Verified computation over real numbers and other continuous objects I

Sewon Park[†]

March 2, 2022 (Akita, Japan)
The second Japan-Russia workshop on effective descriptive set theory, computable analysis and automata

[†] Kyoto University, Japan
[†] supported by JSPS KAKENHI Grant number JP18H03203.

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++
- pure 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++
- pure 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++
- pure 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

iRRAM Example 1

Rump's example:

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

iRRAM Example 1

Rump's example:

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

`double` Rump(`double` x, `double` y)

```
    double x2 := x × x;
    double y2 := y × y;
    double z := 21 × y2 − 2 × x2 + 55 × y2 × y2 − 10 × x2 × y2 + x/(2 × y);

    return z;
```

iRRAM Example 1

Rump's example:

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

`double` Rump(`double` x, `double` y)

$\{x, y \in \mathbb{R}\}$

`double` $x_2 := x \times x$;

`double` $y_2 := y \times y$;

`double` $z := 21 \times y_2 - 2 \times x_2 + 55 \times y_2 \times y_2 - 10 \times x_2 \times y_2 + x/(2 \times y)$;

$\{z = R(x,y)\}$

return z;

this specification is wrong

iRRAM Example 1

Rump's example:

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

`double` Rump(`double` x, `double` y)

$\{x, y \in \mathbb{R}\}$

`double` $x_2 := x \times x$;

`double` $y_2 := y \times y$;

`double` $z := 21 \times y_2 - 2 \times x_2 + 55 \times y_2 \times y_2 - 10 \times x_2 \times y_2 + x/(2 \times y)$;

$\{|z - R(x,y)| \leq E(x,y)\}$

return z;

Error term E analyzed carefully becomes nonsense with large enough x, y.

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, \times, ^{-1}$ computed exactly

Rump's example:

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

iRRAM Example 1

- iRRAM is a C++ library support `REAL`
- `REAL` typed variables store exact real numbers
- overloaded operations $+, -, \times, ^{-1}$ computed exactly

Rump's example:

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

`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 $+, -, \times, {}^{-1}$ computed exactly

Rump's example:

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

`REAL` Rump(`REAL` x, `REAL` y)
$\{x, y \in \mathbb{R}\}$
`REAL` $x_2 := x \times x$;
`REAL` $y_2 := y \times y$;
`REAL` $z := 21 \times y_2 - 2 \times x_2 + 55 \times y_2 \times y_2 - 10 \times x_2 \times y_2 + x/(2 \times y)$;
$\{z = R(x, y)\}$
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 $x \lesssim y$ *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}
    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 $x \lesssim y$ *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 $x \lesssim y$ *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 $x \lesssim y$ *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 $x \lesssim y$ *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}
    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 $x \lesssim y$ *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}
  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 $y \lesssim x + 1$ and $x <\lesssim y$ hold
- at some point, $y \lesssim x + 1$ 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 operations and partial comparisons)
- Convenient-to-use precondition-postcondition-style program verification

Language Overview

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
		$\mid$	$x \coloneqq e$	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
programs	p	::=	$\textbf{input}(x_1 : \tau_1, \cdots, x_d : \tau_d)\ c\ \textbf{return}\ e$	top-level function

Language Overview

While-language based on Peano Arithmetic and Boolean logic:

data types	τ	::=	K	Kleenan (lazy Boolean)
		|	Z	integer
		|	R	real number
expressions	e	::=	$\texttt{true} \mid \texttt{false} \mid 0 \mid 1 \mid \cdots$	constant
		|	$e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$	integer arithmetic
		|	$e_1 = e_2 \mid e_1 \leq e_2$	integer comparison
		|	$e_1 \dot{+} e_2 \mid e_1 \dot{-} e_2 \mid e_1 \dot{\times} e_2 \mid e^{-1}$	real arithmetic
		|	$e_1 \lesssim e_2$	partial real comparison
		|	$\mathsf{choose}(e_1, \cdots, e_n)$	nondeterminism
		|	$2^e \mid \iota(e)$	coercion from Z to R
commands	c	::=	$\texttt{skip}$	do nothing
		|	$c_1; c_2$	composition
		|	$x \coloneqq e$	assignment
		|	$\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}$	conditional
		|	$\texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}$	loop
programs	p	::=	$\textbf{input}(x_1 : \tau_1, \cdots, x_d : \tau_d)\ c\ \textbf{return}\ e$	top-level function
		|	$\textbf{lim}(p : \mathsf{Z};\ x_1 : \tau_1, \cdots, x_d : \tau_d)\ c\ \textbf{return}\ e$	top-level limit function

Overview

Today:

1. Review computable analysis: specify computable (partial) operations and formalize nondeterminism
2. Review domain theory and specify computable domain operations w.r.t. (1)

Overview

Today:

1. Review computable analysis: specify computable (partial) operations and formalize nondeterminism
2. Review domain theory and specify computable domain operations w.r.t. (1)

Tomorrow:

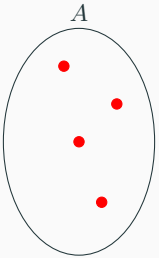
3. Define denotational semantics based on (1) and (2)
4. Define proof rules based on (3)
5. Suggest an extension structure based on (1) and (3)
6. Suggest an extension of the language toward limit and multi-limits (if time allows)

Computable Analysis:
foundation of exact real computation

Section Overview

1. Introduce assemblies over Kleene's second algebra (multi-represented space),
2. and computable functions between them
3. form a category $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.
4. Introduce the concepts of partial and multi-valued functions which are essential concepts in computable analysis
5. and formalize them as some monads in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

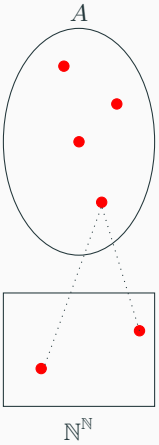
Assemblies (over $\mathbb{N}^{\mathbb{N}}$)



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

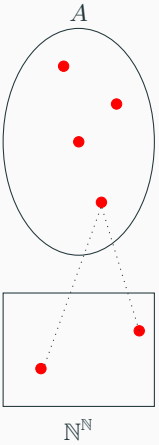
$$\forall x \in A \,.\, \exists \varphi \in \mathbb{N}^{\mathbb{N}} \,.\, \varphi \Vdash_A x$$

Assemblies (over $\mathbb{N}^{\mathbb{N}}$)



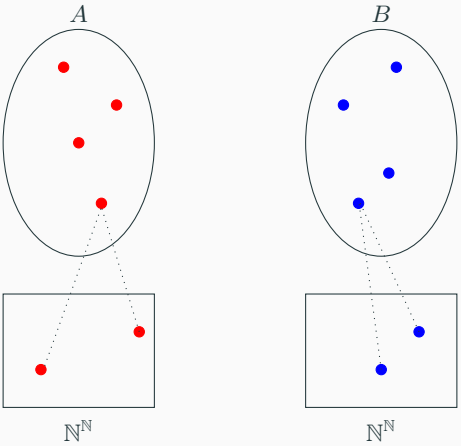
Definition
An assembly is a pair $(A, \Vdash_A)$ of a set A and a relation $\Vdash_A \subseteq \mathbb{N}^{\mathbb{N}} \times A$ such that

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



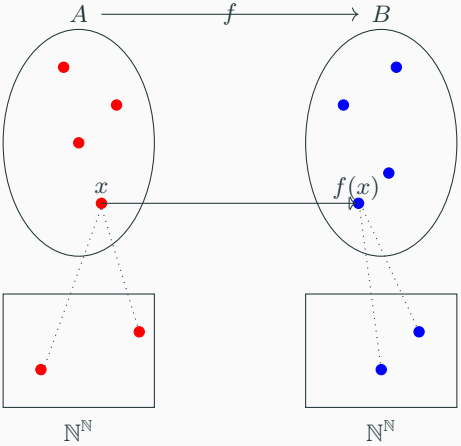
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

