Denotational Semantics

Meven Lennon-Bertrand

Lectures for Part II CST 2024/2025

Practicalities

- My mail: mgapb2@cam.ac.uk. Do not hesitate to ask questions!
- Course notes will be updated, keep an eye on the course webpage.

Introduction

What is this course about?

- Formal methods: mathematical tools for the specification, development, analysis and verification of software and hardware systems.

What is this course about?

- **Formal methods**: mathematical tools for the specification, development, analysis and verification of software and hardware systems.
- **Programming language theory**: design, implementation, tooling and reasoning for/about programming languages.

What is this course about?

- **Formal methods**: mathematical tools for the specification, development, analysis and verification of software and hardware systems.
- **Programming language theory**: design, implementation, tooling and reasoning for/about programming languages.
- **Programming language semantics**: what is the (mathematical) meaning of a program?

What is this course about?

- Formal methods: mathematical tools for the specification, development, analysis and verification of software and hardware systems.
- Programming language theory: design, implementation, tooling and reasoning for/about programming languages.
- Programming language semantics: what is the (mathematical) meaning of a program?

Goal: give an abstract and compositional (mathematical) model of programs.

Why?

- **Insight**: exposes the mathematical "essence" of programming language ideas.

Why?

- **Insight**: exposes the mathematical “essence” of programming language ideas.
- **Documentation**: precise but intuitive, machine-independent specification.

Why?

- **Insight**: exposes the mathematical “essence” of programming language ideas.
- **Documentation**: precise but intuitive, machine-independent specification.
- **Language design**: feedback from semantics (functional programming, monads & handlers, linearity...).

Why?

- **Insight**: exposes the mathematical “essence” of programming language ideas.
- **Documentation**: precise but intuitive, machine-independent specification.
- **Language design**: feedback from semantics (functional programming, monads & handlers, linearity…).
- **Rigour**: powerful way to justify formal methods.

Styles of formal semantics

- Operational
- Axiomatic
- Denotational

Styles of formal semantics

- **Operational**: meaning of a program in terms of the *steps of computation* it takes during execution (see Part IB Semantics).
- **Axiomatic**
- **Denotational**

Styles of formal semantics

- **Operational**: meaning of a program in terms of the *steps of computation* it takes during execution (see Part IB Semantics).
- **Axiomatic**: meaning of a program in terms of a *program logic* to reason about it (see Part II Hoare Logic & Model Checking).
- Denotational

Styles of formal semantics

- **Operational**: meaning of a program in terms of the *steps of computation* it takes during execution (see Part IB Semantics).
- **Axiomatic**: meaning of a program in terms of a *program logic* to reason about it (see Part II Hoare Logic & Model Checking).
- **Denotational**: meaning of a program defined abstractly as object of some suitable *mathematical structure* (see this course).

Denotational semantics in a nutshell

$$
\begin{aligned}
\text{Syntax} &\xrightarrow{[\![-]\!]} \text{Semantics} \\
\text{Program } P &\mapsto \text{Denotation } [\![P]\!] \\
\\
\text{Arithmetic expression} &\mapsto \text{Number} \\
\text{Boolean circuit} &\mapsto \text{Boolean function} \\
\text{Recursive program} &\mapsto \text{Partial recursive function} \\
&\dots
\end{aligned}
$$

Denotational semantics in a nutshell

$$\begin{aligned}
\text{Syntax} &\quad \xrightarrow{[\![-]\!]} \quad \text{Semantics} \\
\text{Program } P &\quad \mapsto \quad \text{Denotation } [\![P]\!]
\end{aligned}$$

$$\begin{aligned}
\text{Arithmetic expression} &\quad \mapsto \quad \text{Number} \\
\text{Boolean circuit} &\quad \mapsto \quad \text{Boolean function} \\
\text{Recursive program} &\quad \mapsto \quad \text{Partial recursive function} \\
&\quad \dots \\
\text{Type} &\quad \mapsto \quad \text{Domain} \\
\text{Program} &\quad \mapsto \quad \text{Continuous functions between domains}
\end{aligned}$$

Properties of denotational semantics

Abstraction

- mathematical object, implementation/machine independent;
- captures the concept of a programming language construct;
- should relate to practical implementations, though…

Properties of denotational semantics

Abstraction

- mathematical object, implementation/machine independent;
- captures the concept of a programming language construct;
- should relate to practical implementations, though…

Compositionality

- The denotation of a whole is defined using the *denotation* of its parts;
- $[\![P]\!]$ represents the contribution of P to *any* program containing P;
- More flexible and expressive than whole-program semantics.

Introduction

A basic example

Imp syntax

Programs

$$C \in \mathbf{Prog} ::= \texttt{skip} \mid L := A \mid C; C \mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C \mid \texttt{while}\ B\ \texttt{do}\ C$$

Programs

ranges over a set $\mathbb{L}$ of *locations*

$$C \in \mathbf{Prog} ::= \mathtt{skip} \mid L := A \mid C; C \mid \mathtt{if}\ B\ \mathtt{then}\ C\ \mathtt{else}\ C \mid \mathtt{while}\ B\ \mathtt{do}\ C$$

Imp syntax

Arithmetic expressions

$$A \in \mathbf{Aexp} ::= \underline{n} \mid L \mid A + A \mid \ldots$$

Programs

$$C \in \mathbf{Prog} ::= \texttt{skip} \mid L := A \mid C; C \mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C \mid \texttt{while}\ B\ \texttt{do}\ C$$

ranges over *integers*

Arithmetic expressions

$$A \in \mathbf{Aexp} ::= \underline{n} \mid L \mid A + A \mid \ldots$$

Programs

$$C \in \mathbf{Prog} ::= \texttt{skip} \mid L := A \mid C;C \mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C \mid \texttt{while}\ B\ \texttt{do}\ C$$

Imp syntax

Arithmetic expressions

$$A \in \mathbf{Aexp} ::= \underline{n} \mid L \mid A + A \mid \ldots$$

Boolean expressions

$$B \in \mathbf{Bexp} ::= \mathtt{true} \mid \mathtt{false} \mid A = A \mid \neg B \mid \ldots$$

Programs

$$C \in \mathbf{Prog} ::= \mathtt{skip} \mid L := A \mid C; C \mid \mathtt{if}\ B\ \mathtt{then}\ C\ \mathtt{else}\ C \mid \mathtt{while}\ B\ \mathtt{do}\ C$$

Denotation functions – naïvely

$$\mathcal{A}: \quad \mathbf{Aexp} \to \mathbb{Z}$$

where

$$\mathbb{Z} \quad = \quad \{\ldots, -1, 0, 1, \ldots\}$$

Denotation functions – naïvely

$$\mathcal{A} : \quad \mathbf{Aexp} \rightarrow \mathbb{Z}$$
$$\mathcal{B} : \quad \mathbf{Bexp} \rightarrow \mathbb{B}$$

where

$$\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$$
$$\mathbb{B} = \{\text{true}, \text{false}\}$$

Arithmetic expressions?

$$\mathcal{A}[\![\underline{n}]\!] = n$$

$$\mathcal{A}[\![A_1 + A_2]\!] = \mathcal{A}[\![A_1]\!] + \mathcal{A}[\![A_2]\!]$$

Arithmetic expressions?

$$\begin{aligned}
\mathcal{A}[\![\underline{n}]\!] &= n \\
\mathcal{A}[\![A_1 + A_2]\!] &= \mathcal{A}[\![A_1]\!] + \mathcal{A}[\![A_2]\!] \\
\mathcal{A}[\![L]\!] &= ???
\end{aligned}$$

Denotation functions – less naïvely

$$\text{State} = (\mathbb{L} \rightarrow \mathbb{Z})$$

Denotation functions – less naïvely

$$\text{State} = (\mathbb{L} \to \mathbb{Z})$$

$$\mathcal{A} : \mathbf{Aexp} \to (\text{State} \to \mathbb{Z})$$
$$\mathcal{B} : \mathbf{Bexp} \to (\text{State} \to \mathbb{B})$$

where

$$\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$$
$$\mathbb{B} = \{\text{true}, \text{false}\}\,.$$

$$\text{State} = (\mathbb{L} \to \mathbb{Z})$$

$$\begin{aligned}
\mathcal{A} &: \textbf{Aexp} \to (\text{State} \to \mathbb{Z}) \\
\mathcal{B} &: \textbf{Bexp} \to (\text{State} \to \mathbb{B}) \\
\mathcal{C} &: \textbf{Prog} \to (\text{State} \to \text{State})
\end{aligned}$$

where

$$\begin{aligned}
\mathbb{Z} &= \{\dots, -1, 0, 1, \dots\} \\
\mathbb{B} &= \{\text{true}, \text{false}\}\,.
\end{aligned}$$

Semantics of arithmetic expressions

$$\mathcal{A}[\![\underline{n}]\!] = \lambda s \in \text{State}.\, n$$

$$\mathcal{A}[\![A_1 + A_2]\!] = \lambda s \in \text{State}.\, \mathcal{A}[\![A_1]\!](s) + \mathcal{A}[\![A_2]\!](s)$$

$$
\begin{aligned}
\mathcal{A}\left[\!\left[\underline{n}\right]\!\right] &= \lambda s \in \text{State}.\, n \\
\mathcal{A}[\![A_1 + A_2]\!] &= \lambda s \in \text{State}.\, \mathcal{A}[\![A_1]\!]\,(s) + \mathcal{A}[\![A_2]\!]\,(s) \\
\mathcal{A}[\![L]\!] &= \lambda s \in \text{State}.\, s(L)
\end{aligned}
$$

Semantics of boolean expressions

$$
\begin{aligned}
\mathcal{B}[\![\texttt{true}]\!] &= \lambda s \in \text{State}.\ \text{true} \\
\mathcal{B}[\![\texttt{false}]\!] &= \lambda s \in \text{State}.\ \text{false} \\
\mathcal{B}[\![A_1 = A_2]\!] &= \lambda s \in \text{State}.\ \text{eq}\,(\mathcal{A}[\![A_1]\!]\,(s), \mathcal{A}[\![A_2]\!]\,(s)) \\
& \quad \text{where } \text{eq}(a, a') = \begin{cases} \text{true} & \text{if } a = a' \\ \text{false} & \text{if } a \neq a' \end{cases}
\end{aligned}
$$

Semantics of programs

$$\mathcal{C}[\![\texttt{skip}]\!] \quad = \quad \lambda s \in \text{State}.\ s$$

$$\mathcal{C}[\![\texttt{skip}]\!] = \lambda s \in \text{State}.\ s$$

$$\mathcal{C}[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C']\!] = \lambda s \in \text{State}.\ \text{if}\,(\mathcal{B}[\![B]\!]\,(s), \mathcal{C}[\![C]\!]\,(s), \mathcal{C}[\![C']\!]\,(s))$$

$$\text{where}\ \text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases}$$

$$\mathcal{C}[\![\texttt{skip}]\!] = \lambda s \in \text{State}.\ s$$

This is compositionality!

$$\mathcal{C}[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C']\!] = \lambda s \in \text{State}.\ \text{if}\,(\mathcal{B}[\![B]\!]\,(s), \mathcal{C}[\![C]\!]\,(s), \mathcal{C}[\![C']\!]\,(s))$$

where $\text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases}$

Semantics of programs

$$\mathcal{C}[\![\texttt{skip}]\!] = \lambda s \in \text{State}.\ s$$

$$\mathcal{C}[\![\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C']\!] = \lambda s \in \text{State}.\ \text{if}\left(\mathcal{B}[\![B]\!](s), \mathcal{C}[\![C]\!](s), \mathcal{C}[\![C']\!](s)\right)$$

$$\text{where } \text{if}(b, x, x') = \begin{cases} x & \text{if } b = \text{true} \\ x' & \text{if } b = \text{false} \end{cases}$$

$$\mathcal{C}[\![L := A]\!] = \lambda s \in \text{State}.\ s[L \mapsto \mathcal{A}[\![A]\!](s)]$$

$$\text{where } s[L \mapsto n](L') = \begin{cases} n & \text{if } L' = L \\ s(L) & \text{otherwise} \end{cases}$$

$$
\begin{aligned}
\mathcal{C}[\![\texttt{skip}]\!] &= \lambda s \in \mathrm{State}.\, s \\[1em]
\mathcal{C}\left[\!\left[\texttt{if } B \texttt{ then } C \texttt{ else } C'\right]\!\right] &= \lambda s \in \mathrm{State}.\, \mathrm{if}\,(\mathcal{B}[\![B]\!]\,(s), \mathcal{C}[\![C]\!]\,(s), \mathcal{C}\left[\!\left[C'\right]\!\right](s)) \\
&\ \text{where } \mathrm{if}(b, x, x') = \begin{cases} x & \text{if } b = \mathrm{true} \\ x' & \text{if } b = \mathrm{false} \end{cases} \\[1em]
\mathcal{C}[\![L := A]\!] &= \lambda s \in \mathrm{State}.\, s[L \mapsto \mathcal{A}[\![A]\!]\,(s)] \\
&\ \text{where } s[L \mapsto n](L') = \begin{cases} n & \text{if } L' = L \\ s(L) & \text{otherwise} \end{cases} \\[1em]
\mathcal{C}\left[\!\left[C; C'\right]\!\right] &= \mathcal{C}\left[\!\left[C'\right]\!\right] \circ \mathcal{C}[\![C]\!] \\
&= \lambda s \in \mathrm{State}.\, \mathcal{C}\left[\!\left[C'\right]\!\right](\mathcal{C}[\![C]\!]\,(s))
\end{aligned}
$$

Introduction

A semantics for loops

Semantics of loops?

This is all very nice, but...

$$[\![\texttt{while}\ B\ \texttt{do}\ C]\!] = ???$$

Semantics of loops?

This is all very nice, but...

$$[\![\texttt{while}\ B\ \texttt{do}\ C]\!] = ???$$

Remember:

- $(\texttt{while}\ B\ \texttt{do}\ C, s) \rightsquigarrow (\texttt{if}\ B\ \texttt{then}\ (C; \texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else}\ \texttt{skip}, s)$
- we want a *compositional* semantic: $[\![\texttt{while}\ B\ \texttt{do}\ C]\!]$ in terms of $[\![C]\!]$ and $[\![B]\!]$

$$\begin{aligned}\llbracket \texttt{while}\ B\ \texttt{do}\ C \rrbracket &= \llbracket \texttt{if}\ B\ \texttt{then}\ (C; \texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else}\ \texttt{skip} \rrbracket \\ &= \lambda s \in \mathrm{State}.\ \mathrm{if}(\llbracket B \rrbracket, \llbracket \texttt{while}\ B\ \texttt{do}\ C \rrbracket \circ \llbracket C \rrbracket (s), s)\end{aligned}$$

$$
\begin{aligned}
[\![\texttt{while}\ B\ \texttt{do}\ C]\!] &= [\![\texttt{if}\ B\ \texttt{then}\ (C;\texttt{while}\ B\ \texttt{do}\ C)\ \texttt{else}\ \texttt{skip}]\!] \\
&= \lambda s \in \mathrm{State}.\ \mathrm{if}([\![B]\!], [\![\texttt{while}\ B\ \texttt{do}\ C]\!] \circ [\![C]\!](s), s)
\end{aligned}
$$

Not a direct definition for $[\![\texttt{while}\ B\ \texttt{do}\ C]\!]$... But a fixed point equation!

$$
[\![\texttt{while}\ B\ \texttt{do}\ C]\!] = F_{[\![B]\!],[\![C]\!]}([\![\texttt{while}\ B\ \texttt{do}\ C]\!])
$$

$$
\text{where} \quad F_{b,c} : \begin{array}{rcl} (\mathrm{State} \to \mathrm{State}) & \to & (\mathrm{State} \to \mathrm{State}) \\ w & \mapsto & \lambda s \in \mathrm{State}.\ \mathrm{if}(b(s), w \circ c(s), s). \end{array}
$$

Now we have a goal

- Why/when does $w = F_{b,c}(w)$ have a solution?
- What if it has several solutions? Which one should be our $[\![\texttt{while}\ B\ \texttt{do}\ C]\!]$?

