \lor (v = \mathtt{false} \land y < x))\right.$$
$$\left.\land \forall v.\ ((v = \mathtt{true} \land x < y) \lor (v = \mathtt{false} \land y < x)) \Rightarrow v = \mathtt{true}\right\}\ b := x \lesssim y\ \left\{b = \mathtt{true}\right\}$$

$$\left\{(x < y \lor y < x) \land x \leq y\right\}\ b := x \lesssim y\ \left\{b = \mathtt{true}\right\}$$

Proof Rule: assignment

$$\frac{}{\left\{\exists v.\ (\!|e|\!)(v) \wedge \forall v.\ (\!|e|\!)(v) \Rightarrow \psi[v/x]\right\}\ x := e\ \left\{\psi\right\}}$$

For an initial state γ,

- γ satisfying $\exists v.\ (\!|e|\!)(v)$ ensures $\bot \notin [\![e]\!]\gamma$
- Resulting state is $\gamma[x \mapsto v]$ for each $v \in [\![e]\!]\gamma$
- If γ satisfies $\forall v.\ (\!|s|\!)(v) \Rightarrow \psi[v/x]$, any $v \in [\![e]\!]\gamma$, γ satisfies $\psi[v/x]$
- Hence, resulting states satisfy ψ

Example:

$$\left\{(\exists v.\ (\!|x \lesssim y|\!)(v)) \wedge \forall v.\ (\!|x \lesssim y|\!)(v) \Rightarrow (b = \texttt{true})[v/b]\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

$$\left\{(\exists v.\ (v = \texttt{true} \wedge x < y) \vee (v = \texttt{false} \wedge y < x))\right.$$
$$\left.\wedge \forall v.\ ((v = \texttt{true} \wedge x < y) \vee (v = \texttt{false} \wedge y < x)) \Rightarrow v = \texttt{true}\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

$$\left\{(x < y \vee y < x) \wedge x \leq y\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

$$\left\{x < y\right\}\ b := x \lesssim y\ \left\{b = \texttt{true}\right\}$$

Proof Rule: while loop

$$\frac{\left\{ (\!|e|\!)(\texttt{true}) \wedge I \wedge V = \xi \wedge L = \xi' \right\}\ c\ \left\{ I \wedge V \leq \xi - \xi' \wedge L = \xi' \right\}}{\left\{ I \right\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \left\{ I \wedge (\!|e|\!)(\texttt{false}) \right\}}$$

- $I \wedge (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \vee (\!|e|\!)(\texttt{false}))$
- $I \wedge V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

Proof Rule: while loop

$$\frac{\left\{ (\!|e|\!)(\texttt{true}) \wedge I \wedge V = \xi \wedge L = \xi' \right\}\, c \,\left\{ I \wedge V \leq \xi - \xi' \wedge L = \xi' \right\}}{\left\{ I \right\}\, \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\, \left\{ I \wedge (\!|e|\!)(\texttt{false}) \right\}}$$

- $I \wedge (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \vee (\!|e|\!)(\texttt{false}))$
- $I \wedge V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

- $(\!|e|\!)(\texttt{true})$ ensures $e \rightsquigarrow \texttt{true}$ and $(\!|e|\!)(\texttt{false})$ ensures $e \rightsquigarrow \texttt{false}$

Proof Rule: while loop

$$\frac{\left\{(\!|e|\!)(\texttt{true}) \land I \land V = \xi \land L = \xi'\right\}\ c\ \left\{I \land V \leq \xi - \xi' \land L = \xi'\right\}}{\left\{I\right\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \left\{I \land (\!|e|\!)(\texttt{false})\right\}}$$

- $I \land (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \lor (\!|e|\!)(\texttt{false}))$
- $I \land V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

- $(\!|e|\!)(\texttt{true})$ ensures $e \rightsquigarrow \texttt{true}$ and $(\!|e|\!)(\texttt{false})$ ensures $e \rightsquigarrow \texttt{false}$
- I is a loop-invariant formula, L is a loop-invariant quantity, and V is a loop-variant quantity

Proof Rule: while loop

$$\frac{\left\{(\!|e|\!)(\texttt{true}) \wedge I \wedge V = \xi \wedge L = \xi'\right\}\ c\ \left\{I \wedge V \leq \xi - \xi' \wedge L = \xi'\right\}}{\left\{I\right\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \left\{I \wedge (\!|e|\!)(\texttt{false})\right\}}$$

- $I \wedge (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \vee (\!|e|\!)(\texttt{false}))$
- $I \wedge V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

- $(\!|e|\!)(\texttt{true})$ ensures $e \rightsquigarrow \texttt{true}$ and $(\!|e|\!)(\texttt{false})$ ensures $e \rightsquigarrow \texttt{false}$
- I is a loop-invariant formula, L is a loop-invariant quantity, and V is a loop-variant quantity
- L bounds the decrement of V in each iteration

Proof Rule: while loop

$$\frac{\left\{(\!|e|\!)(\texttt{true}) \land I \land V = \xi \land L = \xi'\right\}\ c\ \left\{I \land V \leq \xi - \xi' \land L = \xi'\right\}}{\left\{I\right\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \left\{I \land (\!|e|\!)(\texttt{false})\right\}}$$

- $I \land (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \lor (\!|e|\!)(\texttt{false}))$
- $I \land V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