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



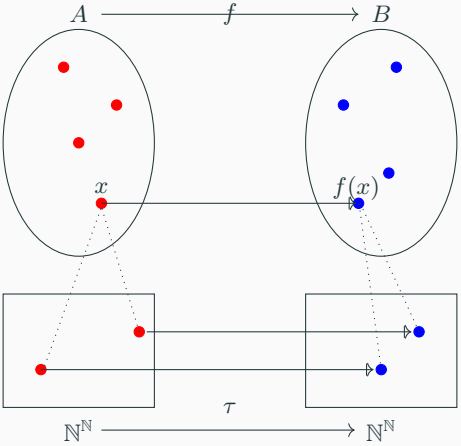
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

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



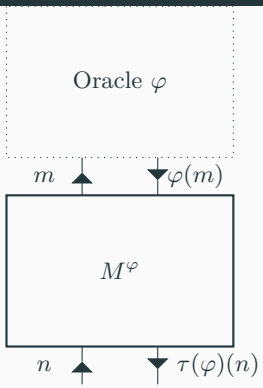
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

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



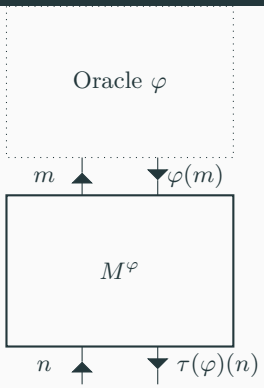
Definition
A function $f : A \to B$ is computable if there is computable $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *tracks* f:

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



Definition
A infinite string function $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is

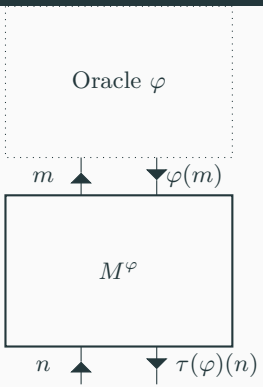
1. *computable* if there is an oracle machine $M^{?}$, when it is equipped with $\varphi \in \mathbb{N}^{\mathbb{N}}$, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$



Definition
A infinite string function $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is

1. *computable* if there is an oracle machine $M^{?}$, when it is equipped with $\varphi \in \mathbb{N}^{\mathbb{N}}$, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$
2. *continuous* if there is an oracle machine $M^{?}$ and an oracle ψ when it is equipped with $\langle \varphi, \psi \rangle$ as its oracle, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$

Oracle Machines



Definition
A infinite string function $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is

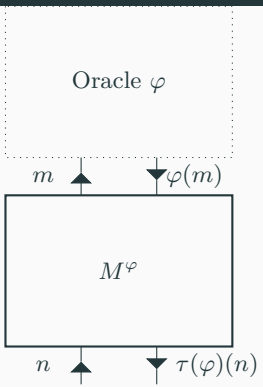
1. *computable* if there is an oracle machine $M^{?}$, when it is equipped with $\varphi \in \mathbb{N}^{\mathbb{N}}$, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$
2. *continuous* if there is an oracle machine $M^{?}$ and an oracle ψ when it is equipped with $\langle \varphi, \psi \rangle$ as its oracle, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$

Formally, oracle machine computes $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \rightharpoonup \mathbb{N}$ not $(\mathbb{N} \to \mathbb{N}) \rightharpoonup (\mathbb{N} \to \mathbb{N})$

Definition

1. A function τ is *strong* if there is an oracle machine $M^{?}$ (with or without extra oracle) computing τ such that $M^{\varphi}(n)$ is undefined (internal infinite loop) for any n when $\varphi \notin \mathrm{dom}(\tau)$

Oracle Machines



Definition
A infinite string function $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is

1. *computable* if there is an oracle machine $M^{?}$, when it is equipped with $\varphi \in \mathbb{N}^{\mathbb{N}}$, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$
2. *continuous* if there is an oracle machine $M^{?}$ and an oracle ψ when it is equipped with $\langle \varphi, \psi \rangle$ as its oracle, prints $\tau(\varphi)(n)$ in its input n, when $\varphi \in \mathrm{dom}(\tau)$

Formally, oracle machine computes $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \rightharpoonup \mathbb{N}$ not $(\mathbb{N} \to \mathbb{N}) \rightharpoonup (\mathbb{N} \to \mathbb{N})$

Definition

1. A function τ is *strong* if there is an oracle machine $M^{?}$ (with or without extra oracle) computing τ such that $M^{\varphi}(n)$ is undefined (internal infinite loop) for any n when $\varphi \notin \mathrm{dom}(\tau)$
2. A partial function $f : X \rightharpoonup Y$ is strongly computable (continuous) if there is a strong realizer tracking f.

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$\varphi \Vdash_{\text{Cauchy}} x \quad \iff \quad \varphi \text{ encodes } (q_i)_{i \in \mathbb{N}} \subseteq \mathbb{Q} \text{ s.t. } \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$\varphi \Vdash_{\text{Cauchy}} x \quad \iff \quad \varphi \text{ encodes } (q_i)_{i\in\mathbb{N}} \subseteq \mathbb{Q} \text{ s.t. } \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
  for any $q \in \mathbb{Q}$ $\quad q, q, q, q, \cdots \Vdash_{\text{Cauchy}} q$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
  When $(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x$ and $(r_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} y$,

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \Longleftrightarrow \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
  When $(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x$ and $(r_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} y$,
  $|(x+y) - (q_{i+1} + r_{i+1})| \leq |x - q_{i+1}| + |y - r_{i+1}| \leq 2^{-i-1} + 2^{-i-1} = 2^{-i}$

$$(q_2 + r_2), (q_3 + r_3), \cdots \Vdash_{\text{Cauchy}} x + y$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<: \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<: \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
  When $(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x$ and $(r_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} y$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<: \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
  When $(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x$ and $(r_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} y$
  $x \neq y$ is equivalent to finding an index $i \in \mathbb{N}$ s.t. $|q_i - r_i| > 2^{-i+1}$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<: \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
  When $(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x$ and $(r_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} y$
  $x \neq y$ is equivalent to finding an index $i \in \mathbb{N}$ s.t. $|q_i - r_i| > 2^{-i+1}$

$$x < y = \begin{cases} tt & \text{if } x < y \\ ff & \text{if } y < x \\ \text{infinite loop} & \text{if } x = y \end{cases}$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i \in \mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

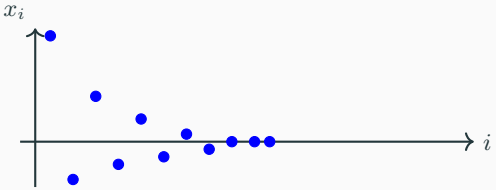
- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable

$$\lim((x_i)_{i\in\mathbb{N}}) = \square\square\square\square\cdots$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable

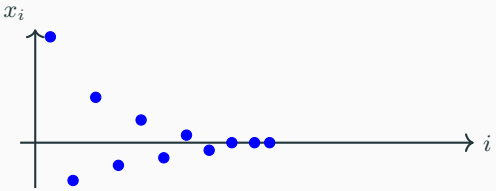


$$\lim((x_i)_{i\in\mathbb{N}}) = \square\square\square\cdots$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable

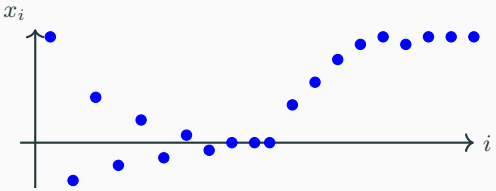


$$\lim((x_i)_{i\in\mathbb{N}}) = 0.0\square\square\cdots$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N}\,.\,|x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable



$$\lim((x_i)_{i\in\mathbb{N}}) = 0.0\square\square\cdots$$

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable
- Partial limit operation $\mathsf{lim} : \{(x_i)_{i\in\mathbb{N}} \mid \exists y \,.\, \forall n \,.\, |y - x_n| \leq 2^{-n}\} \to \mathbb{R}$ is computable (corrects out of the domain)