Introduction

A taste of domain theory

Total functions are not enough

Forget about State for a second, consider these equations ($f \in \mathbb{Z} \to \mathbb{Z}$):

$$f(x) = f(x) + 1 \tag{1}$$

$$f(x) = f(x) \tag{2}$$

What about their fixed points?

Forget about State for a second, consider these equations ($f \in \mathbb{Z} \to \mathbb{Z}$) :

$$f(x) = f(x) + 1 \tag{1}$$
$$f(x) = f(x) \tag{2}$$

What about their fixed points?

- No function satisfies Eq. (1)!
- All functions satisfy Eq. (2)!

Both functions should diverge!

Partial functions to the rescue

Both functions should diverge!

New rule: partial functions $f \in \mathbb{Z} \rightharpoonup \mathbb{Z}$

Partial functions to the rescue

Both functions should diverge!

New rule: partial functions $f \in \mathbb{Z} \rightharpoonup \mathbb{Z}$

$$f(x) = f(x) + 1$$

has a unique solution: the nowhere-defined function $\bot$

Partial functions to the rescue

Both functions should diverge!

New rule: partial functions $f \in \mathbb{Z} \rightharpoonup \mathbb{Z}$

$$f(x) = f(x) + 1$$

has a unique solution: the nowhere-defined function $\bot$

But

$$f(x) = f(x)$$

Has even more solutions now...

Partial order on $\mathbb{Z} \rightharpoonup \mathbb{Z}$:

$w \sqsubseteq w'$ if for all $s \in \mathbb{Z}$, if w is defined at s so is w' and moreover $w(s) = w'(s)$.
if the graph of w is included in the graph of w'.

An order on partial functions

Partial order on $\mathbb{Z} \rightharpoonup \mathbb{Z}$:

$w \sqsubseteq w'$ if for all $s \in \mathbb{Z}$, if w is defined at s so is w' and moreover $w(s) = w'(s)$.
if the graph of w is included in the graph of w'.

Least element $\bot \in \mathbb{Z} \rightharpoonup \mathbb{Z}$:

$\bot$ = totally undefined partial function

An order on partial functions

Partial order on $\mathbb{Z} \rightharpoonup \mathbb{Z}$:

$w \sqsubseteq w'$ if for all $s \in \mathbb{Z}$, if w is defined at s so is w' and moreover $w(s) = w'(s)$.
if the graph of w is included in the graph of w'.

Least element $\bot \in \mathbb{Z} \rightharpoonup \mathbb{Z}$:

$\bot$ = totally undefined partial function

$\bot$ is the least solution to $f(x) = f(x)$, making it "canonical".

Back to loops

$$\mathcal{C} : \mathbf{Prog} \to (\text{State} \rightharpoonup \text{State})$$

Back to loops

$$\mathcal{C} : \mathbf{Prog} \to (\text{State} \rightharpoonup \text{State})$$

$$[\![\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)]\!]$$

Back to loops

$$\mathcal{C} : \mathbf{Prog} \to (\mathrm{State} \rightharpoonup \mathrm{State})$$

$$[\![\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)]\!]$$

should be some w such that:

$$w = F_{[\![X>0]\!],[\![Y:=X*Y;X:=X-1]\!]}(w).$$

$$\mathcal{C} : \textbf{Prog} \to (\text{State} \rightharpoonup \text{State})$$

$$[\![\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)]\!]$$

should be some w such that:

$$w = F_{[\![X>0]\!],[\![Y:=X*Y;X:=X-1]\!]}(w).$$

That is, we are looking for a fixed point of the following F:

$$\begin{aligned} F : (\text{State} \rightharpoonup \text{State}) &\to (\text{State} \rightharpoonup \text{State}) \\ w &\mapsto \lambda[X \mapsto x, Y \mapsto y].\begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w\,([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0 \end{cases} \end{aligned}$$

Approximating the least fixed point

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

Approximating the least fixed point

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_1[X \mapsto x, Y \mapsto y] = F(\bot)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ \text{undefined} & \text{if } x \geq 1 \end{cases}$$

Approximating the least fixed point

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_2[X \mapsto x, Y \mapsto y] = F(w_1)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\ \text{undefined} & \text{if } x \geq 2 \end{cases}$$

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_n[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_n[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

$$w_0 \sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots$$

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_n[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

$$w_0 \sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots \sqsubseteq w_\infty?$$

Approximating the least fixed point

Define $w_n = F^n(w)$, that is $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

$$w_n[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } 0 \leq x < n \\ \text{undefined} & \text{if } x \geq n \end{cases}$$

$$w_0 \sqsubseteq w_1 \sqsubseteq \ldots \sqsubseteq w_n \sqsubseteq \ldots \sqsubseteq w_\infty$$

$$w_\infty[X \mapsto x, Y \mapsto y] = \bigsqcup_{i \in \mathbb{N}} w_i = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } x \geq 0 \end{cases}$$

We have our semantics

$$F(w_\infty)[X \mapsto x, Y \mapsto y]$$

We have our semantics

$$F(w_\infty)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x - 1, Y \mapsto x \cdot y] & \text{if } x > 0 \end{cases} \qquad \text{(definition of } F\text{)}$$

We have our semantics

$$F(w_\infty)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x-1, Y \mapsto x \cdot y] & \text{if } x > 0 \end{cases} \qquad \text{(definition of } F\text{)}$$

$$= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto (x-1)! \cdot x \cdot y] & \text{if } x > 0 \end{cases} \qquad \text{(definition of } w_\infty\text{)}$$

$$
\begin{aligned}
F(w_\infty)[X \mapsto x, Y \mapsto y] &= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x-1, Y \mapsto x \cdot y] & \text{if } x > 0 \end{cases} && \text{(definition of } F\text{)} \\
&= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto (x-1)! \cdot x \cdot y] & \text{if } x > 0 \end{cases} && \text{(definition of } w_\infty\text{)} \\
&= w_\infty[X \mapsto x, Y \mapsto y]
\end{aligned}
$$

We have our semantics

$$
\begin{aligned}
F(w_\infty)[X \mapsto x, Y \mapsto y] &= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w_\infty[X \mapsto x-1, Y \mapsto x \cdot y] & \text{if } x > 0 \end{cases} && \text{(definition of } F\text{)} \\
&= \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto (x-1)! \cdot x \cdot y] & \text{if } x > 0 \end{cases} && \text{(definition of } w_\infty\text{)} \\
&= w_\infty[X \mapsto x, Y \mapsto y]
\end{aligned}
$$

- $F(w_\infty) = w_\infty$ *i.e.* w_∞ is a fixed point of F;
- actually, the least fixed point;
- which agrees with the operational semantics (!)

The rest of this course

Part I domain theory → building mathematical tools

Part II denotational semantics for PCF

Least Fixed Points

Least Fixed Points

Posets and monotone functions

Partially ordered set

A partial order on a set D is a binary relation $\sqsubseteq$ that is

reflexive: $\forall d \in D.\ d \sqsubseteq d$

transitive: $\forall d, d', d'' \in D.\ d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$

antisymmetric: $\forall d, d' \in D.\ d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.

Partially ordered set

A partial order on a set D is a binary relation $\sqsubseteq$ that is

reflexive: $\forall d \in D.\ d \sqsubseteq d$

transitive: $\forall d, d', d'' \in D.\ d \sqsubseteq d' \sqsubseteq d'' \Rightarrow d \sqsubseteq d''$

antisymmetric: $\forall d, d' \in D.\ d \sqsubseteq d' \sqsubseteq d \Rightarrow d = d'$.

$$\textsc{Refl}\ \frac{}{x \sqsubseteq x} \qquad \textsc{Trans}\ \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \textsc{ASym}\ \frac{x \sqsubseteq y \qquad y \sqsubseteq x}{x = y}$$

Domain of partial functions $X \rightharpoonup Y$

Underlying set: partial functions f with domain of definition $\mathbf{dom}(f) \subseteq X$ and taking values in Y;

Domain of partial functions $X \rightharpoonup Y$

Underlying set: partial functions f with domain of definition $\mathrm{dom}(f) \subseteq X$ and taking values in Y;

Order: $f \sqsubseteq g$ if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $\forall x \in \mathrm{dom}(f).\ f(x) = g(x)$, *i.e.* if $\mathrm{graph}(f) \subseteq \mathrm{graph}(g)$.

Domain of partial functions $X \rightharpoonup Y$

Underlying set: partial functions f with domain of definition $\mathrm{dom}(f) \subseteq X$ and taking values in Y;

Order: $f \sqsubseteq g$ if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $\forall x \in \mathrm{dom}(f).\ f(x) = g(x)$, *i.e.* if $\mathrm{graph}(f) \subseteq \mathrm{graph}(g)$.

Proof!

Monotonicity

A function $f\colon D \to E$ between posets is monotone if

$$\forall d, d' \in D.\, d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$

A function $f\colon D \to E$ between posets is monotone if

$$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d').$$

$$\textsc{Mon}\ \frac{x \sqsubseteq y}{f(x) \sqsubseteq f(y)}$$

Least Fixed Points

Least elements and pre-fixed points

Least element

An element $d \in S$ is the least element of S if it satisfies

$$\forall x \in S.\ d \sqsubseteq x.$$

Least element

An element $d \in S$ is the least element of S if it satisfies

$$\forall x \in S.\ d \sqsubseteq x.$$

If it exists, it is unique , and is written $\bot_S$, or simply $\bot$.

$$\textsc{Least}\ \frac{x \in S}{\bot_S \sqsubseteq x}$$

Least element

An element $d \in S$ is the **least** element of S if it satisfies

$$\forall x \in S.\ d \sqsubseteq x.$$

If it exists, it is unique , and is written $\bot_S$, or simply $\bot$.

$$\textsc{Least}\ \frac{x \in S}{\bot_S \sqsubseteq x} \qquad \textsc{Asym}\ \frac{\textsc{Least}\ \dfrac{\bot'_S \in S}{\bot_S \sqsubseteq \bot'_S} \qquad \textsc{Least}\ \dfrac{\bot_S \in S}{\bot'_S \sqsubseteq \bot_S}}{\bot_S = \bot'_S}$$

Fixed point

A fixed point for a function $f\colon D \to D$ is an element $d \in D$ satisfying $f(d) = d$.

(Least) pre-fixed point

An element $d \in D$ is a pre-fixed point of f if it satisfies $f(d) \sqsubseteq d$.

(Least) pre-fixed point

An element $d \in D$ is a **pre-fixed point** of f if it satisfies $f(d) \sqsubseteq d$.

The **least pre-fixed point** of f, if it exists, will be written

$$\mathrm{fix}(f)$$

(Least) pre-fixed point

An element $d \in D$ is a **pre-fixed point** of f if it satisfies $f(d) \sqsubseteq d$.

The **least pre-fixed point** of f, if it exists, will be written

$$\mathrm{fix}(f)$$

It is thus (uniquely) specified by the two properties:

$$\textsc{lfp-fix}\ \frac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \textsc{lfp-least}\ \frac{f(d) \sqsubseteq d}{\mathrm{fix}(f) \sqsubseteq d}$$

Proofs with least fixed points

$$\text{LFP-FIX}\ \frac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)}$$

The least pre-fixed point is a pre-fixed point.

Proofs with least fixed points

$$\text{LFP-FIX}\ \frac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \text{LFP-LEAST}\ \frac{f(d) \sqsubseteq d}{\mathrm{fix}(f) \sqsubseteq d}$$

To prove $\mathrm{fix}(f) \sqsubseteq d$, it is enough to show $f(d) \sqsubseteq d$.

$$\textsc{lfp-fix}\ \frac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \textsc{lfp-least}\ \frac{f(d) \sqsubseteq d}{\mathrm{fix}(f) \sqsubseteq d}$$

Application: least pre-fixed points of monotone functions are (least) fixed points.

$$\textsc{Asym}\ \frac{\textsc{lfp-fix}\ \dfrac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \dfrac{}{\mathrm{fix}(f) \sqsubseteq f(\mathrm{fix}(f))}}{f(\mathrm{fix}(f)) = \mathrm{fix}(f)}$$

Proofs with least fixed points

$$\textsc{lfp-fix}\ \frac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad\qquad \textsc{lfp-least}\ \frac{f(d) \sqsubseteq d}{\mathrm{fix}(f) \sqsubseteq d}$$

Application: least pre-fixed points of monotone functions are (least) fixed points.

$$\textsc{Asym}\ \frac{\textsc{lfp-fix}\ \dfrac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \textsc{lfp-least}\ \dfrac{\textsc{Mon}\ \dfrac{\textsc{lfp-fix}\ \dfrac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)}}{f(f(\mathrm{fix}(f))) \sqsubseteq f(\mathrm{fix}(f))}}{\mathrm{fix}(f) \sqsubseteq f(\mathrm{fix}(f))}}{f(\mathrm{fix}(f)) = \mathrm{fix}(f)}$$

Least Fixed Points

Least upper bounds

Least upper bound of a chain

The **least upper bound** of countable increasing chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$, written $\bigsqcup_{n\geq 0} d_n$, satisfies the two following properties:

$$\text{LUB-BOUND} \quad \frac{}{x_i \sqsubseteq \bigsqcup_{n\geq 0} x_n} \qquad\qquad \text{LUB-LEAST} \quad \frac{\forall n \geq 0 \,.\, x_n \sqsubseteq x}{\bigsqcup_{n\geq 0} x_n \sqsubseteq x}$$

Least upper bound of a chain

The **least upper bound** of countable increasing chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$, written $\bigsqcup_{n \geq 0} d_n$, satisfies the two following properties:

$$\textsc{lub-bound}\ \frac{}{x_i \sqsubseteq \bigsqcup_{n \geq 0} x_n} \qquad \textsc{lub-least}\ \frac{\forall n \geq 0 \,.\, x_n \sqsubseteq x}{\bigsqcup_{n \geq 0} x_n \sqsubseteq x}$$

- Other names: supremum, limit...
- Might write simply $\bigsqcup_n d_n$ or even $\bigsqcup d_n$
- Only lubs of chains – but can be generalized
- $\bigsqcup_{i \geq 0} d_i$ need not be one of the d_i – this is the interesting case!

Properties of lubs

Lubs are unique.

Properties of lubs

Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}.\ d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$.

Properties of lubs

Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}.\ d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$.

$$\textsc{lub-mon}\ \frac{\forall i.\ d_i \sqsubseteq e_i}{\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n}$$

Properties of lubs

Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}.\ d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$.

For any d, $\bigsqcup_n d = d$.

Properties of lubs

Lubs are unique.

Lubs are monotone: if for all $n \in \mathbb{N}.\ d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$.

For any d, $\bigsqcup_n d = d$.

For any chain and $N \in \mathbb{N}$, $\bigsqcup_n d_n = \bigsqcup_n d_{n+N}$.

Properties of lubs

Lubs are unique (if they exist).

Lubs are monotone: if for all $n \in \mathbb{N}.\ d_n \sqsubseteq e_n$, then $\bigsqcup_n d_n \sqsubseteq \bigsqcup_n e_n$ (if they exist).

For any d, $\bigsqcup_n d = d$ (and in particular it exists).

For any chain and $N \in \mathbb{N}$, $\bigsqcup_n d_n = \bigsqcup_n d_{n+N}$ (if any of the two exists).