- $(\!|e|\!)(\texttt{true})$ ensures $e \rightsquigarrow \texttt{true}$ and $(\!|e|\!)(\texttt{false})$ ensures $e \rightsquigarrow \texttt{false}$
- I is a loop-invariant formula, L is a loop-invariant quantity, and V is a loop-variant quantity
- L bounds the decrement of V in each iteration
- Side-conditions ensure
    1. When I holds and $e \rightsquigarrow \texttt{true}$, L is positive
    2. When I holds, $e \rightsquigarrow \bot$ does not happen
    3. When I holds and V is negative, e evaluates only to `false`

Proof Rule: while loop

$$\frac{\left\{ (\!|e|\!)(\texttt{true}) \wedge I \wedge V = \xi \wedge L = \xi' \right\} \; c \; \left\{ I \wedge V \leq \xi - \xi' \wedge L = \xi' \right\}}{\left\{ I \right\} \; \texttt{while } e \texttt{ do } c \texttt{ end} \; \left\{ I \wedge (\!|e|\!)(\texttt{false}) \right\}}$$

- $I \wedge (\!|e|\!)(\texttt{true}) \Rightarrow L > 0$
- $I \Rightarrow ((\!|e|\!)(\texttt{true}) \vee (\!|e|\!)(\texttt{false}))$
- $I \wedge V \leq 0 \Rightarrow \forall k.\ (\!|e|\!)(k) \Rightarrow k = \texttt{false}$

- $(\!|e|\!)(\texttt{true})$ ensures $e \rightsquigarrow \texttt{true}$ and $(\!|e|\!)(\texttt{false})$ ensures $e \rightsquigarrow \texttt{false}$
- I is a loop-invariant formula, L is a loop-invariant quantity, and V is a loop-variant quantity
- L bounds the decrement of V in each iteration
- Side-conditions ensure
  1. When I holds and $e \rightsquigarrow \texttt{true}$, L is positive
  2. When I holds, $e \rightsquigarrow \perp$ does not happen
  3. When I holds and V is negative, e evaluates only to `false`

Theorem
The proof rules are sound w.r.t. the denotational semantics.

Verification Condition

annotated commands	c	$::=$	`skip`	do nothing
		$\mid$	$c_1 ; c_2$	composition
		$\mid$	$x := e$	assignment
		$\mid$	`if` e `then` c_1 `else` c_2 `end`	conditional
		$\mid$	$\{I, V, L\}$`while` e `do` c `end`	loop

$$
\begin{array}{llcll}
\text{annotated commands} & c & ::= & \texttt{skip} & \text{do nothing} \\
 & & | & c_1 ; c_2 & \text{composition} \\
 & & | & x := e & \text{assignment} \\
 & & | & \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end} & \text{conditional} \\
 & & | & \{I, V, L\}\texttt{while } e \texttt{ do } c \texttt{ end} & \text{loop}
\end{array}
$$

Theorem
For c and a postcondition ψ, there is a precondition $\texttt{vc}(c, \psi)$

$$\left\{\texttt{vc}(c, \psi)\right\} \ c \ \left\{\psi\right\}$$

In order to show

$$\left\{\phi\right\} \ c \ \left\{\psi\right\}$$

it suffices to show

$$\phi \Rightarrow \texttt{vc}(c, \psi)$$

Verification Condition Example

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

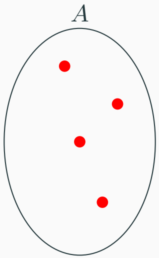
```
int magnitude(REAL x){
    {x ∈ ℝ}
    let y := 1;
    let n := 0;

    while choose(y < x + 1, x < y) = 1
    do
        y = y × 2;
        n = n + 1
    end
    {x < 2^n}
    return n
}
```

Verification Condition Example

For any $x \in \mathbb{R}$, compute $n \in \mathbb{Z}$ such that $x < 2^n$

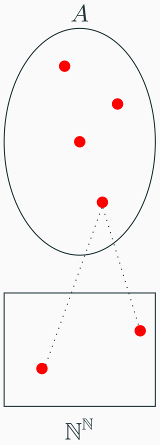
```
int magnitude(REAL x){
    {x ∈ ℝ}
    let y := 1;
    let n := 0;
    {I ≡ y = 2^n ∧ n ≥ 0     V ≡ x − y + 1     L ≡ 1}
    while choose(y < x + 1, x < y) = 1
    do
        y = y × 2;
        n = n + 1
    end
    {x < 2^n}
    return n
}
```

Extension



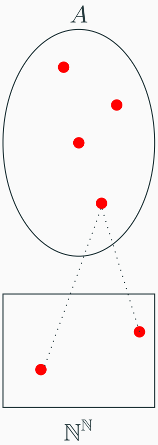
Definition
An assembly is a pair $(A, \Vdash_A)$ of a set A and a relation $\Vdash_A \subseteq \mathbb{N}^{\mathbb{N}} \times A$ such that

$$\forall x \in A \,.\, \exists \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x$$



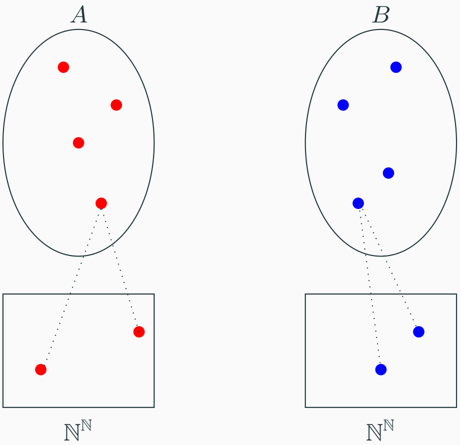
Definition
An assembly is a pair $(A, \Vdash_A)$ of a set A and a relation $\Vdash_A \subseteq \mathbb{N}^{\mathbb{N}} \times A$ such that

$$\forall x \in A \,.\, \exists \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x$$