Assemblies of Real Numbers

Definition
Cauchy assembly is $\mathbf{R} = (\mathbb{R}, \Vdash_{\text{Cauchy}})$ such that

$$(q_i)_{i\in\mathbb{N}} \Vdash_{\text{Cauchy}} x \quad \iff \quad \forall n \in \mathbb{N} \,.\, |x - q_n| \leq 2^{-n}$$

- $\mathbb{Q} \subseteq \mathbb{R}$ are computable.
- Field arithmetic $+, -, \times, /$ is computable
- Order comparison $<\colon \mathbb{R}^2 \to \{tt, ff\}$ is not computable at $\Delta = \{(x, x) \mid x \in \mathbb{R}\}$
- Partial comparison $\lesssim\colon \mathbb{R}^2 \setminus \Delta \to \{tt, ff\}$ is strongly computable (diverges out of the domain)
- Limit operation $\lim : \{\text{sequence } (x_i)_{i\in\mathbb{N}} \text{ converges}\} \to \mathbb{R}$ is not computable
- Partial limit operation $\mathsf{lim} : \{(x_i)_{i\in\mathbb{N}} \mid \exists y \,.\, \forall n \,.\, |y - x_n| \leq 2^{-n}\} \to \mathbb{R}$ is computable (corrects out of the domain)

Lemma

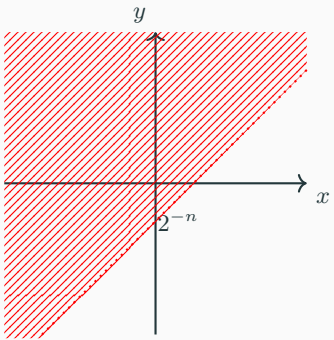
1. *[Wei00, Theorem 4.1.16] There is no assembly that makes the total comparison computable.*
2. *Assembly that makes total limit computable makes any non-trivial computable function to $\{tt, ff\}$ uncomputable*
3. *[Her99, Theorem 3.5] There is unique (up to isomorphism) assembly of real numbers that satisfy the above computability properties.*

Nondeterminism

- $x \lesssim y$ diverges when $x = y$.

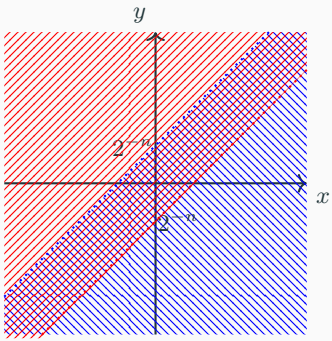
Nondeterminism

- $x \lesssim y$ diverges when $x = y$.
- $x \lesssim y + 2^{-n}$ diverges when $x = y + 2^{-n}$



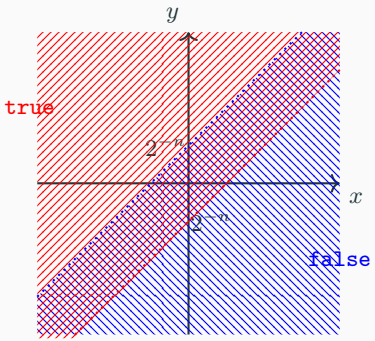
Nondeterminism

- $x \lesssim y$ diverges when $x = y$.
- $x \lesssim y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y \lesssim x + 2^{-n}$ diverges when $y = x + 2^{-n}$



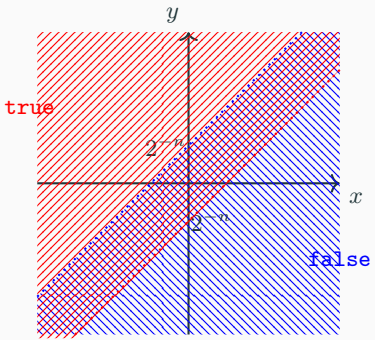
Nondeterminism

- $x \lesssim y$ diverges when $x = y$.
- $x \lesssim y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y \lesssim x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x \lesssim y + 2^{-n}$ and $y \lesssim x + 2^{-n}$ in parallel
  Return `true` when $x \lesssim y + 2^{-n}$
  Return `false` when $y \lesssim x + 2^{-n}$



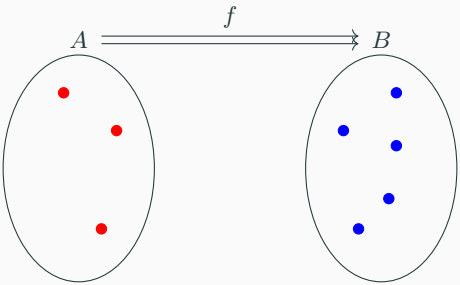
Nondeterminism

- $x \lesssim y$ diverges when $x = y$.
- $x \lesssim y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y \lesssim x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x \lesssim y + 2^{-n}$ and $y \lesssim x + 2^{-n}$ in parallel
  Return `true` when $x \lesssim y + 2^{-n}$
  Return `false` when $y \lesssim x + 2^{-n}$



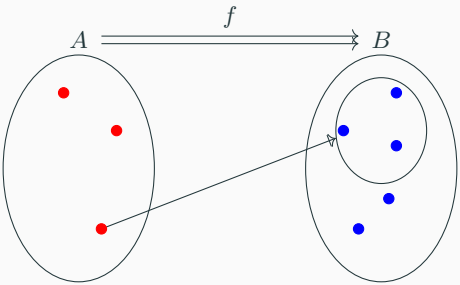
$$x \lesssim_n y = \begin{cases} \texttt{true} & x \leq y - 2^{-n}, \\ \texttt{true or false} & y - 2^{-n} < x < y + 2^{-n}, \\ \texttt{false} & y \leq x - 2^{-n}. \end{cases}$$

Nondeterminism and Multi-functions



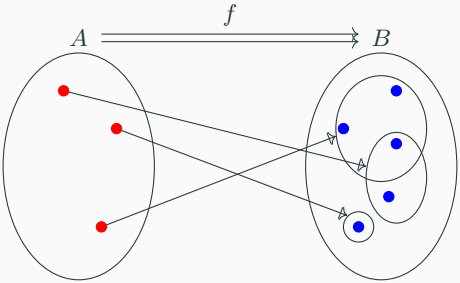
- Multi-function $f : A \rightrightarrows B$, return value $f(x)$ may be different in each trial

Nondeterminism and Multi-functions



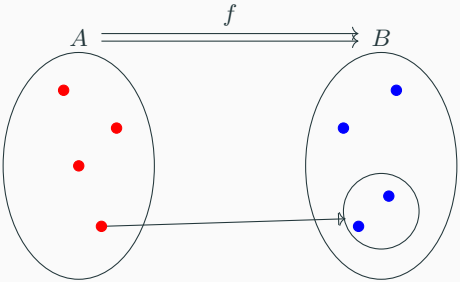
- Multi-function $f : A \rightrightarrows B$, return value $f(x)$ may be different in each trial
- $f(x)$ denotes the set of all possible outputs

Nondeterminism and Multi-functions

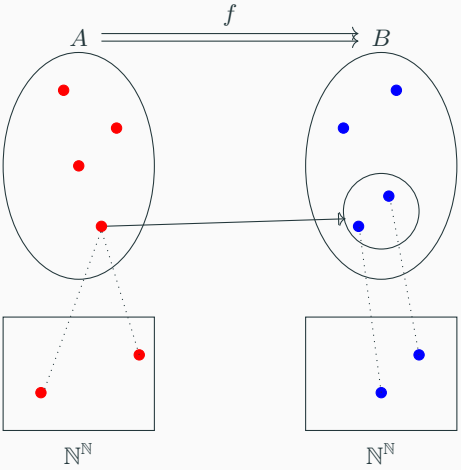


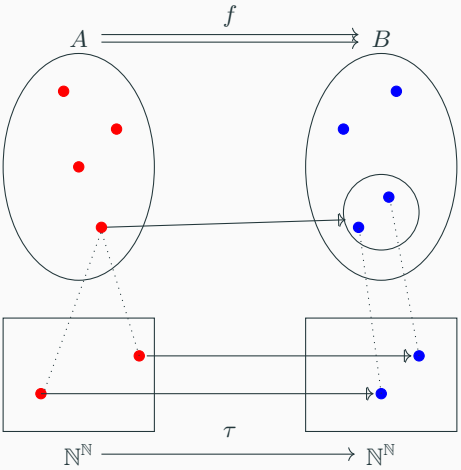
- Multi-function $f : A \rightrightarrows B$, return value $f(x)$ may be different in each trial
- $f(x)$ denotes the set of all possible outputs