Assume $d_{m,n} \in D$ $(m, n \geq 0)$ satisfies

$$m \leq m' \wedge n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'}.$$

Assume $d_{m,n} \in D$ $(m, n \geq 0)$ satisfies

$$m \leq m' \wedge n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'}. \tag{†}$$

Then, assuming they exist, the lubs form two chains

$$\bigsqcup_{n\geq 0} d_{0,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{1,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{2,n} \sqsubseteq \ldots$$

and

$$\bigsqcup_{m\geq 0} d_{m,0} \sqsubseteq \bigsqcup_{m\geq 0} d_{m,1} \sqsubseteq \bigsqcup_{m\geq 0} d_{m,2} \sqsubseteq \ldots$$

Assume $d_{m,n} \in D$ $(m, n \geq 0)$ satisfies

$$m \leq m' \wedge n \leq n' \Rightarrow d_{m,n} \sqsubseteq d_{m',n'}. \qquad (\dagger)$$

Then, assuming they exist, the lubs form two chains

$$\bigsqcup_{n\geq 0} d_{0,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{1,n} \sqsubseteq \bigsqcup_{n\geq 0} d_{2,n} \sqsubseteq \ldots$$

and

$$\bigsqcup_{m\geq 0} d_{m,0} \sqsubseteq \bigsqcup_{m\geq 0} d_{m,1} \sqsubseteq \bigsqcup_{m\geq 0} d_{m,2} \sqsubseteq \ldots$$

Moreover, again assuming the lubs of these chains exist,

$$\bigsqcup_{m\geq 0}\left(\bigsqcup_{n\geq 0} d_{m,n}\right) = \bigsqcup_{k\geq 0} d_{k,k} = \bigsqcup_{n\geq 0}\left(\bigsqcup_{m\geq 0} d_{m,n}\right) .$$

Least Fixed Points

Complete partial orders and domains

A chain complete poset/cpo is a poset $(D, \sqsubseteq)$ in which all chains have least upper bounds.

Cpos and domains

A chain complete poset/cpo is a poset $(D, \sqsubseteq)$ in which all chains have least upper bounds.

Beware: the lub need only exist if the x_i form a chain!

A chain complete poset/cpo is a poset $(D, \sqsubseteq)$ in which all chains have least upper bounds.

Beware: the lub need only exist if the x_i form a chain!

A domain is a cpo with a least element $\perp$.

Domain of partial functions

Least element: $\perp$ is the totally undefined function.

Least element: $\bot$ is the totally undefined function.

Lub of a chain: $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ has lub f such that

$$f(x) = \begin{cases} f_n(x) & \text{if } x \in \mathbf{dom}(f_n) \text{ for some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

Domain of partial functions

Least element: $\bot$ is the totally undefined function.

Lub of a chain: $f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq \ldots$ has lub f such that

$$f(x) = \begin{cases} f_n(x) & \text{if } x \in \mathbf{dom}(f_n) \text{ for some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

Beware: the definition of $\bigsqcup_{n \geq 0} f_n$ is unambiguous only if the f_i form a chain!

Finite cpos

Finite posets are always cpos – why?

Finite cpos

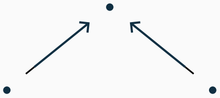
Finite posets are always cpos – why?

Are they always domains?

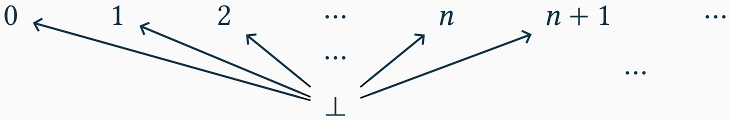
Finite cpos

Finite posets are always cpos – why?

Are they always domains?



The flat natural numbers $\mathbb{N}_\perp$



$$\begin{array}{c} n+1 \\ \uparrow \\ n \\ \vdots \\ 1 \\ \uparrow \\ 0 \end{array}$$

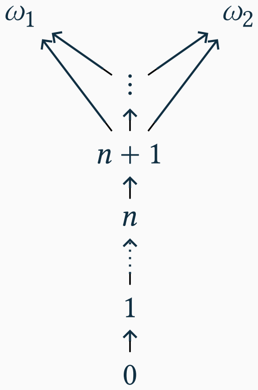
No! (Why?)

Vertical natural numbers

$$\begin{array}{c} \omega \\ \uparrow \\ \vdots \\ n+1 \\ \uparrow \\ n \\ \uparrow \\ \vdots \\ 1 \\ \uparrow \\ 0 \end{array}$$

Yes!

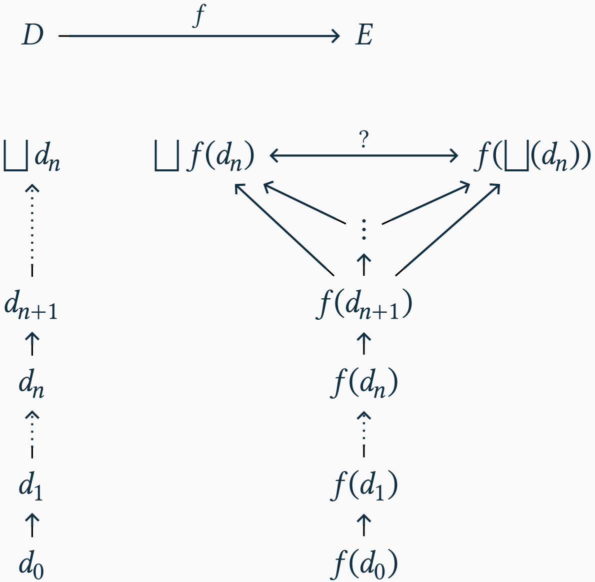
Vertical natural numbers

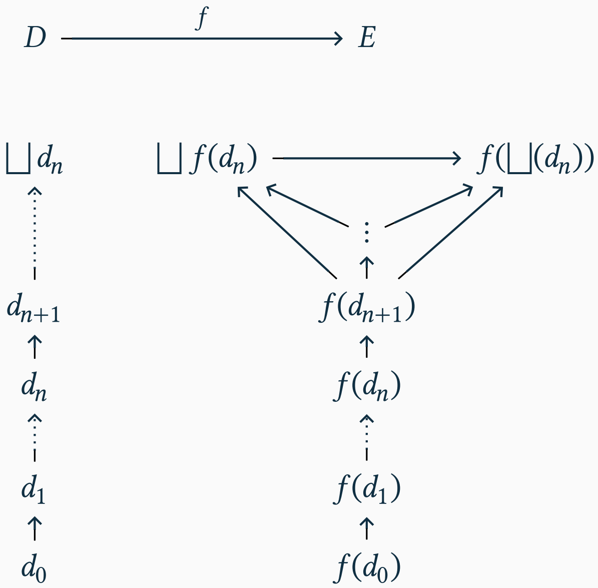


No! (Why?)

Least Fixed Points

Continuous functions


D
f
E
$\bigsqcup d_n$
$\bigsqcup f(d_n)$
?
$f(\bigsqcup(d_n))$
d_{n+1}
$f(d_{n+1})$
d_n
$f(d_n)$
d_1
$f(d_1)$
d_0
$f(d_0)$


D
f
E
⊔ d_n
⊔ f(d_n)
f(⊔(d_n))
d_{n+1}
f(d_{n+1})
d_n
f(d_n)
d_1
f(d_1)
d_0
f(d_0)

Continuity and strictness

Given two cpos D and E, a function $f : D \to E$ is continuous if

- it is monotone, and
- it preserves lubs of chains, *i.e.* for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ in D, we have

$$f\left(\bigsqcup_{n \geq 0} d_n\right) = \bigsqcup_{n \geq 0} f(d_n)$$

Note: one direction is automatic.

Continuity and strictness

Given two cpos D and E, a function $f : D \to E$ is continuous if

- it is monotone, and
- it preserves lubs of chains, *i.e.* for all chains $d_0 \sqsubseteq d_1 \sqsubseteq \dots$ in D, we have

$$f(\bigsqcup_{n \geq 0} d_n) = \bigsqcup_{n \geq 0} f(d_n)$$

Note: one direction is automatic.

A function f is strict if $f(\bot_D) = \bot_E$.

Thesis

All computable functions are continuous.

All computable functions are continuous.

All computable functions are continuous.

Typical non-continuous function: "is a sequence the constant 0"? $(\mathbb{N} \rightharpoonup \mathbb{B}) \rightharpoonup \mathbb{B}$

$$
\begin{array}{llllllll}
0 & 0 & \bot & \ldots & & & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 & \ldots & & \mapsto 1 \\
\\
0 & 0 & 0 & 0 & 0 & \overline{0} & & \mapsto 0
\end{array}
$$

All computable functions are continuous.

Typical non-continuous function: “is a sequence the constant $\mathbf{0}$”? $(\mathbb{N} \rightharpoonup \mathbb{B}) \rightharpoonup \mathbb{B}$

$$
\begin{array}{llllllll}
0 & 0 & \bot & \ldots & & & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 & \ldots & & \mapsto 1 \\
0 & 0 & 0 & 0 & 0 & \ldots & & \mapsto ? \\
 & & & & & & & \\
0 & 0 & 0 & 0 & 0 & \overline{0} & & \mapsto 0
\end{array}
$$

All computable functions are continuous.

Typical non-continuous function: "is a sequence the constant $\mathbf{0}$"? $(\mathbb{N} \rightharpoonup \mathbb{B}) \rightharpoonup \mathbb{B}$

$$
\begin{array}{llllllllll l}
0 & 0 & \bot & \ldots & & & & & & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 & \ldots & & & & & \mapsto 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bot & \ldots & \mapsto \bot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & \mapsto ? \\
0 & 0 & 0 & 0 & 0 & \overline{0} & & & & & \mapsto 0
\end{array}
$$

All computable functions are continuous.

Typical non-continuous function: "is a sequence the constant 0"? $(\mathbb{N} \rightharpoonup \mathbb{B}) \rightharpoonup \mathbb{B}$

$$
\begin{array}{llllllllllr}
0 & 0 & \bot & \dots & & & & & & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 & \dots & & & & & \mapsto 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bot & \dots & \mapsto \bot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \mapsto ? \\
0 & 0 & 0 & 0 & 0 & \overline{0} & & & & & \mapsto 0
\end{array}
$$

Intuition: non-continuity ≈ "jump at infinity" ≈ non-computability

All computable functions are continuous.

Typical non-continuous function: "is a sequence the constant 0"? $(\mathbb{N} \rightharpoonup \mathbb{B}) \rightharpoonup \mathbb{B}$

$$
\begin{array}{llllllllllr}
0 & 0 & \bot & \dots & & & & & & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 & \dots & & & & & \mapsto 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bot & \dots & \mapsto \bot \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \mapsto ? \\
0 & 0 & 0 & 0 & 0 & \overline{0} & & & & & \mapsto 0
\end{array}
$$

Intuition: non-continuity $\approx$ "jump at infinity" $\approx$ non-computability

Later in the course: show the thesis… by giving a denotational semantics.

Least Fixed Points

Kleene's fixed point theorem

Let $f : D \to D$ be a continuous function on a domain D. Then f possesses a least pre-fixed point, given by

$$\mathrm{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot).$$

Kleene's fixed point theorem

Let $f\colon D \to D$ be a continuous function on a domain D. Then f possesses a least pre-fixed point, given by

$$\mathrm{fix}(f) = \bigsqcup_{n \geq 0} f^n(\bot).$$

It is thus also the least fixed point of f!

Constructions on Domains

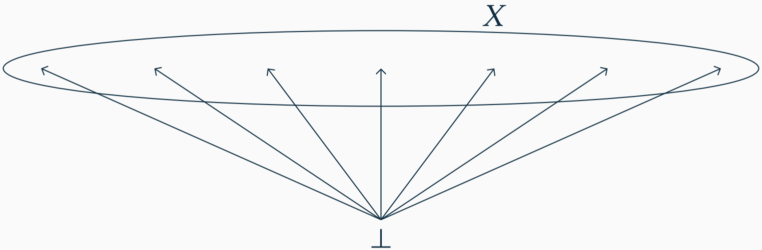
Constructions on Domains

Flat domains

Flat domain on X

The flat domain on a set X is defined by:

- its underlying set $X \biguplus \{\bot\}$;
- $x \sqsubseteq x'$ if either $x = \bot$ or $x = x'$.



Flat domain lifting

Let $f : X \rightharpoonup Y$ be a partial function between two sets. Then

$$
\begin{aligned}
f_\perp : \quad X_\perp \quad &\to \quad Y_\perp \\
d \quad &\mapsto \begin{cases} f(d) & \text{if } d \in X \text{ and } f \text{ is defined at } d \\ \perp & \text{if } d \in X \text{ and } f \text{ is not defined at } d \\ \perp & \text{if } d = \perp \end{cases}
\end{aligned}
$$

defines a strict continuous function between the corresponding flat domains.

Constructions on Domains

Products of domains

Binary product

The product of two posets $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ has underlying set

$$D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \wedge d_2 \in D_2\}$$

and partial order $\sqsubseteq$ defined by

$$(d_1, d_2) \sqsubseteq (d_1', d_2') \stackrel{\text{def}}{\Leftrightarrow} d_1 \sqsubseteq_1 d_1' \wedge d_2 \sqsubseteq_2 d_2'$$

Binary product

The **product** of two posets $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ has underlying set

$$D_1 \times D_2 = \{(d_1, d_2) \mid d_1 \in D_1 \land d_2 \in D_2\}$$

and partial order $\sqsubseteq$ defined by

$$(d_1, d_2) \sqsubseteq (d_1', d_2') \overset{\text{def}}{\Leftrightarrow} d_1 \sqsubseteq_1 d_1' \land d_2 \sqsubseteq_2 d_2'$$

$$\textsc{pox}\ \frac{d_1 \sqsubseteq_1 d_1' \qquad d_2 \sqsubseteq_2 d_2'}{(d_1, d_2) \sqsubseteq (d_1', d_2')}$$

lubs of chains are computed componentwise:

$$\bigsqcup_{n \geq 0}(d_{1,n}, d_{2,n}) = (\bigsqcup_{i \geq 0} d_{1,i}, \bigsqcup_{j \geq 0} d_{2,j}).$$

lubs of chains are computed componentwise:

$$\bigsqcup_{n\geq 0}(d_{1,n}, d_{2,n}) = (\bigsqcup_{i\geq 0} d_{1,i}, \bigsqcup_{j\geq 0} d_{2,j}).$$

If $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ have least elements, so does $(D_1 \times D_2, \sqsubseteq)$ with

$$\bot_{D_1\times D_2} = (\bot_{D_1}, \bot_{D_2})$$

Componentwise lubs and least elements

lubs of chains are computed componentwise:

$$\bigsqcup_{n\geq 0}(d_{1,n}, d_{2,n}) = (\bigsqcup_{i\geq 0} d_{1,i}, \bigsqcup_{j\geq 0} d_{2,j}).$$

If $(D_1, \sqsubseteq_1)$ and $(D_2, \sqsubseteq_2)$ have least elements, so does $(D_1 \times D_2, \sqsubseteq)$ with

$$\bot_{D_1 \times D_2} = (\bot_{D_1}, \bot_{D_2})$$

Products of cpos (domains) are cpos (domains).