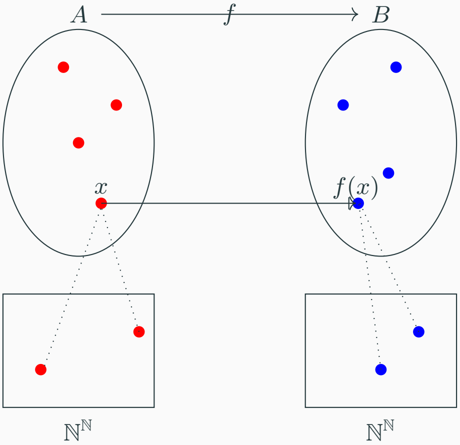
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

$$\forall x \in A \,.\, \forall \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x \Rightarrow \tau(\varphi) \Vdash_B f(x)$$



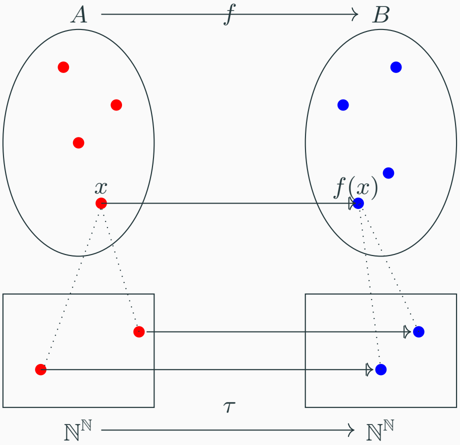
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

$$\forall x \in A \,.\, \forall \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x \Rightarrow \tau(\varphi) \Vdash_B f(x)$$



Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

$$\forall x \in A \,.\, \forall \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x \Rightarrow \tau(\varphi) \Vdash_B f(x)$$



Definition

A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

$$\forall x \in A \,.\, \forall \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x \Rightarrow \tau(\varphi) \Vdash_B f(x)$$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- There are monads $\natural, \flat, \sharp, \mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\flat A = A \cup \{\flat\}, \;\; \sharp A = A \cup \{\sharp\}, \;\; \natural A = A \cup \{\natural\}, \;\; \mathsf{M}A = \text{power-set of } A$$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- There are monads $\natural, \flat, \sharp, \mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\flat A = A \cup \{\flat\}, \;\; \sharp A = A \cup \{\sharp\}, \;\; \natural A = A \cup \{\natural\}, \;\; \mathsf{M}A = \text{power-set of } A$$

- $\flat$ classifies partial functions which diverge outside of their domains

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \flat\{\mathtt{true}, \mathtt{false}\} \qquad \text{s.t.} \qquad \pi \lesssim \pi = \flat$$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- There are monads $\natural, \flat, \sharp, \mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\flat A = A \cup \{\flat\}, \quad \sharp A = A \cup \{\sharp\}, \quad \natural A = A \cup \{\natural\}, \quad \mathsf{M}A = \text{power-set of } A$$

- $\flat$ classifies partial functions which diverge outside of their domains

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \flat\{\texttt{true}, \texttt{false}\} \qquad \text{s.t.} \qquad \pi \lesssim \pi = \flat$$

- $\sharp$ classifies partial functions which correct outside of their domains

$$\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \to \sharp\mathbb{R} \qquad \text{s.t.} \qquad \mathsf{lim}(0, 1, 2, 3, \cdots) = \sharp$$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- There are monads $\natural, \flat, \sharp, \mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\flat A = A \cup \{\flat\}, \quad \sharp A = A \cup \{\sharp\}, \quad \natural A = A \cup \{\natural\}, \quad \mathsf{M}A = \text{power-set of } A$$

- $\flat$ classifies partial functions which diverge outside of their domains

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \flat\{\mathtt{true}, \mathtt{false}\} \qquad \text{s.t.} \qquad \pi \lesssim \pi = \flat$$

- $\sharp$ classifies partial functions which correct outside of their domains

$$\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \to \sharp\mathbb{R} \qquad \text{s.t.} \qquad \mathsf{lim}(0, 1, 2, 3, \cdots) = \sharp$$

- $\natural$ classifies general partial functions without any out-of-domain specification

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \natural\{\mathtt{true}, \mathtt{false}\} \qquad \text{and} \qquad \mathsf{lim} : \mathbb{R}^{\mathbb{N}} \to \natural\mathbb{R}$$

Monads

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- There are monads $\natural, \flat, \sharp, \mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\flat A = A \cup \{\flat\}, \quad \sharp A = A \cup \{\sharp\}, \quad \natural A = A \cup \{\natural\}, \quad \mathsf{M}A = \text{power-set of } A$$

- $\flat$ classifies partial functions which diverge outside of their domains

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \flat\{\mathtt{true}, \mathtt{false}\} \qquad \text{s.t.} \qquad \pi \lesssim \pi = \flat$$

- $\sharp$ classifies partial functions which correct outside of their domains

$$\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \to \sharp\mathbb{R} \qquad \text{s.t.} \qquad \mathsf{lim}(0, 1, 2, 3, \cdots) = \sharp$$

- $\natural$ classifies general partial functions without any out-of-domain specification

$$\lesssim : \mathbb{R} \times \mathbb{R} \to \natural\{\mathtt{true}, \mathtt{false}\} \qquad \text{and} \qquad \mathsf{lim} : \mathbb{R}^{\mathbb{N}} \to \natural\mathbb{R}$$