Nondeterminism and Multi-functions



Nondeterminism and Multi-functions

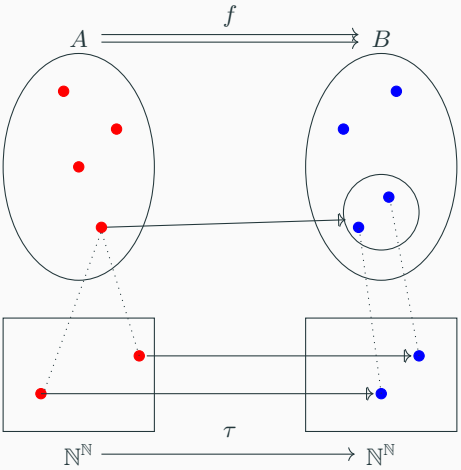




- A multi-function $f : A \rightrightarrows B$ is computable if there is computable τ tracks it

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

Nondeterminism and Multi-functions



- A multi-function $f : A \rightrightarrows B$ is computable if there is computable τ tracks it

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

- the nondeterministic approximation $\lesssim_n : \mathbf{R} \times \mathbf{R} \rightrightarrows \mathbf{2}$ is computable

Category of Assemblies $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

- Assmeblies and computable functions form $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Category of represented sets is a subcategory of $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$
- Category of sets Set is a reflective subcategory of $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ by

$$\nabla X = (X, \mathbb{N}^{\mathbb{N}} \times X)$$

- $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is a quasitopos whose regular-subobject classifier is $\nabla 2$
- $\mathbf{Y}^{\mathbf{X}}$ is assembly over all continuously realizable functions though $\mathrm{Hom}_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\mathbf{X}, \mathbf{Y})$ is the set of all computable functions.

Nondeterminism monad in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition

Nondeterminism monad $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\mathsf{M}(\mathbf{X})$ whose underlying set is the non-empty power-set of X such that

$$\varphi \Vdash_{\mathsf{M}(X, \Vdash_X)} S \quad :\Leftrightarrow \quad \exists x \in S \wedge \varphi \Vdash_X x$$

The monad on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\mathsf{M}(f : \mathbf{X} \to \mathbf{Y}) \coloneqq S \mapsto \bigcup_{x \in S} \{f(x)\}$$

Nondeterminism monad in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition

Nondeterminism monad $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\mathsf{M}(\mathbf{X})$ whose underlying set is the non-empty power-set of X such that

$$\varphi \Vdash_{\mathsf{M}(X, \Vdash_X)} S \quad :\Leftrightarrow \quad \exists x \in S \wedge \varphi \Vdash_X x$$

The monad on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\mathsf{M}(f : \mathbf{X} \to \mathbf{Y}) \coloneqq S \mapsto \bigcup_{x \in S} \{f(x)\}$$

Examples

For any $n \in \mathbb{N}$, the nondeterministic comparison appears as a morhpism:

$$\lesssim_n : \mathbf{R} \times \mathbf{R} \to \mathsf{M}(\mathbf{2})$$

Nondeterminism monad in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition
Nondeterminism monad $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\mathsf{M}(\mathbf{X})$ whose underlying set is the non-empty power-set of X such that

$$\varphi \Vdash_{\mathsf{M}(X, \Vdash_X)} S \quad :\Leftrightarrow \quad \exists x \in S \wedge \varphi \Vdash_X x$$

The monad on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\mathsf{M}(f : \mathbf{X} \to \mathbf{Y}) \coloneqq S \mapsto \bigcup_{x \in S} \{f(x)\}$$

Examples
For any $n \in \mathbb{N}$, the nondeterministic comparison appears as a morhpism:

$$\lesssim_n : \mathbf{R} \times \mathbf{R} \to \mathsf{M}(\mathbf{2})$$

Remark
For any represented sets $\mathbf{X}, \mathbf{Y}$,

the set of computable multi-functions $\mathbf{X} \rightrightarrows \mathbf{Y}$ is identical to $\mathrm{Hom}_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\mathbf{X}, \mathsf{M}(\mathbf{Y}))$ and

the set of continuously realizable multi-functions $\mathbf{X} \rightrightarrows \mathbf{Y}$ is identical to $|\mathbf{X} \to \mathsf{M}(\mathbf{Y})|$.

Partiality monad $\flat$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition

Lazy partiality $\flat : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\flat\mathbf{X}$ whose underlying set is $X \cup \{\flat_{\mathbf{X}}\}$ such that

$$\begin{array}{llll} \mathtt{NotYet}^{\mathbb{N}} & \Vdash_{\flat\mathbf{X}} & \flat_{\mathbf{X}} & \\ \mathtt{NotYet}^{m} :: \varphi & \Vdash_{\flat\mathbf{X}} & x & :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{X}} x \text{ for any } m \in \mathbb{N}. \end{array}$$

Partiality monad $\flat$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition
Lazy partiality $\flat : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\flat\mathbf{X}$ whose underlying set is $X \cup \{\flat_{\mathbf{X}}\}$ such that

$$\begin{array}{lllll} \mathtt{NotYet}^{\mathbb{N}} & \Vdash_{\flat\mathbf{X}} & \flat_{\mathbf{X}} & & \\ \mathtt{NotYet}^{m} :: \varphi & \Vdash_{\flat\mathbf{X}} & x & :\Leftrightarrow & \varphi \Vdash_{\mathbf{X}} x \text{ for any } m \in \mathbb{N}. \end{array}$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \flat_{\mathbf{X}} \ , \\ \flat_{\mathbf{Y}} & \text{otherwise .} \end{cases}$$

Partiality monad $\flat$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition
Lazy partiality $\flat : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\flat\mathbf{X}$ whose underlying set is $X \cup \{\flat_{\mathbf{X}}\}$ such that

$$\begin{array}{lllll} \mathtt{NotYet}^{\mathbb{N}} & \Vdash_{\flat\mathbf{X}} & \flat_{\mathbf{X}} & & \\ \mathtt{NotYet}^{m} :: \varphi & \Vdash_{\flat\mathbf{X}} & x & :\Leftrightarrow & \varphi \Vdash_{\mathbf{X}} x \text{ for any } m \in \mathbb{N}. \end{array}$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \flat_{\mathbf{X}} \ , \\ \flat_{\mathbf{Y}} & \text{otherwise} \ . \end{cases}$$

Examples
This partiality monad classifies partial functions that diverge out of their domains:

1. The partial comparison $\lesssim$ appears as a (computable) morphism

$$\lesssim : \mathbf{R} \times \mathbf{R} \to \flat\mathbf{2}$$

2. Nondeterminism choose_n appears as a

$$\mathsf{choose}_n : (\flat\mathbf{2})^n \to \flat\mathsf{M}\mathbf{Z}$$

Definition
Colazy partiality $\sharp : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\sharp\mathbf{X}$ whose underlying set is $X \cup \{\sharp_{\mathbf{X}}\}$ such that

$$\begin{array}{llllll} a :: \mathtt{Wrong} :: \varphi & \Vdash_{\sharp\mathbf{X}} & \sharp_{\mathbf{X}} & \text{for any } a_i \in \mathbb{N}^* \text{ and } \varphi \in \mathbb{N}^{\mathbb{N}} \\ \varphi & \Vdash_{\sharp\mathbf{X}} & x & :\Leftrightarrow & \varphi \Vdash_{\mathbf{X}} x \end{array}$$