Functions of two arguments

A function $f : (D \times E) \to F$ is monotone if and only if it is monotone in each argument separately:

$$\forall d, d' \in D, e \in E.\, d \sqsubseteq d' \Rightarrow f(d, e) \sqsubseteq f(d', e)$$
$$\forall d \in D, e, e' \in E.\, e \sqsubseteq e' \Rightarrow f(d, e) \sqsubseteq f(d, e').$$

Functions of two arguments

A function $f : (D \times E) \to F$ is monotone if and only if it is monotone in each argument separately:

$$\forall d, d' \in D, e \in E.\, d \sqsubseteq d' \Rightarrow f(d, e) \sqsubseteq f(d', e)$$
$$\forall d \in D, e, e' \in E.\, e \sqsubseteq e' \Rightarrow f(d, e) \sqsubseteq f(d, e').$$

Moreover, it is continuous if and only if it preserves lubs in each argument separately:

$$f(\bigsqcup_{m \geq 0} d_m\, ,\, e) = \bigsqcup_{m \geq 0} f(d_m, e)$$
$$f(d\, ,\, \bigsqcup_{n \geq 0} e_n) = \bigsqcup_{n \geq 0} f(d, e_n).$$

Derived rules for functions of two arguments

$$\text{MON}\times \; \frac{f \text{ monotone} \qquad x \sqsubseteq x' \qquad y \sqsubseteq y'}{f(x, y) \sqsubseteq f(x', y')}$$

$$f\left(\bigsqcup_m x_m, \bigsqcup_n y_n\right) = \bigsqcup_m \bigsqcup_n f(x_m, y_n) = \bigsqcup_k f(x_k, y_k)$$

Projection and pairing

Let D_1 and D_2 be cpos. The projections

$$\begin{aligned} \pi_1 : \quad D_1 \times D_2 &\rightarrow D_1 \\ (d_1, d_2) &\mapsto d_1 \end{aligned} \qquad\qquad \begin{aligned} \pi_2 : \quad D_1 \times D_2 &\rightarrow D_2 \\ (d_1, d_2) &\mapsto d_2 \end{aligned}$$

are continuous functions.

Projection and pairing

Let D_1 and D_2 be cpos. The projections

$$\begin{array}{rrcl} \pi_1 : & D_1 \times D_2 & \to & D_1 \\ & (d_1, d_2) & \mapsto & d_1 \end{array} \qquad \begin{array}{rrcl} \pi_2 : & D_1 \times D_2 & \to & D_2 \\ & (d_1, d_2) & \mapsto & d_2 \end{array}$$

are continuous functions.

If $f_1 : D \to D_1$ and $f_2 : D \to D_2$ are continuous functions from a cpo D, then the pairing function

$$\begin{array}{rrcl} \langle f_1, f_2 \rangle : & D & \to & D_1 \times D_2 \\ & d & \mapsto & (f_1(d), f_2(d)) \end{array}$$

is continuous.

For any domain D, the conditional function

$$\begin{aligned} \mathrm{if} : \quad \mathbb{B}_\perp \times (D \times D) \quad &\to \quad D \\ (x, d) \qquad\qquad &\mapsto \quad \begin{cases} \pi_1(d) & \text{if } x = \mathrm{true} \\ \pi_2(d) & \text{if } x = \mathrm{false} \\ \perp_D & \text{if } x = \perp \end{cases} \end{aligned}$$

is continuous.

General product

Given a set I, suppose that for each $i \in I$ we are given a set X_i. The (cartesian) product of the X_i is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\dots, x_i, \dots)_{i \in I}$ such that $x_i \in X_i$;

General product

Given a set I, suppose that for each $i \in I$ we are given a set X_i. The (cartesian) product of the X_i is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\ldots, x_i, \ldots)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: p defined on I such that $p(i) \in X_i$.

General product

Given a set I, suppose that for each $i \in I$ we are given a set X_i. The (cartesian) product of the X_i is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\dots, x_i, \dots)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: p defined on I such that $p(i) \in X_i$.

Special case: $\prod_{i \in \mathbb{B}} D_i$ corresponds to $D_{\mathrm{true}} \times D_{\mathrm{false}}$.

Given a set I, suppose that for each $i \in I$ we are given a set X_i. The (cartesian) product of the X_i is

$$\prod_{i \in I} X_i$$

Two ways to see it:

- tuples: $(\dots, x_i, \dots)_{i \in I}$ such that $x_i \in X_i$;
- heterogeneous functions: p defined on I such that $p(i) \in X_i$.

Special case: $\prod_{i \in \mathbb{B}} D_i$ corresponds to $D_{\mathrm{true}} \times D_{\mathrm{false}}$.

Projections (for any $i \in I$):

$$\pi_i : \left(\prod_{i \in I} X_i\right) \to X_i$$

General product of domains

Given a set I, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The product of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;

Given a set I, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The product of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;
- componentwise order

$$p \sqsubseteq p' \overset{\text{def}}{\Leftrightarrow} \forall i \in I.\, p_i \sqsubseteq_i p'_i.$$

General product of domains

Given a set I, suppose that for each $i \in I$ we are given a cpo $(D_i, \sqsubseteq_i)$. The product of this whole family of cpos has

- underlying set equal to $\prod_{i \in I} D_i$;
- componentwise order

$$p \sqsubseteq p' \overset{\text{def}}{\Leftrightarrow} \forall i \in I.\ p_i \sqsubseteq_i p'_i.$$

I-indexed products of cpos (domains) are cpos (domains), and projections are continuous.

Constructions on Domains

Function domains

Given two cpos $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$, the **function cpo** $(D \to E, \sqsubseteq)$ has underlying set

$$\{f : D \to E \mid \text{ is a } \textit{continuous} \text{ function}\}$$

equipped with the pointwise order:

$$f \sqsubseteq f' \overset{\text{def}}{\Leftrightarrow} \forall d \in D.\, f(d) \sqsubseteq_E f'(d).$$

Cpo/domain of continuous functions

Given two cpos $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$, the function cpo $(D \to E, \sqsubseteq)$ has underlying set

$$\{f : D \to E \mid \text{ is a } \textit{continuous} \text{ function}\}$$

equipped with the pointwise order:

$$f \sqsubseteq f' \overset{\text{def}}{\Leftrightarrow} \forall d \in D.\ f(d) \sqsubseteq_E f'(d).$$

$$\frac{f \sqsubseteq_{D \to E} g \qquad x \sqsubseteq_D y}{f(x) \sqsubseteq_E g(y)}$$

Cpo/domain of continuous functions

Given two cpos $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$, the function cpo $(D \to E, \sqsubseteq)$ has underlying set

$$\{f : D \to E \mid \text{ is a } \textit{continuous} \text{ function}\}$$

equipped with the pointwise order:

$$f \sqsubseteq f' \overset{\text{def}}{\Leftrightarrow} \forall d \in D.\ f(d) \sqsubseteq_E f'(d).$$

Argumentwise least elements and lubs:

$$\bot_{D \to E}(d) = \bot_E \qquad \left(\bigsqcup_{n \geq 0} f_n\right)(d) = \bigsqcup_{n \geq 0} f_n(d)$$

Evaluation, currying ($f : (D' \times D) \to E$) and composition

$$\begin{aligned} \text{eval} : \quad (D \to E) \times D &\to E \\ (f, d) &\mapsto f(d) \end{aligned}$$

$$\begin{aligned} \text{cur}(f) : \quad D' &\to (D \to E) \\ d' &\mapsto \lambda d \in D.\ f(d', d) \end{aligned}$$

$$\begin{aligned} \circ : \quad \big((E \to F) \times (D \to E)\big) &\longrightarrow (D \to F) \\ (f, g) &\mapsto \lambda d \in D.\ g(f(d)) \end{aligned}$$

are all well-defined and continuous.

Continuity of the fixed point operator

$$\text{fix:} \quad (D \to D) \quad \to \quad D$$

is continuous.

Constructions on Domains

Back to the introduction

$$[\![\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)]\!]$$

is a fixed point of the following $F : D \to D$, where D is $(\mathbf{State} \rightharpoonup \mathbf{State})$:

$$F(w)([X \mapsto x, Y \mapsto y]) \quad = \quad \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0. \end{cases}$$

The semantics of a while loop

$$[\![\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)]\!]$$

is a fixed point of the following $F : D \to D$, where D is $(\text{State}_\bot \to \text{State}_\bot)$:

$$F(w)([X \mapsto x, Y \mapsto y]) = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ w([X \mapsto x - 1, Y \mapsto x \cdot y]) & \text{if } x > 0. \end{cases}$$

$$F(\bot) = \bot$$

$\text{State}_\bot \to \text{State}_\bot$ is a domain!

Kleene's fixed point theorem

Kleene's fixed point theorem:

$$w_\infty = \bigsqcup_{i \in \mathbb{N}} F^n(\bot)$$

is the least fixed point of F, and in particular a fixed point.

Kleene's fixed point theorem:

$$w_\infty = \bigsqcup_{i \in \mathbb{N}} F^n(\bot)$$

is the least fixed point of F, and in particular a fixed point.

We can compute explicitly

$$w_\infty[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x < 0 \\ [X \mapsto 0, Y \mapsto (x!) \cdot y] & \text{if } x \geq 0 \end{cases}$$

And check this agrees with the operational semantics.

Scott Induction

Reasoning on fixed points: Scott induction

Let D be a domain, $f\colon D \to D$ be a continuous function and $S \subseteq D$ be a subset of D. If the set S

(i) contains $\bot$,

(ii) is chain-closed, *i.e.* the lub of any chain of elements of S is also in S,

(iii) is stable for f, *i.e.* $f(S) \subseteq S$,

then $\mathrm{fix}(f) \in S$.

Reasoning on fixed points: Scott induction

Let D be a domain, $f : D \to D$ be a continuous function and $S \subseteq D$ be a subset of D. If the set S

(i) contains $\bot$,

(ii) is chain-closed, *i.e.* the lub of any chain of elements of S is also in S,

(iii) is stable for f, *i.e.* $f(S) \subseteq S$,

then $\mathrm{fix}(f) \in S$.

$$\textsc{ScottInd}\ \frac{\Phi(\bot) \qquad \Phi(x) \Rightarrow \Phi(f(x)) \qquad (\forall i \in \mathbb{N}.\ \Phi(x_i)) \Rightarrow \Phi(\bigsqcup_{i \in \mathbb{N}} x_i)}{\Phi(\mathrm{fix}(f))}$$

Building chain-closed sets

All the following are chain-closed:

All the following are chain-closed:

$$\{(x, y) \in D{\times}D \mid x \sqsubseteq y\}\ , \qquad d \downarrow \stackrel{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\} \qquad \text{and} \qquad \{(x, y) \in D{\times}D \mid x = y\}$$

Building chain-closed sets

All the following are chain-closed:

$$\{(x,y) \in D{\times}D \mid x \sqsubseteq y\}\ ,\qquad d\downarrow \stackrel{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\}\qquad \text{and}\qquad \{(x,y) \in D{\times}D \mid x = y\}$$

$$f^{-1}S = \{x \in D \mid f(x) \in S\}\qquad \text{if } S \subseteq E \text{ is chain-closed, and } f : D \to E \text{ is continuous}$$

All the following are chain-closed:

$\{(x,y) \in D{\times}D \mid x \sqsubseteq y\}$, $\quad d\downarrow \stackrel{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\}$ and $\{(x,y) \in D{\times}D \mid x = y\}$

$f^{-1}S = \{x \in D \mid f(x) \in S\}$ if $S \subseteq E$ is chain-closed, and $f\colon D \to E$ is continuous

$S \cup T$ and $\bigcap_{i \in I} S_i$ if S, T and S_i are

All the following are chain-closed:

$$\{(x,y) \in D \times D \mid x \sqsubseteq y\}\ , \qquad d\downarrow \stackrel{\text{def}}{=} \{x \in D \mid x \sqsubseteq d\} \quad \text{and} \quad \{(x,y) \in D \times D \mid x = y\}$$

$f^{-1}S = \{x \in D \mid f(x) \in S\}$ if $S \subseteq E$ is chain-closed, and $f : D \to E$ is continuous

$S \cup T$ and $\bigcap_{i \in I} S_i$ if S, T and S_i are

$\forall S \stackrel{\text{def}}{=} \{y \in E \mid \forall x \in D.\, (x,y) \in S\} \subseteq E$ if $S \subseteq D \times E$ is

The "logical" view

Any formula written using:

- signature: continuous functions + constants
- relations: equality, inequality
- logical connectives: conjuction, disjunction, universal quantification

is chain-closed.

The "logical" view

Any formula written using:

- signature: continuous functions + constants
- relations: equality, inequality
- logical connectives: conjuction, disjunction, universal quantification

is chain-closed.

Given any set I, domains D, E, functions $(f_i)_{i \in I}, g \colon D \to E$, $e \in E$,

$$\Phi(x) := \forall y \in E, (\forall i \in I,\ f_i(x) \sqsubseteq y) \vee g(x) = e$$

is chain-closed.

Example: downset

Assume $f(d) \sqsubseteq d$, *i.e.* d is a pre-fixed point of the continuous $f : D \to D$. By Scott induction on $d\downarrow$, $\mathrm{fix}(f) \sqsubseteq d$.

Example: downset

Assume $f(d) \sqsubseteq d$, *i.e.* d is a pre-fixed point of the continuous $f : D \to D$. By Scott induction on $d \downarrow$, $\mathrm{fix}(f) \sqsubseteq d$.

Proof!

Example: partial correctness

Let $w_\infty \colon \mathrm{State}_\perp \to \mathrm{State}_\perp$ be the denotation of

$$\texttt{while } X > 0 \texttt{ do } (Y := X * Y; X := X - 1)$$

Recall that $w_\infty = \mathrm{fix}(F)$ where

$$F(w)(x,y) = \begin{cases} (x,y) & \text{if } x \leq 0 \\ w(x-1, x \cdot y) & \text{if } x > 0 \end{cases}$$

$$F(w)(\perp) = \perp$$

Example: partial correctness

Let $w_\infty \colon \mathrm{State}_\perp \to \mathrm{State}_\perp$ be the denotation of

$$\texttt{while } X > 0 \texttt{ do } (Y := X * Y; X := X - 1)$$

Recall that $w_\infty = \mathrm{fix}(F)$ where

$$F(w)(x,y) = \begin{cases} (x,y) & \text{if } x \leq 0 \\ w(x-1, x \cdot y) & \text{if } x > 0 \end{cases}$$
$$F(w)(\perp) = \perp$$

Claim:

$$\forall x.\, \forall y \geq 0.\, w_\infty(x,y) \Downarrow \implies \pi_Y(w_\infty(x,y)) \geq 0$$

Example: partial correctness

Let $w_\infty \colon \mathrm{State}_\perp \to \mathrm{State}_\perp$ be the denotation of

$$\texttt{while}\ X > 0\ \texttt{do}\ (Y := X * Y; X := X - 1)$$

Recall that $w_\infty = \mathrm{fix}(F)$ where

$$F(w)(x, y) = \begin{cases} (x, y) & \text{if } x \leq 0 \\ w(x - 1, x \cdot y) & \text{if } x > 0 \end{cases}$$
$$F(w)(\perp) = \perp$$

Claim:

$$\forall x.\ \forall y \geq 0.\ w_\infty(x, y) \Downarrow \implies \pi_Y\left(w_\infty(x, y)\right) \geq 0$$

Proof: by Scott induction!

PCF

Pcf

Syntax

Syntax of Pcf

Types: $$\tau ::= \texttt{nat} \mid \texttt{bool} \mid \tau \to \tau$$

Syntax of Pcf

Types:

$$\tau ::= \texttt{nat} \mid \texttt{bool} \mid \tau \to \tau$$

Terms:

$$\begin{aligned} t \quad ::= \quad & \texttt{0} \mid \texttt{succ}(t) \mid \texttt{pred}(t) \mid \\ & \texttt{true} \mid \texttt{false} \mid \texttt{zero?}(t) \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \\ & x \mid \texttt{fun}\ x{:}\tau.\ t \mid t\ t \mid \texttt{fix}(t) \end{aligned}$$

Syntax of Pcf

Types: $\tau ::= \texttt{nat} \mid \texttt{bool} \mid \tau \to \tau$

Terms:

$$
\begin{array}{rcl}
t & ::= & \texttt{0} \mid \texttt{succ}(t) \mid \texttt{pred}(t) \mid \\
 & & \texttt{true} \mid \texttt{false} \mid \texttt{zero?}(t) \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \\
 & & x \mid \texttt{fun}\ x{:}\tau.\ t \mid t\ t \mid \texttt{fix}(t)
\end{array}
$$

- λ-calculus + base types/functions + `fix`
- tiny ML (without references, ADTs, polymorphism...)