- M defines nondeterministic functions

$$\lesssim_n : \mathbb{R} \times \mathbb{R} \to \mathsf{M}\{\mathtt{true}, \mathtt{false}\} \qquad \text{s.t.} \qquad \pi \lesssim_n \pi = \{\mathtt{true}, \mathtt{false}\}$$

Interpretation

Denotational Semantics

In Set:

Denotational Semantics

In Set:

$$\bullet \; [\![\tau]\!]$$

- Denotation of data type $[\![\tau]\!]$

Denotational Semantics

In Set:

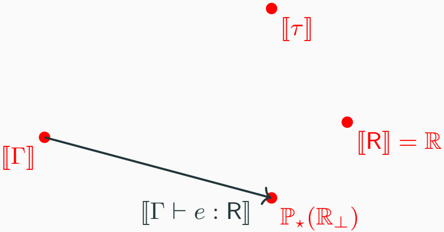
$[\![\tau]\!]$

$[\![\mathsf{R}]\!] = \mathbb{R}$

- Denotation of data type $[\![\tau]\!]$

Denotational Semantics

In Set:



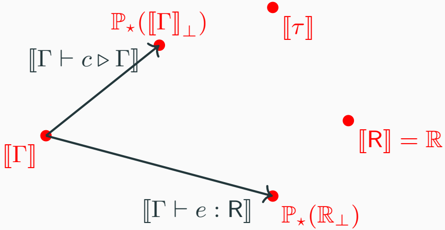
- Denotation of data type $[\![\tau]\!]$
- Denotation of a context $[\![\Gamma]\!]$ of all states
- Denotation of a expression

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathbb{P}_\star([\![\tau]\!]_\perp)$$

Interpretation

Denotational Semantics

In Set:

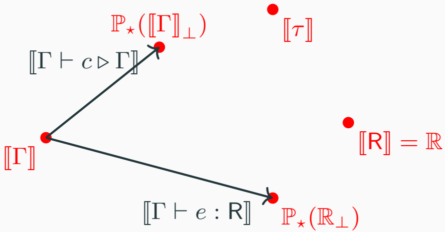


- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

Interpretation

Denotational Semantics

In Set:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

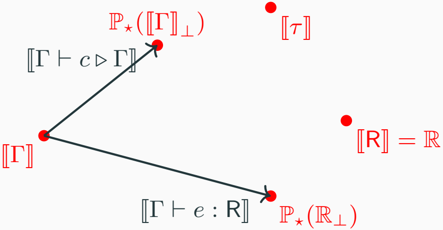
Interpretation

In $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$:

Interpretation

Denotational Semantics

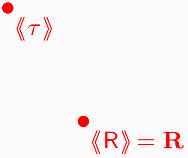
In Set:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

Interpretation

In $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$:

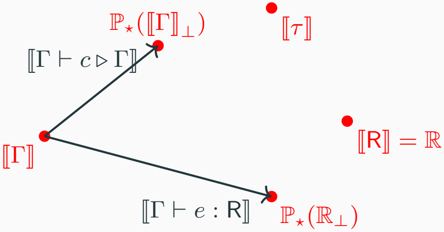


- Interpretation of data type $\langle\!\langle\tau\rangle\!\rangle$

Interpretation

Denotational Semantics

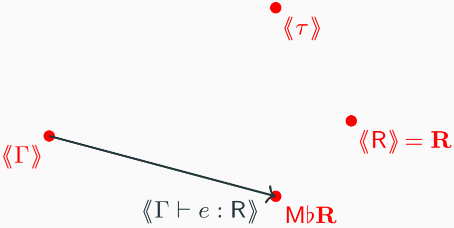
In Set:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

Interpretation

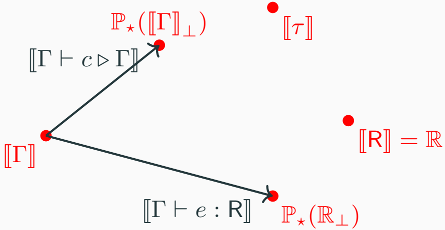
In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Interpretation of data type $\langle\!\langle\tau\rangle\!\rangle$
- Interpretation of a context $\langle\!\langle\Gamma\rangle\!\rangle$ of all states
- Interpretation of a expression
$$\langle\!\langle\Gamma \vdash e : \tau\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\tau\rangle\!\rangle$$

Denotational Semantics

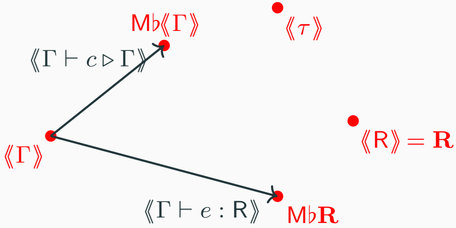
In Set:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
  $$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\bot)$$
- Denotation of a command
  $$\llbracket\Gamma \vdash c \rhd \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\bot)$$