Definition
Colazy partiality $\sharp : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\sharp\mathbf{X}$ whose underlying set is $X \cup \{\sharp_{\mathbf{X}}\}$ such that

$$\begin{array}{llll} a :: \texttt{Wrong} :: \varphi & \Vdash_{\sharp\mathbf{X}} & \sharp_{\mathbf{X}} & \text{for any } a_i \in \mathbb{N}^* \text{ and } \varphi \in \mathbb{N}^{\mathbb{N}} \\ \varphi & \Vdash_{\sharp\mathbf{X}} & x & :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{X}} x \end{array}$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \sharp_{\mathbf{X}} \ , \\ \sharp_{\mathbf{Y}} & \text{otherwise} \ . \end{cases}$$

Definition
Colazy partiality $\sharp : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\sharp\mathbf{X}$ whose underlying set is $X \cup \{\sharp_{\mathbf{X}}\}$ such that

$$\begin{array}{llll} a :: \texttt{Wrong} :: \varphi & \Vdash_{\sharp\mathbf{X}} & \sharp_{\mathbf{X}} & \text{for any } a_i \in \mathbb{N}^* \text{ and } \varphi \in \mathbb{N}^{\mathbb{N}} \\ \varphi & \Vdash_{\sharp\mathbf{X}} & x & :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{X}} x \end{array}$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \sharp_{\mathbf{X}} \ , \\ \sharp_{\mathbf{Y}} & \text{otherwise} \ . \end{cases}$$

Examples
This partiality monad classifies partial functions that correct out of their domains:

1. The partial limit operation appears as a (computable) mortphism $\mathbf{lim} : \mathbf{R}^{\mathbf{N}} \to \sharp\mathbf{R}$
2. There is no partial function from uncountable set that is both $\flat$ and $\sharp$

Partiality monad $\natural$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition
Natural partiality $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\natural\mathbf{X}$ whose underlying set is $X \cup \{\natural_{\mathbf{X}}\}$ such that

$$\varphi \Vdash_{\natural\mathbf{X}} x \quad :\Leftrightarrow \quad \varphi \text{ contains infinitely many non-}\texttt{JUNK}\text{ elements and the subsequence removing }\texttt{JUNK}\text{s realizes } x \in \mathbf{X}$$

$$\varphi \Vdash_{\natural\mathbf{X}} \natural_{\mathbf{X}}$$

Partiality monad $\natural$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Definition
Natural partiality $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\natural\mathbf{X}$ whose underlying set is $X \cup \{\natural_{\mathbf{X}}\}$ such that

$$\varphi \Vdash_{\natural\mathbf{X}} x \quad :\Leftrightarrow \quad \varphi \text{ contains infinitely many non-}\texttt{JUNK}\text{ elements and the subsequence removing }\texttt{JUNK}\text{s realizes } x \in \mathbf{X}$$

$$\varphi \Vdash_{\natural\mathbf{X}} \natural_{\mathbf{X}}$$

Definition
Natural partiality $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\natural\mathbf{X}$ whose underlying set is $X \cup \{\natural_{\mathbf{X}}\}$ such that

$$\varphi \Vdash_{\natural\mathbf{X}} x \quad :\Leftrightarrow \quad \varphi \text{ contains infinitely many non-}\texttt{JUNK}\text{ elements and the subsequence removing }\texttt{JUNK}\text{s realizes } x \in \mathbf{X}$$

$$\varphi \Vdash_{\natural\mathbf{X}} \natural_{\mathbf{X}}$$

$$a_1 :: \texttt{JUNK} :: \texttt{JUNK} :: a_2 :: \texttt{JUNK} :: \cdots \Vdash_{\natural\mathbf{X}} x \Leftrightarrow a_1 :: a_2 :: \cdots \Vdash_{\mathbf{X}} x$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \natural_{\mathbf{X}} \ , \\ \natural_{\mathbf{Y}} & \text{otherwise .} \end{cases}$$

Definition
Natural partiality $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ on $\mathbf{X}$ is defined to be an assembly $\natural\mathbf{X}$ whose underlying set is $X \cup \{\natural_{\mathbf{X}}\}$ such that

$$\varphi \Vdash_{\natural\mathbf{X}} x \quad :\Leftrightarrow \quad \varphi \text{ contains infinitely many non-}\texttt{JUNK}\text{ elements and the subsequence removing }\texttt{JUNK}\text{s realizes } x \in \mathbf{X}$$

$$\varphi \Vdash_{\natural\mathbf{X}} \natural_{\mathbf{X}}$$

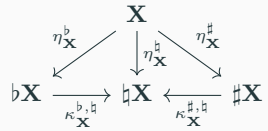
$$a_1 :: \texttt{JUNK} :: \texttt{JUNK} :: a_2 :: \texttt{JUNK} :: \cdots \Vdash_{\natural\mathbf{X}} x \Leftrightarrow a_1 :: a_2 :: \cdots \Vdash_{\mathbf{X}} x$$

The functor on a morphism $f : \mathbf{X} \to \mathbf{Y}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \natural_{\mathbf{X}} \text{ ,} \\ \natural_{\mathbf{Y}} & \text{otherwise .} \end{cases}$$

Remark
This partiality monad classifies any partial functions:

there are nat. trans $\kappa^{\flat,\natural} : \flat \to \natural$ and $\kappa^{\sharp,\natural} : \sharp \to \natural$



For assemblies $\mathbf{X}, \mathbf{Y}$,

1. the assembly of strong partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which does not terminate for any inputs out of its domain
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$

Partiality and nondeterminism monads

For assemblies $\mathbf{X}, \mathbf{Y}$,

1. the assembly of strong partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which does not terminate for any inputs out of its domain
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$
2. the assembly of partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which later signals that its input is invalid for any inputs out of its domain
   is isomorphic to $(\sharp\mathbf{Y})^{\mathbf{X}}$

Partiality and nondeterminism monads

For assemblies $\mathbf{X}, \mathbf{Y}$,

1. the assembly of strong partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which does not terminate for any inputs out of its domain
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$
2. the assembly of partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which later signals that its input is invalid for any inputs out of its domain
   is isomorphic to $(\sharp\mathbf{Y})^{\mathbf{X}}$
3. the assembly of continuously realizable partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ in the ordinary sense of computable analysis,
   which does not specify any out-of-domain behaviour
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$

Partiality and nondeterminism monads

For assemblies $\mathbf{X}, \mathbf{Y}$,

1. the assembly of strong partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which does not terminate for any inputs out of its domain
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$
2. the assembly of partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ which admits
   a (continuous) realizer which later signals that its input is invalid for any inputs out of its domain
   is isomorphic to $(\sharp\mathbf{Y})^{\mathbf{X}}$
3. the assembly of continuously realizable partial functions $f :\subseteq \mathbf{X} \to \mathbf{Y}$ in the ordinary sense of computable analysis,
   which does not specify any out-of-domain behaviour
   is isomorphic to $(\flat\mathbf{Y})^{\mathbf{X}}$
4. A partial multi-function $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ is computable in the sense of computable analysis
   if and only if
   $f_\natural$ appears as a morphism $\mathbf{X} \to \natural\mathsf{M}\mathbf{Y}$.