Variables: up to α-equivalence

Variables, substitutions and contexts

Variables: up to α-equivalence

Substitution: $t[u/x]$

Variables, substitutions and contexts

Variables: up to α-equivalence

Substitution: $t[u/x]$

Contexts: $\cdot$ and $\Gamma, x\colon \tau$

Variables: up to α-equivalence

Substitution: $t[u/x]$

Contexts: $\cdot$ and $\Gamma, x\colon\tau$

- partial maps from variable to types
- finite lists $x_1\colon\tau_1, \ldots, x_n\colon\tau_n$

Typing for Pcf (I)

$\boxed{\Gamma \vdash t : \tau}$ The term t has type τ in context Γ

$$\textsc{Zero}\ \frac{}{\Gamma \vdash \mathtt{0} : \mathtt{nat}} \qquad \textsc{Succ}\ \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(t) : \mathtt{nat}} \qquad \textsc{Pred}\ \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{pred}(t) : \mathtt{nat}}$$

Typing for Pcf (I)

$\boxed{\Gamma \vdash t : \tau}$ The term t has type τ in context Γ

$$\textsc{Zero}\ \frac{}{\Gamma \vdash \mathtt{0} : \mathtt{nat}} \qquad \textsc{Succ}\ \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{succ}(t) : \mathtt{nat}} \qquad \textsc{Pred}\ \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{pred}(t) : \mathtt{nat}}$$

$$\textsc{True}\ \frac{}{\Gamma \vdash \mathtt{true} : \mathtt{bool}} \qquad \textsc{False}\ \frac{}{\Gamma \vdash \mathtt{false} : \mathtt{bool}} \qquad \textsc{IsZ}\ \frac{\Gamma \vdash t : \mathtt{nat}}{\Gamma \vdash \mathtt{zero?}(t) : \mathtt{bool}}$$

$$\textsc{If}\ \frac{\begin{array}{c}\Gamma \vdash b : \mathtt{bool} \\ \Gamma \vdash t : \tau \qquad \Gamma \vdash t' : \tau\end{array}}{\Gamma \vdash \mathtt{if}\ b\ \mathtt{then}\ t\ \mathtt{else}\ t' : \tau}$$

Typing for Pcf (II)

$$\textsc{Var}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \textsc{Fun}\ \frac{\Gamma, x{:}\sigma \vdash t : \tau}{\Gamma \vdash \mathtt{fun}\, x{:}\sigma.\, t : \sigma \to \tau} \qquad \textsc{App}\ \frac{\Gamma \vdash f : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash f\ u : \tau}$$

$$\textsc{Fix}\ \frac{\Gamma \vdash f : \tau \to \tau}{\Gamma \vdash \mathtt{fix}(f) : \tau}$$

Typing for Pcf (II)

$$\textsc{Var}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \textsc{Fun}\ \frac{\Gamma, x{:}\sigma \vdash t : \tau}{\Gamma \vdash \mathtt{fun}\, x{:}\sigma.\, t : \sigma \to \tau} \qquad \textsc{App}\ \frac{\Gamma \vdash f : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash f\, u : \tau}$$

$$\textsc{Fix}\ \frac{\Gamma \vdash f : \tau \to \tau}{\Gamma \vdash \mathtt{fix}(f) : \tau}$$

$$\textsc{Pcf}_{\Gamma,\tau} \stackrel{\text{def}}{=} \{t \mid \Gamma \vdash t : \tau\} \qquad\qquad \textsc{Pcf}_{\tau} \stackrel{\text{def}}{=} \textsc{Pcf}_{\cdot,\tau}$$

Typing for Pcf (II)

$$\text{Var}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \text{Fun}\ \frac{\Gamma, x{:}\sigma \vdash t : \tau}{\Gamma \vdash \mathtt{fun}\, x{:}\sigma.\, t : \sigma \to \tau} \qquad \text{App}\ \frac{\Gamma \vdash f : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash f\, u : \tau}$$

$$\text{Fix}\ \frac{\Gamma \vdash f : \tau \to \tau}{\Gamma \vdash \mathtt{fix}(f) : \tau}$$

$$\text{Pcf}_{\Gamma,\tau} \stackrel{\text{def}}{=} \{t \mid \Gamma \vdash t : \tau\} \qquad \text{Pcf}_{\tau} \stackrel{\text{def}}{=} \text{Pcf}_{\cdot,\tau}$$

The **only** programs we care about!

Typing and substitution

If $\Gamma \vdash t : \tau$ and $\Gamma, x{:}\tau \vdash t' : \tau'$ both hold, then so does $\Gamma \vdash t'[t/x] : \tau'$.

Pcf

Operational semantics

Pcf values

Values:

$$v ::= \underbrace{\mathtt{0} \mid \mathtt{succ}(v)}_{\underline{n}} \mid \mathtt{true} \mid \mathtt{false} \mid \underbrace{\mathtt{fun}\, x{:}\tau.\, t}_{\text{All functions } (< \mathtt{fun} >)}$$

PCF VALUES

Values: $v ::= \underbrace{\mathtt{0} \mid \mathtt{succ}(v)}_{\underline{n}} \mid \mathtt{true} \mid \mathtt{false} \mid \underbrace{\mathtt{fun}\, x{:}\tau.\, t}_{\text{All functions } (<\, \mathtt{fun}\, >)}$

We will only evaluate closed term to values.

PCF evaluation

$$\textsc{Val}\ \frac{\vdash v : \tau}{v \Downarrow_\tau v}$$

PCF evaluation

$$\textsc{Val}\ \frac{\vdash v : \tau}{v \Downarrow_\tau v} \qquad \textsc{Succ}\ \frac{t \Downarrow_{\mathtt{nat}} v}{\mathtt{succ}(t) \Downarrow_{\mathtt{nat}} \mathtt{succ}(v)} \qquad \textsc{Pred}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{succ}(v)}{\mathtt{pred}(t) \Downarrow_{\mathtt{nat}} v}$$

PCF evaluation

$$\textsc{Val}\ \frac{\vdash v : \tau}{v \Downarrow_{\tau} v} \qquad \textsc{Succ}\ \frac{t \Downarrow_{\mathtt{nat}} v}{\mathtt{succ}(t) \Downarrow_{\mathtt{nat}} \mathtt{succ}(v)} \qquad \textsc{Pred}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{succ}(v)}{\mathtt{pred}(t) \Downarrow_{\mathtt{nat}} v}$$

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{0}}{\mathtt{zero?}(t) \Downarrow_{\mathtt{bool}} \mathtt{true}} \qquad \textsc{ZeroS}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{succ}(v)}{\mathtt{zero?}(t) \Downarrow_{\mathtt{bool}} \mathtt{false}}$$

PCF EVALUATION

$$\textsc{Val}\ \frac{\vdash v : \tau}{v \Downarrow_\tau v} \qquad \textsc{Succ}\ \frac{t \Downarrow_{\texttt{nat}} v}{\texttt{succ}(t) \Downarrow_{\texttt{nat}} \texttt{succ}(v)} \qquad \textsc{Pred}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{succ}(v)}{\texttt{pred}(t) \Downarrow_{\texttt{nat}} v}$$

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{0}}{\texttt{zero?}(t) \Downarrow_{\texttt{bool}} \texttt{true}} \qquad \textsc{ZeroS}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{succ}(v)}{\texttt{zero?}(t) \Downarrow_{\texttt{bool}} \texttt{false}}$$

$$\textsc{IfT}\ \frac{b \Downarrow_{\texttt{bool}} \texttt{true} \qquad t_1 \Downarrow_\tau v}{\texttt{if}\ b\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_\tau v} \qquad \textsc{IfF}\ \frac{b \Downarrow_{\texttt{bool}} \texttt{false} \qquad t_2 \Downarrow_\tau v}{\texttt{if}\ b\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_\tau v}$$

PCF evaluation

$$\textsc{Val}\ \frac{\vdash v : \tau}{v \Downarrow_\tau v} \qquad \textsc{Succ}\ \frac{t \Downarrow_{\texttt{nat}} v}{\texttt{succ}(t) \Downarrow_{\texttt{nat}} \texttt{succ}(v)} \qquad \textsc{Pred}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{succ}(v)}{\texttt{pred}(t) \Downarrow_{\texttt{nat}} v}$$

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{0}}{\texttt{zero?}(t) \Downarrow_{\texttt{bool}} \texttt{true}} \qquad \textsc{ZeroS}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{succ}(v)}{\texttt{zero?}(t) \Downarrow_{\texttt{bool}} \texttt{false}}$$

$$\textsc{IfT}\ \frac{b \Downarrow_{\texttt{bool}} \texttt{true} \qquad t_1 \Downarrow_\tau v}{\texttt{if}\ b\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_\tau v} \qquad \textsc{IfF}\ \frac{b \Downarrow_{\texttt{bool}} \texttt{false} \qquad t_2 \Downarrow_\tau v}{\texttt{if}\ b\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_\tau v}$$

$$\textsc{Fun}\ \frac{t \Downarrow_{\sigma\to\tau} \texttt{fun}\ x{:}\sigma.\, t' \qquad t'[u/x] \Downarrow_\tau v}{t\ u \Downarrow_\tau v} \qquad \textsc{Fix}\ \frac{t\ (\texttt{fix}(t)) \Downarrow_\tau v}{\texttt{fix}(t) \Downarrow_\tau v}$$

$$\mathtt{plus} \stackrel{\text{def}}{=} \mathtt{fun}\, x\colon \mathtt{nat}.\ \mathtt{fix}(\mathtt{fun}(p\colon \mathtt{nat} \to \mathtt{nat})(y\colon \mathtt{nat}).\ \mathtt{if}\ \mathtt{zero?}(y)\ \mathtt{then}\ x\ \mathtt{else}\ \mathtt{succ}(p\ \mathtt{pred}(y)))$$

$$\mathtt{plus}\ \underline{3}\ \underline{1} \Downarrow_{\mathtt{nat}} \underline{4}$$

Evaluation (I)

$$\textsc{Fun}\ \frac{\texttt{plus} \Downarrow \texttt{plus} \qquad \texttt{plus}_{\underline{3}}\ \underline{1} \Downarrow \underline{4}}{\texttt{plus}\ \underline{3}\ \underline{1} \Downarrow_{\texttt{nat}} \underline{4}}$$

$$\begin{aligned}\texttt{plus}_x \quad &\stackrel{\text{def}}{=} \texttt{fix}(\texttt{fun}(p\colon \texttt{nat} \to \texttt{nat})(y\colon \texttt{nat}).\\ &\quad \texttt{if zero?}(y)\ \texttt{then}\ x\ \texttt{else succ}(p\ \texttt{pred}(y)))\end{aligned}$$

Evaluation (I)

$$\textsc{Fun}\ \frac{\texttt{plus} \Downarrow \texttt{plus} \qquad \texttt{plus}_{\underline{3}}\ \underline{1} \Downarrow \underline{4}}{\texttt{plus}\ \underline{3}\ \underline{1} \Downarrow_{\texttt{nat}} \underline{4}}$$

$$\begin{aligned}\texttt{plus}_x \ \stackrel{\text{def}}{=}\ & \texttt{fix}(\texttt{fun}(p\colon \texttt{nat} \to \texttt{nat})(y\colon \texttt{nat}).\\ & \texttt{if zero?}(y)\ \texttt{then}\ x\ \texttt{else succ}(p\ \texttt{pred}(y)))\end{aligned}$$

$$\textsc{Fix}\ \dfrac{\textsc{Fun}\ \dfrac{\textsc{Val}\ \dfrac{}{(\texttt{fun}\ p\colon \texttt{nat} \to \texttt{nat}.\ \ldots) \Downarrow \ldots} \qquad \textsc{Val}\ \dfrac{}{(\texttt{fun}\ y\colon \texttt{nat}.\ldots)[p/\texttt{plus}_x] \Downarrow r_x}}{(\texttt{fun}(p\colon \texttt{nat} \to \texttt{nat})(y\colon \texttt{nat}).\ \ldots)\ \texttt{plus}_x \Downarrow r_x}}{\texttt{plus}_x \Downarrow \underbrace{\texttt{fun}\ y\colon \texttt{nat}.\ \texttt{if zero?}(y)\ \texttt{then}\ x\ \texttt{else succ}(\texttt{plus}_x\ \texttt{pred}(y))}_{r_x}}$$

$$
\textsc{Fun}\;\dfrac{
\mathtt{plus}_{\underline{3}} \Downarrow \underline{r_3}
\qquad
\textsc{IfF}\;\dfrac{
\textsc{ZeroS}\;\dfrac{\textsc{Val}\;\dfrac{}{\underline{1} \Downarrow \underline{1}}}{\mathtt{zero?}(\underline{1}) \Downarrow \mathtt{false}}
\qquad
\textsc{Succ}\;\dfrac{\dfrac{\dfrac{\textsc{ZeroZ}\;\dfrac{\textsc{Pred}\;\dfrac{\cdots}{\mathtt{pred}(\underline{1}) \Downarrow \mathtt{0}}}{\mathtt{zero?}(\mathtt{pred}(\underline{1})) \Downarrow \mathtt{true}}}{\cdots}}{\mathtt{plus}_{\underline{3}}\ \mathtt{pred}(\underline{1}) \Downarrow \underline{3}}}{\mathtt{succ}(\mathtt{plus}_{\underline{3}}\ \mathtt{pred}(\underline{1})) \Downarrow \underline{4}}
}{
\mathtt{if\ zero?}(\underline{1})\ \mathtt{then}\ \underline{3}\ \mathtt{else\ succ}(\mathtt{plus}_{\underline{3}}\ \mathtt{pred}(\underline{1})) \Downarrow \underline{4}
}
}{
\mathtt{plus}_{\underline{3}}\ \underline{1} \Downarrow_{\mathtt{nat}} \underline{4}
}
$$

Divergence

Divergence ($t \Uparrow_\tau$):

$$t : \tau \quad \wedge \quad \nexists v.\, t \Downarrow_\tau v$$

Divergence ($t \Uparrow_\tau$):

$$t : \tau \quad \wedge \quad \nexists v.\, t \Downarrow_\tau v$$

$$\Omega_\tau \stackrel{\text{def}}{=} \texttt{fix}(\texttt{fun}\, x{:}\tau.\, x)$$

$$\Omega_\tau \Uparrow_\tau \qquad \text{(diverges)}$$

Divergence

Divergence ($t \Uparrow_\tau$):

$$t : \tau \quad \wedge \quad \nexists v.\, t \Downarrow_\tau v$$

$$\Omega_\tau \stackrel{\text{def}}{=} \mathtt{fix}(\mathtt{fun}\, x{:}\, \tau.\, x)$$

$$\Omega_\tau \Uparrow_\tau \qquad \text{(diverges)}$$

$$\dfrac{\dfrac{\mathtt{fun}\, x{:}\, \tau.\, x \Downarrow \mathtt{fun}\, x{:}\, \tau.\, x \qquad \overset{\mathcal{P}}{\mathtt{fix}(\mathtt{fun}\, x{:}\, \tau.\, x) \Downarrow v}}{(\mathtt{fun}\, x{:}\, \tau.\, x)\,(\mathtt{fix}(\mathtt{fun}\, x{:}\, \tau.\, x)) \Downarrow v}}{\mathtt{fix}(\mathtt{fun}\, x{:}\, \tau.\, x) \Downarrow v}$$

$$\textsc{Fun-CBN}\ \frac{t \Downarrow_{\sigma\to\tau} \mathsf{fun}\, x{:}\,\sigma.\, t' \qquad t'[u/x] \Downarrow_\tau v}{t\ u \Downarrow_\tau v}$$

$$\textsc{Fun-CBV}\ \frac{t \Downarrow_{\sigma\to\tau} \mathsf{fun}\, x{:}\,\sigma.\, t' \qquad u \Downarrow_\sigma v' \qquad t'[v'/x] \Downarrow_\tau v}{t\ u \Downarrow_\tau v}$$

Call-by-name and call-by-value

$$\textsc{Fun-CBN}\ \frac{t \Downarrow_{\sigma\to\tau} \mathtt{fun}\, x{:}\, \sigma.\, t' \qquad t'[u/x] \Downarrow_\tau v}{t\ u \Downarrow_\tau v}$$

$$\textsc{Fun-CBV}\ \frac{t \Downarrow_{\sigma\to\tau} \mathtt{fun}\, x{:}\, \sigma.\, t' \qquad u \Downarrow_\sigma v' \qquad t'[v'/x] \Downarrow_\tau v}{t\ u \Downarrow_\tau v}$$

What does $(\mathtt{fun}\, x{:}\, \mathtt{nat}.\, \mathtt{0})\ \Omega_{\mathtt{nat}}$ denote?

$$\textsc{Fun-CBN}\ \frac{t \Downarrow_{\sigma\to\tau} \mathtt{fun}\, x{:}\,\sigma.\, t' \qquad t'[u/x] \Downarrow_\tau v}{t\; u \Downarrow_\tau v}$$

$$\textsc{Fun-CBV}\ \frac{t \Downarrow_{\sigma\to\tau} \mathtt{fun}\, x{:}\,\sigma.\, t' \qquad u \Downarrow_\sigma v' \qquad t'[v'/x] \Downarrow_\tau v}{t\; u \Downarrow_\tau v}$$

What does $(\mathtt{fun}\, x{:}\, \mathtt{nat}.\, \mathtt{0})\ \Omega_{\mathtt{nat}}$ denote?