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



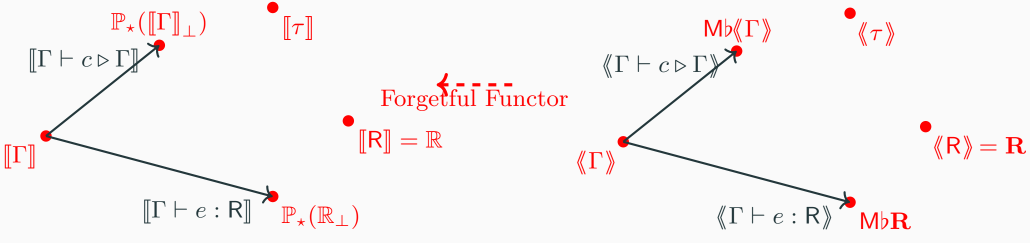
- Interpretation of data type $\langle\!\langle\tau\rangle\!\rangle$
- Interpretation of a context $\langle\!\langle\Gamma\rangle\!\rangle$ of all states
- Interpretation of a expression
  $$\langle\!\langle\Gamma \vdash e : \tau\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\tau\rangle\!\rangle$$
- Interpretation of a command
  $$\langle\!\langle\Gamma \vdash e \rhd \Gamma\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\Gamma\rangle\!\rangle$$

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

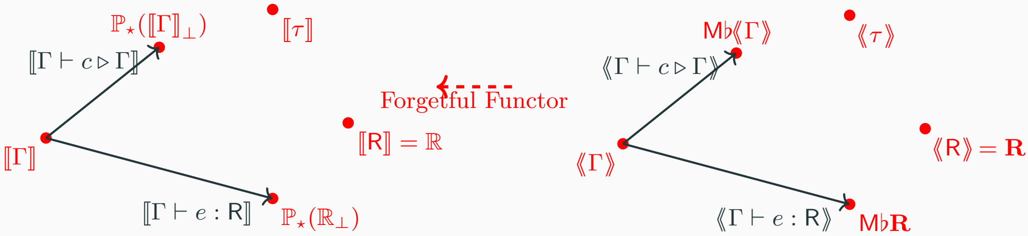
- Interpretation of data type $\langle\!\langle\tau\rangle\!\rangle$
- Interpretation of a context $\langle\!\langle\Gamma\rangle\!\rangle$ of all states
- Interpretation of a expression
$$\langle\!\langle\Gamma \vdash e : \tau\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\tau\rangle\!\rangle$$
- Interpretation of a command
$$\langle\!\langle\Gamma \vdash e \triangleright \Gamma\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\Gamma\rangle\!\rangle$$

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Denotation of data type $\llbracket\tau\rrbracket$
- Denotation of a context $\llbracket\Gamma\rrbracket$ of all states
- Denotation of a expression
$$\llbracket\Gamma \vdash e : \tau\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\tau\rrbracket_\perp)$$
- Denotation of a command
$$\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket : \llbracket\Gamma\rrbracket \to \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp)$$

- Interpretation of data type $\langle\!\langle\tau\rangle\!\rangle$
- Interpretation of a context $\langle\!\langle\Gamma\rangle\!\rangle$ of all states
- Interpretation of a expression
$$\langle\!\langle\Gamma \vdash e : \tau\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\tau\rangle\!\rangle$$
- Interpretation of a command
$$\langle\!\langle\Gamma \vdash e \triangleright \Gamma\rangle\!\rangle : \langle\!\langle\Gamma\rangle\!\rangle \to \mathsf{M}\flat\langle\!\langle\Gamma\rangle\!\rangle$$

Theorem
The denotational semantics is computable.

Extension

Denotational Semantics

In Set:

$$\begin{tikzcd} & \mathbb{P}_\star(\llbracket\Gamma\rrbracket_\perp) & \llbracket\tau\rrbracket \\ \llbracket\Gamma\rrbracket \arrow[ur, "\llbracket\Gamma \vdash c \triangleright \Gamma\rrbracket"] \arrow[rr, "\llbracket\Gamma \vdash e : \mathsf{R}\rrbracket"'] & & \mathbb{P}_\star(\mathbb{R}_\perp) \end{tikzcd}$$

$\llbracket\mathsf{R}\rrbracket = \mathbb{R}$

$\xleftarrow{\text{Forgetful Functor}}$

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:

$$\begin{tikzcd} & \mathsf{M}\flat\langle\!\langle\Gamma\rangle\!\rangle & \langle\!\langle\tau\rangle\!\rangle \\ \langle\!\langle\Gamma\rangle\!\rangle \arrow[ur, "\langle\!\langle\Gamma \vdash c \triangleright \Gamma\rangle\!\rangle"] \arrow[rr, "\langle\!\langle\Gamma \vdash e : \mathsf{R}\rangle\!\rangle"'] & & \mathsf{M}\flat\mathbf{R} \end{tikzcd}$$

$\langle\!\langle\mathsf{R}\rangle\!\rangle = \mathbf{R}$

Denotational Semantics

In Set:

$$\begin{tikzcd}[column sep=small]
 & \mathbb{P}_\star(\llbracket \Gamma \rrbracket_\perp) & \llbracket \tau \rrbracket & \\
\llbracket \Gamma \rrbracket \arrow[ur, "\llbracket \Gamma \vdash c \triangleright \Gamma \rrbracket"] \arrow[rr, "\llbracket \Gamma \vdash e : \mathsf{R} \rrbracket"'] & & \mathbb{P}_\star(\mathbb{R}_\perp) & \llbracket \mathsf{R} \rrbracket = \mathbb{R}
\end{tikzcd}$$

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:

$$\begin{tikzcd}[column sep=small]
 & \mathsf{M}\flat\langle\!\langle \Gamma \rangle\!\rangle & \langle\!\langle \tau \rangle\!\rangle & \\
\langle\!\langle \Gamma \rangle\!\rangle \arrow[ur, "\langle\!\langle \Gamma \vdash c \triangleright \Gamma \rangle\!\rangle"] \arrow[rr, "\langle\!\langle \Gamma \vdash e : \mathsf{R} \rangle\!\rangle"'] & & \mathsf{M}\flat\mathbf{R} & \langle\!\langle \mathsf{R} \rangle\!\rangle = \mathbf{R}
\end{tikzcd}$$

$$\mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \xrightarrow{\text{Forgetful Functor}} \mathsf{Set}$$

- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$

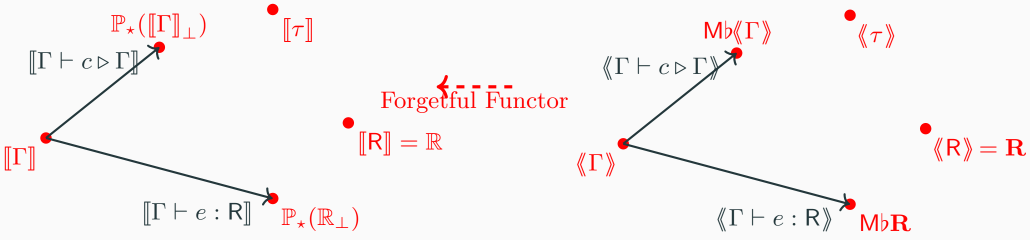
Extension

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



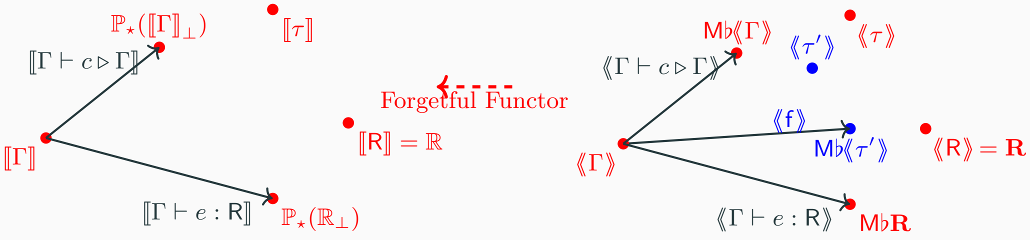
- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

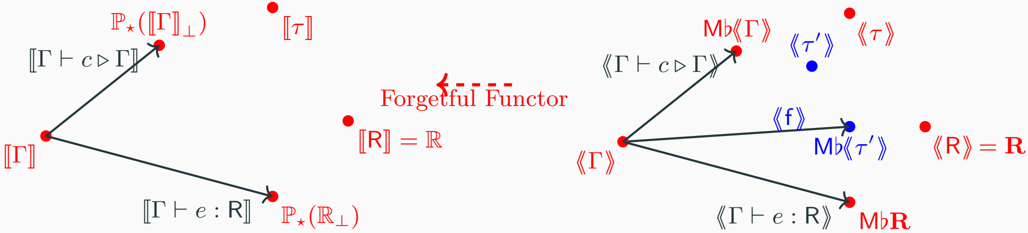
Extension

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



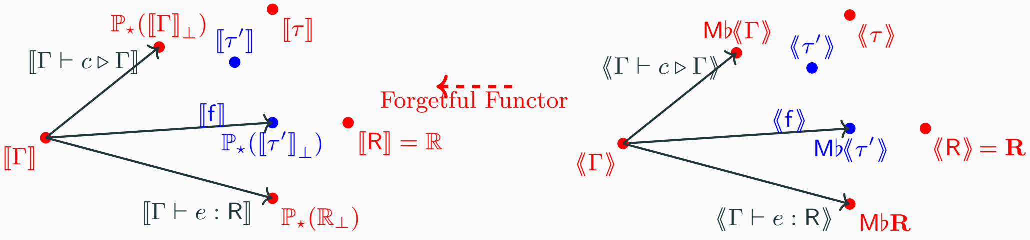
- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Define the denotations by the forgetful functor

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Define the denotations by the forgetful functor

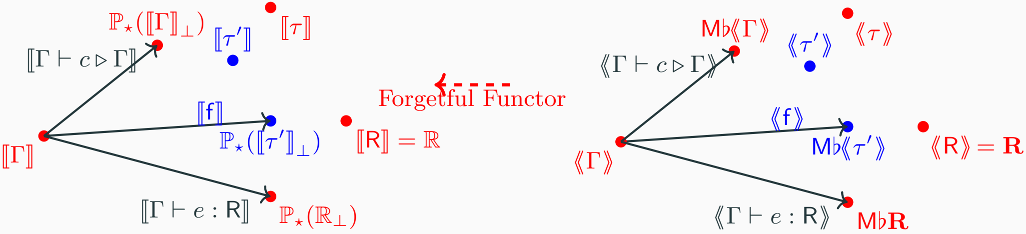
Extension

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Define the denotations by the forgetful functor
- Commands remain the same, for a specification language that is expressive for the expression language, the proof rule remains sound

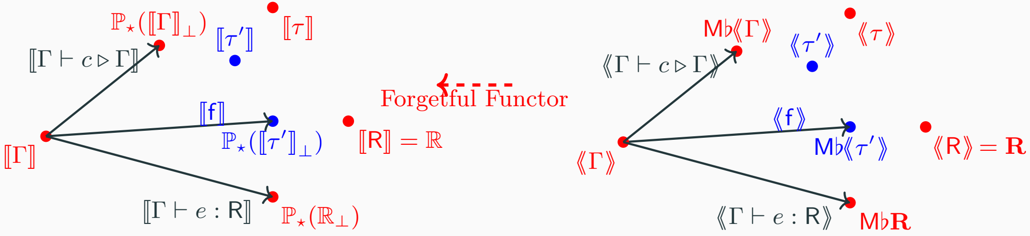
Extension

Denotational Semantics

In Set:

Interpretation

In $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:



- Add a new data type τ' and a new expression construct $\mathsf{f}(t_1, \cdots, t_n) : \tau'$
- Assign $\langle\!\langle\tau'\rangle\!\rangle$ an assembly and $\langle\!\langle\mathsf{f}\rangle\!\rangle$ a morphism to $\mathsf{M}\flat\langle\!\langle\tau'\rangle\!\rangle$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Define the denotations by the forgetful functor
- Commands remain the same, for a specification language that is expressive for the expression language, the proof rule remains sound
- per collection of assemblies and morphisms $\mathcal{E}$ (with consistency property), we define a while language over $\mathcal{E}$
  - with laziness
  - with matrices
  - with continuous real functions

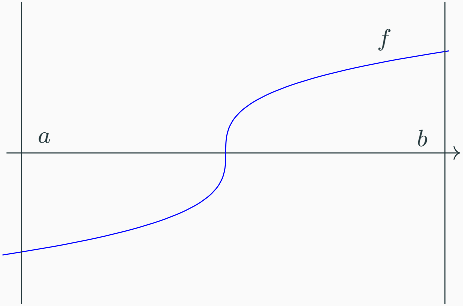
Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $\quad a, b : \mathsf{R}, f : \mathsf{C}$

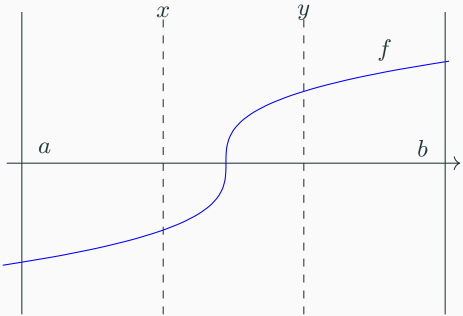


Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $\quad x, y, a, b : \mathsf{R}, f : \mathsf{C}$

$$x := (2 \times a + b)/3;$$
$$y := (a + 2 \times b)/3;$$

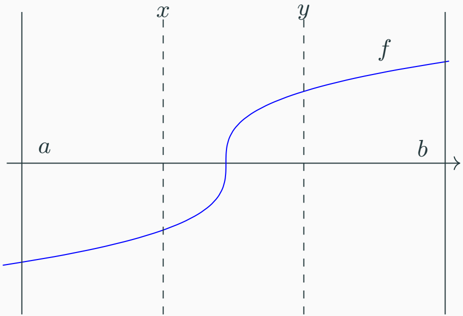


Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $\quad x, y, a, b : \mathsf{R}, f : \mathsf{C}$

$x := (2 \times a + b)/3;$
$y := (a + 2 \times b)/3;$
$\mathtt{if}\ \mathsf{choose}(\mathsf{eval}(f, x) \lesssim 0, 0 \lesssim \mathsf{eval}(f, y)) = 1$

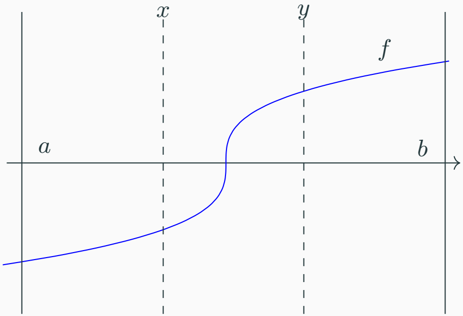


Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $\quad x, y, a, b : \mathsf{R}, f : \mathsf{C}$

$x := (2 \times a + b)/3;$
$y := (a + 2 \times b)/3;$
$\mathtt{if}\ \mathsf{choose}(\mathsf{eval}(f, x) \lesssim 0, 0 \lesssim \mathsf{eval}(f, y)) = 1$
  $\mathtt{then}\ a := x$
  $\mathtt{else}\ b := y$
$\mathtt{end}$



Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $x, y, a, b : \mathsf{R}, f : \mathsf{C}$

$x := (2 \times a + b)/3;$

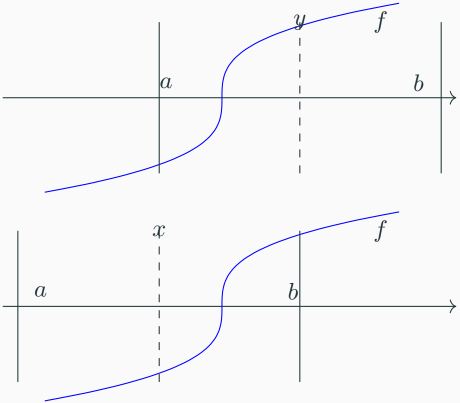
$y := (a + 2 \times b)/3;$

$\texttt{if}\ \mathsf{choose}(\mathsf{eval}(f, x) \lesssim 0, 0 \lesssim \mathsf{eval}(f, y)) = 1$