Exact Real Computation in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Exact real computation in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is

the collection of morhpisms for any effective assembly of real numbers $\mathbf{R}$:

1. constants $0, 1 : \mathbf{1} \to \mathbf{R}$
2. field operators $+, \times : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, $- : \mathbf{R} \to \mathbf{R}$, and ${}^{-1} : \mathbf{R} \to \flat\mathbf{R}$,
3. order relation $\lesssim : \mathbf{R} \times \mathbf{R} \to \flat\mathbf{2}$
4. limit operation $\mathsf{lim} : \mathbf{R}^{\mathbf{N}} \to \sharp\mathbf{R}$
5. nondeterminism construct $\mathsf{choose}_n : (\flat\mathbf{2})^n \to \flat\mathsf{M}(\mathbf{n})$

Exact Real Computation in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Exact real computation in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is

the collection of morhpisms for any effective assembly of real numbers $\mathbf{R}$:

1. constants $0, 1 : \mathbf{1} \to \mathbf{R}$
2. field operators $+, \times : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, $- : \mathbf{R} \to \mathbf{R}$, and ${}^{-1} : \mathbf{R} \to \flat\mathbf{R}$,
3. order relation $\lesssim : \mathbf{R} \times \mathbf{R} \to \mathbf{K}$
4. limit operation $\mathsf{lim} : \mathbf{R}^{\mathbf{N}} \to \sharp\mathbf{R}$
5. nondeterminism construct $\mathsf{choose}_n : \mathbf{K}^n \to \flat\mathsf{M}(\mathbf{n})$

when $\mathbf{K} := \flat\mathbf{2}$

Computable Analysis Theoretic Domain Theory

Section Overview

1. denotational semantics of while loops:

$$\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end} \simeq \texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}$$

1. denotational semantics of while loops:

$$\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end} \simeq \texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}$$

$$[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] = [\![\texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}]\!]$$

$$= \begin{cases} [\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] \circ [\![c]\!] & \text{if } [\![b]\!] = \texttt{true} \\ * & \text{if } [\![b]\!] = \texttt{false} \end{cases}$$

1. denotational semantics of while loops:

$$\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end} \simeq \texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}$$

$$[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] = [\![\texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}]\!]$$

$$= \begin{cases} [\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] \circ [\![c]\!] & \text{if } [\![b]\!] = \texttt{true} \\ * & \text{if } [\![b]\!] = \texttt{false} \end{cases}$$

defined by the least fixed point of

$$f \mapsto \textbf{if}\ \ [\![b]\!]\ \ \textbf{then}\ \ f \circ [\![c]\!]\ \ \textbf{else}\ \ *$$

Section Overview

1. denotational semantics of while loops:

$$\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end} \simeq \texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}$$

$$[\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] = [\![\texttt{if}\ b\ \texttt{then}\ c; (\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end})\ \texttt{else skip end}]\!]$$

$$= \begin{cases} [\![\texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}]\!] \circ [\![c]\!] & \text{if } [\![b]\!] = \texttt{true} \\ * & \text{if } [\![b]\!] = \texttt{false} \end{cases}$$

   defined by the least fixed point of

$$f \mapsto \textbf{if}\ \ [\![b]\!]\ \ \textbf{then}\ \ f \circ [\![c]\!]\ \ \textbf{else}\ \ *$$

2. Review Plotkin powerdomain and some continuity related properties
3. Express the powerdomain in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ attaching computability notion

Definition
A poset $(X, \sqsubseteq_X)$ is a ω-CPO with a bottom element if there is a bottom element $\bot \in X$

$$\bot \sqsubseteq x \quad \text{for any } x \in X$$

and for any increasing chain

$$x_1 \sqsubseteq x_2 \sqsubseteq \cdots$$

there is a least upper bound $\bigsqcup_{i \in \mathbb{N}} x_i$ in X.

Definition
A poset $(X, \sqsubseteq_X)$ is a ω-CPO with a bottom element if there is a bottom element $\bot \in X$

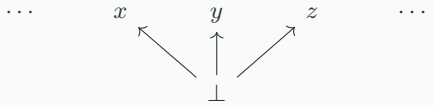
$$\bot \sqsubseteq x \quad \text{for any } x \in X$$

and for any increasing chain

$$x_1 \sqsubseteq x_2 \sqsubseteq \cdots$$

there is a least upper bound $\bigsqcup_{i \in \mathbb{N}} x_i$ in X.

Examples
For any set X, the flat domain $X_\bot = (X \cup \{\bot\}, \sqsubseteq_{X_\bot})$ is an ω-cpo with bottom:



Definition
A poset $(X, \sqsubseteq_X)$ is a ω-CPO with a bottom element if there is a bottom element $\bot \in X$

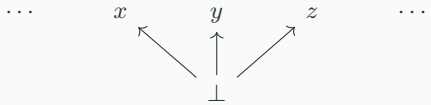
$$\bot \sqsubseteq x \quad \text{for any } x \in X$$

and for any increasing chain

$$x_1 \sqsubseteq x_2 \sqsubseteq \cdots$$

there is a least upper bound $\bigsqcup_{i \in \mathbb{N}} x_i$ in X.

Examples
For any set X, the flat domain $X_\bot = (X \cup \{\bot\}, \sqsubseteq_{X_\bot})$ is an ω-cpo with bottom:



- $\bot$ a state where computation is still going on
- $x \in X$ a state where computation results in $x \in X$
- $p \sqsubseteq q$: state p yields state q

Least Fixed Points for while loops

Definition
A monotone function $f : X \to Y$ is continuous if for any chain $x_i \subseteq X$

$$f\Big(\bigsqcup_{i \in \mathbb{N}} x_i\Big) = \bigsqcup_{i \in \mathbb{N}} f(x_i)$$

Least Fixed Points for while loops

Definition
A monotone function $f : X \to Y$ is continuous if for any chain $x_i \subseteq X$

$$f\Big(\bigsqcup_{i\in\mathbb{N}} x_i\Big) = \bigsqcup_{i\in\mathbb{N}} f(x_i)$$

Fact

1. *For any continuous $f : X \to X$, the element $\bigsqcup_{i\in\mathbb{N}} f^i(\bot)$ is the least fixed point of f.*

Least Fixed Points for while loops

Definition
A monotone function $f : X \to Y$ is continuous if for any chain $x_i \subseteq X$

$$f\Big(\bigsqcup_{i\in\mathbb{N}} x_i\Big) = \bigsqcup_{i\in\mathbb{N}} f(x_i)$$

Fact

1. *For any continuous $f : X \to X$, the element $\bigsqcup_{i\in\mathbb{N}} f^i(\bot)$ is the least fixed point of f.*
2. *For any ω-cpo Y with $\bot$, the set of functions $X \to Y$ is also ω-cpo where $x \mapsto \bot$ is the bottom element.*

Least Fixed Points for while loops

Definition
A monotone function $f : X \to Y$ is continuous if for any chain $x_i \subseteq X$

$$f\Big(\bigsqcup_{i\in\mathbb{N}} x_i\Big) = \bigsqcup_{i\in\mathbb{N}} f(x_i)$$

Fact

1. *For any continuous $f : X \to X$, the element $\bigsqcup_{i\in\mathbb{N}} f^i(\bot)$ is the least fixed point of f.*
2. *For any ω-cpo Y with $\bot$, the set of functions $X \to Y$ is also ω-cpo where $x \mapsto \bot$ is the bottom element.*

Examples
Σ a set of states "x stores 3, y stores 42, ..." $\in \Sigma$.

Denotations of commands is $[\![c]\!] : \Sigma \to \Sigma_\bot$

$$[\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!] = \mathsf{LFP}(W_{b,c})$$

where $W_{b,c}(f : \Sigma \to \Sigma_\bot) := \sigma \mapsto \begin{cases} [\![c]\!](f(\sigma)) & \text{if } [\![b]\!]\sigma = \texttt{true} \wedge f(\sigma) \neq \bot \\ \bot & \text{if } [\![b]\!]\sigma = \texttt{true} \wedge f(\sigma) = \bot \\ \sigma & \text{otherwise} \end{cases}$

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

1. $P \in \mathcal{P}(X_\perp)$: set of results of a nondeterministic computation in X, $\perp$ denoting *computation still on progress*

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

1. $P \in \mathcal{P}(X_\perp)$: set of results of a nondeterministic computation in X, $\perp$ denoting *computation still on progress*
2. $P \sqsubseteq Q$: "Q is a nondeterministic computer state that can come after P"

Plotkin Powerdomain

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

1. $P \in \mathcal{P}(X_\perp)$: set of results of a nondeterministic computation in X, $\perp$ denoting *computation still on progress*
2. $P \sqsubseteq Q$: "Q is a nondeterministic computer state that can come after P"
3. $\{x, y, \perp\} \sqsubseteq \{x, y, z\}$: after an evaluation $\perp$ terminated resulted in z