In call-by-value, all functions are **strict**... but the least-fixed points of a strict function is **always $\bot$**!

Small-step $t \rightsquigarrow_\tau u$:

$$\frac{}{(\mathsf{fun}\, x\colon \sigma.\, t)\; u \rightsquigarrow_\tau t[u/x]} \qquad \frac{t \rightsquigarrow_{\sigma \to \tau} t'}{t\; u \rightsquigarrow_\tau t'\; u} \qquad \ldots$$

Small-step $t \rightsquigarrow_\tau u$:

$$\frac{}{(\mathsf{fun}\, x\colon \sigma.\, t)\; u \rightsquigarrow_\tau t[u/x]} \qquad \frac{t \rightsquigarrow_{\sigma \to \tau} t'}{t\; u \rightsquigarrow_\tau t'\; u} \qquad \dots$$

We have $t \Downarrow_\tau v$ iff $t \rightsquigarrow_\tau^\star u$.

PCF is **Turing-complete**: for every partial recursive function ϕ, there is a PCF term $\underline{\phi} \in \mathrm{PCF}_{\mathtt{nat} \to \mathtt{nat}}$ such that for all $n \in \mathbb{N}$, if $\phi(n)$ is defined then $\underline{\phi}\ \underline{n} \Downarrow_{\mathtt{nat}} \underline{\phi(n)}$.

Turing-completeness

PCF is Turing-complete: for every partial recursive function ϕ, there is a PCF term $\underline{\phi} \in \text{PCF}_{\mathtt{nat} \to \mathtt{nat}}$ such that for all $n \in \mathbb{N}$, if $\phi(n)$ is defined then $\underline{\phi}\ \underline{n} \Downarrow_{\mathtt{nat}} \underline{\phi(n)}$.

(Later on: $\phi = \llbracket \underline{\phi} \rrbracket$).

Determinism

Evaluation in PCF is deterministic: if both $t \Downarrow_\tau v$ and $t \Downarrow_\tau v'$ hold, then $v = v'$.

Determinism

Evaluation in PCF is **deterministic**: if both $t \Downarrow_\tau v$ and $t \Downarrow_\tau v'$ hold, then $v = v'$.

By (rule) induction on evaluation $\Downarrow$:

$$P(t, \tau, v) \quad \stackrel{\text{def}}{=} \quad \forall v' \in \mathrm{PCF}_\tau \,.(t \Downarrow_\tau v' \Rightarrow v = v')$$

Intuition: there is always exactly one rule which applies.

Pcf

Contextual equivalence

Contextual equivalence – informal

Two phrases of a programming language are contextually equivalent if any occurrences of the first phrase in a complete program can be replaced by the second phrase without affecting the observable results of executing the program.

Contextual equivalence – informal

Two phrases of a programming language are **contextually equivalent** if any occurrences of the first phrase in a **complete program** can be replaced by the second phrase without affecting the **observable results** of executing the program.

The intuitive notion of **program equivalence** for programmers.

Contextual equivalence – informal

Two phrases of a programming language are **contextually equivalent** if any occurrences of the first phrase in a **complete program** can be replaced by the second phrase without affecting the **observable results** of executing the program.

The intuitive notion of **program equivalence** for programmers.

But what's a complete program? What's an observable result?

Evaluation contexts

"Term with a hole":

$$\mathcal{C} ::= - \mid \texttt{succ}(\mathcal{C}) \mid \texttt{pred}(\mathcal{C}) \mid \texttt{zero?}(\mathcal{C}) \mid$$
$$\texttt{if}\ \mathcal{C}\ \texttt{then}\ t\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ \mathcal{C}\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ \mathcal{C} \mid$$
$$\texttt{fun}\ x{:}\tau.\ \mathcal{C} \mid \mathcal{C}\ t \mid t\ \mathcal{C} \mid \texttt{fix}(\mathcal{C})$$

Evaluation contexts

"Term with a hole":

$$\begin{array}{rcl} \mathcal{C} & ::= & - \mid \texttt{succ}(\mathcal{C}) \mid \texttt{pred}(\mathcal{C}) \mid \texttt{zero?}(\mathcal{C}) \mid \\ & & \texttt{if}\ \mathcal{C}\ \texttt{then}\ t\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ \mathcal{C}\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ \mathcal{C} \mid \\ & & \texttt{fun}\ x{:}\,\tau.\,\mathcal{C} \mid \mathcal{C}\ t \mid t\ \mathcal{C} \mid \texttt{fix}(\mathcal{C}) \end{array}$$

Typing extended to evaluation contexts: $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$.

"Term with a hole":

$$\mathcal{C} \;::=\; - \mid \texttt{succ}(\mathcal{C}) \mid \texttt{pred}(\mathcal{C}) \mid \texttt{zero?}(\mathcal{C}) \mid$$
$$\texttt{if}\ \mathcal{C}\ \texttt{then}\ t\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ \mathcal{C}\ \texttt{else}\ t \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ \mathcal{C} \mid$$
$$\texttt{fun}\ x{:}\tau.\ \mathcal{C} \mid \mathcal{C}\ t \mid t\ \mathcal{C} \mid \texttt{fix}(\mathcal{C})$$

Typing extended to evaluation contexts: $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$.

$$\frac{}{\Gamma \vdash_{\Gamma,\tau} - : \tau} \qquad \frac{\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash_{\Delta,\sigma} \mathcal{C}\ u : \tau_2} \qquad \dots$$

Given a type τ, a typing context Γ and terms $t, t' \in \mathrm{PCF}_{\Gamma,\tau}$, contextual equivalence, written $\Gamma \vdash t \cong_{\mathrm{ctx}} t' : \tau$ is defined to hold if for all evaluation contexts $\mathcal{C}$ such that $\cdot \vdash_{\Gamma,\tau} \mathcal{C} : \gamma$, where γ is `nat` or `bool`, and for all values $v \in \mathrm{PCF}_{\gamma}$,

$$\mathcal{C}[t] \Downarrow_{\gamma} v \Leftrightarrow \mathcal{C}[t'] \Downarrow_{\gamma} v.$$

When Γ is the empty context, we simply write $t \cong_{\mathrm{ctx}} t' : \tau$ for $\cdot \vdash t \cong_{\mathrm{ctx}} t' : \tau$.

Given a type τ, a typing context Γ and terms $t, t' \in \textsc{Pcf}_{\Gamma,\tau}$, **contextual equivalence**, written $\Gamma \vdash t \cong_{\mathrm{ctx}} t' : \tau$ is defined to hold if for all evaluation contexts $\mathcal{C}$ such that $\cdot \vdash_{\Gamma,\tau} \mathcal{C} : \gamma$, where γ is `nat` or `bool`, and for all values $v \in \textsc{Pcf}_{\gamma}$,

$$\mathcal{C}[t] \Downarrow_{\gamma} v \Leftrightarrow \mathcal{C}[t'] \Downarrow_{\gamma} v.$$

When Γ is the empty context, we simply write $t \cong_{\mathrm{ctx}} t' : \tau$ for $\cdot \vdash t \cong_{\mathrm{ctx}} t' : \tau$.

Divergence is implicitly covered.

Denotational Semantics for Pcf

Denotational Semantics for Pcf

Introducing denotational semantics

The aims of denotational semantics

- a mapping of PCF types τ to domains $[\![\tau]\!]$;
- a mapping of closed, well-typed PCF terms $\cdot \vdash t : \tau$ to elements $[\![t]\!] \in [\![\tau]\!]$;
- denotation of open terms will be continuous functions.

- a mapping of PCF types τ to domains $[\![\tau]\!]$;
- a mapping of closed, well-typed PCF terms $\cdot \vdash t : \tau$ to elements $[\![t]\!] \in [\![\tau]\!]$;
- denotation of open terms will be continuous functions.

Compositionality: $[\![t]\!] = [\![t']\!] \Rightarrow [\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[t']]\!]$.

Soundness: for any type τ, $t \Downarrow_\tau v \Rightarrow [\![t]\!] = [\![v]\!]$.

Adequacy: for $\gamma = \texttt{bool}$ or $\texttt{nat}$, if $t \in \mathrm{PCF}_\gamma$ and $[\![t]\!] = [\![v]\!]$ then $t \Downarrow_\gamma v$.

$$v \stackrel{\text{def}}{=} \texttt{fun}\, x\colon \texttt{nat}.\, (\texttt{fun}\, y\colon \texttt{nat}.\, y)\, \texttt{0} \quad \text{and} \quad v' \stackrel{\text{def}}{=} \texttt{fun}\, x\colon \texttt{nat}.\, \texttt{0}.$$

Proof principle: to show

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau$$

it suffices to establish

$$[\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

Proof principle: to show

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau$$

it suffices to establish

$$[\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

$$\begin{aligned}
\mathcal{C}[t_1] \Downarrow_{\mathtt{nat}} v &\Rightarrow [\![\mathcal{C}[t_1]]\!] = [\![v]\!] && \text{(soundness)}\\
&\Rightarrow [\![\mathcal{C}[t_2]]\!] = [\![v]\!] && \text{(compositionality on } [\![t_1]\!] = [\![t_2]\!]\text{)}\\
&\Rightarrow \mathcal{C}[t_2] \Downarrow_{\mathtt{nat}} v && \text{(adequacy)}
\end{aligned}$$

Proof principle: to show

$$t_1 \cong_{\text{ctx}} t_2 : \tau$$

it suffices to establish

$$[\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

$$\begin{aligned} \mathcal{C}[t_1] \Downarrow_{\texttt{nat}} v &\Rightarrow [\![\mathcal{C}[t_1]]\!] = [\![v]\!] && \text{(soundness)} \\ &\Rightarrow [\![\mathcal{C}[t_2]]\!] = [\![v]\!] && \text{(compositionality on } [\![t_1]\!] = [\![t_2]\!]\text{)} \\ &\Rightarrow \mathcal{C}[t_2] \Downarrow_{\texttt{nat}} v && \text{(adequacy)} \end{aligned}$$

and symmetrically for $\mathcal{C}[t_2] \Downarrow_{\texttt{nat}} v \Rightarrow \mathcal{C}[t_1] \Downarrow_{\texttt{nat}} v$, and similarly for `bool`.

The power of denotational semantics

Proof principle: to show

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau$$

it suffices to establish

$$[\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

Denotational equality is **sound**, but is it **complete**?
Does equality in the model imply contextual equivalence?

Proof principle: to show

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau$$

it suffices to establish

$$[\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

Denotational equality is sound, but is it complete?
Does equality in the model imply contextual equivalence?

Full abstraction.

Denotational Semantics for Pcf

Definition

$$[\![\texttt{nat}]\!] \stackrel{\text{def}}{=} \mathbb{N}_\perp \qquad \text{(flat domain)}$$

$$[\![\texttt{bool}]\!] \stackrel{\text{def}}{=} \mathbb{B}_\perp \qquad \text{(flat domain)}$$

$$[\![\tau \to \tau']\!] \stackrel{\text{def}}{=} [\![\tau]\!] \to [\![\tau']\!] \qquad \text{(function domain)}$$

Semantics of contexts

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \prod_{x \in \mathrm{dom}(\Gamma)} [\![\Gamma(x)]\!] \qquad \text{(environment)}$$

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \prod_{x \in \text{dom}(\Gamma)} [\![\Gamma(x)]\!] \qquad \text{(environment)}$$

- $[\![\cdot]\!] = \mathbb{1}$ (one element set)
- $[\![x : \tau]\!] = (\{x\} \to [\![\tau]\!]) \cong [\![\tau]\!]$
- $[\![x_1 : \tau_1, \ldots, x_n : \tau_n]\!] \cong [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$

Denotational semantics of Pcf

To every typing judgement

$$\Gamma \vdash t : \tau$$

we associate a continuous function

$$[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

between domains. In other words,

$$[\![-]\!] : \textsc{Pcf}_{\Gamma,\tau} \to [\![\Gamma]\!] \to [\![\tau]\!]$$

Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

$$\begin{aligned} \text{succ} : \quad \mathbb{N} &\to \mathbb{N} \\ n &\mapsto n+1 \end{aligned}$$

$$\begin{aligned} \text{pred} : \quad \mathbb{N} &\rightharpoonup \mathbb{N} \\ n+1 &\mapsto n \\ 0 &\quad \text{undefined} \end{aligned}$$

$$\begin{aligned} \text{zero?} : \quad \mathbb{N} &\to \mathbb{B} \\ 0 &\mapsto \text{true} \\ n+1 &\mapsto \text{false} \end{aligned}$$

Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

$$\begin{aligned} \text{succ}_\bot : \quad \mathbb{N}_\bot &\to \mathbb{N}_\bot \\ n &\mapsto n+1 \\ \bot &\mapsto \bot \end{aligned}$$

$$\begin{aligned} \text{pred}_\bot : \quad \mathbb{N}_\bot &\to \mathbb{N}_\bot \\ n+1 &\mapsto n \\ 0 &\mapsto \bot \\ \bot &\mapsto \bot \end{aligned}$$

$$\begin{aligned} \text{zero?}_\bot : \quad \mathbb{N}_\bot &\to \mathbb{B}_\bot \\ 0 &\mapsto \text{true} \\ n+1 &\mapsto \text{false} \\ \bot &\mapsto \bot \end{aligned}$$

Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

$$
\begin{aligned}
[\![\texttt{0}]\!](\rho) &\stackrel{\text{def}}{=} 0 && \in \mathbb{N}_\perp \\
[\![\texttt{true}]\!](\rho) &\stackrel{\text{def}}{=} \text{true} && \in \mathbb{B}_\perp \\
[\![\texttt{false}]\!](\rho) &\stackrel{\text{def}}{=} \text{false} && \in \mathbb{B}_\perp
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathtt{0}]\!]\,(\rho) &\stackrel{\text{def}}{=} 0 && \in \mathbb{N}_\perp \\
[\![\mathtt{true}]\!]\,(\rho) &\stackrel{\text{def}}{=} \text{true} && \in \mathbb{B}_\perp \\
[\![\mathtt{false}]\!]\,(\rho) &\stackrel{\text{def}}{=} \text{false} && \in \mathbb{B}_\perp
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathtt{succ}(t)]\!]\,(\rho) &\stackrel{\text{def}}{=} \text{succ}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{N}_\perp \\
[\![\mathtt{pred}(t)]\!]\,(\rho) &\stackrel{\text{def}}{=} \text{pred}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{N}_\perp \\
[\![\mathtt{zero?}(t)]\!]\,(\rho) &\stackrel{\text{def}}{=} \text{zero?}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{B}_\perp
\end{aligned}
$$

$$
[\![\mathtt{succ}(t)]\!] = \text{succ}_\perp \circ [\![t]\!]
$$

Denotation of operations on $\mathbb{B}$ and $\mathbb{N}$

$$
\begin{aligned}
[\![\texttt{0}]\!]\,(\rho) &\overset{\text{def}}{=} 0 && \in \mathbb{N}_\perp \\
[\![\texttt{true}]\!]\,(\rho) &\overset{\text{def}}{=} \text{true} && \in \mathbb{B}_\perp \\
[\![\texttt{false}]\!]\,(\rho) &\overset{\text{def}}{=} \text{false} && \in \mathbb{B}_\perp
\end{aligned}
$$

$$
\begin{aligned}
[\![\texttt{succ}(t)]\!]\,(\rho) &\overset{\text{def}}{=} \text{succ}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{N}_\perp \\
[\![\texttt{pred}(t)]\!]\,(\rho) &\overset{\text{def}}{=} \text{pred}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{N}_\perp \\
[\![\texttt{zero?}(t)]\!]\,(\rho) &\overset{\text{def}}{=} \text{zero?}_\perp([\![t]\!]\,(\rho)) && \in \mathbb{B}_\perp
\end{aligned}
$$

$$
[\![\texttt{if}\ b\ \texttt{then}\ t\ \texttt{else}\ t']\!] \overset{\text{def}}{=} \text{if}([\![b]\!]\,(\rho), [\![t]\!]\,(\rho), [\![t']\!]\,(\rho)) \quad \in [\![\tau]\!]
$$

$$
[\![\texttt{if}\ b\ \texttt{then}\ t\ \texttt{else}\ t']\!] = \text{if} \circ \langle [\![b]\!]\,, \langle [\![t]\!]\,, [\![t']\!] \rangle\rangle
$$

$$[\![x]\!]\,(\rho) \stackrel{\text{def}}{=} \rho(x) \qquad \in [\![\Gamma(x)]\!]$$

$$[\![x]\!]\,(\rho) = \pi_x(\rho)$$

Denotation of the λ-calculus operations

$$\llbracket x \rrbracket (\rho) \stackrel{\text{def}}{=} \rho(x) \in \llbracket \Gamma(x) \rrbracket$$

$$\llbracket t_1\ t_2 \rrbracket (\rho) \stackrel{\text{def}}{=} (\llbracket t_1 \rrbracket (\rho))\ (\llbracket t_2 \rrbracket (\rho))$$

$$\llbracket t_1\ t_2 \rrbracket = \mathrm{eval} \circ \langle \llbracket t_1 \rrbracket , \llbracket t_2 \rrbracket \rangle$$

Denotation of the λ-calculus operations

$$\begin{aligned}
[\![x]\!]\,(\rho) &\stackrel{\text{def}}{=} \rho(x) && \in [\![\Gamma(x)]\!] \\
[\![t_1\ t_2]\!]\,(\rho) &\stackrel{\text{def}}{=} ([\![t_1]\!]\,(\rho))\ ([\![t_2]\!]\,(\rho)) \\
[\![\mathtt{fun}\,x{:}\,\tau.\,t]\!]\,(\rho) &\stackrel{\text{def}}{=} \lambda d \in [\![\tau]\!]\,.\ [\![t]\!]\,(\rho, d)
\end{aligned}$$

$$[\![\mathtt{fun}\,x{:}\,\tau.\,t]\!] = \mathrm{cur}([\![t]\!])$$

Denotation of fixed points

$$[\![\texttt{fix}\ f]\!]\,(\rho) \quad \stackrel{\text{def}}{=} \quad \mathrm{fix}([\![f]\!]\,(\rho))$$

Denotation of Pcf terms

For any Pcf term t such that $\Gamma \vdash t : \tau$, the object $[\![t]\!]$
is well-defined and a continuous function $[\![t]\!] : [\![\Gamma]\!] \to \tau$.

For any Pcf term t such that $\Gamma \vdash t : \tau$, the object $[\![t]\!]$ is well-defined and a continuous function $[\![t]\!] : [\![\Gamma]\!] \to \tau$.

If $t \in \text{Pcf}_\tau$: $[\![t]\!] \in [\![\cdot]\!] \to [\![\tau]\!] = \mathbb{1} \to [\![\tau]\!] \cong [\![\tau]\!]$

Denotational Semantics for Pcf

Compositionality

Suppose $t, u \in \textsc{Pcf}_{\Delta,\sigma}$, such that

$$[\![t]\!] = [\![u]\!] : [\![\Delta]\!] \to [\![\sigma]\!]$$

Suppose moreover that $\mathcal{C}[-]$ is a $\textsc{Pcf}$ context such that $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$. Then

$$[\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[u]]\!] : [\![\Gamma]\!] \to [\![\tau]\!] .$$

A denotation for evaluation contexts

If $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$, then define $[\![\mathcal{C}]\!]$ such that

$$[\![\mathcal{C}]\!] : ([\![\Delta]\!] \to [\![\sigma]\!]) \to [\![\Gamma]\!] \to [\![\tau]\!]$$

A denotation for evaluation contexts

If $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$, then define $[\![\mathcal{C}]\!]$ such that

$$[\![\mathcal{C}]\!] : ([\![\Delta]\!] \to [\![\sigma]\!]) \to [\![\Gamma]\!] \to [\![\tau]\!]$$

$$\begin{aligned} [\![-]\!]\,(d) &= d \\ [\![\mathcal{C}\ t]\!]\,(d)(\rho) &= ([\![\mathcal{C}]\!]\,(d)(\rho))([\![t]\!]\,(\rho)) \\ &\vdots \end{aligned}$$

A denotation for evaluation contexts

If $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$, then define $[\![\mathcal{C}]\!]$ such that

$$[\![\mathcal{C}]\!] : ([\![\Delta]\!] \to [\![\sigma]\!]) \to [\![\Gamma]\!] \to [\![\tau]\!]$$

$$\begin{aligned}
[\![-]\!]\,(d) &= d \\
[\![\mathcal{C}\;t]\!]\,(d)(\rho) &= ([\![\mathcal{C}]\!]\,(d)(\rho))([\![t]\!]\,(\rho)) \\
&\vdots
\end{aligned}$$

If $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$ and $\Delta \vdash t : \sigma$, then

$$[\![\mathcal{C}[t]]\!] = [\![\mathcal{C}]\!]\,([\![t]\!])$$

Assume

$$\Gamma \vdash u : \sigma$$
$$\Gamma, x{:}\sigma \vdash t : \tau$$

Then for all $\rho \in [\![\Gamma]\!]$

$$[\![t[u/x]]\!]\,(\rho) = [\![t]\!]\,(\rho[x \mapsto [\![u]\!]\,(\rho)]).$$

In particular when $\Gamma = \cdot$, $[\![t]\!] : [\![\sigma]\!] \to [\![\tau]\!]$ and

$$[\![t[u/x]]\!] = [\![t]\!]\,([\![u]\!])$$

Denotational Semantics for Pcf

Soundness

Soundness

For all PCF types τ and all closed terms $t, v \in \text{PCF}_\tau$ with v a value, if $t \Downarrow_\tau v$ is derivable, then

$$[\![t]\!] = [\![v]\!] \in [\![\tau]\!]$$

Divergence

If $t \in \textsc{Pcf}_{\mathtt{nat}}$ and $[\![t]\!] = \bot$, then $t \Uparrow_{\mathtt{nat}}$.

Adequacy

Reminder: Adequacy

For any closed Pcf term t and value v of ground type $\gamma \in \{\mathtt{nat}, \mathtt{bool}\}$

$$[\![t]\!] = [\![v]\!] \in [\![\gamma]\!] \Rightarrow t \Downarrow_\gamma v$$

Reminder: Adequacy

For any closed PCF term t and value v of ground type $\gamma \in \{\mathtt{nat}, \mathtt{bool}\}$

$$[\![t]\!] = [\![v]\!] \in [\![\gamma]\!] \Rightarrow t \Downarrow_\gamma v$$

Adequacy does not hold at function types or for open terms

Reminder: Adequacy

For any **closed** PCF term t and value v of **ground** type $\gamma \in \{\mathtt{nat}, \mathtt{bool}\}$

$$[\![t]\!] = [\![v]\!] \in [\![\gamma]\!] \Rightarrow t \Downarrow_\gamma v$$

Adequacy does **not** hold at function types or for open terms

$$[\![\mathtt{fun}\, x\colon\tau.\,(\mathtt{fun}\, y\colon\tau.\, y)\, x]\!] \quad = \quad [\![\mathtt{fun}\, x\colon\tau.\, x]\!] \quad : [\![\tau]\!] \to [\![\tau]\!]$$

but

$$\mathtt{fun}\, x\colon\tau.\,(\mathtt{fun}\, y\colon\tau.\, y)\, x \not\Downarrow_{\tau\to\tau} \mathtt{fun}\, x\colon\tau.\, x$$

Reminder: Adequacy

For any **closed** PCF term t and value v of **ground** type $\gamma \in \{\texttt{nat}, \texttt{bool}\}$

$$[\![t]\!] = [\![v]\!] \in [\![\gamma]\!] \Rightarrow t \Downarrow_\gamma v$$

Adequacy does **not** hold at function types or for open terms

More serious:

$$\begin{aligned} & [\![\texttt{fun}\, x\colon \texttt{nat}.\, (\texttt{if zero?}(f\ x)\ \texttt{then true else true})]\!] \\ \stackrel{?}{=}\ & [\![\texttt{fun}\, x\colon \texttt{nat}.\, \texttt{true}]\!] \end{aligned}$$

Adequacy

Formal approximation relation

Proof idea: introduce a relation R such that

1. if $t \in \textsc{Pcf}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans);
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

Proof idea: introduce a relation R such that

1. if $t \in \textsc{Pcf}_{\mathtt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans);
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

But at non-base types, adequacy does not hold.

Proof idea: introduce a relation R such that

1. if $t \in \text{PCF}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans);
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

But at non-base types, adequacy does not hold.

We must define a family of relations, tailored for each type: formal approximation

$$\triangleleft_\tau \subseteq \llbracket \tau \rrbracket \times \text{PCF}_\tau$$

Proof idea: introduce a relation R such that

1. if $t \in \textsc{Pcf}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans);
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

But at non-base types, adequacy does not hold.

We must define a **family** of relations, tailored for each type: **formal approximation**

$$\lhd_\tau \subseteq \llbracket \tau \rrbracket \times \textsc{Pcf}_\tau$$

A **logical relation**.

Proof idea: introduce a relation R such that

1. if $t \in \textsc{Pcf}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n,t)$, then $t \Downarrow_{\gamma} \underline{n}$ (same for booleans);
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

But at non-base types, **adequacy does not hold**.

We must define a **family** of relations, tailored for each type: **formal approximation**

$$\lhd_{\tau} \subseteq \llbracket \tau \rrbracket \times \textsc{Pcf}_{\tau}$$

A **logical relation**.

$$d \lhd_{\texttt{nat}} t \overset{\text{def}}{\Leftrightarrow} (d \in \mathbb{N} \Rightarrow t \Downarrow_{\texttt{nat}} \underline{d})$$

$$\begin{aligned} d \lhd_{\texttt{bool}} t \overset{\text{def}}{\Leftrightarrow} & (d = \text{true} \Rightarrow t \Downarrow_{\texttt{bool}} \texttt{true}) \\ & \wedge (d = \text{false} \Rightarrow t \Downarrow_{\texttt{bool}} \texttt{false}) \end{aligned}$$

$$d \lhd_{\mathtt{nat}} t \overset{\text{def}}{\Leftrightarrow} (d \in \mathbb{N} \Rightarrow t \Downarrow_{\mathtt{nat}} \underline{d})$$

$$\begin{aligned} d \lhd_{\mathtt{bool}} t \overset{\text{def}}{\Leftrightarrow} & (d = \text{true} \Rightarrow t \Downarrow_{\mathtt{bool}} \mathtt{true}) \\ & \wedge (d = \text{false} \Rightarrow t \Downarrow_{\mathtt{bool}} \mathtt{false}) \end{aligned}$$

Exactly what we need to get 1.

Formal approximation at base types

$$d \lhd_{\texttt{nat}} t \overset{\text{def}}{\Leftrightarrow} (d \in \mathbb{N} \Rightarrow t \Downarrow_{\texttt{nat}} \underline{d})$$

$$\begin{aligned} d \lhd_{\texttt{bool}} t \overset{\text{def}}{\Leftrightarrow} & (d = \text{true} \Rightarrow t \Downarrow_{\texttt{bool}} \texttt{true}) \\ & \wedge (d = \text{false} \Rightarrow t \Downarrow_{\texttt{bool}} \texttt{false}) \end{aligned}$$

Exactly what we need to get 1.

Note though that $\bot \lhd_{\texttt{nat}} t$ for any $t \in \mathrm{PCF}_{\texttt{nat}}$.

1. if $t \in \textsc{Pcf}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans); ✓
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.

Formal approximation at function types

1. if $t \in \textsc{Pcf}_{\mathtt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_{\gamma} \underline{n}$ (same for booleans); ✓
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.
   2.1 By induction on (the typing derivation of) t;
   2.2 we need to interpret each typing rule.

1. if $t \in \textsc{Pcf}_{\mathtt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans); ✓
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.
   - 2.1 By induction on (the typing derivation of) t;
   - 2.2 we need to interpret each typing rule.

$$\textsc{App}\ \frac{\vdash t : \tau \to \tau' \qquad \vdash u : \tau}{\vdash t\ u : \tau'}$$

Formal approximation at function types

1. if $t \in \textsc{Pcf}_{\mathtt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans); ✓
2. for any well-typed term t, $R([\![t]\!], t)$.
   2.1 By induction on (the typing derivation of) t;
   2.2 we need to interpret each typing rule.

$$\textsc{App}\ \frac{\vdash t : \tau \to \tau' \qquad \vdash u : \tau}{\vdash t\ u : \tau'}$$

Assume $[\![u]\!] \lhd_\tau u$ and $[\![t]\!] \lhd_{\tau \to \tau'} t$, how do we get $[\![t\ u]\!] = [\![t]\!]\,([\![u]\!]) \lhd_\tau t\ u$?

1. if $t \in \text{PCF}_{\texttt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_\gamma \underline{n}$ (same for booleans); ✓
2. for any well-typed term t, $R(\llbracket t \rrbracket, t)$.
   2.1 By induction on (the typing derivation of) t;
   2.2 we need to interpret each typing rule.

$$\textsc{App}\ \frac{\vdash t : \tau \to \tau' \qquad \vdash u : \tau}{\vdash t\ u : \tau'}$$

Assume $\llbracket u \rrbracket \lhd_\tau u$ and $\llbracket t \rrbracket \lhd_{\tau \to \tau'} t$, how do we get $\llbracket t\ u \rrbracket = \llbracket t \rrbracket (\llbracket u \rrbracket) \lhd_\tau t\ u$?