  $\texttt{then}\ a := x$

  $\texttt{else}\ b := y$

$\texttt{end}$

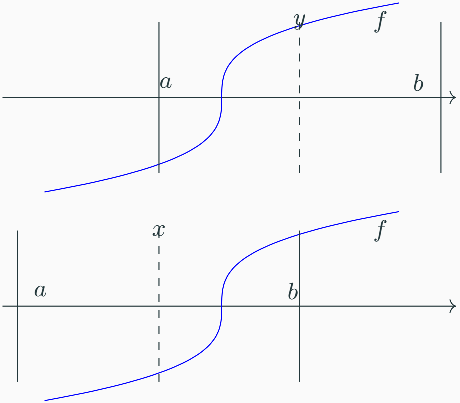


Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $p : \mathsf{Z}, x, y, a, b : \mathsf{R}, f : \mathsf{C}$

while $b - a \lesssim 2^p$ **do**
  $x := (2 \times a + b)/3;$
  $y := (a + 2 \times b)/3;$
  if $\mathsf{choose}(\mathsf{eval}(f, x) \lesssim 0, 0 \lesssim \mathsf{eval}(f, y)) = 1$
    then $a := x$
    else $b := y$
  end
end

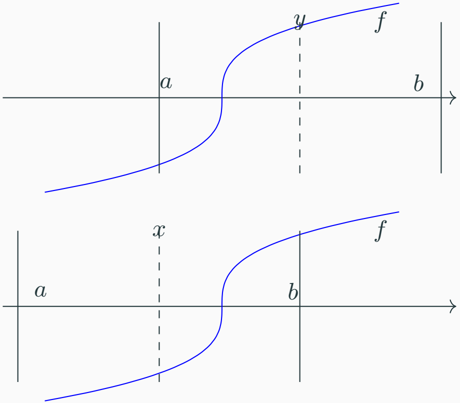


Extension with Continuous Real Functions

- Add a new data type C and a term construct $\mathsf{eval} : \mathsf{C} \times \mathsf{R} \to \mathsf{R}$
- Interpret $\langle\!\langle \mathsf{C} \rangle\!\rangle = \mathbf{R}^{\mathbf{R}}$ and $\langle\!\langle \mathsf{eval} \rangle\!\rangle$ the evaluation map of $\mathbf{R}^{\mathbf{R}}$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Derived $[\![\mathsf{C}]\!] = \mathcal{C}(\mathbb{R}, \mathbb{R})$ and $[\![\mathsf{eval}]\!]$ is the evaluation map in Set

When $p : \mathsf{Z}, x, y, a, b : \mathsf{R}, f : \mathsf{C}$

$\{f$ has unique root in $(a, b) \wedge f(a) < 0 < f(b)\}$
`while` $b - a \lesssim 2^p$ `do`
  $x := (2 \times a + b)/3;$
  $y := (a + 2 \times b)/3;$
  `if` $\mathsf{choose}(\mathsf{eval}(f, x) \lesssim 0, 0 \lesssim \mathsf{eval}(f, y)) = 1$
    `then` $a := x$
    `else` $b := y$
  `end`
`end`
$\{a$ approximates the root of f by $2^p\}$



Concluding Remark (of the section)

So far,

- we defined a simple imperative language that supports exact real computation and nondeterminism
- we devised a sound verification calculus and VC generator
- we suggested a way to extend the language and the verification calculus with other continuous objects

In the future,

- strong verification automation
- practical tool for verifying iRRAM programs

Concluding Remark (of the section)

So far,

- we defined a simple imperative language that supports exact real computation and nondeterminism
- we devised a sound verification calculus and VC generator
- we suggested a way to extend the language and the verification calculus with other continuous objects

In the future,

- strong verification automation
- practical tool for verifying iRRAM programs
- imperative language with explicit limit operator (j.w.w. Andrej Bauer and Alex Simpson, CCA 2020)

Program Extraction

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories
- In a dependent type theory that admits $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$-interpretation, axiomatize the nondeterminism monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories
- In a dependent type theory that admits $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$-interpretation, axiomatize the nondeterminism monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$
- Using the monad, axiomatize the set of real numbers; i.e., axiomatize an assembly that is (provably) equivalent to the Cauchy assembly

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories
- In a dependent type theory that admits $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$-interpretation, axiomatize the nondeterminism monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$
- Using the monad, axiomatize the set of real numbers; i.e., axiomatize an assembly that is (provably) equivalent to the Cauchy assembly
- Implement the axioms in Coq, adjust Coq's extraction mechanism: axiomatic real $\to$ AERN's primitive data type `CReal`

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories
- In a dependent type theory that admits $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$-interpretation, axiomatize the nondeterminism monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$
- Using the monad, axiomatize the set of real numbers; i.e., axiomatize an assembly that is (provably) equivalent to the Cauchy assembly
- Implement the axioms in Coq, adjust Coq's extraction mechanism: axiomatic real $\to$ AERN's primitive data type `CReal`
- Proved interesting lemmas including intermediate value theorem, boundedness of real numbers, square roots, and complex square roots.
- Experiment with extracted AERN programs.

Program Extraction

- real numbers as a primitive data type in exact real computation
- axiomatic real in dependent type theories
- In a dependent type theory that admits $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$-interpretation, axiomatize the nondeterminism monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$
- Using the monad, axiomatize the set of real numbers; i.e., axiomatize an assembly that is (provably) equivalent to the Cauchy assembly
- Implement the axioms in Coq, adjust Coq's extraction mechanism: axiomatic real $\to$ AERN's primitive data type `CReal`
- Proved interesting lemmas including intermediate value theorem, boundedness of real numbers, square roots, and complex square roots.
- Experiment with extracted AERN programs.
- j.w.w. Michal Konečný and Holger Thies. Talk on October 9 (Saturday) at WoLLIC (Workshop on Logic, Language, Information and Computation) workshop
- implementation available at `https://github.com/holgerthies/coq-aern`

Thank you for listening! :)