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

1. $P \in \mathcal{P}(X_\perp)$: set of results of a nondeterministic computation in X, $\perp$ denoting *computation still on progress*
2. $P \sqsubseteq Q$: "Q is a nondeterministic computer state that can come after P"
3. $\{x, y, \perp\} \sqsubseteq \{x, y, z\}$: after an evaluation $\perp$ terminated resulted in z
4. $\{x, y, \perp\} \sqsubseteq \{x, y, z, \perp\}$: after an evaluation $\perp$ nondeterministically terminated resulted in z and nondeterministically still goes on

Definition
For a set X, the Plotkin powerdomain $\mathcal{P}(X_\perp)$ over its flat domain $X_\perp$ is a poset on

$$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$$

whose partial order is

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

Examples

1. $P \in \mathcal{P}(X_\perp)$: set of results of a nondeterministic computation in X, $\perp$ denoting *computation still on progress*
2. $P \sqsubseteq Q$: "Q is a nondeterministic computer state that can come after P"
3. $\{x, y, \perp\} \sqsubseteq \{x, y, z\}$: after an evaluation $\perp$ terminated resulted in z
4. $\{x, y, \perp\} \sqsubseteq \{x, y, z, \perp\}$: after an evaluation $\perp$ nondeterministically terminated resulted in z and nondeterministically still goes on
5. $\{x, y\} \not\sqsubseteq \{x, y, z\}$: since $\{x, y\}$ is already finished computation, z cannot be added

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Lemma

1. *For any set X, $\mathcal{P}(X_\bot)$ is a ω-cpo which admits a bottom element $\{\bot\}$*

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Lemma

1. *For any set* X, $\mathcal{P}(X_\bot)$ *is a* ω*-cpo which admits a bottom element* $\{\bot\}$
2. *For a function* $f : X_1 \times \cdots \times X_d \to \mathcal{P}(Y_\bot)$,
   define $f^\dagger : \mathcal{P}((X_1)_\bot) \times \cdots \times \mathcal{P}((X_d)_\bot) \to \mathcal{P}(Y_\bot)$ *by*

$$f^\dagger(P_1, \cdots, P_d) \coloneqq \bigcup_{x_i \in P_i} \begin{cases} f(x_1, \cdots, x_d) & \text{if } x_i \neq \bot \\ \{\bot\} & \text{otherwise} \end{cases}$$

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Lemma

1. *For any set X, $\mathcal{P}(X_\bot)$ is a ω-cpo which admits a bottom element $\{\bot\}$*
2. *For a function $f : X_1 \times \cdots \times X_d \to \mathcal{P}(Y_\bot)$,*
   define $f^\dagger : \mathcal{P}((X_1)_\bot) \times \cdots \times \mathcal{P}((X_d)_\bot) \to \mathcal{P}(Y_\bot)$ by

$$f^\dagger(P_1, \cdots, P_d) \coloneqq \bigcup_{x_i \in P_i} \begin{cases} f(x_1, \cdots, x_d) & \text{if } x_i \neq \bot \\ \{\bot\} & \text{otherwise} \end{cases}$$

The Kleisli-composition $g^\dagger \circ f : X \to \mathcal{P}(Y_\bot)$ is continuous in both arguments: for any chain f_i

$$\bigsqcup_{i \in \mathbb{N}} (g^\dagger \circ f_i) = g^\dagger \circ \bigsqcup_{i \in \mathbb{N}} f_i \quad \text{and} \quad \bigsqcup_{i \in \mathbb{N}} (f_i^\dagger \circ g) = \Big(\bigsqcup_{i \in \mathbb{N}} f_i\Big)^\dagger \circ g.$$

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Lemma

3 *For any set* X *and an index set* $S \in \mathcal{P}(X_\bot)$,
any bottom preserving index $f : S \to \mathcal{P}(Y_\bot)$ *s.t.* $f(\bot) = \{\bot\}$

$$\Big(\bigcup_{s \in S} f(s) \Big) \in \mathcal{P}(Y_\bot)$$

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \bot \in S\}$ and $P \sqsubseteq Q \iff (\bot \in P \wedge P \subseteq Q \cup \{\bot\}) \vee (\bot \notin P \wedge P = Q)$.

Lemma

3 *For any set* X *and an index set* $S \in \mathcal{P}(X_\bot)$,
any bottom preserving index $f : S \to \mathcal{P}(Y_\bot)$ *s.t.* $f(\bot) = \{\bot\}$

$$\Big(\bigcup_{s \in S} f(s)\Big) \in \mathcal{P}(Y_\bot)$$

4 *The union is monotone: For any* $f, g : S \to \mathcal{P}(Y_\bot)$

$$f \sqsubseteq g \Rightarrow \bigcup_{s \in S} f(s) \subseteq \bigcup_{s \in S} g(s)$$

Definition

$\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$ and $P \sqsubseteq Q \iff (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q)$.

Lemma

3 *For any set* X *and an index set* $S \in \mathcal{P}(X_\perp)$,
any bottom preserving index $f : S \to \mathcal{P}(Y_\perp)$ *s.t.* $f(\perp) = \{\perp\}$

$$\Big(\bigcup_{s \in S} f(s)\Big) \in \mathcal{P}(Y_\perp)$$

4 *The union is monotone: For any* $f, g : S \to \mathcal{P}(Y_\perp)$

$$f \sqsubseteq g \Rightarrow \bigcup_{s \in S} f(s) \subseteq \bigcup_{s \in S} g(s)$$

5 *The union is continuous: For any chain* $f_i : S \to \mathcal{P}(Y_\perp)$

$$\bigcup_{s \in S} \bigsqcup_{i \in \mathbb{N}} f_i(s) = \bigsqcup_{i \in \mathbb{N}} \bigcup_{s \in S} f_i(s)$$

Plotkin Powerdomain - Conditions and While

For any set X, define $\mathrm{Cond}_X : \mathcal{P}(2_\perp) \times \mathcal{P}(X_\perp) \times \mathcal{P}(X_\perp) \to \mathcal{P}(X_\perp)$ by

$$\mathrm{Cond}_X(B, P, Q) := \bigcup_{b \in B} \begin{cases} P & \text{if } b = \texttt{true} \\ Q & \text{if } b = \texttt{false} \\ \{\perp\} & \text{if } b = \perp \end{cases}$$

Plotkin Powerdomain - Conditions and While

For any set X, define $\mathrm{Cond}_X : \mathcal{P}(2_\perp) \times \mathcal{P}(X_\perp) \times \mathcal{P}(X_\perp) \to \mathcal{P}(X_\perp)$ by

$$\mathrm{Cond}_X(B, P, Q) := \bigcup_{b \in B} \begin{cases} P & \text{if } b = \texttt{true} \\ Q & \text{if } b = \texttt{false} \\ \{\perp\} & \text{if } b = \perp \end{cases}$$

- nondeterministic condition $b : X \to \mathcal{P}(2_\perp)$
- nondeterministic command $c : X \to \mathcal{P}(X_\perp)$

$$[\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]\gamma = \mathrm{Cond}(b(\gamma), [\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]^\dagger \circ c(\gamma), \{\gamma\})$$

Plotkin Powerdomain - Conditions and While

For any set X, define $\mathrm{Cond}_X : \mathcal{P}(2_\perp) \times \mathcal{P}(X_\perp) \times \mathcal{P}(X_\perp) \to \mathcal{P}(X_\perp)$ by

$$\mathrm{Cond}_X(B, P, Q) := \bigcup_{b \in B} \begin{cases} P & \text{if } b = \texttt{true} \\ Q & \text{if } b = \texttt{false} \\ \{\perp\} & \text{if } b = \perp \end{cases}$$

- nondeterministic condition $b : X \to \mathcal{P}(2_\perp)$
- nondeterministic command $c : X \to \mathcal{P}(X_\perp)$

$$[\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]\gamma = \mathrm{Cond}(b(\gamma), [\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]^\dagger \circ c(\gamma), \{\gamma\})$$

is the fixed point of

$$\mathcal{W}_{b,c}(f : X \to \mathcal{P}(X_\perp)) := \gamma \mapsto \mathrm{Cond}_X(b(\gamma), f^\dagger \circ c(\gamma), \{\gamma\})$$