Define

$$d \lhd_{\tau \to \tau'} t \overset{\text{def}}{\Leftrightarrow} \forall e \in \llbracket \tau \rrbracket, u \in \text{PCF}_\tau .(e \lhd_\tau u \Rightarrow d(e) \lhd_{\tau'} t\ u)$$

$$\textsc{Abs}\ \frac{\Gamma, x{:}\tau \vdash t : \tau'}{\Gamma \vdash \mathsf{fun}\, x{:}\tau.\, t : \tau \to \tau'}$$

To prove Item 2, we need to talk about **open** terms.

$$\text{Abs}\ \frac{\Gamma, x{:}\tau \vdash t : \tau'}{\Gamma \vdash \mathsf{fun}\, x{:}\tau.\, t : \tau \to \tau'}$$

To prove Item 2, we need to talk about **open** terms.

$$[\![t]\!]\,([\![u]\!]) = [\![(t[u/x])]\!]$$ Semantic application ≈ syntactic substitution

Formal approximation for open terms

$$\textsc{Abs}\ \frac{\Gamma, x{:}\tau \vdash t : \tau'}{\Gamma \vdash \mathsf{fun}\, x{:}\tau.\, t : \tau \to \tau'}$$

To prove Item 2, we need to talk about open terms.

$$[\![t]\!]\left([\![u]\!]\right) = [\![(t[u/x])]\!] \qquad \text{Semantic application} \approx \text{syntactic substitution}$$

Parallel substitution: maps each $x \in \mathrm{dom}(\Gamma)$ to $\sigma(x) \in \textsc{Pcf}_{\Gamma(x)}$.

$$\rho \lhd_\Gamma \sigma \overset{\text{def}}{\Leftrightarrow} \forall x \in \mathrm{dom}(\Gamma), \rho(x) \lhd_{\Gamma(x)} \sigma(x)$$

The fundamental theorem

For any

- context Γ and type τ
- term t such that $\Gamma \vdash t : \tau$
- environment ρ
- substitution σ
- such that $\rho \lhd_\Gamma \sigma$

we have

$$[\![t]\!]\,(\rho) \lhd_\tau t[\sigma].$$

The fundamental theorem

For any

- context Γ and type τ
- term t such that $\Gamma \vdash t : \tau$
- environment ρ
- substitution σ
- such that $\rho \lhd_\Gamma \sigma$

we have

$$[\![t]\!]\,(\rho) \lhd_\tau t[\sigma].$$

Corollary: if $\cdot \vdash t : \tau$,

$$[\![t]\!] \lhd_\tau t.$$

Adequacy

Proof of the fundamental property of formal approximation

Properties of formal approximation

1. The least element approximates any program: for any τ and $t \in \textsc{Pcf}_\tau$, $\bot_{[\![\tau]\!]} \lhd_\tau t$;
2. if $d' \sqsubseteq d$ and $d \lhd_\tau t$, then $d' \lhd_\tau t$;
3. the set $\{d \in [\![\tau]\!] \mid d \lhd_\tau t\}$ is chain-closed;

Properties of formal approximation

1. The least element approximates any program: for any τ and $t \in \textsc{Pcf}_\tau$, $\bot_{[\![\tau]\!]} \lhd_\tau t$;
2. if $d' \sqsubseteq d$ and $d \lhd_\tau t$, then $d' \lhd_\tau t$;
3. the set $\{d \in [\![\tau]\!] \mid d \lhd_\tau t\}$ is chain-closed;
4. if $\forall v.\, t \Downarrow_\tau v \Rightarrow t' \Downarrow_\tau v$, and $d \lhd_\tau t$, then $d \lhd_\tau t'$.

Fundamental property

For any

- context Γ, type τ and term t such that $\Gamma \vdash t : \tau$
- environment ρ
- substitution σ
- such that $\rho \lhd_\Gamma \sigma$

we have $[\![t]\!]\,(\rho) \lhd_\tau t[\sigma]$.

For any

- context Γ, type τ and term t such that $\Gamma \vdash t : \tau$
- environment ρ
- substitution σ
- such that $\rho \lhd_\Gamma \sigma$

we have $[\![t]\!]\,(\rho) \lhd_\tau t[\sigma]$.

Proof! Induction on $\Gamma \vdash t : \tau$:

$$\forall \rho, \sigma.\, (\rho \lhd_\Gamma \sigma \;\Rightarrow [\![t]\!]\,(\rho) \lhd_\tau t[\sigma])$$

Adequacy

Extensionality

Contextual preorder is the one-sided version of contextual equivalence: $\Gamma \vdash t \leq_{\mathrm{ctx}} t' : \tau$ if for all $\mathcal{C}$ such that $\cdot \vdash_{\Gamma,\tau} \mathcal{C} : \gamma$ and for all values v,

$$\mathcal{C}[t] \Downarrow_\gamma v \Rightarrow \mathcal{C}[t'] \Downarrow_\gamma v.$$

Contextual preorder is the one-sided version of contextual equivalence: $\Gamma \vdash t \leq_{\text{ctx}} t' : \tau$ if for all $\mathcal{C}$ such that $\cdot \vdash_{\Gamma,\tau} \mathcal{C} : \gamma$ and for all values v,

$$\mathcal{C}[t] \Downarrow_\gamma v \Rightarrow \mathcal{C}[t'] \Downarrow_\gamma v.$$

$$\Gamma \vdash t \cong_{\text{ctx}} t' : \tau \Leftrightarrow (\Gamma \vdash t \leq_{\text{ctx}} t' : \tau \wedge \Gamma \vdash t' \leq_{\text{ctx}} t : \tau)$$

Let τ be a type, and assume $t_1, t_2 \in \textsc{Pcf}_\tau$ are such that $t_1 \leq_{\mathrm{ctx}} t_2 : \tau$. Then

$$d \lhd_\tau t_1 \Rightarrow d \lhd_\tau t_2.$$

Lemma: application contexts

To characterise contextual preorder between closed terms, applicative contexts are enough.

Lemma: application contexts

To characterise contextual preorder between closed terms, applicative contexts are enough.

Let t_1, t_2 be closed terms of type τ. Then $t_1 \leq_{\mathrm{ctx}} t_2 : \tau$ if and only if, for every term $f : \tau \to \texttt{bool}$,

$$f\ t_1 \Downarrow_{\texttt{bool}} \texttt{true} \Rightarrow f\ t_2 \Downarrow_{\texttt{bool}} \texttt{true}.$$

Contextual preorder and formal approximation

Formal approximation corresponds to the contextual preorder.

Formal approximation **corresponds to** the contextual preorder.

For all PCF types τ and all closed terms $t_1, t_2 \in \mathrm{PCF}_\tau$

$$t_1 \leq_{\mathrm{ctx}} t_2 : \tau \Leftrightarrow [\![t_1]\!] \lhd_\tau t_2.$$

Extensionality properties of contextual preorder

For $\gamma = \texttt{bool}$ or $\texttt{nat}$, $t_1 \leq_{\text{ctx}} t_2 : \gamma$ holds if and only if

$$\forall v.\, (t_1 \Downarrow_\gamma v \Rightarrow t_2 \Downarrow_\gamma v).$$

For $\gamma = \texttt{bool}$ or $\texttt{nat}$, $t_1 \leq_{\text{ctx}} t_2 : \gamma$ holds if and only if

$$\forall v.\, (t_1 \Downarrow_\gamma v \Rightarrow t_2 \Downarrow_\gamma v).$$

At a function type $\tau \to \tau'$, $t_1 \leq_{\text{ctx}} t_2 : \tau \to \tau'$ holds if and only if

$$\forall t \in \textsc{Pcf}_\tau\, .\, (t_1\, t \leq_{\text{ctx}} t_2\, t : \tau').$$

Full abstraction

Full abstraction

Failure of full abstraction

Full abstraction

A denotational model is fully abstract if

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau \Rightarrow [\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

Full abstraction

A denotational model is fully abstract if

$$t_1 \cong_{\mathrm{ctx}} t_2 : \tau \Rightarrow [\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

A form of completeness of semantic equivalence wrt. program equivalence.

Full abstraction

A denotational model is fully abstract if

$$t_1 \cong_{\text{ctx}} t_2 : \tau \Rightarrow [\![t_1]\!] = [\![t_2]\!] \in [\![\tau]\!]$$

A form of completeness of semantic equivalence wrt. program equivalence.

The domain model of PCF is not fully abstract.

The *parallel or* function $\mathbf{por} : \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$ is defined as given by the following table:

por	true	false	$\perp$
true	true	true	true
false	true	false	$\perp$
$\perp$	true	$\perp$	$\perp$

Left sequential or

The (left) sequential or function $\mathbf{or} : \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$ is defined as

$$\mathbf{or} \stackrel{\text{def}}{=} [\![\texttt{fun}\, x\colon \texttt{bool}.\ \texttt{fun}\, y\colon \texttt{bool}.\ \texttt{if}\ x\ \texttt{then}\ \texttt{true}\ \texttt{else}\ y]\!]$$

It is given by the following table:

or	true	false	$\perp$
true	true	true	true
false	true	false	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

Parallel vs sequential or

por	true	false	⊥
true	true	true	true
false	true	false	⊥
⊥	true	⊥	⊥

or	true	false	⊥
true	true	true	true
false	true	false	⊥
⊥	⊥	⊥	⊥

Parallel vs sequential or

por	true	false	⊥
true	true	true	true
false	true	false	⊥
⊥	true	⊥	⊥

or	true	false	⊥
true	true	true	true
false	true	false	⊥
⊥	⊥	⊥	⊥

or is sequential, but por is not.

Undefinability or parallel or

There is **no** closed Pcf term

$$t : \texttt{bool} \to \texttt{bool} \to \texttt{bool}$$

satisfying

$$[\![t]\!] = \mathrm{por} : \mathbb{B}_\perp \to \mathbb{B}_\perp \to \mathbb{B}_\perp \ .$$

Failure of full abstraction

The denotational model of PCF in domains and continuous functions is not fully abstract.

Failure of full abstraction

The denotational model of PCF in domains and continuous functions is not fully abstract.

For well-chosen $T_{\texttt{true}}$ and $T_{\texttt{false}}$,

$$T_{\texttt{true}} \cong_{\text{ctx}} T_{\texttt{false}} : (\texttt{bool} \to \texttt{bool} \to \texttt{bool}) \to \texttt{bool}$$

$$[\![T_{\texttt{true}}]\!] \neq [\![T_{\texttt{false}}]\!] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$

The denotational model of PCF in domains and continuous functions is not fully abstract.

For well-chosen $T_{\mathtt{true}}$ and $T_{\mathtt{false}}$,

$$T_{\mathtt{true}} \cong_{\text{ctx}} T_{\mathtt{false}} : (\mathtt{bool} \to \mathtt{bool} \to \mathtt{bool}) \to \mathtt{bool}$$

$$[\![T_{\mathtt{true}}]\!] \neq [\![T_{\mathtt{false}}]\!] \in (\mathbb{B} \to \mathbb{B} \to \mathbb{B}) \to \mathbb{B}$$

Idea:

- for all $f \in PCF_{\mathtt{bool}\to\mathtt{bool}\to\mathtt{bool}}$, ensure $T_b\ f \Uparrow_{\mathtt{bool}}$...
- but $[\![T_b]\!]\ (\mathrm{por}) = [\![b]\!]$.

Example of full abstraction failure

$$
\begin{aligned}
T_b \stackrel{\text{def}}{=}\ & \texttt{fun}\ f\colon \texttt{bool} \to (\texttt{bool} \to \texttt{bool}).\\
& \quad \texttt{if}(f\ \texttt{true}\ \Omega_{\texttt{bool}})\ \texttt{then}\\
& \quad\quad \texttt{if}\ (f\ \Omega_{\texttt{bool}}\ \texttt{true})\ \texttt{then}\\
& \quad\quad\quad \texttt{if}\ (f\ \texttt{false}\ \texttt{false})\ \texttt{then}\ \Omega_{\texttt{bool}}\ \texttt{else}\ b\\
& \quad\quad \texttt{else}\ \Omega_{\texttt{bool}}\\
& \quad \texttt{else}\ \Omega_{\texttt{bool}}
\end{aligned}
$$

Full abstraction

Beyond full abstraction failure

Interpreting full abstraction failure

- Pcf is not expressive enough to present the model?
- The model does not adequately capture Pcf?
- Contexts are too weak: they do not distinguish enough programs?

$\boxed{\Gamma \vdash t : \tau}$

$$\ldots \qquad \textsc{Por}\ \frac{\Gamma \vdash t_1 : \tau \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash \mathtt{por}(t_1, t_2) : \tau}$$

$\boxed{t \Downarrow_\tau v}$

$$\textsc{PorL}\ \frac{t_1 \Downarrow_{\mathtt{bool}} \mathtt{true}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{true}} \qquad \textsc{PorR}\ \frac{t_2 \Downarrow_{\mathtt{bool}} \mathtt{true}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{true}}$$

$$\textsc{PorF}\ \frac{t_1 \Downarrow_{\mathtt{bool}} \mathtt{false} \qquad t_2 \Downarrow_{\mathtt{bool}} \mathtt{false}}{\mathtt{por}(t_1, t_2) \Downarrow_{\mathtt{bool}} \mathtt{false}}$$

If we extend the semantics of Pcf to Pcf+`por` with

$$[\![\texttt{por}]\!] = \mathrm{por}$$

the resulting denotational semantics is fully abstract.

Full abstraction for Pcf+`por`

If we extend the semantics of Pcf to Pcf+`por` with

$$[\![\texttt{por}]\!] = \mathrm{por}$$

the resulting denotational semantics is fully abstract…

but is Pcf+`por` still a reasonable model of programming language?

Fully abstract semantics

Fully abstract semantics for Pcf

- first step: dI-domains & stable functions $\rightarrow$ no $\mathbf{por}$ any more, but still not fully abstract...
- only proper answers in the late 90s (!): logical relations and game semantics

Fully abstract semantics

Fully abstract semantics for Pcf

- first step: dI-domains & stable functions → no **por** any more, but still not fully abstract...
- only proper answers in the late 90s (!): logical relations and game semantics

Real languages have effects

- If you add effects (references, control flow...) to a language, contexts become *much more* expressive.
- Full abstraction becomes different: somewhat easier... but is contextual equivalence still a reasonable idea?

Where to go from here?

Towards full abstraction

Source of a very rich literature:

- linear logic
- logical relations
- game semantics
- bisimulations techniques
- ...

Categorical semantics

Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)

Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)

Interpret:

- a type τ as an object in a category;
- a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

Separate

- the structure needed to interpret a language (generic)
- how to construct this structure in particular examples (specific)

Interpret:

- a type τ as an object in a category;
- a term $\Gamma \vdash t : \tau$ as a morphism/arrow $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$.

Example: λ-calculus → cartesian closed categories

OCaml's ADT:

```
type 'a tree =
  | Leaf
  | Node of 'a * 'a tree * 'a tree
```

It is a fixed point equation! We can use domain theory to solve it.

Beyond pure languages

Effects: control flow (errors), mutability/state, input-output...

An important aspect of programming languages!

Effects: control flow (errors), mutability/state, input-output...
An important aspect of programming languages!

Modelled as a monad T (example: $T(A) \stackrel{\text{def}}{=} (A \times \mathrm{State})^{\mathrm{State}}$)

Effects: control flow (errors), mutability/state, input-output…
An important aspect of programming languages!

Modelled as a monad T (example: $T(A) \stackrel{\text{def}}{=} (A \times \mathrm{State})^{\mathrm{State}}$)

Denotation of a computation: $[\![\Gamma]\!] \to T([\![\tau]\!])$

More semantics

Easter: axiomatic semantic (Hoare Logic and Model Checking)

Easter: **axiomatic semantic** (Hoare Logic and Model Checking)

In the end, the most interesting aspects of semantics is in the **interaction** between different approaches.