Plotkin Powerdomain - Conditions and While

For any set X, define $\mathrm{Cond}_X : \mathcal{P}(\mathbb{2}_\perp) \times \mathcal{P}(X_\perp) \times \mathcal{P}(X_\perp) \to \mathcal{P}(X_\perp)$ by

$$\mathrm{Cond}_X(B, P, Q) := \bigcup_{b \in B} \begin{cases} P & \text{if } b = \texttt{true} \\ Q & \text{if } b = \texttt{false} \\ \{\perp\} & \text{if } b = \perp \end{cases}$$

- nondeterministic condition $b : X \to \mathcal{P}(\mathbb{2}_\perp)$
- nondeterministic command $c : X \to \mathcal{P}(X_\perp)$

$$[\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]\gamma = \mathrm{Cond}(b(\gamma), [\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]^\dagger \circ c(\gamma), \{\gamma\})$$

is the fixed point of

$$\mathcal{W}_{b,c}(f : X \to \mathcal{P}(X_\perp)) := \gamma \mapsto \mathrm{Cond}_X(b(\gamma), f^\dagger \circ c(\gamma), \{\gamma\})$$

which is continuous:

$$\begin{aligned} \mathcal{W}_{b,c}\Big(\bigsqcup f_i\Big)(\gamma) &= \mathrm{Cond}_X\Big(b(\gamma), \Big(\bigsqcup f_i\Big)^\dagger \circ c(\gamma), \{\gamma\}\Big) \\ &= \mathrm{Cond}_X\Big(b(\gamma), \bigsqcup (f_i^\dagger \circ c)(\gamma), \{\gamma\}\Big) \\ &= \bigsqcup \mathrm{Cond}_X(b(\gamma), f_i^\dagger \circ c(\gamma), \{\gamma\}) = \bigsqcup \mathcal{W}_{b,c}(f_i)(\gamma) \end{aligned}$$

Plotkin Powerdomain - Conditions and While

For any set X, define $\mathrm{Cond}_X : \mathcal{P}(\mathbb{2}_\perp) \times \mathcal{P}(X_\perp) \times \mathcal{P}(X_\perp) \to \mathcal{P}(X_\perp)$ by

$$\mathrm{Cond}_X(B, P, Q) := \bigcup_{b \in B} \begin{cases} P & \text{if } b = \texttt{true} \\ Q & \text{if } b = \texttt{false} \\ \{\perp\} & \text{if } b = \perp \end{cases}$$

- nondeterministic condition $b : X \to \mathcal{P}(\mathbb{2}_\perp)$
- nondeterministic command $c : X \to \mathcal{P}(X_\perp)$

$$[\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]\gamma = \mathrm{Cond}(b(\gamma), [\![\texttt{while } b \texttt{ do } c \texttt{ end}]\!]^\dagger \circ c(\gamma), \{\gamma\})$$

is the fixed point of

$$\mathcal{W}_{b,c}(f : X \to \mathcal{P}(X_\perp)) := \gamma \mapsto \mathrm{Cond}_X(b(\gamma), f^\dagger \circ c(\gamma), \{\gamma\})$$

which is continuous:

$$\begin{aligned} \mathcal{W}_{b,c}\big(\bigsqcup f_i\big)(\gamma) &= \mathrm{Cond}_X\Big(b(\gamma), \big(\bigsqcup f_i\big)^\dagger \circ c(\gamma), \{\gamma\}\Big) \\ &= \mathrm{Cond}_X\Big(b(\gamma), \bigsqcup (f_i^\dagger \circ c)(\gamma), \{\gamma\}\Big) \\ &= \bigsqcup \mathrm{Cond}_X(b(\gamma), f_i^\dagger \circ c(\gamma), \{\gamma\}) = \bigsqcup \mathcal{W}_{b,c}(f_i)(\gamma) \end{aligned}$$

- Define $\mathsf{While}(b, c) := \mathsf{LFP}(\mathcal{W}_{b,c})$

Plotkin Powerdomain in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Depending on the meaning of $\bot$, there can be different Plotkin powerdomains in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$:

Definition (For $\star \in \{\natural, \sharp, \natural\}$)
Define the closure: for any non-empty subset $S \subseteq X \cup \{\star\}$,

$$S_\star \coloneqq \begin{cases} S \cup \{\star\} & \text{if } S \text{ is infinite} \\ S & \text{otherwise} \end{cases}$$

Then monad $\mathbf{P}_\star : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ which on an assembly $\mathbf{X}$ is the natural sub-assembly of $\mathsf{M}(\star\mathbf{X})$ induced by $_\star$:

$$\varphi \Vdash_{\mathbf{P}(\mathbf{X}_\star)} S \Leftrightarrow \varphi \Vdash_{\mathsf{M}(\star\mathbf{X})} S$$

Lemma

1. $\mathbf{P}_\flat$, *when* $\perp$ *is identified as nontermination:*
   the least fixed-point is uniformly computable: there is

$$\mathbf{LFP} : (\mathbf{P}(\mathbf{X}_\flat))^{\mathbf{N}} \to \flat\mathbf{P}(\mathbf{X}_\flat)$$

   defined on chains that computes the least fixed points.

Lemma

1. $\mathbf{P}_\flat$, *when* $\bot$ *is identified as nontermination:*
   the least fixed-point is uniformly computable: there is

$$\mathbf{LFP} : (\mathbf{P}(\mathbf{X}_\flat))^{\mathbf{N}} \to \flat\mathbf{P}(\mathbf{X}_\flat)$$

   defined on chains that computes the least fixed points.

2. $\mathbf{P}_\natural$, *when* $\bot$ *is identified as general partiality:*
   the least fixed-point is not uniformly computable.
   However, the while operator is uniformly computable: there is

$$\mathbf{While} : \mathbf{P}(\mathbf{2}_\natural)^{\mathbf{X}} \times \mathbf{P}(\mathbf{X}_\natural)^{\mathbf{X}} \to (\mathbf{X}_\natural)^{\mathbf{X}}$$

   computing the least fixed point of $\mathcal{W}_{b,c}$

Lemma

1. $\mathbf{P}_\flat$, *when* $\bot$ *is identified as nontermination:*
   the least fixed-point is uniformly computable: there is

$$\mathbf{LFP} : (\mathbf{P}(\mathbf{X}_\flat))^{\mathbf{N}} \to \flat\mathbf{P}(\mathbf{X}_\flat)$$

   defined on chains that computes the least fixed points.

2. $\mathbf{P}_\natural$, *when* $\bot$ *is identified as general partiality:*
   the least fixed-point is not uniformly computable.
   However, the while operator is uniformly computable: there is

$$\mathbf{While} : \mathbf{P}(\mathbf{2}_\natural)^{\mathbf{X}} \times \mathbf{P}(\mathbf{X}_\natural)^{\mathbf{X}} \to (\mathbf{X}_\natural)^{\mathbf{X}}$$

   computing the least fixed point of $\mathcal{W}_{b,c}$

3. $\mathbf{P}_\sharp$, *when* $\bot$ *identified as fixing: the least fixed-point nor the while operator is computable.*

Conclusion

Today, we

- identified a core fragment of imperative exact real computation software as a simple imperative language with partial comparison of reals and choose
- reviewed computable analysis in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ and specified some monads for various partiality and nondeterminism
- reviewed Plotkin powerdomain and specified some representations on them

Conclusion

Today, we

- identified a core fragment of imperative exact real computation software as a simple imperative language with partial comparison of reals and choose
- reviewed computable analysis in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ and specified some monads for various partiality and nondeterminism
- reviewed Plotkin powerdomain and specified some representations on them

Tomorrow, we

- define formal semantics both denotational and operational using the elements specified today
- devise proof rules for formal verification that is sound w.r.t. the denotational semantics
- suggest an extension structure of the language with further continuous objects
- introduce some further designs toward having limits and multi-limits

References


Peter Hertling, *A real number structure that is effectively categorical*, Math. Log. Q. **45** (1999), 147–182.


Klaus Weihrauch, *Computable analysis: An introduction (texts in theoretical computer science. an EATCS series)*.